Faculty of Science
Master of Statistics and Data Science
Academic year: 2021 - 2022

Support Vector Machines [H02D3a]

Report

by

Khachatur Papikyan r0825613

Assignment 1

1.1 A simple example: two Gaussians

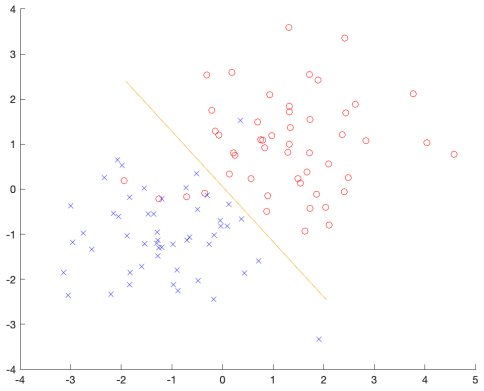

Figure 1: Classification with 2 Gaussians

Discriminant Analysis Classification was used to obtain a line to classify the observations of the given 2 classes. In particular, the linear version of this classifier was used since the data was generated from 2 classes of Gaussians. Thus, since for the Linear Discriminant Analysis the underlying assumption is that the data follows a normal distribution, the choice of this method seemed to be reasonable also taking into account the fact that it is often advisable to use LDA when the data is linearly separable. For the current toy example from Figure 1 as well, it can be stated that the orange line has classified the examples of the red and blue classes quite accurately.

Nevertheless, since the data at hand is not clearly linearly separable because of certain red points overlapping with the distribution of the blue points and vice versa, it can equally be stated that the linear separation is probably not the most optimal choice to make in terms of misclassified examples given the overlapping distributions.

1.2 Support vector machine classifier

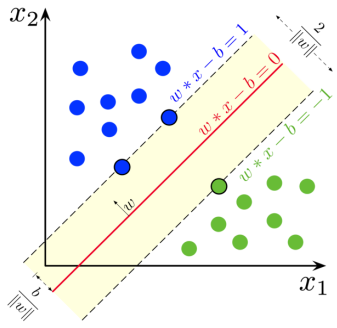

Figure 2: Maximum-margin hyperplane and margins for an SVM trained with samples from two classes. Samples on the margin are called the support vectors.

Support vectors, shown in Figure 2, are data points that are the nearest to the hyperplane separating different classes and influence the position and orientation of the hyperplane. In other words, support vectors are those training points that define the maximum margin of the hyperplane to the data points. When a new point is added that is close to the previous boundary, then it has good chances of becoming a support vector and the closer it is to the decision boundary, the more important it is with regards to the other datapoints. Moreover, it can be that a certain point was not a support vector, but when a point of the opposite class is added somewhere close to that point, then chances are high that this initially unimportant point will become a support vector with high importance. On top of that, Figure 3 suggests that the datapoints situated on the wrong side of the linear decision boundary also gain high importance as support vectors.

All in all, the goal of Support Vector Machines (SVM) is to find a hyperplane (i.e. decision boundaries) in an N-dimensional feature space such that it classifies the data distinctly and

maximizes the margin between different classes in data.
The regularization parameter C is the term that balances the tolerance towards misclassifications, i.e. whether the model is allowed to have a lot of errors or only a few of them. Figures 3 and 4 show that the smaller values of C tend to give a larger margin hyperplane thus allowing more misclassification, whereas with a bigger C it gives a smaller margin, which means that the model becomes very conservative longing for very small misclassification. As a side remark, it can be noted that with a larger value of C the number of support vectors required for the creation of the decision boundary tends to decrease since the further away datapoints do not play a significant role in the creation of the decision boundary anymore.
The parameter sigma is the sigma denominator in the RBF kernel, i.e. the bandwidth of the Gaussian kernel that shows how much the data is spread. Figures 5 and 6 show that the more the data is spread, i.e. the larger the value of sigma is, the more the decision boundary tends to become linear, in the meanwhile, with smaller values of sigma the decision boundary becomes highly nonlinear. In other words, when the sigma value increases, the kernel function becomes less dependent on the differences between the classes, hence making the exponential part to behave linearly which is also depicted in Figure 6 that shows that the decision boundary has now become highly linear with a very large value of sigma.

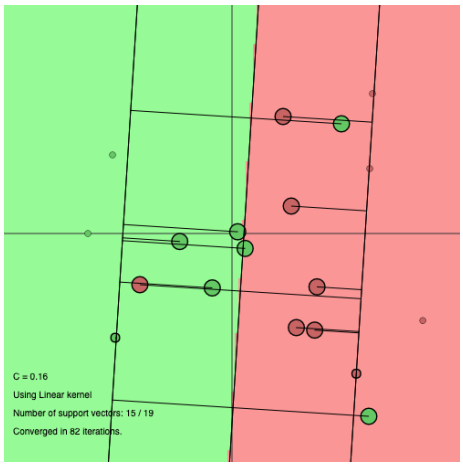

Figure 3: Linear kernel with C = 0.15

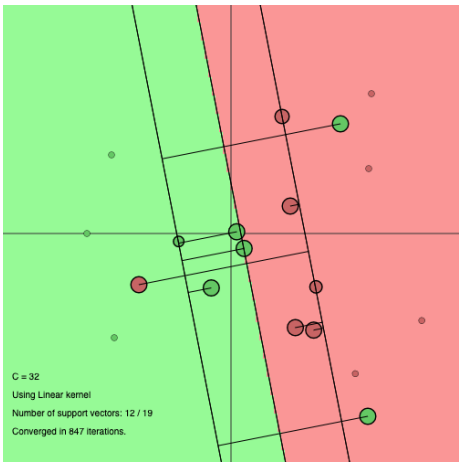

Figure 4: Linear kernel with C = 32

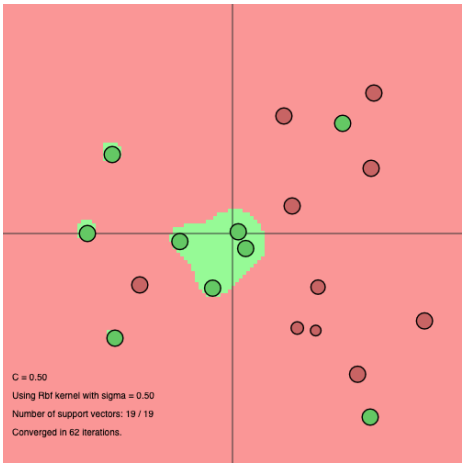

Figure 5: RBF kernel with sigma = 0.5

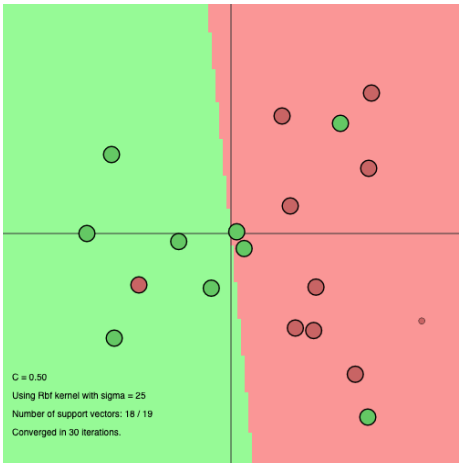

Figure 6: RBF kernel with sigma = 25

1.3 Least-squares support vector machine classifier

1.3.1 Influence of hyperparameters and kernel parameters

For degree-d polynomials, the polynomial kernel is defined as

$$K(x, x_k) = (x_k^T x + \tau)^d \qquad (1)$$

where x and x_k are vectors in the input space and $\tau \geq 0$, which determines the coefficient of the polynomial, is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. The polynomial kernel is used to transform the input space into a higher dimensional space which helps with the better separation of data considering that not everything might be linearly separable in the original input space. In other words, it computes the high-dimensional relationships between the input observations by giving their high-dimensional coordinates in order to later on find a good support vector classifier based on these high-dimensional relationships. In a special case of τ being set to 0, all that the polynomial kernel does is shifting the datapoints further away from each other within the original dimension no matter what the degree of the polynomial is. So the degree of polynomial kernel impacts the transformation and also controls the flexibility of the resulting classifier.

For the degree 1 polynomial kernel (i.e. linear kernel) the misclassification rate was 55% meaning that 11 out of 20 observations on the test set were misclassified. With degree 2 polynomial (a.k.a. quadratic kernel) the results were already much more convincing since only 1 observation was misclassified out of 20. The polynomial kernels of degrees 3-11 turned to be the most accurate ones since they have managed to classify all the test observations correctly. For $d \geq 12$ kernels the test set accuracy was starting to downgrade yielding misclassification rate of e.g. 20% and 50% for degree 20 and 30 polynomials respectively. That being said, it can be stated that because of the nonlinear nature of the data with degree 1 polynomial kernel it was really hard to make accurate classification even on training set as shown Figure 7, whereas with more flexible models the classification accuracy made a significant jump leading to highly accurate classification boundaries as shown in Figure 8. However, from Figure 9 it becomes clear that with $d \geq 12$ kernels the classification boundaries were starting to become too flexible (i.e. learning the training data too well) leading to overfitting and downfalling generalization performance according to the results from the test set.

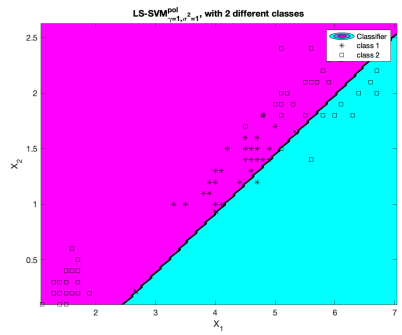

Figure 7: Degree 1 polynomial kernel

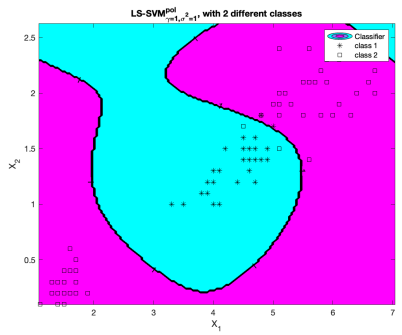

Figure 8: Degree 4 polynomial kernel

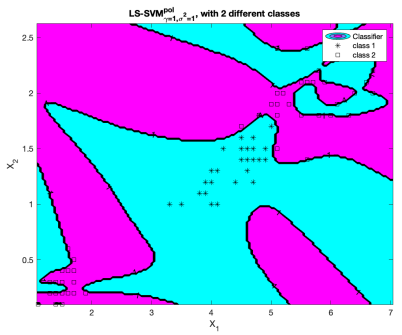

Figure 9: Degree 20 polynomial kernel

The RBF kernel can be defined as

$$K(x, x_k) = exp(-\frac{||x - x_k||_2^2}{\sigma^2}) \qquad (2)$$

where $||x - x_k||_2^2$ may be recognized as the squared Euclidean distance between the two feature vectors.

Different values in the range from 0.01 to 50 were checked for $sig2$ while keeping the gamma value constant at 1. Of these values, the ones between 0.05 and 12 included seemed to constitute a good range for $sig2$ due to the fact of yielding a misclassification rate of 0% on the test data meaning that no test datapoint was misclassified. Figure 10 summarizes these results.

From the definition of the RBF kernel it is clear that the closest observations (a.k.a nearest neighbors) have a lot of influence on how the new observation is classified, and observations that are further away have relatively little influence on the classification. So since certain observations are closer to the new observation, the RBF kernel uses their classification for the new observation. The amount of influence one observation has on another is a function of the squared distance. Gamma, which can be determined by cross validation, scales the squared distance, and thus, it scales the influence. E.g. if $\gamma = 1$ is used for observations that are relatively close to each other and it is compared with that of $\gamma = 2$, then for $\gamma = 2$ this high-dimensional relationship will be smaller. So by scaling the distance, gamma scales the amount of influence two points have on each other. The same way, if $\gamma = 1$ and the two observations are relatively far from each other, then it will result in a number very close to 0 for this high-dimensional relationship. Thus, the further two observations are from each other, the less influence they have on each other.

Therefore, apart from $sig2$, it is also very important to choose a good value for gamma in order not to make the model too constrained with too small values so that it is able to capture the complexity of the data well enough. As such, $sig2 = 5$ was fixed from the above-discussed

range of optimal values for $sig2$ and different values for gamma were considered. Of the values tried, the results suggest that a good range for gamma may potentially be the one between 0.5 and 150, since for the gamma values below 0.5 the model was still making significant errors by misclassifying half of the test observations. The results of this experiment can be consulted in Figure 11.

