Support Vector Machines: Report

Thibeau Wouters

June 2023

Contents

All code, figures and data generated for this assignment can be found here.

1 Classification

In this first assignment, we will explore the basics of SVMs and LS-SVMs in the context of classification problems.

1.1 Simple example: two Gaussians

We start off with a simple classification problem where the instances of the two classes, denoted $+$ and $-$, are drawn from two Gaussian distributions over the (x, y)-plane, having the same covariance matrix Σ and mean vectors $\boldsymbol{\mu}_+ = (1, 1)$ and $\boldsymbol{\mu}_- = (-1, -1)$ respectively. Using the knowledge of the distributions, we can determine the optimal decision boundary for this problem. In lecture 1, we saw that the decision boundary is determined by

$$d_+(\boldsymbol{x}) = d_-(\boldsymbol{x})\,, \tag{1.1}$$

where d_i is the discriminant function for the two classes. For a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ, we saw that the discriminant function takes the form

$$d_i(x) = -\frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{\mu}_i\right)^T \Sigma^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i) - \frac{1}{2}\log|\Sigma| + \log P(\mathcal{C}_i)\,, \quad i \in \{+, -\}\,. \tag{1.2}$$

with $P(\mathcal{C}_i)$ representing the prior class probabilities. In our case, the covariance matrices are both equal to the identity matrix, such that the discriminant function is simplified to

$$d_i(x) = -\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2^2}{2} + \log P(\mathcal{C}_i)\,, \quad i \in \{+, -\}\,. \tag{1.3}$$

Substituting this into Eq. (1.1), we derive that the decision boundary is given by

$$y = \frac{\mu_{x,+} - \mu_{x,-}}{\mu_{y,-} - \mu_{y,+}}x + \frac{\mu_{x,-}^2 - \mu_{x,+}^2 + \mu_{y,-}^2 - \mu_{y,+}^2}{2(\mu_{y,-} - \mu_{y,+})} + \frac{1}{\mu_{y,-} - \mu_{y,+}}\log\left(\frac{P(\mathcal{C}_-)}{P(\mathcal{C}_+)}\right)\,, \tag{1.4}$$

using the notation $\boldsymbol{\mu}_i = (\mu_{x,i}, \mu_{y,i})$ for $i \in \{+, -\}$. For our example, with mean vectors $\boldsymbol{\mu}_+ = (1, 1)$ and $\boldsymbol{\mu}_- = (-1, -1)$, the slope equals -1 and the first constant term vanishes. Hence, the decision boundary is

$$y = -x - \frac{1}{2}\log\left(\frac{P(\mathcal{C}_-)}{P(\mathcal{C}_+)}\right)\,. \tag{1.5}$$

In the assignment, we have to sample 50 points from the first class and 51 points from the second class, which is a negligible difference such that the decision boundary is well approximated by the line $y = -x$. This decision boundary is shown in Figure 1.1. A classifier using this decision boundary is optimal in the sense that it minimizes the probability of misclassification, which follows from the fact that the equation for the discriminant function is derived from the posterior distribution, *i.e.*

$$d_i(x) \propto p(c|\mathcal{C}_i)P(\mathcal{C}_i)\,, \tag{1.6}$$

where p represents a Gaussian distribution in our case.

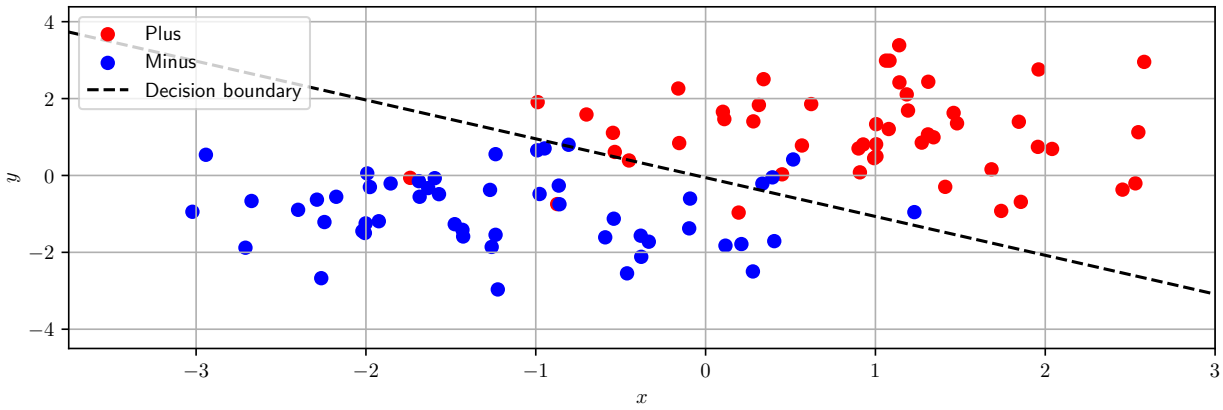

Figure 1.1: Classification problem from two Gaussian distributions with identical covariance matrix.

1.2 Support vectors and hyperparameters

A generic SVM classifier for a binary classification problem can be written as

$$y(\boldsymbol{x}) = \texttt{sign}\left[\sum_{k=1}^{N} \alpha_k y_k K(\boldsymbol{x}, \boldsymbol{x}_k) + b\right] . \tag{1.7}$$

For the construction of the classifiers, support vectors are important. Support vectors are typically the datapoints that are closest to the decision boundary of the classifier. They are hence the most important datapoints to construct that boundary, especially since the theory of SVMs is based on maximizing the margin of the decision boundary (either in the original space or the feature space). Therefore, datapoints become support vectors if they are close to the margin or influence it in the feature space. Their importance is determined by their distance to the margin of the classifier in the (feature) space. In case the datapoint is misclassified, it also becomes a support vector.

To support this explanation and study the behaviour of the SVM hyperparameter C and the bandwidth σ^2 of the RBF kernel, we use the application tool from Ref. [1]. In Figure 1.2, we first compare a linear SVM with a SVM using a non-linear RBF kernel. We clearly notice, especially for the linear case, that the support vectors are the datapoints close to the decision boundary. The influence of the C parameter is shown in Figure 1.3. Recall from the lectures that the hyperparameter C is defined in the objective function of the SVM optimization problem in the primal representation, *i.e.*

$$\mathcal{J} = \frac{1}{2} w^T w + C \sum_{k=1}^{N} \xi_k \, . \tag{1.8}$$

Therefore, this hyperparameter determines the penalty value for misclassifications. For low C, the classifier tolerates a few misclassifications and has a simpler decision boundary. For higher C, misclassifications are avoided at all costs, and the decision boundary is more flexible, which has the risk of leading to overfitting. Therefore, as our experiments will also show, it is always advised to set C sufficiently high to guarantee generalization.

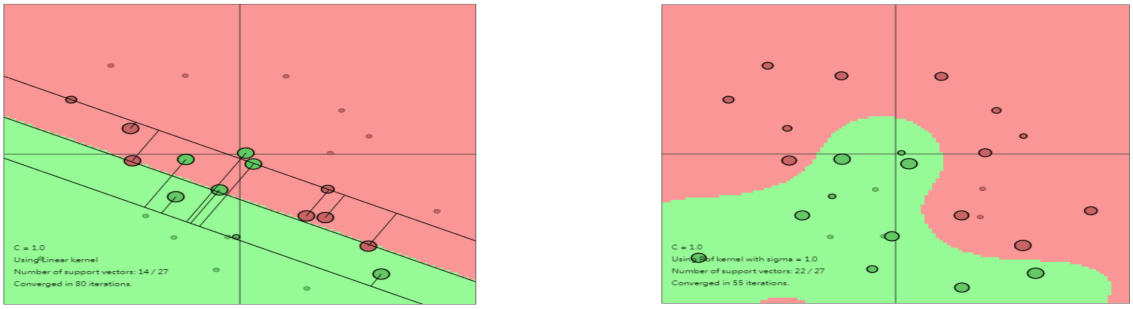

Figure 1.2: Left: Linear SVM classifier with $C = 1$. *Right*: Non-linear SVM classifier with RBF kernel, with $C = 1$ and $\sigma^2 = 1$.

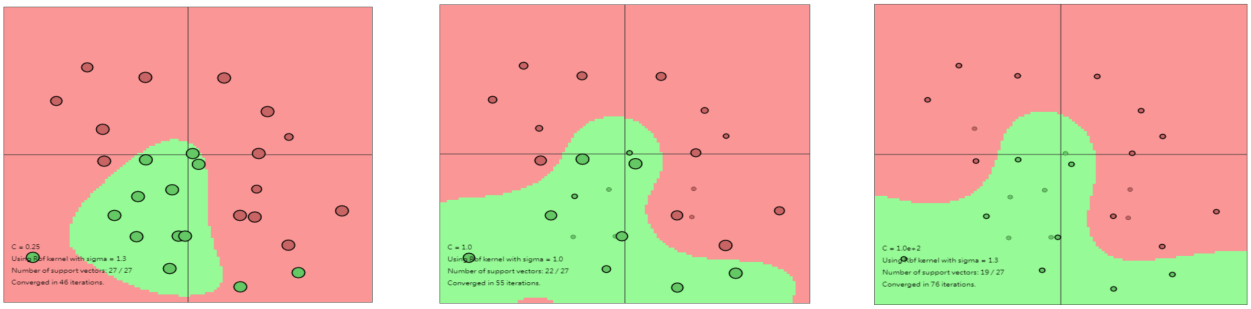

Figure 1.3: Influence of the C parameter on the SVM classifier. The C parameter equals, from left to right, 0.25, 1 and 100.

We now study the RBF kernel and its kernel hyperparameter σ^2. The RBF kernel is given by

$$K(\boldsymbol{x}, \boldsymbol{x}_k) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{x}_k\|_2^2/\sigma^2\right) . \tag{1.9}$$

Throughout all assignments, we will frequently encounter the RBF kernel. It is therefore important to consider the behaviour of the RBF kernel as a function of the bandwidth σ^2. For low σ^2, the RBF kernel essentially becomes a Dirac delta distribution or step function. For high σ^2, the RBF kernel behaves almost like a constant function with value 1. For intermediate values of σ^2, only points within a small neighbourhood of $\boldsymbol{x}_k$ have significant values for the kernel function. The influence of the width σ of the kernel is studied experimentally on the dataset given in Figure 1.4. Recall that the kernel function encodes a similarity measure between datapoints in the feature space and its effect is hence to group together datapoints that are "near" each other. If the width of the Gaussian kernel is small, then the datapoints become isolated from each other and only influence a region in their immediate neighbourhood for the classification, as seen from Eq. (1.7). In the limit of $\sigma \to 0$, the classifier would essentially become a superposition of Dirac delta functions which leads to overfitting on the training data and poor generalization. On the other hand, if σ becomes very large, all datapoints will be viewed as similar to each other and will be grouped together in one large region. This property reminds us of the linear classifier, and indeed, in the region of interest (close to all datapoints), the decision boundary will seem to be linear. Far away from this region, however, we would most likely see that the decision surface is, in fact, curved and non-linear.

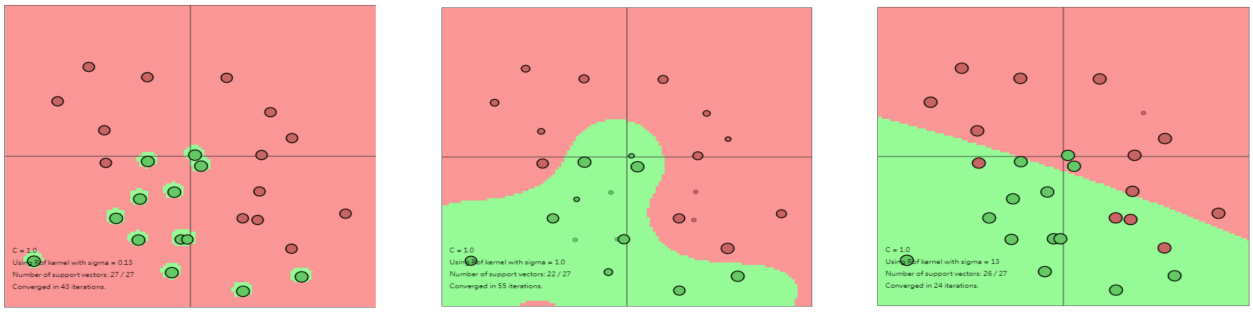

Figure 1.4: Influence of the σ parameter on the SVM classifier. The σ parameter equals, from left to right, 0.13, 1 and 13.