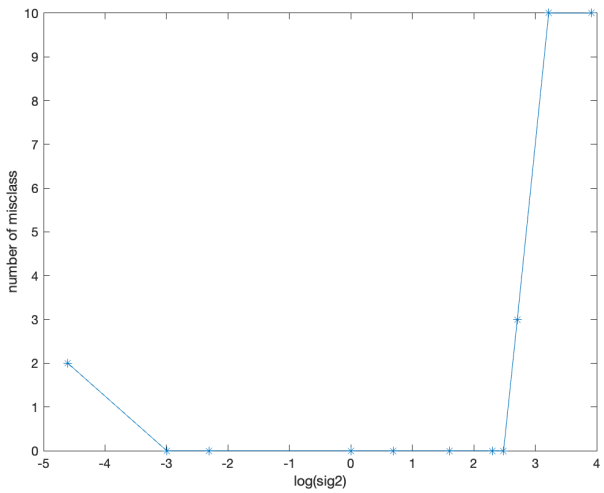

Figure 10: Misclassification rate wrt log(sig2)

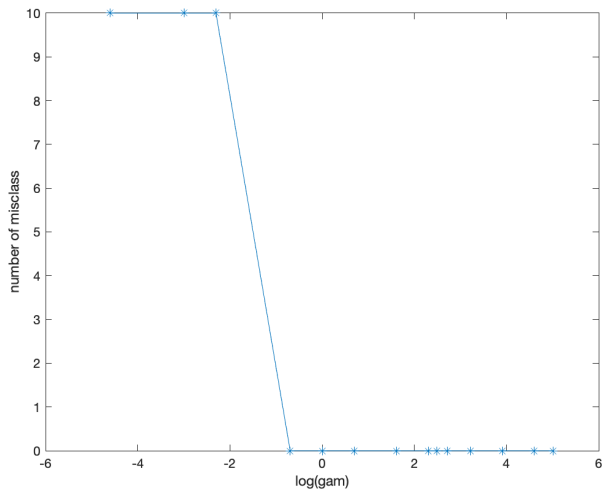

Figure 11: Misclassification rate wrt log(gam)

1.3.2 Tuning parameters using validation

Table 1 shows the misclassification rates of the models with different σ^2 and γ values ranging from 10^{-3} to 10^{3} using three methods for validation, namely: random split, cross validation and leave-one-out cross validation. According to these results, the best performance based on misclassification rates were provided by random split classifying every single observation of the validation set correctly. Despite this fact, it might be a questionable statement to claim this method to be the absolute best since the results of random split might be and in fact are variable from the results of e.g. the second run of parameter tuning with this same method where for certain combinations of parameters the error rates were increased to 0.2 instead of the previous 0s.

On the other hand, the two other methods seem to have yielded quite similar results in terms of the winning combinations of parameters with 0.04 being the lowest misclassification rate achieved. So it can be stated that in a way cross-validation might be preferred over random split. A potential reason is that random split is based on random, yet a very particular subdivision of the training data into training and validation sets as opposed to cross-validation where the validation sets are chosen in a more systematic way making those latter results better reproducible and generalizable in a sense that with random split there might be a certain level of exposure to bias by risking the validation set to be largely influenced by outlying observations. Consequently, cross-validation is deemed to be more robust due to taking into account more variability between the possible splits of training and validation sets. As a matter of fact, it takes into account all the training data via running multiple iterations on data splits helping to take away randomness of the results.

sig2	gam	random split	CV	LOOCV
0,001	0,001	0,55	0,33	0,33
0,001	0,01	0,35	0,33	0,33
0,001	0,1	0,35	0,33	0,33
0,001	1	0,2	0,21	0,18
0,001	10	0,1	0,2	0,18
0,001	100	0,2	0,17	0,18
0,001	1000	0,25	0,21	0,18
0,01	0,001	0,3	0,33	0,33
0,01	0,01	0,3	0,33	0,33
0,01	0,1	0,35	0,2	0,2
0,01	1	0	0,05	0,05
0,01	10	0	0,05	0,05
0,01	100	0	0,08	0,05
0,01	1000	0,05	0,04	0,05
0,1	0,001	0,25	0,33	0,33
0,1	0,01	0,45	0,33	0,33
0,1	0,1	0,05	0,04	0,04
0,1	1	0,05	0,04	0,04
0,1	10	0,05	0,04	0,04
0,1	100	0	0,07	0,04
0,1	1000	0,05	0,05	0,05
1	0,001	0,25	0,33	0,33
1	0,01	0,25	0,33	0,33
1	0,1	0,05	0,05	0,05
1	1	0	0,05	0,05
1	10	0,05	0,05	0,05
1	100	0,05	0,04	0,04
1	1000	0,05	0,04	0,04
10	0,001	0,2	0,33	0,33
10	0,01	0,3	0,33	0,33
10	0,1	0,3	0,33	0,33
10	1	0,2	0,04	0,05
10	10	0,1	0,05	0,06
10	100	0,1	0,06	0,04
10	1000	0,05	0,05	0,05
100	0,001	0,4	0,33	0,33
100	0,01	0,25	0,33	0,33
100	0,1	0,2	0,33	0,33
100	1	0,25	0,33	0,33
100	10	0,45	0,34	0,34
100	100	0,05	0,04	0,05
100	1000	0,05	0,06	0,05
1000	0,001	0,25	0,33	0,33
1000	0,01	0,3	0,33	0,33
1000	0,1	0,25	0,33	0,33
1000	1	0,35	0,33	0,33
1000	10	0,35	0,33	0,33
1000	100	0,35	0,37	0,37
1000	1000	0,35	0,37	0,36

Table 1: Tuning parameters using validation

As a rule of thumb, $k = 10$ is often chosen as a good compromise because it allows 90% of data to be used for training on every run. The choice of k may also depend on the sample size, i.e. if the sample is really small, then selecting a small value for k will be more recommended in order not to have too few observations left for validation during the cross-validation runs. To decide the best value for k, a range of values can be taken and the performance of the model accessed. The value providing the smallest error might be a good candidate for k.

1.3.3 Automatic parameter tuning

Grid Search is the most simple and one of the widely used methods to determine the minimum of a cost function (and hence to optimize the hyperparameters) with possibly multiple optima. At the very heart of this algorithm is evaluating a grid over the parameter space and picking the minimum. However, it takes a lot of computational time and, as such, is the most expensive method to use as it suffers from the curse of dimensionality, i.e. the cost increases exponentially with the number of parameters. The reason for grid search to be slow is that it spends a lot of time investigating hyperparameter settings that are nowhere near the optimal. An alternative and possibly a better solution is the Nelder-Mead simplex algorithm, which is a derivative free optimization method (i.e. does not require any gradient information) doing multidimensional unconstrained nonlinear optimization. It finds a local minimum of a function starting from a random initial point and going through the most likely search direction.

Table 2 shows the results of both of the algorithms at different runs. As it was expected, on average grid search used to take more time to run in comparison with simplex. What is interesting is that for both of the algorithms the optimal combination of hyperparameters was highly variable leading to almost identical values of the cost functions. A possible reason for these differences might be the existence of multiple optima. This might especially be the case with the simplex algorithm since it is essentially solving for the local optimum.

Algorithm	γ	σ^2	$cost$	Runs
Grid Search	101.5983	79.7127	0.03	1
	0.1594	0.5537	0.04	3
	0.6075	0.1079	0.04	5
	0.3461	0.2225	0.04	10
Simplex	0.1226	1.5499	0.04	1
	28.5501	0.1168	0.04	3
	206.0971	0.1178	0.03	5
	178.9488	0.1289	0.03	10

Table 2: Comparison of Grid Search and Simplex algorithms

1.3.4 Using ROC curves

Generally, the ROC curve is created by plotting the sensitivity or the true positive rates (i.e. proportion of positive cases correctly classified as positive) against the false positive rates (i.e. proportion of negative cases wrongly classified as positive) at different thresholds. It helps to measure how well a model distinguishes the classes in a sense that a model has to be able to classify the positive cases as accurately as possible in the meanwhile making as little mistakes as possible in misclassifying negatives as positives. In practice, the ROC curve is computed on the test set rather than on the training set since the primary goal is the generalization performance of a model as opposed to overfitting and memorizing the training data. Therefore, computing the ROC curve on a standalone test data that has neither been used for training the model nor for hyperparameter tuning is more representative in terms of performance expectations from the trained model.

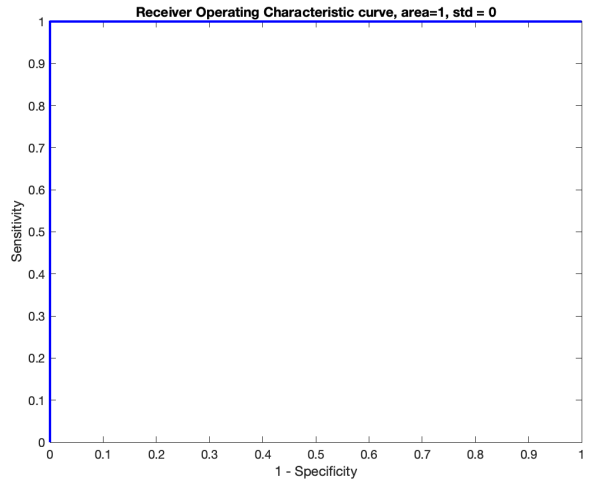

Figure 12: ROC curve

Figure 12 shows the ROC curve generated from the tuned values for γ and σ^2 from the grid search algorithm. The goal was to maximize the area under the ROC curve via maximizing the sensitivity and minimizing the false positive rate. According to the generated curve, the model has been proven to show excellent performance on the test set in this regard since the area under the demonstrated curve is 1. In the worst case scenario when the model is doing no better than randomly assigning labels to the observations this area would be equal to 0.5.

1.3.5 Bayesian framework

The colors in Figure 13 correspond to the probability of the observations in that color area to belong to a certain class. Pink colored areas denote the areas of the dominance of the probability of the observations belonging to the positive class and the blue colored areas denote the areas of the negative class' probability dominance. In other words, the higher the probability of belonging to the positive class is for the observations in a certain area, the closer the color in that area will be to pink and the lower that probability is, the closer the color will be to blue signifying that the observations in that area have very low probability of belonging to the positive class and thus, they have much higher probability to belong to the negative class.

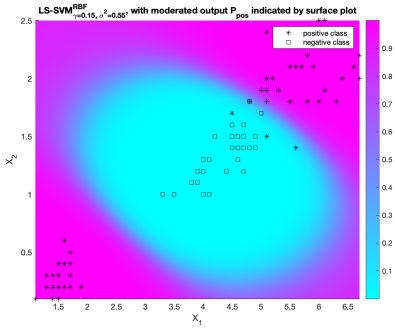

Figure 13: Optimized parameters

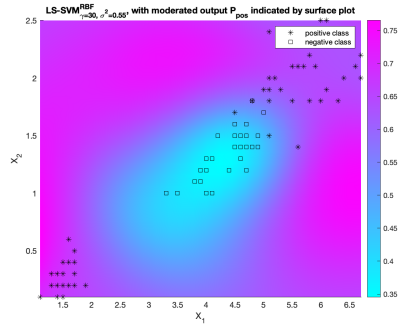

Figure 14: Adjusted γ

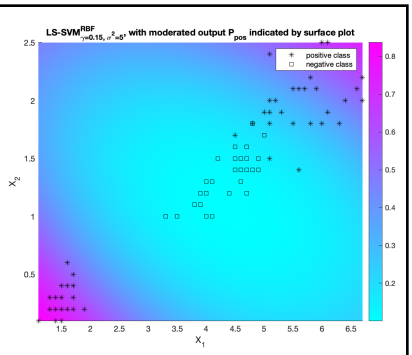

Figure 15: Adjusted σ^2

Figures 14 and 15 demonstrate the impacts of adjusting the parameter values for γ and σ^2. It can be observed that whichever parameter is deviated while keeping the other one constant, the posterior probability boundaries get affected. In particular, Figure 14 shows that by increasing γ while keeping σ^2 constant at its tuned value, the pink coloured area has increased indicating that in those areas the posterior probabilities of the data points to be assigned to the positive class have become higher. In case of holding γ constant at its tuned value and adjusting σ^2, it is observed from Figure 15 that the blue colored area has increased indicating an increase in the dominance of posterior probabilities of the points to belong to the negative class. Moreover, it has to be noted that by further increasing the σ^2 values (e.g. $\sigma^2 = 10$) the whole posterior probability area gets dominated by the negative class and no color assignment gets depicted on the graph as such.

2 Homework problems

Figure 16 shows the Ripley dataset with red circles representing the negative class and the blue circles representing the positive class. From the overlaps in the distributions of the two classes it is clear that the two classes are not linearly separable. Taking into account this fact together with that of the classes following Gaussian distribution, SVM classifier with RBF kernel might potentially be a good candidate for the classification problem at hand.