1.3 LS-SVM hyperparameters

We continue with LS-SVMs and study their hyperparameters and kernel parameters. For LS-SVMs, the objective function is changed to

$$\mathcal{J} = \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{k=1}^{N} e_k^2 \,, \tag{1.10}$$

such that the hyperparameter γ plays a role similar to C in the SVM case. The classifiers are trained on the iris dataset and their performance on the test data is evaluated through their accuracy. We show the influence of the degree of the polynomial kernel, the width σ of the RBF kernel and the hyperparameter γ in Figure 1.5. When the degree equals one, we have a linear classifier, which clearly is unable to classify the dataset. Increasing the degree introduces non-linearity and increases the capacity of the model until it can classify the dataset. If the degree becomes too high, the model overfits on the training data. For the RBF kernel, a small width similarly leads to overfitting (also see the discussion regarding Figure 1.4). This analysis shows that the complexity of the polynomial and RBF kernels are determined by the degree and bandwidth, respectively, which can lead to flexible models that overfit if not tuned properly. If the RBF bandwidth is too large, datapoints are easily grouped into one large region, which leads to underfitting. We already remarked in context of the SVM formulation that C has to be set sufficiently high to avoid overfitting. Equivalently, in the LS-SVM case it is important to make sure that γ is set sufficiently high, as the figure shows.[1] Since this hyperparameter determines the penalty for misclassifications, it improves the generalizability of the model.

1.4 Tuning parameters using validation sets

In actual classification problems, we tune (hyper)parameters through more sophisticated methods, such as using a validation set. Here, we explore different tuning algorithms. We train RBF SVMs on a grid of (σ^2, γ) values, with both σ^2 and γ taking values in $\{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. Their

[1] While it is generally advised to tune γ on a logarithmic scale, we used a small range with linearly spaced values to demonstrate the influence on generalizability for this discussion.

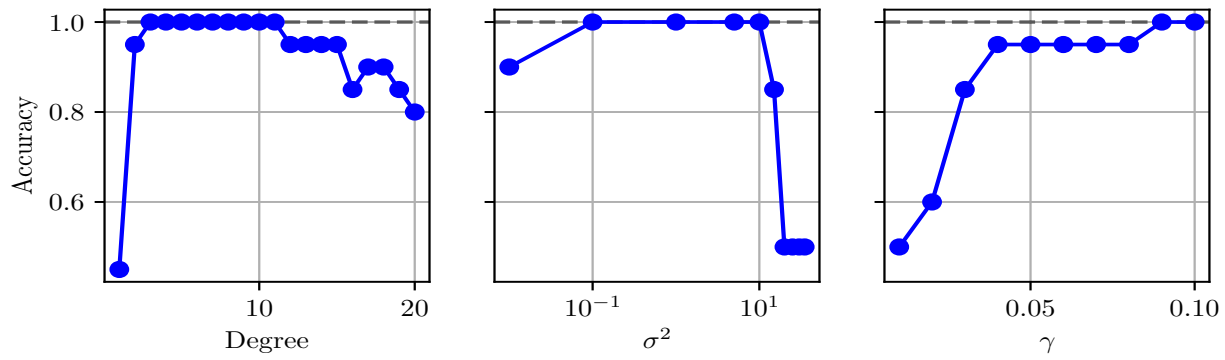

Figure 1.5: Accuracy of classifiers on the iris test data. *Left*: Polynomial kernels with varying degree. *Middle*: RBF kernels with varying width. *Right*: RBF kernels with $\sigma = 1$ and varying values for γ.

misclassification rates with different tuning algorithms are shown in Figure 1.6. The cross-validation and leave-one-out misclassification rates are roughly identical to each other. The random split, on the other hand, can occasionally report significantly different error rates. This is due to the fact that the random split only has a single validation set, which gives a biased estimate of the true accuracy of the classifier if the split is not representative of the true data distribution. As this example shows, cross-validation is preferred over simple validation since it provides a more accurate and unbiased estimate of the model's generalization to unseen data. The tuning algorithms clearly indicate that the optimal classifiers are those for which $\sigma^2 \in [10^{-1}, 10^2]$, while γ seems to be less influential. To choose the parameter k of k-fold cross-validation, one often chooses $k = 10$ as a general rule.

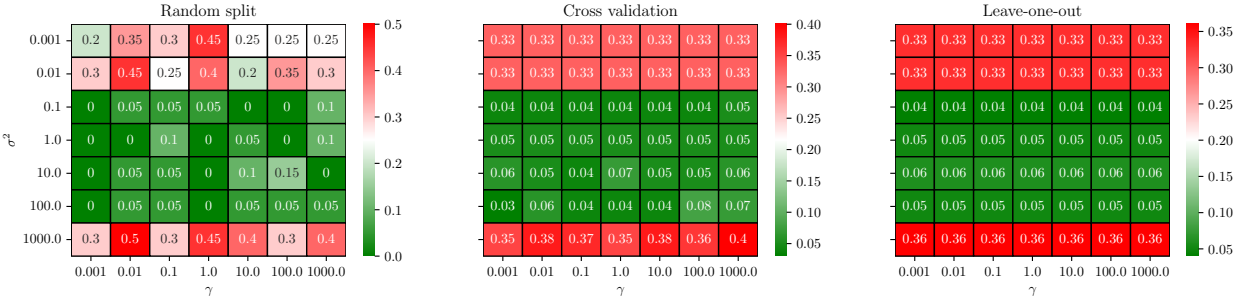

Figure 1.6: Misclassification rates of RBF SVMs obtained through different tuning algorithms.

However, the optimal choice of k depends on the dataset. For smaller datasets, a higher value of k ensures that each iteration has sufficient training data. On the other hand, for large datasets, training can be computationally expensive, and a lower k might be more suitable. If the dataset has high variance, a higher value of k might also be appropriate to have a more accurate estimate.

1.5 Automated tuning algorithms

The tuning procedure is fully automated in the LS-SVMlab toolbox via `tunelssvm` and uses either the Nelder-Mead method (a simplex method) or a brute-force grid search. We compare the results

of both algorithms in Figure 1.6. Both optimization algorithms are initialized with starting values determined by coupled simulated annealing [2]. The obtained hyperparameters differ a lot between runs since simulated annealing is a stochastic process which limits the grid and chooses the starting values, which hence both vary wildly as a result [3]. Despite this randomness, the tuned SVMs have similar performances, with misclassification rates around $3-4\%$, which agrees with the best models we found in the previous section. Moreover, the results of `tunelssvm` agree with the remarks made above, *i.e.* the best models have $\sigma^2 \in [10^{-1}, 10^2]$ with γ having less influence in this problem.

While the determined hyperparameters are more or less within the same range for both algorithms, the grid search method occasionally returns slightly higher values. This is perhaps due to the fact that grid search performs an exhaustive search over a predefined grid whereas the simplex method is a directed search algorithm. Both algorithms of `tunelssvm` seem to be equally fast in the tuning procedure, although the grid search algorithm is simpler in design since it evaluates all grid points and returns the minimum, whereas the simplex algorithm is a direct search algorithm. However, this implies that the grid search method likely will not scale well in case many more parameters would have to be tuned.

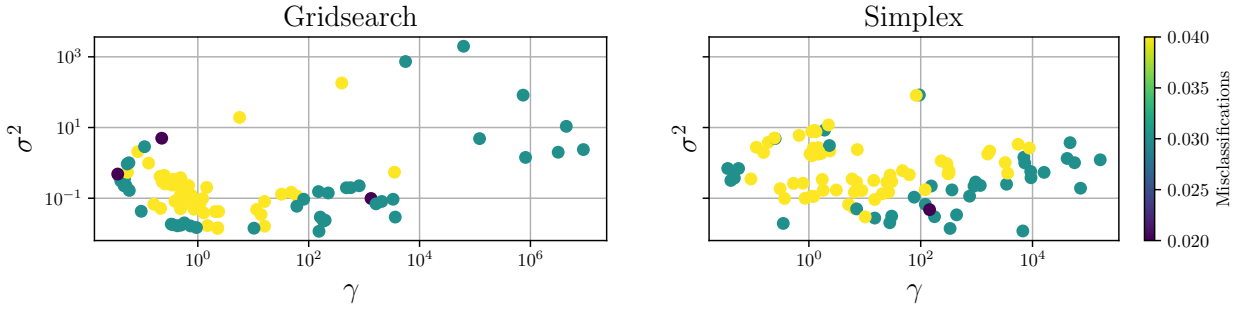

Figure 1.7: Results of `tunelssvm` with the grid search and simplex algorithms.

1.6 ROC curves

We can also analyze the performance of classifiers by plotting their ROC curve and comparing their area under the curve (AUC). In a ROC diagram, a classifier corresponds to a single point with a certain sensitivity and specificity. In the context of SVMs, we obtain a curve by varying a threshold. That is, after computing the output of the SVM, we make the classification by comparing the output against a threshold value, the default being zero such that the classification coincides with taking the sign of the output of the SVM as seen in the lectures. By varying this threshold, we obtain a curve in the ROC diagram, of which we can compute the area. In practice, we compute the ROC curve on the test set rather than the training set, since the estimate computed on the training set is biased as the model's parameter values were configured to perform well on that dataset. The test set, on the other hand, is an unobserved dataset for the model and hence provides an unbiased estimate of its performance and its ability to generalize well to new data.

For the iris dataset, we compute the ROC curve on the training data for simplicity. When computing the ROC curve for an RBF SVM with tuned (hyper)parameters, we find that this

classifier has the maximal AUC score. This is as expected, since these optimal classifiers achieve an accuracy of 100% on the test set (see for instance Figure 1.5). Since sensitivity, specificity and accuracy are related by[2] [4]

$$\text{acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{\text{TP} - \text{FP} + 1}{2}\,, \tag{1.11}$$

we can deduce that a classifier has 100% accuracy if and only if its sensitivity and specificity are equal to one, and hence has a maximal AUC score.

1.7 Bayesian framework

Using the Bayesian framework, we can output probability estimates instead of classification predictions. Examples of such SVMs are shown in Figure 1.8. The output is no longer binary, denoting the predicted class label, but varies continuously between 0 and 1. Values close to 0, respectively 1, indicate a preference for the negative, respectively positive class. Put differently, the output values indicate the probability that a datapoint at that location corresponds to the positive class. We notice that an SVM using the RBF kernel with tuned (hyper)parameters has a region containing all negative examples with very low output values, while the output is close to 1 everywhere else. SVMs without tuned (hyper)parameters, on the other hand, have regions with low and high values, and regions where the output is about 0.5. Hence, the classifier is unsure what to predict in regions without any examples if we do not tune the hyperparameters. Much of the discussion of the influence of the parameters also holds for the Bayesian framework. For instance, increasing the width of the RBF kernel again creates larger connected regions where the output favours the negative class. The key difference is that these classifiers do not have clear decision boundaries anymore, which are replaced instead by smooth transitions from high output values to low output values.