Since both the Wisconsin breast cancer dataset and the Diabetes dataset are multidimensional, $tsne$ was used to reduce the dimensionality of the data for visualization purposes. According to Figure 17, there is some overlap in the distributions of the classes but not too much. Consequently, SVM classifier with linear kernel may also be a good candidate for the Wisconsin breast cancer dataset apart from the one with RBF kernel. Regarding the Diabetes dataset from Figure 18, the classification in this case may potentially get more troublesome because of the class distributions being highly mixed up.

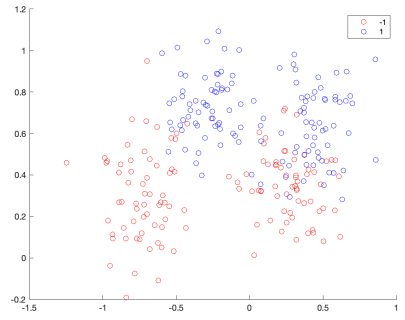

Figure 16: Ripley dataset

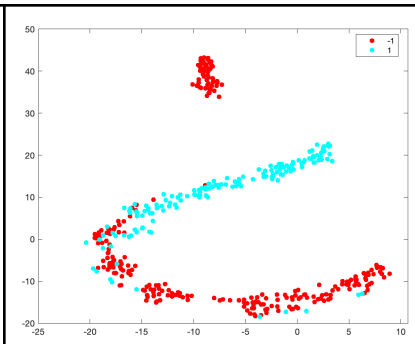

Figure 17: Breast cancer dataset

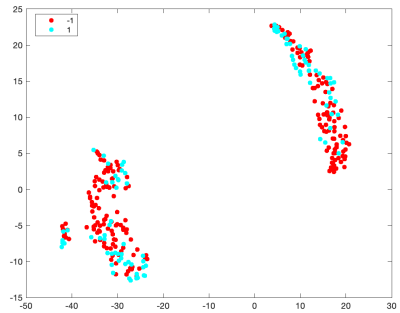

Figure 18: Diabetes dataset

Despite the initial guesses about the potential best performing models, different models (linear, polynomial, RBF kernel) were tried with automatic parameter tuning via simplex algorithm.

Ripley dataset

According to Figures 19, 20 and 21 the best performing model was the one with RBF kernel ($gam = 10.0019$ and $sig2 = 1.02916$) where the AUC computed on the test set was 0.968. It is worth mentioning that this result is also highly coherent with the initial guess. However, the other two models were also very accurate in their predictions.

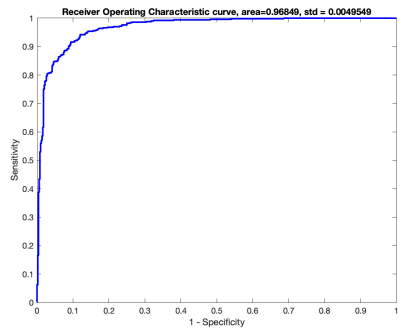

Figure 19: ROC for SVM with RBF kernel (gam=10.0019, sig2=1.02916)

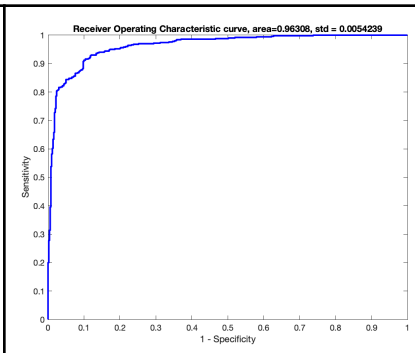

Figure 20: ROC for SVM with Polynomial kernel (gam = 0.12723, t = 2.4087, degree = 5)

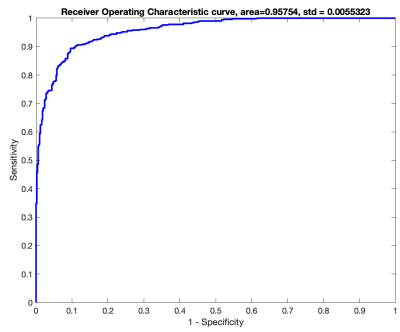

Figure 21: ROC for SVM with Linear kernel (gam = 0.0070996)

Wisconsin Breast Cancer dataset

For the Wisconsin breast cancer dataset as well the best performing model was the one with RBF kernel ($gam = 27.8422$ and $sig2 = 67.1227$). Figure 22 shows that the AUC for this model

was 0.998. Figures 23 and 24 show that the other two models were also highly accurate being only slightly behind the best model in their performances.

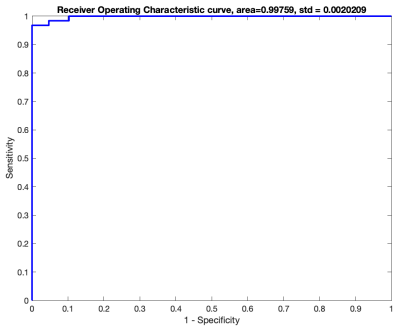

Figure 22: ROC for SVM with RBF kernel (gam = 27.8422, sig2 = 67.1227)

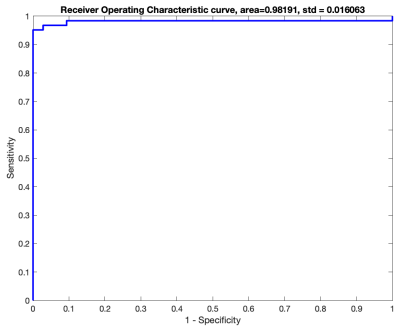

Figure 23: ROC for SVM with Polynomial kernel (gam = 1.922698e-06, t = 185.8068, degree = 3)

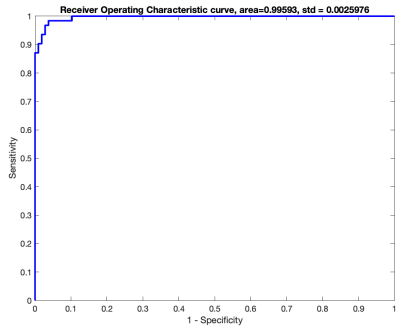

Figure 24: ROC for SVM with Linear kernel (gam = 0.39122)

Diabetes dataset

Things got predictably harder with the Diabetes dataset. The best performing model here was also the SVM classifier with RBF kernel. The AUC for this model from Figure 25 was 0.846. An almost equivalent performance was achieved with the linear kernel as well as shown in Figure 27. In the meanwhile, from Figure 26 it is clear that the performance of the model with polynomial kernel was very unsatisfying since the AUC of 0.58 signifies that the model was only slightly better than a random model blindly assigning class labels to the examples.

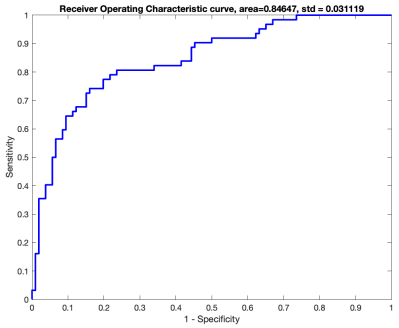

Figure 25: ROC for SVM with RBF kernel (gam = 7.831134, sig2 = 602.128)

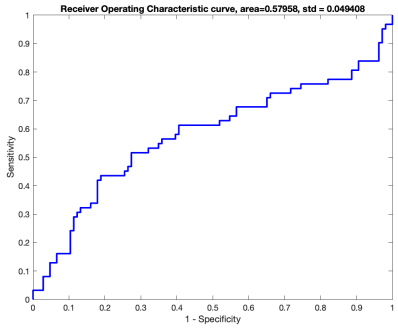

Figure 26: ROC for SVM with Polynomial kernel (gam = 33.0137, t = 2.04071, degree = 5)

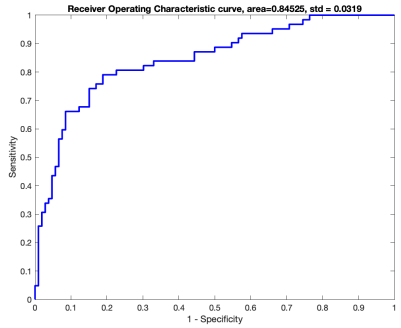

Figure 27: ROC for SVM with Linear kernel (gam = 0.016899)

In conclusion, the models for the Ripley and Wisconsin breast cancer datasets demonstrated very good performances, whereas the ones for the Diabetes dataset were not really satisfying. Considering the complexity of the latter dataset, it might be worth trying e.g. a tree based method since the decision trees might potentially be able to come up with more flexible decision boundaries. Moreover, to provide a certain level of robustness and to enhance the generalization performance of the model, it might perhaps be a good idea to utilize the wisdom of crowds by combining multiple decision trees in an ensemble method such as the random forest classifier.

Assignment 2

1.1 Support vector machine for function estimation

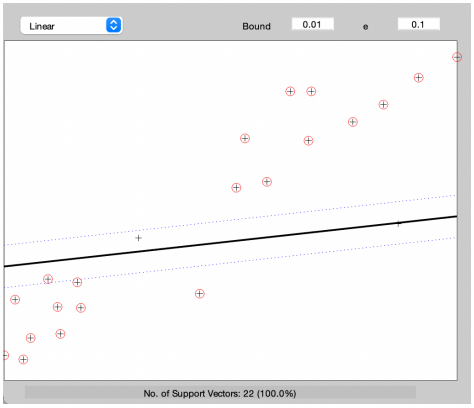

Figure 28: Linear kernel with ϵ=0.1

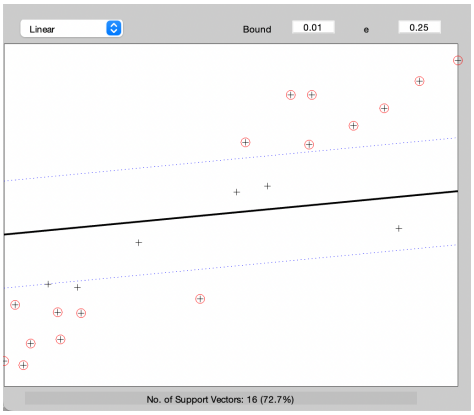

Figure 29: Linear kernel with ϵ=0.25

The value of ϵ controls the width of the ϵ-insensitive area that is used to fit the training data, i.e. it defines the margin of tolerance for errors. Consequently, if ϵ is taken very small, then the tolerance for errors will be small leading to a very accurate estimation as opposed to the case with very large ϵ leading to a very wide ϵ -tube and thus a rather rough approximation. That is why the number of support vectors decreases with the increase in the width of the ϵ-tube and vice versa since the better the required fit is, the more support vectors may be needed to accomplish that task.

Bound	ϵ	Number and % of support vectors
0.01	0.1	22 (100.0%)
0.01	0.25	16 (72.7%)
0.01	0.5	8 (36.4%)

Table 3: Results from deviating ϵ

Figures 28, 29 and Table 3 demonstrate the above discussed results where the bound is fixed at the value 0.01.

In general, the goal of function estimation with SVMs is to find a function $f(x)$ that deviates from y_n by a value no greater than ϵ for each training point x, and at the same time is as flat as possible. As such, bound is a trade-off between the flatness of the function $f(x)$ and the amount to which the deviations larger than ϵ are tolerated. The flatness of the function $f(x)$ refers to finding the minimal norm value $w^T w$. The larger the value of bound is, the less flat the function $f(x)$ is and the more punishment is given to the errors.
A certain kind of factual support for the discussion above is depicted and can be found in Figures 30 and

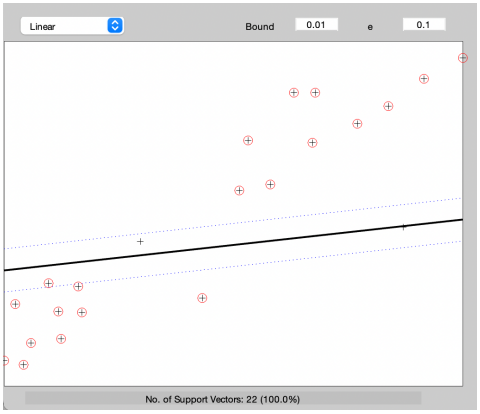

Figure 30: Linear kernel with bound=0.01

31 as well as in Table 4.