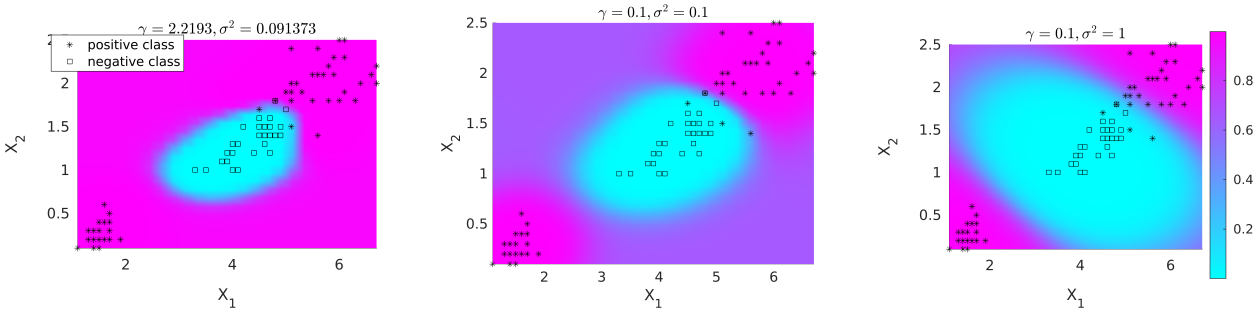

Figure 1.8: RBF SVMs in the Bayesian framework with varying parameters. *Left*: $\gamma = 2.2193, \sigma^2 = 0.091373$ (automatically tuned). *Middle*: $\gamma = 0.1, \sigma^2 = 0.1$. *Right*: $\gamma = 0.1, \sigma^2 = 1$.

[2] We use the notation of true positive ratio (TP), true negative ratio (TN), false positive ratio (FP) and false negative ratio (FN). We have that *sensitivity* is equal to TP and *specificity* is equal to TN, which equals $1 - \text{FP}$.

1.8 Synthetic and real-life datasets

After having digested the concepts of SVMs, we are ready to try and classify synthetic and real-life datasets. We will look at the Ripley dataset, the Wisconsin breast cancer dataset and the diabetes dataset, which are obtained from the *UCI Machine Learning Repository* [5] and are shown in Figure 1.9. The breast, respectively diabetes, dataset contains 30, respectively 8 features, such that we only plot the first two input features. However, we can gain additional insight regarding the structure of the dataset through a t-SNE plot [6, 7]. In Figure 1.10, we show the t-SNE plots for these two datasets. We can predict that the diabetes dataset will be harder to classify, as it seems that datapoints of different classes are often close together.

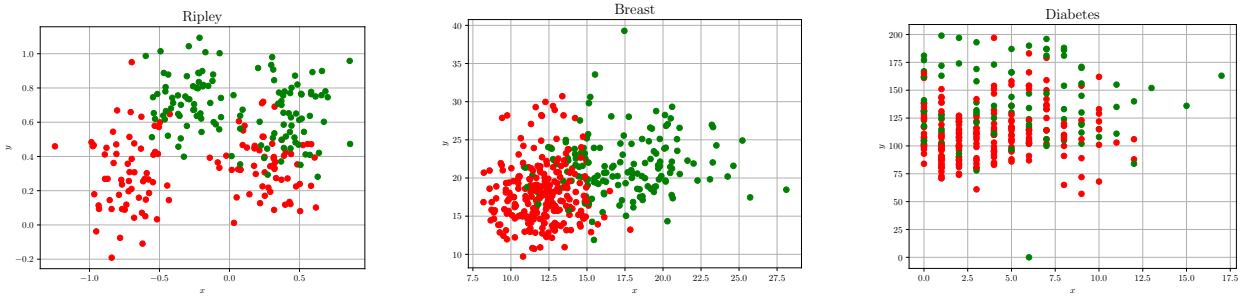

Figure 1.9: The Ripley, breast cancer and diabetes datasets. Colors denote the class.

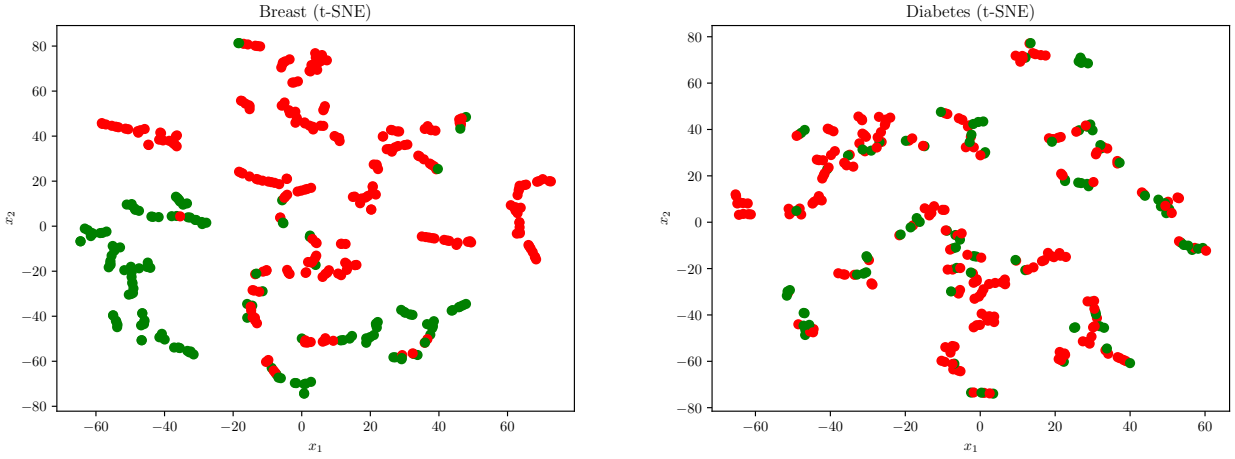

Figure 1.10: t-SNE plots for the breast cancer and diabetes datasets.

1.8.1 Ripley dataset

We first train SVM classifiers on the Ripley dataset. Their ROC curves and corresponding AUC scores are shown in Figure 1.11. The corresponding decision boundaries are shown in Figure 1.12. The Ripley dataset is generated from a mixture of two Gaussian distributions. Therefore, the RBF kernel seems the most appropriate choice for the kernel function, which our analysis seems to confirm. However, the data is quite easy to classify, as other kernels have a good performance as

well, even the linear kernel. The accuracy of the RBF, linear and polynomial kernel was 88.80%, 85.60% and 87.20% respectively after tuning with `tunelssvm`. As the majority class distribution is 50%, all the SVM classifiers improve upon a basic classifier that always predicts the majority.