Bound	ϵ	# and % of SVs
0.01	0.1	22 (100%)
0.1	0.1	15 (68.2%)
1	0.1	11 (50%)
10	0.1	11 (50%)
100	0.1	11 (50%)

Table 4: Results from deviating bound

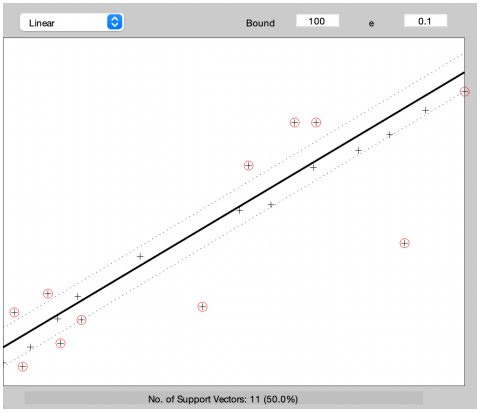

Figure 31: Linear kernel with bound=100

The sparsity property comes in when the number of support vectors needed for accomplishing the function approximation task constitutes the smaller portion of the training datapoints. In other words, due to the sparsity property (i.e. the case when $\alpha_k = 0$ from the dual representation) only the observations for which $\alpha_k \neq 0$ become support vectors (thus indicating that these observations are more important with regards to the task) instead of utilizing every single observation as such. According to Tables 3 and 4, in terms of ϵ, small changes are enough to provide sparsity. In case of deviations in bound, larger increments seem to be needed to make this property visible.

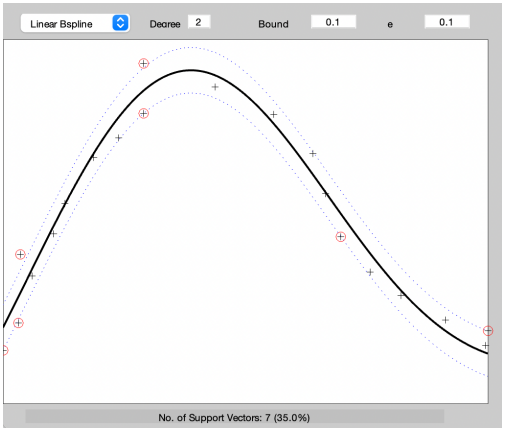

Figure 32: Linear B-Spline kernel of degree 2 with bound and ϵ of 0.1

In order to decide which kernel was better suited for the more challenging dataset from Figure 32, both bound and ϵ were fixed at 0.1 and the models with different kernels were tried. The results from these kernels are depicted in Table 1. According to it, the best performing model was the one with Linear B-Spline kernel of degree 2, the fit of which can be consulted in Figure 32 as well. A B-spline function is a combination of flexible bands that is controlled by a number of points called control points, creating smooth curves. These functions enable the creation and management of complex shapes and surfaces using a number of points. That is why the model with Linear B-Spline kernel was a good fit for the data at hand.

Kernel	Number and percentage of support vectors
Linear	16 (80%)
Polynomial (Degree 5)	13 (65%)

Gaussian RBF (Sigma 0.7)	13 (65%)
MLP (Scale 10)	18 (90%)
Linear Spline	17 (85%)
Linear B-Spline (Degree 2)	7 (35%)
Trigonometric Polynomial (Degree 1)	17 (85.0%)
Exponential RBF (Sigma 0.7)	15 (75%)

Table 5: Results with different kernels

Perhaps the main point in which the SVM regression is different from the classical least squares fit is the loss function. The loss function of classical least squares is the one that finds the line minimizing the distance from the observations, whereas the SVM regression uses a function where the observations that are within the ϵ-tube do not produce any error and only the ones outside that range produce errors.

1.2 A simple example: the sinc function

1.2.1 Regression of the sinc function

Table 6, where γ is the regularization parameter determining the trade-off between the fitting error minimization and smoothness of the estimated function and σ^2 is the kernel function parameter of the RBF kernel, shows the mean squared errors (MSE) of function estimation with different combinations of γ and σ^2 parameter values. As it can be observed from these results, with increments in γ the models start fitting the data slightly better. Nevertheless, it is also evident that the fit becomes relatively poorer when the value of σ^2 increases since this change seems to be accompanied with increases in MSE.

	$\sigma^2 = 0.01$	$\sigma^2 = 1$	$\sigma^2 = 100$
$\gamma = 10$	0.011291	0.034141	0.13679
$\gamma = 10^3$	0.012041	0.013029	0.11676
$\gamma = 10^6$	0.012684	0.010641	0.11083

Table 6: MSE for different combinations of γ and σ^2

Figures 33, 34 and 35 demonstrate the declines in the fit of the function when increasing the values of σ^2 while keeping γ fixed at 10.

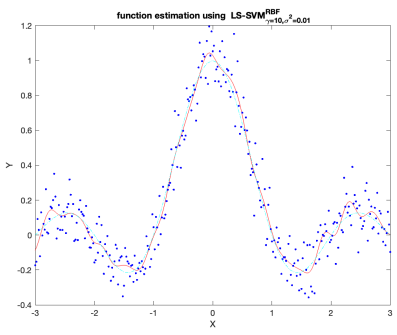

Figure 33: $\gamma = 10,\ \sigma^2 = 0.01$

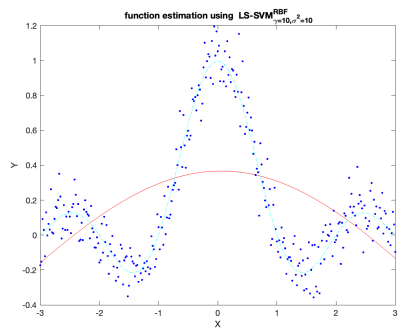

Figure 34: $\gamma = 10,\ \sigma^2 = 10$

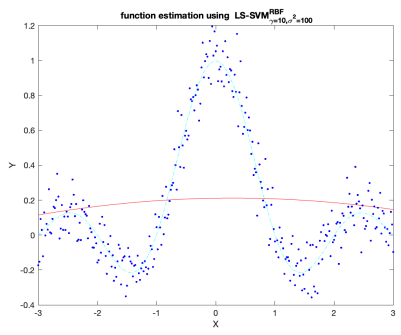

Figure 35: $\gamma = 10,\ \sigma^2 = 100$

Considering the settings of the problem at hand, it may be hard to claim a particular pair of hyperparameters to be the optimal one since the hyperparameters' space is predefined before optimizing it and the best pair is to be selected from that predetermined space only, whereas there are no formal guarantees that the optimal pair was included in the predefined hyperparameter space.

According to Table 7, the tuned results produced by both of the methods particularly in terms of the MSE and σ^2 values are quite similar for the given dataset.

Algorithm	γ	σ^2	*cost*	Runs
Simplex	36.8153	0.3685	0.0107	1
	41.4906	0.3854	0.0106	5
	190.675	0.4472	0.0107	10
Grid Search	34.836	0.3751	0.0106	1
	113.5917	0.4131	0.0109	5
	206.2897	0.4135	0.0108	10

Table 7: Tuned hyperparameters with Simplex and Grid Search algorithms

1.2.2 Application of the Bayesian framework

The basic result from the Bayesian framework for the LS-SVM is the derivation of the probability that the data points are generated by the given model. This criterion is called the posterior probability and is expressed as a number. There are 3 variants: the posterior with respect to the model parameters α and b, the posterior with respect to the regularization constant γ and the posterior with respect to the choice of the kernel and its parameter σ^2. The corresponding costs are obtained via taking the negative logarithms of the posteriors and neglecting all constants.

Figure 36 demonstrates in a schematic way how parameter tuning works using the Bayesian framework. As shown there, the likelihood at a certain level equals the evidence at the previous

level. In this way, by gradually integrating out the parameters at different levels, the subsequent levels are linked to each other. In particular, the starting points are the prior distributions of level 1 parameters, i.e. α and b, whose posterior distributions are tried to estimate, which, on their turns, are then maximized to get the solutions for support values α and the bias term b. The maximization of posteriors is done in order to obtain values of α and b that give the highest probability of the datapoints to be generated by the given model. Similar approach is retained for the hyperparameters of the second and third level as well. The function bay_lssvm is used to estimate the costs associated with the posteriors of the parameters of each of the 3 inference levels correspondingly. $crit\_L1$, $crit\_L2$ and $crit\_L3$ represent these costs, i.e. the cost associated with the posteriors of α and b for level 1, the cost associated with the posterior of γ for level 2 and the cost associated with the posterior of σ^2 for level 3 since the RBF kernel was used. For the problem at hand, $crit\_L1 = 139.1514$, $crit\_L2 = 792.1836$ and $crit\_L3 = -720.211$. The function $bay\_optimise$ is used to optimize the posterior probabilities of model parameters and hyperparameters with respect to the 3 different levels in Bayesian inference. The results are depicted in Figure 37.

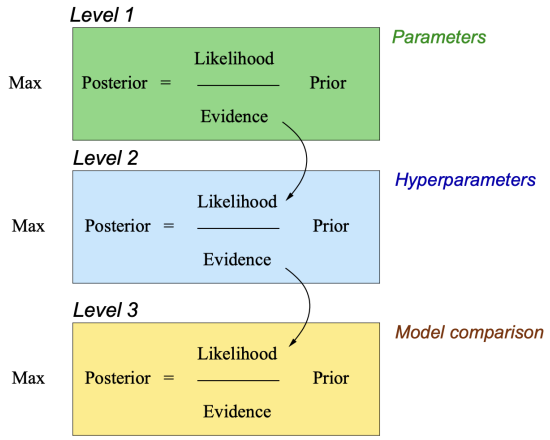

Figure 36: Bayesian Inference

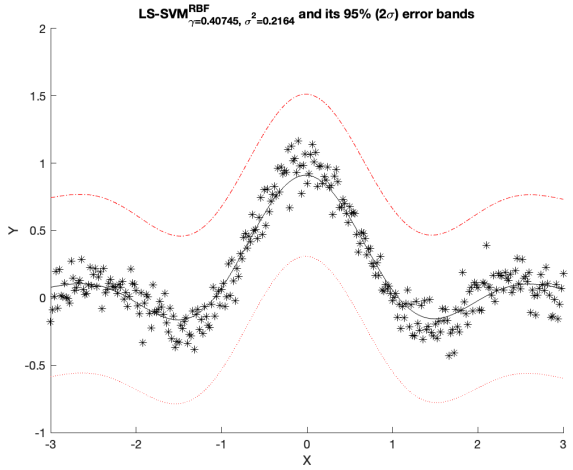

Figure 37: Bayesian inference results and error bars

1.3 Automatic Relevance Determination

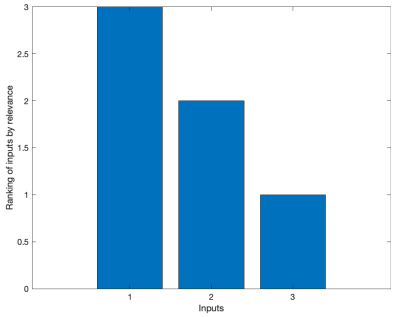

Figure 38: Rankings by relevance

Figure 38 shows the results of the given three dimensional input selection task where the inputs are ordered according to their relevance with input 1 being the most relevant one.
A similar way of accomplishing this task using the $crossvalidate$ function instead of the Bayesian framework may consist of starting from the initial set of inputs, temporarily removing an input at a time and using cross validation to check the MSE. Then, if the training set with a certain input temporarily removed produces lower MSE, that input may be considered of a lower relevance and may even

be permanently disregarded if its relevance is lower than a certain benchmark.

1.4 Robust regression

The comparison of Figures 39 and 40 shows that the non-robust regression model is very sensitive to the outliers in the data and thus fails to fit the data well enough, whereas the robust regression model tackles this challenge quite well in the view of not giving too much specific attention to the outliers at the time of estimation.

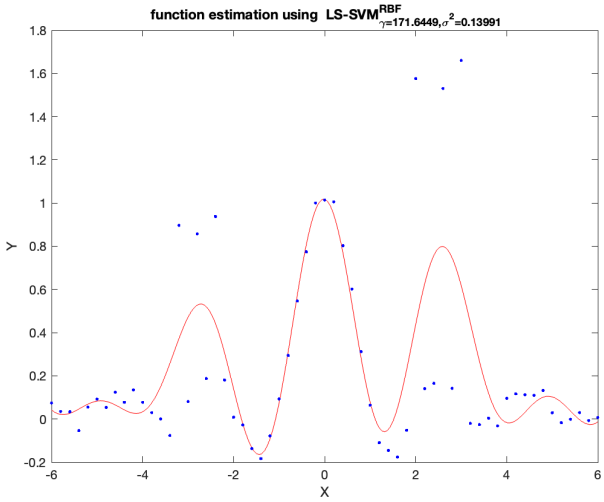

Figure 39: Non-robust regression model

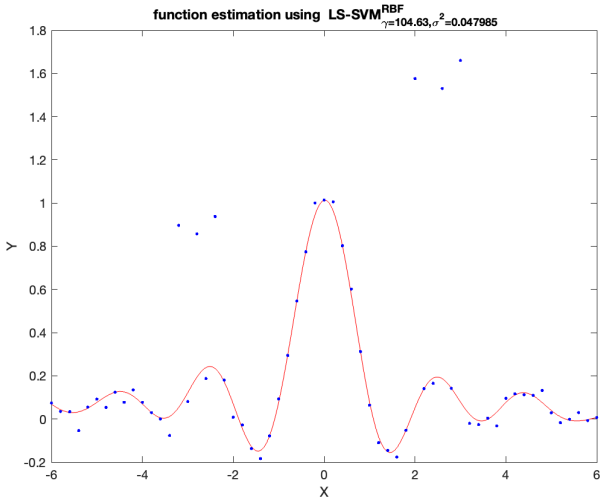

Figure 40: Robust regression model

The mean absolute error (MAE) is preferred over the classical mean squared error (MSE) in this case since it treats all the errors the same whereas with MSE the larger errors would become even larger because of squaring them and more attention would be drawn towards minimizing them while fitting the model thus resulting in a poor fit because of the sensitivity to the outliers (i.e. with MSE the higher values of residuals would be penalized more than with MAE).

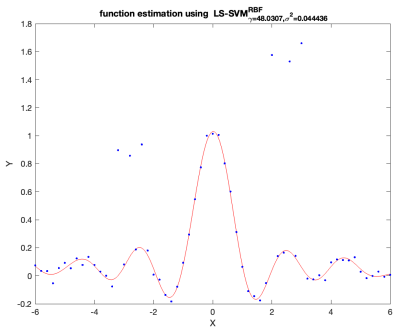

Figure 41: Robust regression model with Hampel weight function used

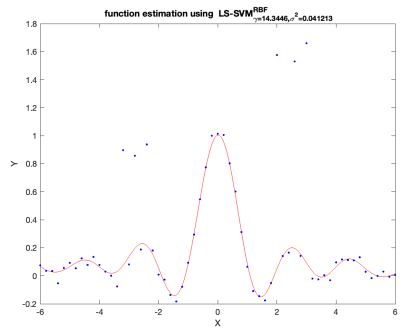

Figure 42: Robust regression model with Logistic weight function used

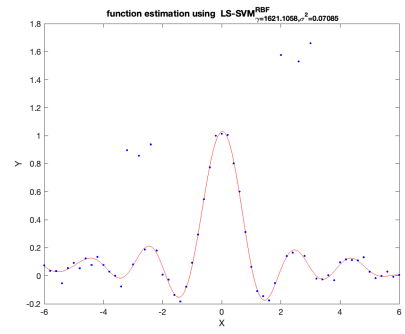

Figure 43: Robust regression model with Myriad weight function used

As it can be observed from Figures 40, 41, 42 and 43, the robust regression models with all the four of the weighting functions fit the data similarly well disregarding the outliers. Following the above-mentioned, it comes not as a surprise that the results are quite similar in terms of costs. However, it was also observed that with Hampel and Huber weight functions the convergence was much faster than with Logistic or Myriad in terms of the number of iterations needed.

2 Homework problems

2.2 Logmap dataset

Figure 44 demonstrates the predicted time series (in red) along with the actual ones (in black). Based on that, one may observe that the predicted values are opposite to the actual values. That is to say, if the actual value is low, then the predicted value is high and vice versa. Consequently, the quality of the prediction is not satisfying.

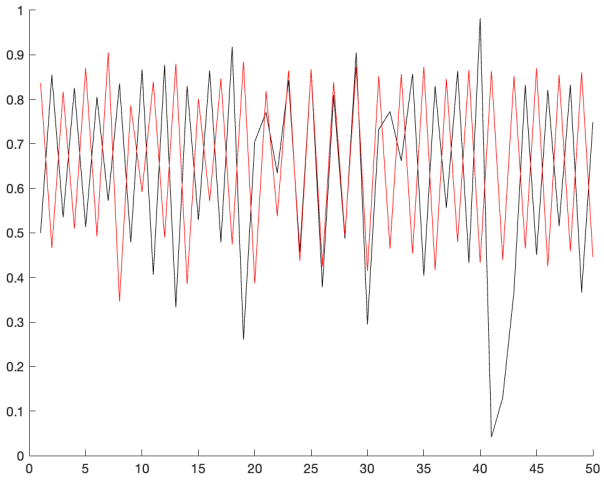

Figure 44: TS prediction with order=10

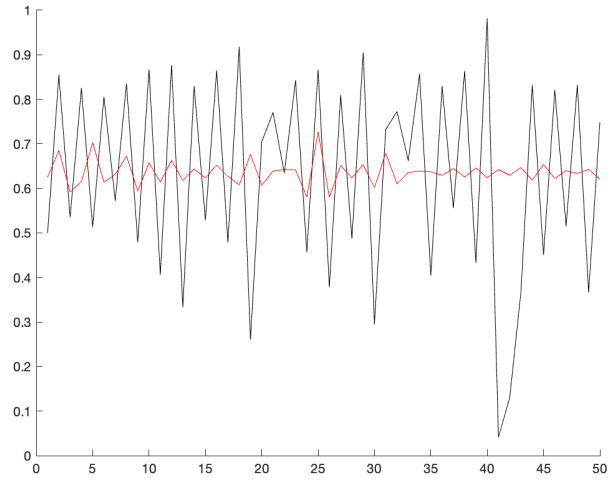

Figure 45: TS prediction with order=30

The following strategy may be implemented to tune the 3 parameters, namely γ, σ^2 and $order$:

1. Create an array of $order$ parameters ([1;5;10;20;30;40;50;60;70;80;90;100] in this case).
2. For each value of $order$, tune the parameters γ and σ^2 using cross validation and make predictions on the test set using the tuned set of parameters. Calculate the MSE by comparing the actual time series with the predicted one.
3. Compare the MSE for different orders and select the order which gives the minimum MSE.

According to the above-mentioned strategy, $order = 30$ yielded the minimum MSE. These results can be consulted in Figure 46 and Table 8.

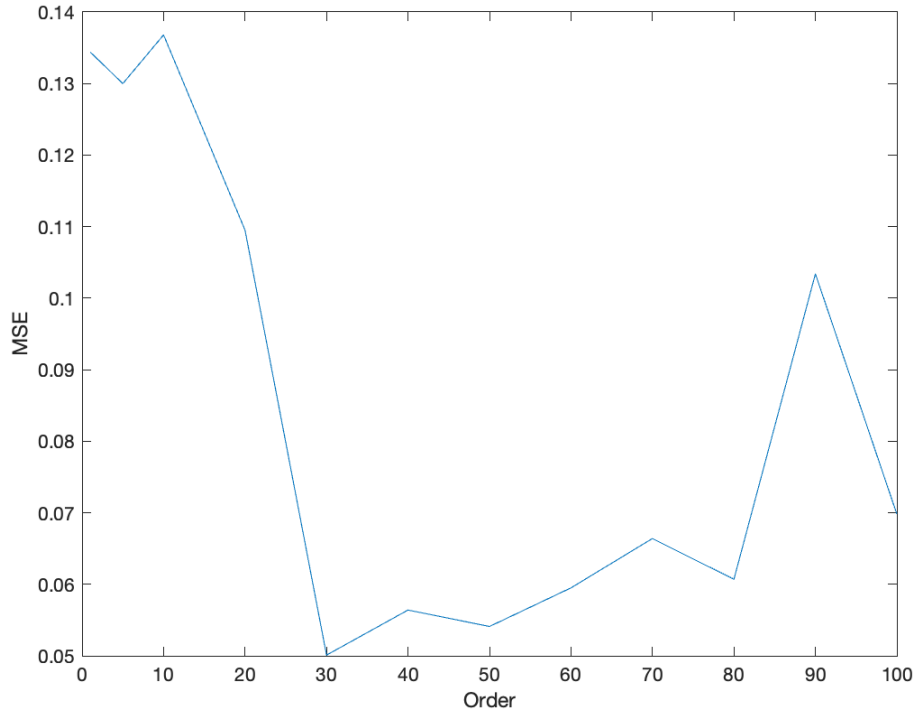

Figure 46: MSE per Order

Order	MSE
1	0.1344
5	0.13
10	0.1368
20	0.1095
30	0.0501
40	0.0564
50	0.0541
60	0.0595
70	0.0664
80	0.0607
90	0.1034
100	0.0698

Table 8: MSE per Order

Figure 45 shows the actual time series as well as the prediction with $order = 30$. As it can be observed from the figure, the model still does not really perform well since it highly underestimates the values even though it behaves considerably differently from the previous model when it was predicting almost the opposite for every single value.

2.3 Santa Fe dataset

The time series prediction for $order = 50$ (in red) along with the actual time series (in black) are depicted in Figure 47. Evidently, $order = 50$ does not sound like a good choice for the utilized auto-regressive model. In particular, up until around time point 50 the predictions are quite accurate whereas in terms of the later points the fit significantly degrades. This is also accompanied with a high value of MSE as a way of support for the above drawn conclusion.

Even though using the performance of the recurrent prediction on the validation set might be closer to the real life application, it might still be more sensible to try to use everything that is already known when training the model since it is more stable and explainable. In other words, if there is observed data available, obviously in no way the recurrent prediction can be more accurate than what has already been observed since the task of prediction is exactly about predicting something that is as close to the observed data as possible.

Just like in Section 2.2, here as well cross-validation was used in order to fine tune the 3 parameters of the model. For the parameter $order$ values in the range from 10 to 100 were tested with 10 point increments and the corresponding MSE values are depicted in Figure 49 and Table

9. The best performing model turned out to be the one with $order = 40$. The results from the implementation of this model can be consulted in Figure 48. According to these results, the fine tuned model indeed seems to perform better than the previous one. Nevertheless, even for the obtained results there is still some room for improvement in order to better fit the second half of the series.

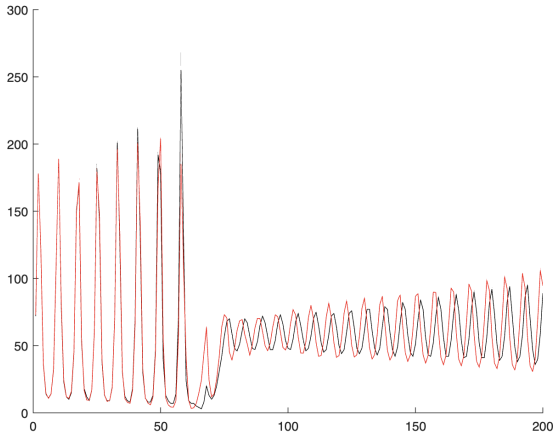

Figure 47: TS prediction with order=50

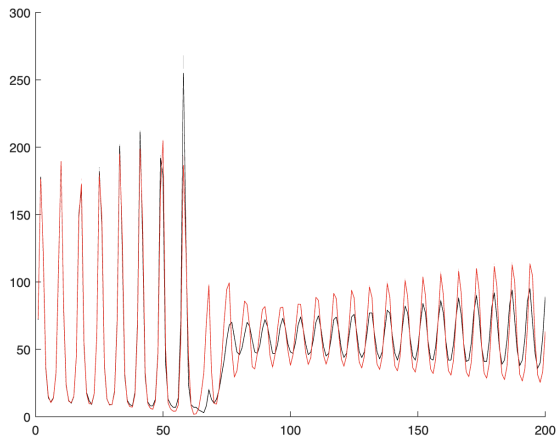

Figure 48: TS prediction with order=40

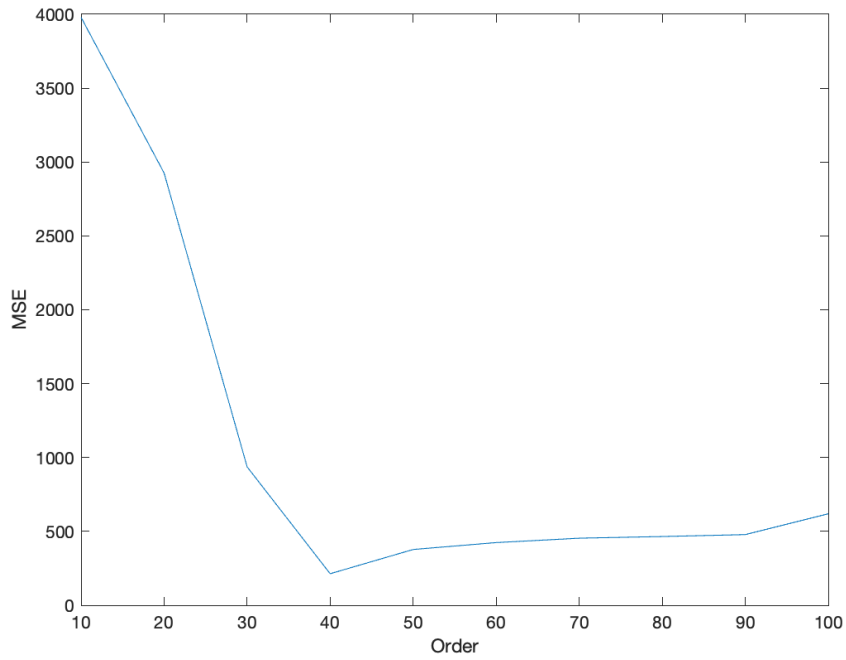

Figure 49: MSE per Order

Order	MSE
10	3979.17
20	2925.59
30	936.67
40	213.56
50	377.53
60	424.31
70	453.99
80	465.25
90	478.33
100	619.23

Table 9: MSE per Order

Assignment 3

1.1 Kernel principal component analysis

The idea of denoising basically consists of finding the most important aspects in the dataset and using them to reconstruct the main structure of the original dataset as accurately as possible. As such, PCA can be considered a good technique to accomplish this task since it tries to find the directions in the input space with maximal variance through choosing the largest eigenvalues with their corresponding eigenvectors. This way, the most important components shaping the dataset are preserved and through the use of this information bottleneck the main structure of the original dataset is attempted to reconstruct based on these few most important components. So the general idea is that the main structure in the dataset is captured only by the first n_c components and the remaining components just capture the noise in the data.