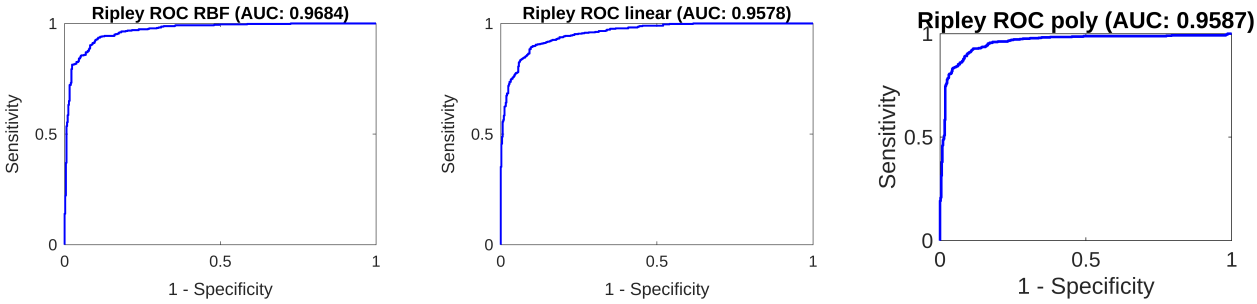

Figure 1.11: ROC diagrams and AUC scores for SVM classifiers trained on the Ripley dataset.

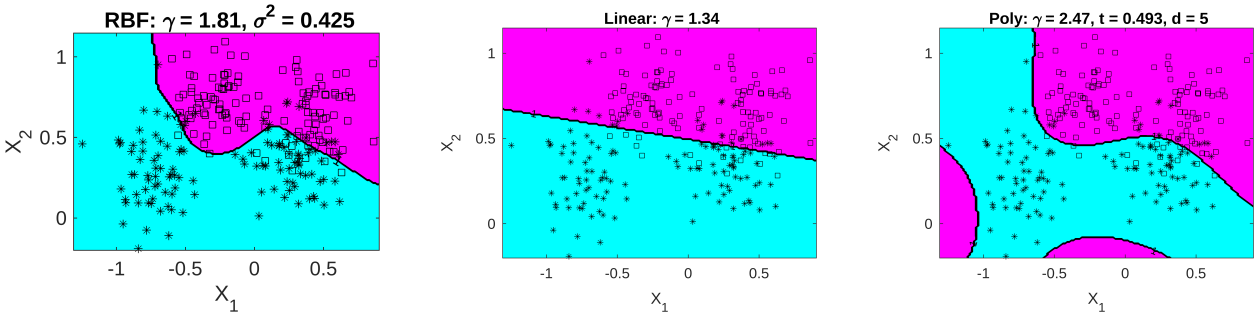

Figure 1.12: Decision boundaries of SVM classifiers trained on the Ripley dataset.

1.8.2 Breast cancer dataset

Next, we investigate the breast cancer dataset. The t-SNE plot seems to indicate that the data is likely easy to separate, and gives the impression that a linear classifier would already achieve high performance. Our analysis from Figure 1.13 seems to support this, although the RBF kernel provides comparable results. This is naturally explained by the fact that the tuned bandwidth is given by $\sigma^2 = 23.7798$, which is quite high such that the decision boundary likely closely resembles that of the linear kernel. The accuracy of the SVM classifiers with an RBF, linear and polynomial kernel are 98.50%, 97% and 87.25%, respectively. The majority class distribution is 62.5%, such that all classifiers greatly improve upon the simplest classifier which always predicts the majority class. Since the feature data is high-dimensional, we do not plot the classifiers and their decision boundaries for this dataset.

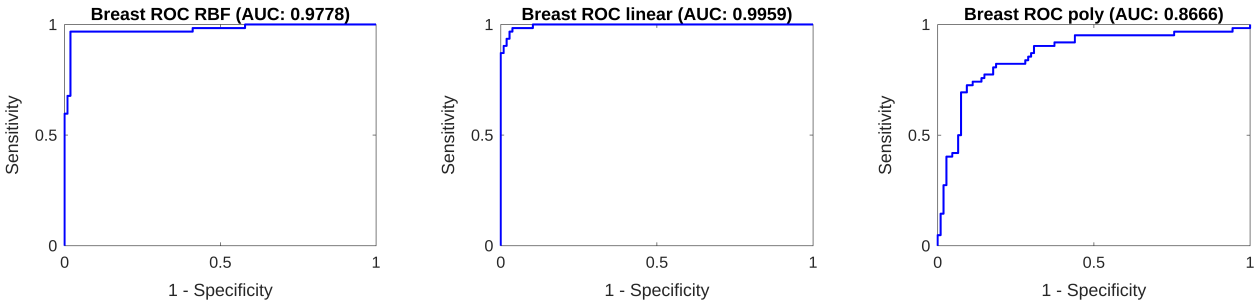

Figure 1.13: ROC diagrams and AUC scores for SVM classifiers trained on the breast cancer dataset.

1.8.3 Diabetes dataset

The visualization shows that the structure of this dataset is completely different from the previous two datasets and the t-SNE plot seems to indicate that the data is harder to classify since datapoints are placed close to each other. Our analysis in Figure 1.14 indeed seems to indicate that the SVMs overall have a poorer performance compared to the previous two datasets. The accuracy of the SVM classifiers with an RBF kernel, linear kernel and polynomial kernel are 76%, 75.67% and 75.33%, respectively. Since the majority class distribution is equal to 68.33%, the SVM classifiers only give a slight improvement over predicting the majority class.

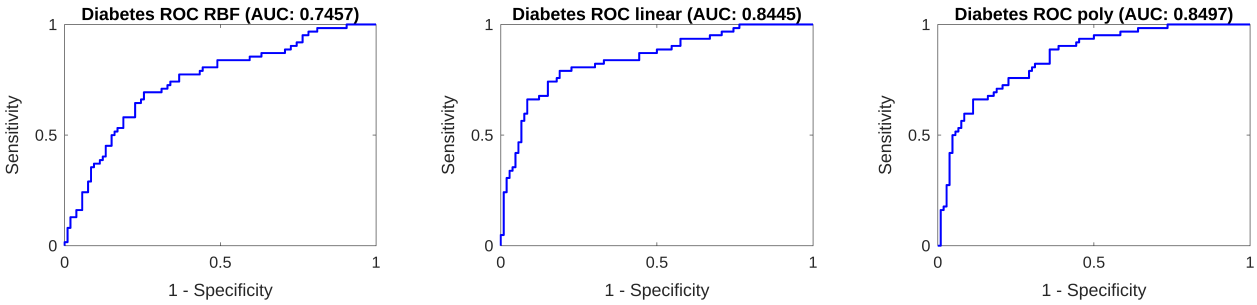

Figure 1.14: ROC diagrams and AUC scores for SVM classifiers trained on the diabetes dataset.

There is likely an underlying structure present in this dataset which is hard to capture with our SVM classifiers. The decision boundaries that the SVMs have learned are probably not suited for this dataset due to its regular structure. It is possible that other machine learning algorithms can provide better classification results. For instance, a decision tree is potentially an interesting candidate, since it is capable of learning a jagged decision surface. Moreover, it is likely that having diabetes is determined by a certain combination of specific features having certain values. Therefore, the classification formulated by if-then rules. Since decision trees can be converted into such if-then rules, they are a promising algorithm to use for this dataset, as it has the appropriate bias and is an interpretable model such that we can gain insight from the learned classifier.

2 Function Estimation and Time Series Prediction

We now consider applications of SVMs and LS-SVMs for regression problems.

2.1 SVM for function estimation

In this first example, we consider SVM regression problems using the `uiregress` demo. Recall that the SVM formulation for regression[3] in the linear case was to find the optimal linear function $\hat{y}(x) = w^T x + b$ by solving

$$\min_{w,b,\xi_i} \frac{1}{2} w^T w + c \sum_{i=1}^{N} \xi_i \,, \tag{2.1}$$

subject to the constraints

$$|y_i - \hat{y}_i| \leq \varepsilon + \xi_i \tag{2.2}$$

$$\xi_i \geq 0 \,, \tag{2.3}$$

for $i \in \{1, \ldots, N\}$. Here, ε is the width of the ε-tube and ξ_i indicates the error between the model predictions (taking the width of the tube into account) and the true values. First, we consider a dataset for which a linear kernel is most suited. The demo takes in a parameter `Bound` and a parameter `e`, where the former acts as a regularization parameter and the latter is equal to ε. The figure shows clearly that ε determines the sparsity of the model. Data points that fall within the tube are not support vectors. Hence, a larger ε would imply a larger sparsity. This can be understood by considering the Lagrangian of the optimization problem, given in the slides. If we have a datapoint x_k in the ε-tube, then Eq. (2.2) is satisfied with $\xi_k = 0$ and the term in the Langrangian corresponding to α_k is maximized by taking $\alpha_k = 0$, *i.e.*, the datapoint is not a support vector. A similar relationship holds in the LS-SVM case, where one can derive from the optimization problem that $\alpha_k = \gamma e_k$ such that the magnitude of the Lagrange multiplier of a datapoint is proportional to its contribution to the loss. In the standard SVM formulation, the loss is zero in the ε-tube, leading to sparsity of the model. Intuitively, the regressor is "blind" to datapoints within the ε-tube, which is precisely the crux of the Vapnik ε-insensitive loss function. In the extreme case that ε is larger than the spread of the y-values of the datapoints, all datapoints easily fall within the ε tube and we can minimize the second term of the above objective function by setting $\xi_i = 0$ for all i. Then, the objective function only contains the regularization term $\frac{1}{2} w^T w$ and the optimal solution is given by a horizontal line, *i.e.* $w = 0$.

In Figure 2.2, we vary the `Bound` parameter and fix the `e` parameter. We notice that a low `Bound` parameter implies a higher regularization (smaller w), whereas a high `Bound` parameter puts a lot of weight on the training error and allows the model to be more flexible but more prone to overfitting. Hence, the `Bound` parameter likely corresponds to the C parameter in Eq. (2.1). Indeed, in the extreme case of `Bound` $= 0$, the second term in Eq. (2.1) vanishes and only the regularization term remains, which again gives a flat horizontal line as optimal solution.

[3] While the original formulation from the lectures was more sophisticated and relied on two sets of slack variables, we simplify the presentation for a clearer discussion.

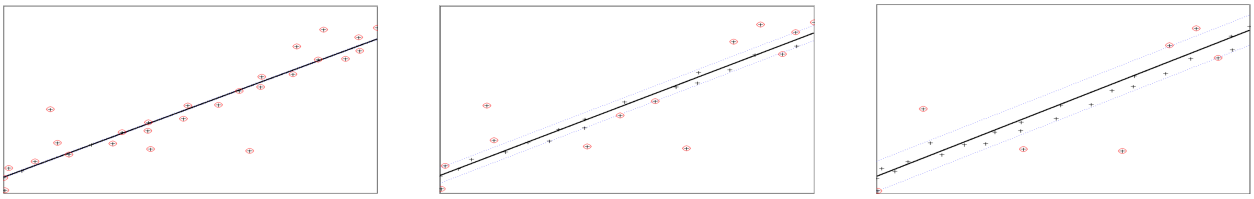

Figure 2.1: SVM regressors for varying values of `e`, with `Bound` = 10. From left to right: `e` = 0.01, `e` = 0.1, `e` = 0.2.