By increasing the number of principal components (Figures 50-53) and, in particular, making it even equal to the dataset size, the denoised points become as close as possible to the actual points meaning that they start capturing the noise as well instead of just capturing the main structure. As a side effect of this, the original noisy dataset may get reconstructed.

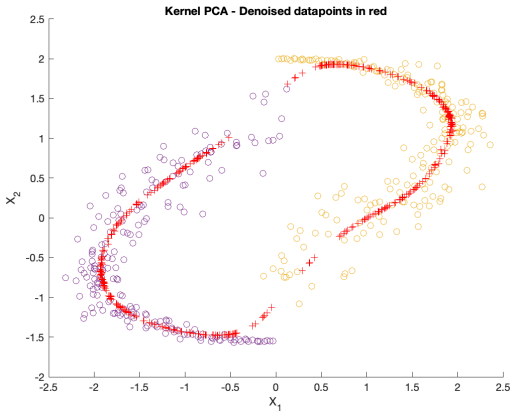

Figure 50: KPCA with 6 components

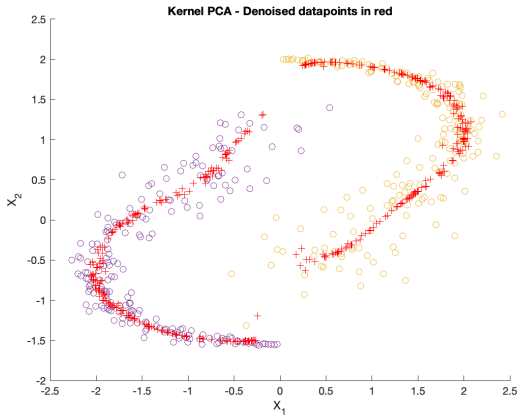

Figure 51: KPCA with 10 components

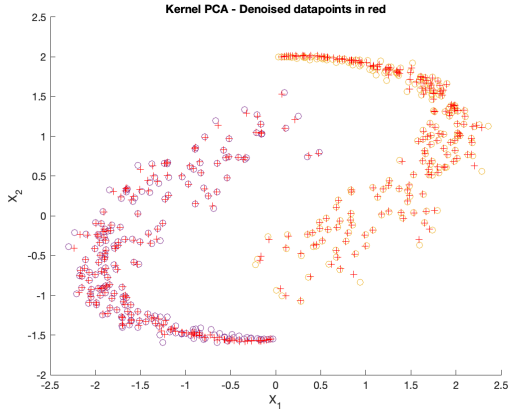

Figure 52: KPCA with 25 components

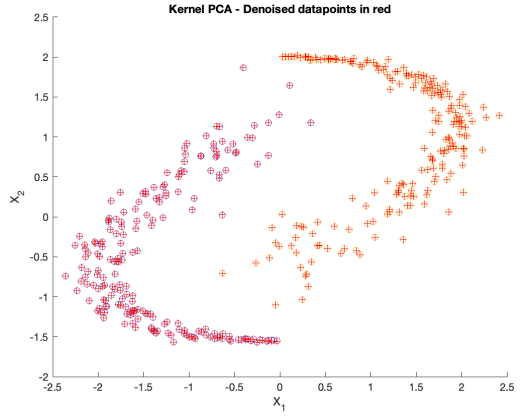

Figure 53: KPCA with 50 components

Linear PCA is a technique that reduces the dimension of the data through the use of the principal components, where each principal component explains the variance in the data with the first component explaining the highest amount of variance. It is a well-known method for feature extraction that does a linear mapping of the data from high-dimensional space to low-dimensional space in such a way that the variance of the data in the low-dimensional space is maximized. Nevertheless, linear PCA cannot be directly applied to non-linear cases.

Kernel PCA is an extension of linear PCA that creates a non-linear version of the method via applying the kernel trick. Thus, it is capable of constructing a non-linear mapping that maximizes the variance of the data. In other words, kernel PCA first maps the original input space into high-dimensional feature space using the kernel trick and then does the PCA in the high-dimensional feature space. It can be stated that linear PCA in the high-dimensional feature space corresponds to non-linear PCA in the original input space.

As it can be observed from Figures 54 and 55, it comes not as a surprise that in the view of the non-linear nature of the original dataset, linear PCA performs significantly worse compared to kernel PCA. Moreover, it fails to capture the main structure of the data.

The number of principal components that can be obtained by linear PCA is equal to the dimension of the input space, whereas the kernel PCA allows them to be greater than that dimension. Consequently, unlike the linear PCA, it is not merely a dimensionality reduction technique.

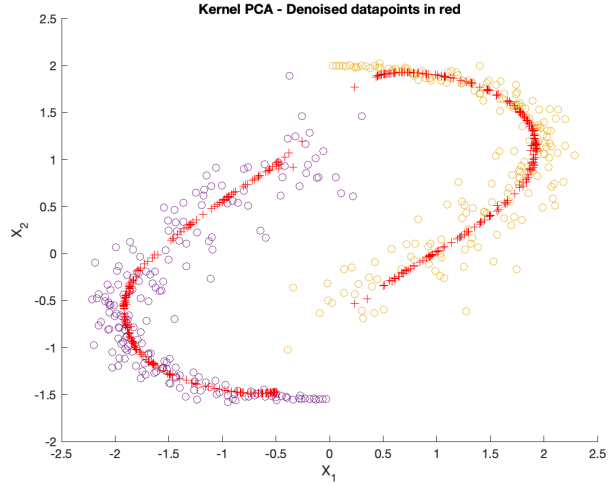

Figure 54: Kernel PCA

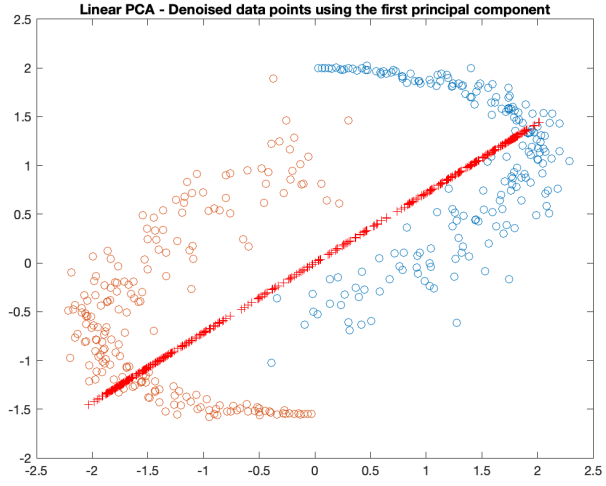

Figure 55: Linear PCA

For the dataset at hand, the number of components, the hyperparameter and the kernel parameters can be fine tuned via the following technique:

1. Create an array consisting of values corresponding to the potential number of principal components.
2. For each of such values, optimize the hyperparameter and kernel parameters via cross-validation or the like.

3. Calculate the corresponding reconstruction errors.
4. Select the value for the number of components that gives the minimum reconstruction error in its combination with the optimal values of the hyperparameter and kernel parameters from step 2.

1.2 Spectral clustering (optional)

Clustering, in general, is aimed at grouping the datapoints into smaller subgroups based on certain similarity measures so that within the group the datapoints are more similar than between the groups. However, a problem may arise when this grouping is not trivial. In particular, it is possible that the datapoints that have to be in the same group are not situated close to each other (e.g. distributed spherically) and in such a situation the classical clustering algorithms such as k-means will have hard times distinguishing these groups since the best that they will do is to behave like a cookie cutter by hardcutting the dataset. That is exactly where spectral clustering may come in handy. In spectral clustering, the datapoints are treated as nodes of the graph. The nodes are then mapped to a low-dimensional space so that the datapoints that belong to the same group can be close to each other in order to be separated by a traditional clustering algorithm. This projection of the data can be done by making use of the eigenvectors of a Laplacian matrix (which is obtained by rescaling the kernel matrix) derived from the data to create groups of data points that are similar. In this context, the kernel function acts as a similarity measure between two data points. The number of the first k eigenvectors' eigenvalues after which there appears the first large gap between the eigenvalues of the Laplacian matrix will be the number of clusters expressed in the data.

The main difference between classification and spectral clustering is that classification is a supervised learning method whereas spectral clustering belongs to the family of unsupervised learning algorithms meaning that with spectral clustering the datapoints are unlabelled and the goal is to come up with a reasonable number of clusters and ways to "label" or "classify" the datapoints into one of these clusters, whereas with classification the datapoints are already labeled (i.e. the true labels are known) and the goal is to build a model that is as accurate as possible in classifying them into their true labels.

The choice of the σ^2 value influences the clarity of the clusters. As demonstrated in Figures 56-63, very small and very large values of σ^2 have led to unclear and messy clusters. In particular, the values in the range from 0.001 to 0.02 seem to have been giving clear lines in the projections' chart resulting in good clustering whereas the ones far from the above-mentioned range seem to be producing much less clear clusters.

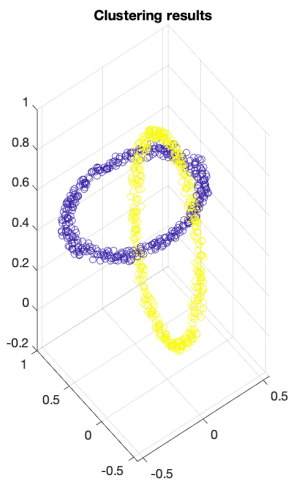

Figure 56: Spectral clustering with $\sigma^2 = 0.001$

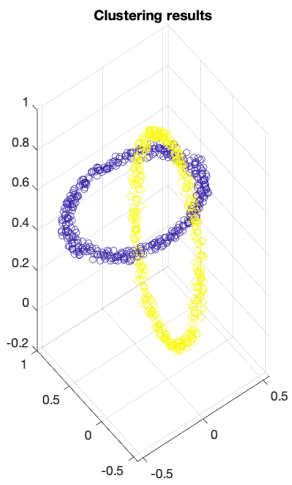

Figure 57: Spectral clustering with $\sigma^2 = 0.005$

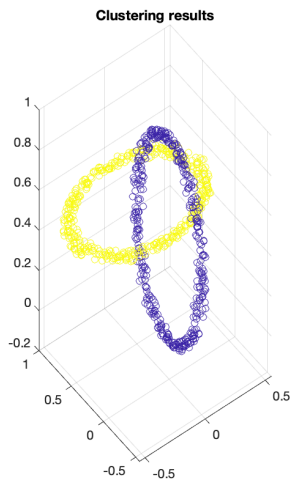

Figure 58: Spectral clustering with $\sigma^2 = 0.01$

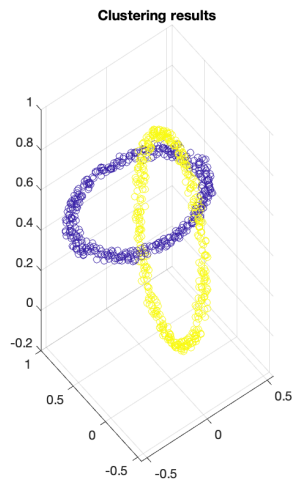

Figure 59: Spectral clustering with $\sigma^2 = 0.02$

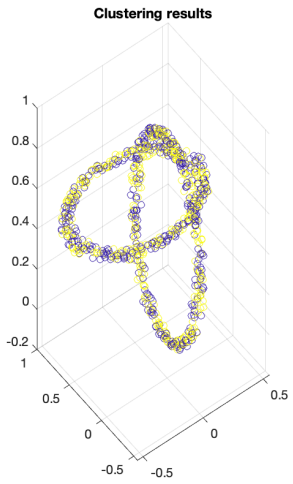

Figure 60: Spectral clustering with $\sigma^2 = 0.00000001$

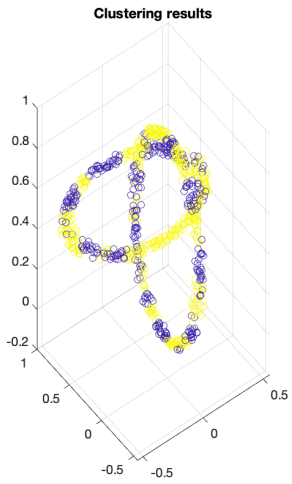

Figure 61: Spectral clustering with $\sigma^2 = 0.00005$

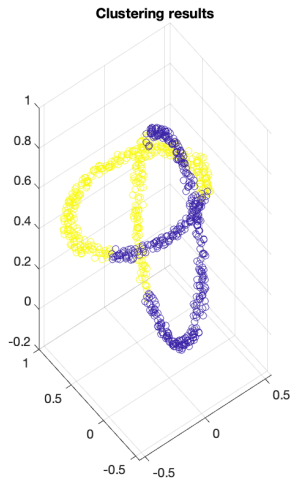

Figure 62: Spectral clustering with $\sigma^2 = 0.1$

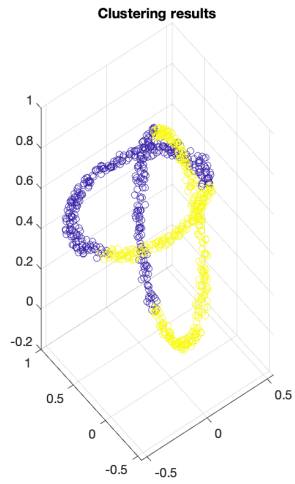

Figure 63: Spectral clustering with $\sigma^2 = 10$

1.3 Fixed-size LS-SVM

Generally, the choice between solving the primal or the dual problem boils down to evaluating the influence of the dataset size in comparison with that related to the dimensionality of input space. In particular, with many datapoints it is better to solve the primal problem and with many inputs solving the dual problem is more recommended. That is to say, the dual formulation is more advantageous when working with large-dimensional input space or when the dimensionality of the input space is higher than the size of the dataset. On the other hand, it is hard to solve the dual problem for a very large dataset since an NxN square matrix will have to be computed in that case (i.e. the sample size becomes too large making the implementation of

the standard dual setting harder) where N is the number of datapoints in the dataset. This is where the fixed-size LS-SVM comes to help via computing an approximation of the non-linear mapping in order to perform estimations directly in primal space. The fixed-size LS-SVM allows to build a large scale non-linear regression model in a primal space by using only a subsample of the selected support vectors from the dataset.

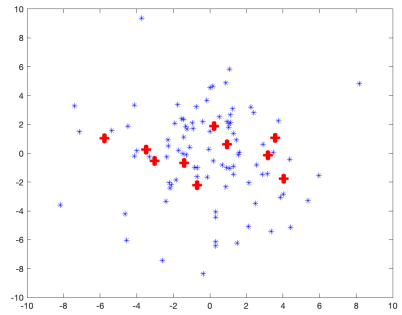

Figure 64: $\sigma^2 = 0.01$

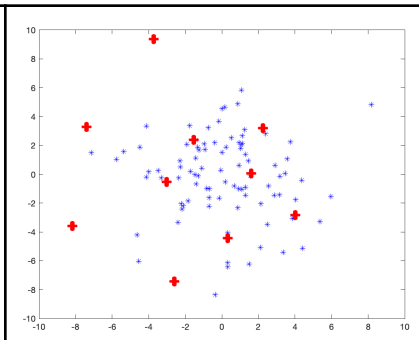

Figure 65: $\sigma^2 = 1$

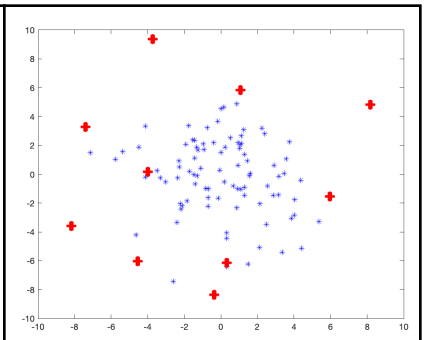

Figure 66: $\sigma^2 = 100$

Figures 64, 65 and 66 show the results for different values of σ^2 and based on them it can be observed that the positions of the support vectors (i.e. red crosses) used to define the structure of the data set has changed with the changes in the values of σ^2. That being said, in comparison with the smaller values of σ^2, for the larger values of it the red crosses are more dispersed potentially concentrating more on describing the general shape of the data. For the smaller values of σ^2, on the contrary, the support vectors are more inside the data cloud thus describing the details within the data. The algorithm to make the corresponding method work can be consulted in Figures 67 and 68.

Algorithm:

1. Given N training data
2. Choose a working set with fixed size M (i.e. M support vectors) (typically $M \ll N$).
3. Randomly select a SV x^* from the working set of M support vectors.
4. Randomly select a point x^{t*} from the training data and replace x^* by x^{t*}. If the entropy increases by taking the point x^{t*} instead of x^* then this point x^{t*} is accepted for the working set of M SVs, otherwise the point x^{t*} is rejected (and returned to the training data pool) and the SV x^* stays in the working set.
5. Calculate the entropy value for the present working set. The quadratic Renyi entropy equals $H_R = -\log \frac{1}{M^2} \sum_{ij} \Omega_{(M,M)_{ij}}$.

Figure 67: Fixed-size method: using quadratic Renyi entropy (1)

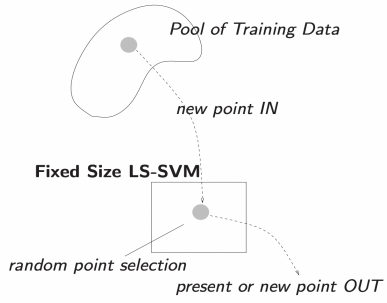

Figure 68: Fixed-size method: using quadratic Renyi entropy (2)

As it can be observed from Figure 69, with fixed-size LS-SVM the produced test errors are lower in comparison with that of the l_0-approximation. Nevertheless, it can also be noticed that the variability in these errors is higher for the results from fixed-size LS-SVM, though even with this

fact taken into account the errors with fixed-size LS-SVM are still lower overall. The number of support vectors is 16 and is the same for both of the approaches and for all the runs. In regards to computational time, the one for fixed-size LS-SVM seems to be slightly higher. The median value of computational time for fixed-size LS-SVM is 3.645 and for l_0-approximation it is equal to 3.64. With both of the approaches 1 outlier was produced in terms of computational time, that being 5.27 for the former and 5.23 for the latter and that is still lower compared to the former.

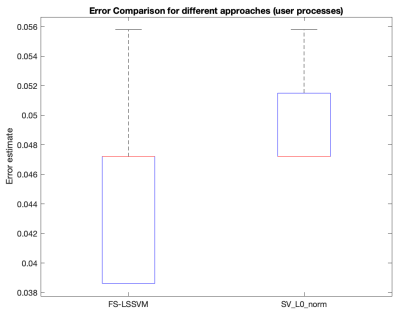

Figure 69: Test error comparison

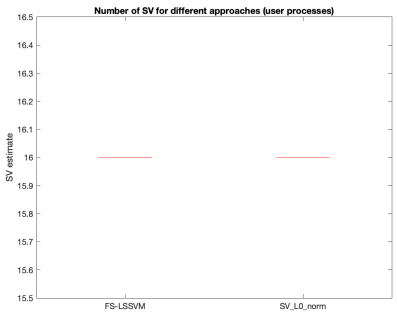

Figure 70: Comparison of number of support vectors

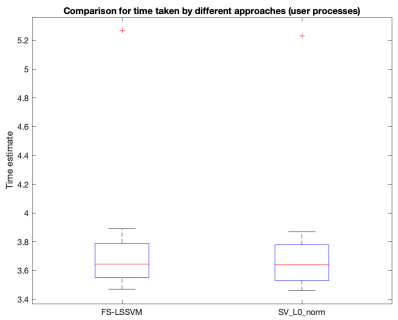

Figure 71: Time comparison

2 Homework problems

2.1 Kernel principal component analysis

Based on the visual inspection of Figures 72 and 73, it can be derived that kernel PCA seems to be performing better in comparison with linear PCA no matter what the predefined number of principal components is. Moreover, it can also be observed that with the increase in the number of principal components, the results from kernel PCA get ameliorated, whereas those from linear PCA degrade because of starting to heavily capture the noise in the data.

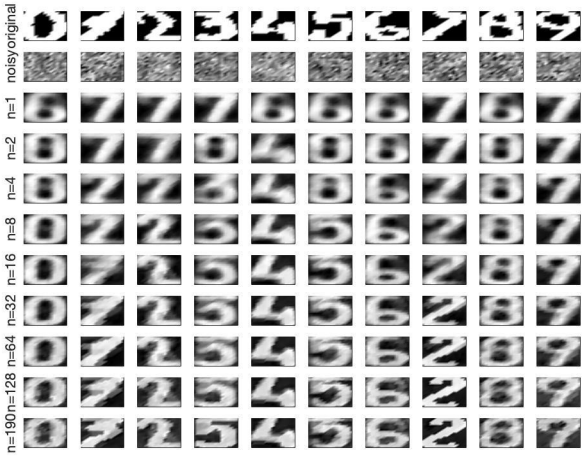

Figure 72: Results with kernel PCA

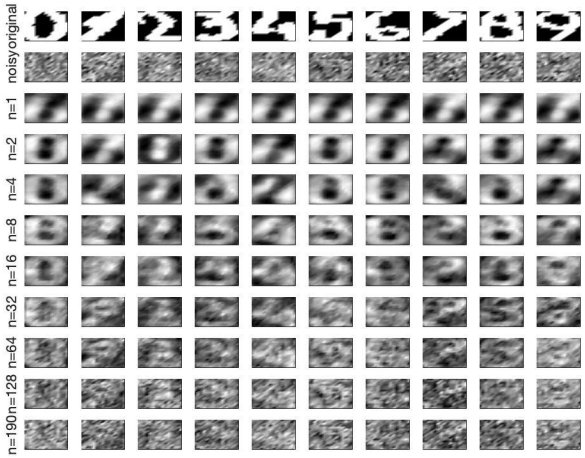

Figure 73: Results with linear PCA

Figures 74-76 demonstrate the reconstruction results from kernel PCA with changed $sigmafactor$ parameter for equispaced values in logarithmic scale. From these figures it follows that for large values of $sigmafactor$ and, as a consequence, σ^2, kernel PCA starts performing worse, becoming unable to reconstruct clear digits. Similar problem arises with very small values of σ^2 where only the noise was captured and reconstructed as a result as seen in Figure 74. With $sigmafactor = 0.07$ the digits are much more clearly visible and sterile, but certain digits are not reconstructed correctly.

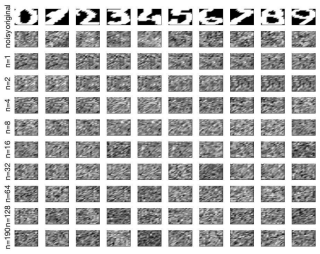

Figure 74: sigmafactor=0.007

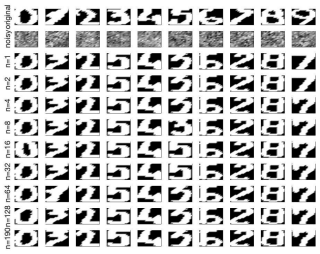

Figure 75: sigmafactor=0.07

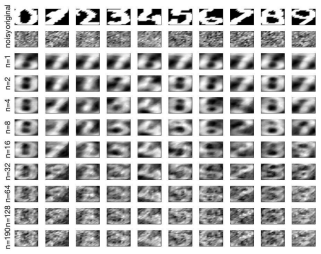

Figure 76: sigmafactor=7

By comparing the reconstruction errors (i.e. MSE scores) between training and the two validation sets, the optimal results seem to be achieved by fixing the $sigmafactor$ at 0.7 and taking 5 principal components. It was also observed that based on these results the model was having less difficulties in correctly reconstructing the numbers as opposed to the previous cases. However, there were still problems with distinguishing the number 7 which was still misreconstructed as 2.

2.2 Fixed-size LS-SVM

2.2.1 Shuttle (statlog) dataset

The Shuttle dataset consists of 58000 observations. In terms of dimensionality, it contains 9 independent attributes all of which are numerical. Since approximately 80% of the data belongs to class 1, the default accuracy or the expected classification performance is 80%. Nevertheless, the goal is to obtain an accuracy of 99 - 99.9%. The examples in the original dataset were in time order, and this time order could presumably be relevant in classification. However, this was not deemed relevant for statlog purposes, so the order of the examples in the original dataset was randomized. The original 7 labels were regrouped into 2 labels, where the original values of 1 were kept as such and the others were recoded into -1. As it can be seen from Figure 77, even though the decision boundary is not really linear, the datapoint clouds belonging to the same class are quite well-concentrated, so it is indeed expected to have a good classification performance.

In terms of computational time and number of support vectors used, both of the methods seem to have similar performance. For fixed-size LS-SVM as well as for l_0-approximation 52 support

vectors were required to tackle the classification task (see Figure 79). While performing very similarly timewise, it took slightly less time for fixed-size LS-SVM to accomplish its mission as depicted in Figure 80 (228.425 vs 228.61). Despite the similarities in regards to computational time and number of support vectors, the error produced by fixed-size LS-SVM is lower than the one from l_0-approximation even with the larger variance and interquartile range taken into account for the results of fixed-size LS-SVM which can be consulted in Figure 78.

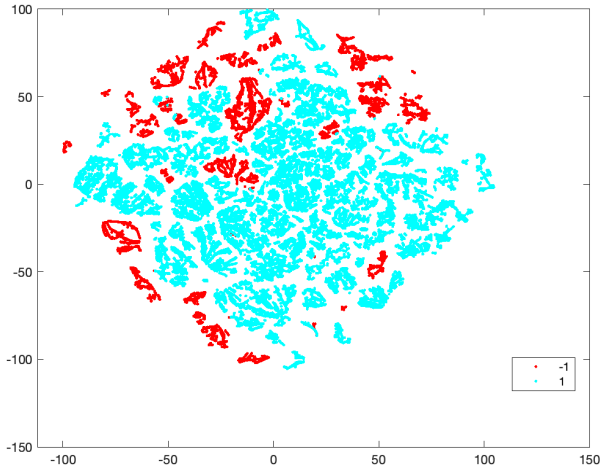

Figure 77: Shuttle dataset

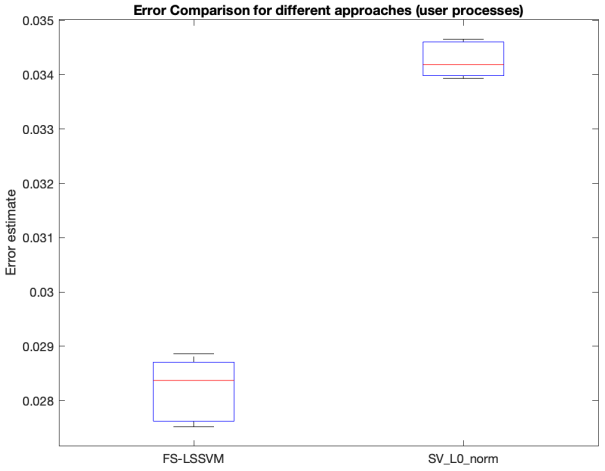

Figure 78: Error comparison

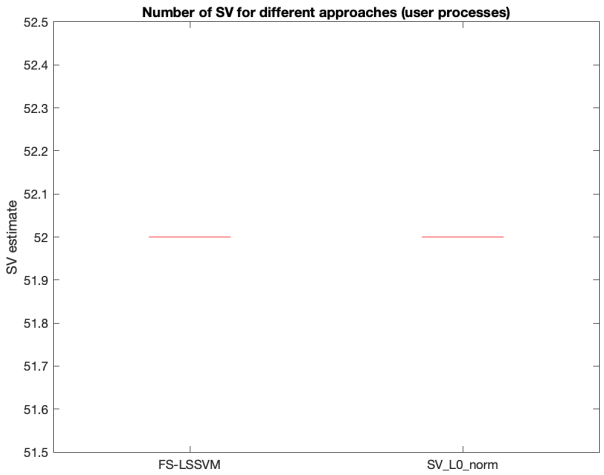

Figure 79: Comparison of number of SVs

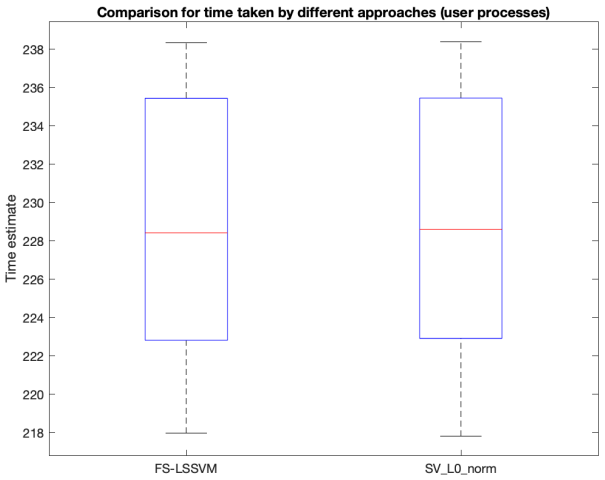

Figure 80: Time comparison

2.2.2 California dataset

```
:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude
```

Figure 81: Attribute information

The California housing dataset contains data drawn from the 1990 U.S. Census and consists of 20640 observations on 9 variables. The dependent variable is ln(median house value). Both the independent and dependent variables are numerical.

The attribute information is depicted in Figure 81. A block group is the smallest geographical unit for which the U.S. Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). According to Figure 82, the median income is a distribution with a long tail. It means that the salary of people is more or less normally distributed but there are some people getting a high salary. Regarding the average house age, the distribution is more or less uniform. The target distribution has a long tail as well. In addition, a threshold-effect for high-valued houses is added: all houses with a price above 5 are given the value 5. Focusing on the average rooms, average bedrooms, average occupation, and population, the range of the data is large with unnoticeable bin for the largest values. It means that there are very few high values, perhaps outliers.

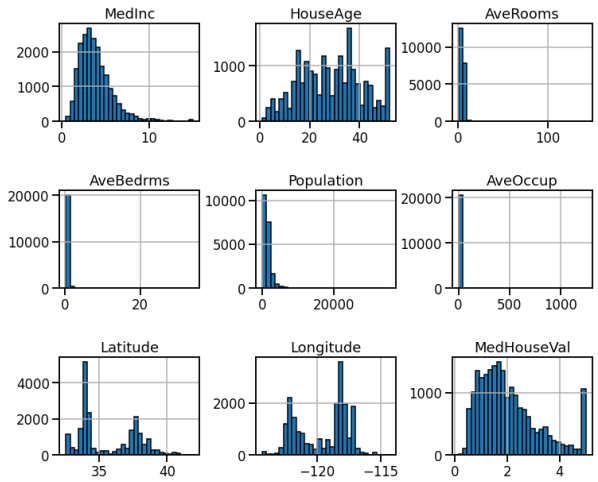

Figure 82: Distribution of features of California dataset

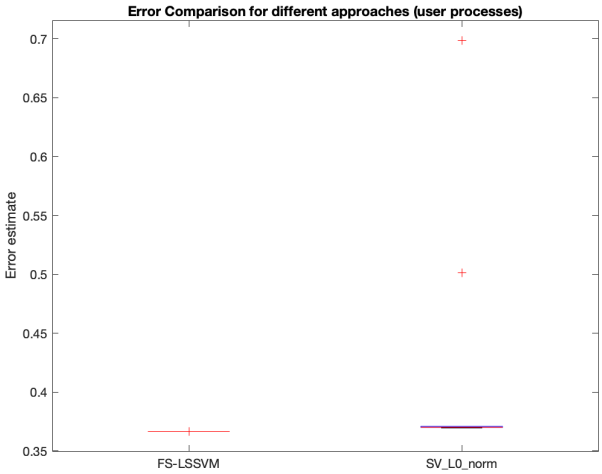

Figure 83: Error comparison

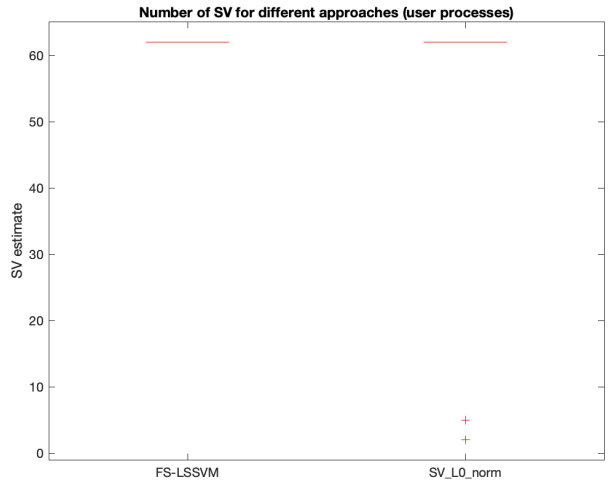

Figure 84: Comparison of number of SVs

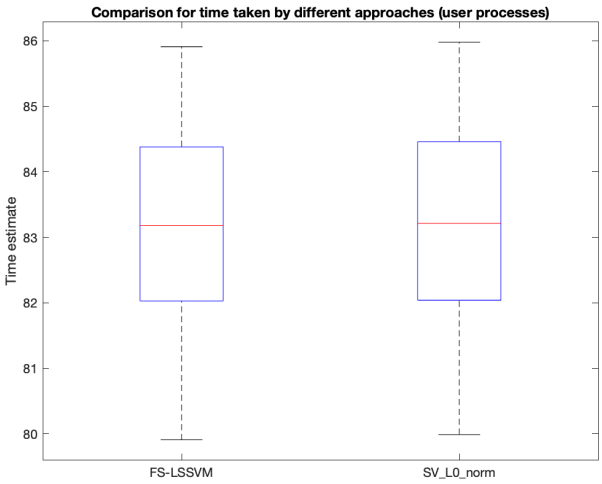

Figure 85: Time comparison

Figures 83-85 show that both of the models were quite similar in terms of errors (0.366 vs 0.3703), number of support vectors (62) and computational time (83.18 vs 83.215), with fixed-size LS-SVM performing slightly better again.

References

[1] Suykens et al., Least Squares Support Vector Machines, 2002
[2] Suykens et al., LS-SVMlab Toolbox User's Guide version 1.8, 2011 (https://www.esat.kuleuven.be/sista/lssvmlab/downloads/tutorialv1_8.pdf)
[3] Suykens, Support Vector Machines: Methods and Applications [H02D3a] course slides, 2019
[4] MathWorks documentation: Discriminant Analysis Classification (https://nl.mathworks.com/help/stats/discriminant-analysis.html, https://nl.mathworks.com/help/stats/create-and-visualize-discriminant-analysis-classifier.html)
[5] Scholarpedia: Nelder-Mead algorithm (http://var.scholarpedia.org/article/Nelder-Mead_algorithm)
[6] Wikipedia: Radial basis function kernel (https://en.wikipedia.org/wiki/Radial_basis_function_kernel)
[7] MathWorks documentation: Understanding Support Vector Machine Regression (https://nl.mathworks.com/help/stats/understanding-support-vector-machine-regression.html)
[8] Wikipedia: B-spline (https://en.wikipedia.org/wiki/B-spline)
[9] Fleshman, Spectral Clustering, 2019 (https://towardsdatascience.com/spectral-clustering-aba2640c0d5b#:~:text=Spectral%20clustering%20is%20a%20technique,non%20graph%20data%20as%20well.)
[10] StatQuest with Josh Starmer: Support Vector Machines (https://statquest.org/video-index/)
[11] Scikit-learn documentation: California housing dataset (https://inria.github.io/scikit-learn-mooc/python_scripts/datasets_california_housing.html)
[12] Statlog (Shuttle) Data Set (https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle))
[13] Wikipedia: Polynomial kernel (https://en.wikipedia.org/wiki/Polynomial_kernel)
[14] Starmer, The StatQuest Illustrated Guide To Machine Learning, 2022
[15] Cristianini et al., An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, 2000
[16] Scikit-learn documentation: Ensemble methods (https://scikit-learn.org/stable/modules/ensemble.html#forest)