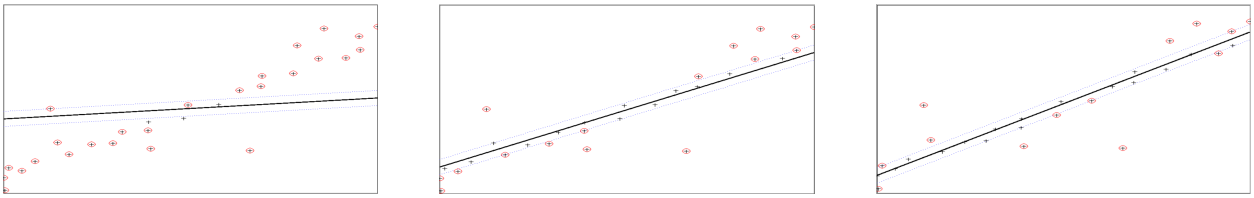

Figure 2.2: SVM regressors for varying values of `Bound`, with `e` = 0.1. From left to right: `Bound` = 0.01, `Bound` = 0.1, `Bound` = 10.

Now, we construct a more challenging dataset. In our case, the best kernel is likely either a polynomial or an RBF kernel, due to the shape and distribution of the dataset, which clearly is not suitable for the linear kernel. We choose to work with an RBF kernel, since the σ^2 parameter is more intuitive to tune and the polynomial kernel has two hyperparameters to tune instead of one. Figure 2.3 shows a few examples of SVM regressors. We notice that a low σ^2 and high `Bound` can lead to overfitting. This can be understood from our above remarks and the discussion of Section 1. A low bandwidth implies a flexible model, while a large `Bound` puts more weight on the training error term, which puts us in the regime of overfitting. By keeping the same value for `Bound` but increasing σ^2 we can improve the results.

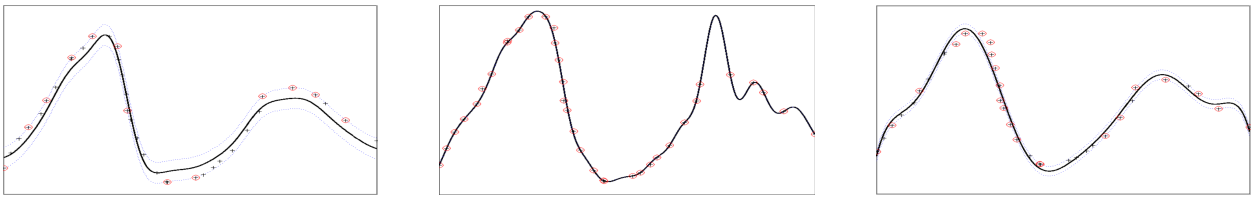

Figure 2.3: SVM models with RBF kernels for varying (hyper)parameters. *Left*: $\sigma^2 = 0.1$, `Bound` = 100, `e` = 0.1. *Middle*: $\sigma^2 = 0.1$, `Bound` = 100, `e` = 0.0001. *Right*: $\sigma^2 = 0.5$, `Bound` = 1000000, `e` = 0.05

There are several differences between SVM regression and linear least squares fits. First of all, they differ in their objective function. Whereas least squares fits aim to minimize the sum of squared residuals (differences between model predictions and true values), SVM regression uses the Vapnik ε-insensitive loss function and takes into account a regularization term when computing the optimal solution. As a consequence, least squares fits can be sensitive to outliers, since these give high residual values. SVM regression, on the other hand, is less sensitive to outliers due to the loss function being linear and because the regularization improves generalization. Second, least squares regression always fits a straight line through datapoints. SVM regression can achieve this as well, by relying on a linear kernel, but can furthermore give non-linear fits by choosing a non-linear

kernel instead. Therefore, SVM regressors are much more general and are capable of handling a wide variety of datasets. Third, the complexity of the resulting model can be controlled in SVM regression by tuning hyperparameters, since the complexity is controlled by the importance of the regularization term compared to the training error term in the objective function. On the other hand, this implies that SVM regression is more complicated to perform since this requires the practitioner to tune the kernel parameters and hyperparameters through *e.g.* cross-validation. As a result, SVMs can be computationally more expensive to train. Moreover, due to the high bias of a linear fit, this method has a low variance and less likely suffers from overfitting. Finally, while SVMs can be sparse, linear fits are guaranteed to be even simpler and interpretable, which might be harder for non-linear SVM regressors.

2.2 Toy example: the sinc function

As a simple example, we consider SVM regression on the sinc function. The data is generated at equidistant points with separation $\Delta x = 0.01$ for $x \in [-3, 3]$. In the first part of this section, we divide the data into 50% training data and 50% test data, ensuring that both still consist of equidistant points. We try different SVM regressors obtained from different (γ, σ^2) combinations and determine the validation MSE through cross-validation. The results are shown in Figure 2.4. The models obtained from three distinct combinations are shown in Figure 2.5. The performance of the model is mainly determined by the σ^2 parameter. It is preferable to have an intermediate value for σ^2, since low values easily overfit on the training data and large values lead to a model that is too smooth such that the model underfits, which is also apparent from Figure 2.5. Similar conclusions were also observed in the tuning procedure in Section 1. Therefore, we can conclude that the flexibility of the model, which is controlled by σ^2, greatly influences the results of SVMs with RBF kernels.

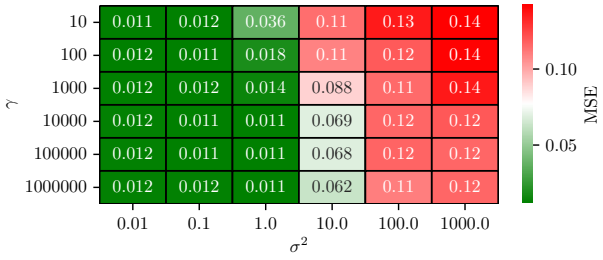

Figure 2.4: MSE measured on a validation set for different SVM regressors trained on the sinc function.

We can also use the `tunelssvm` method to automatically tune the hyperparameters, using either the simplex or the grid search as explained in the previous assignment. The resulting parameter values, as well as their MSE, are shown in Figure 2.6 for 100 repetitions. We observe that both algorithms return quite similar parameter values, as we already observed in Section 1.5. Moreover, the results from `tunelssvm` confirm our observations made above that the tuned σ^2 values are around 1 or lower to provide a trade-off between flexibility and generalizability, while γ is less restricted by the tuning procedure.

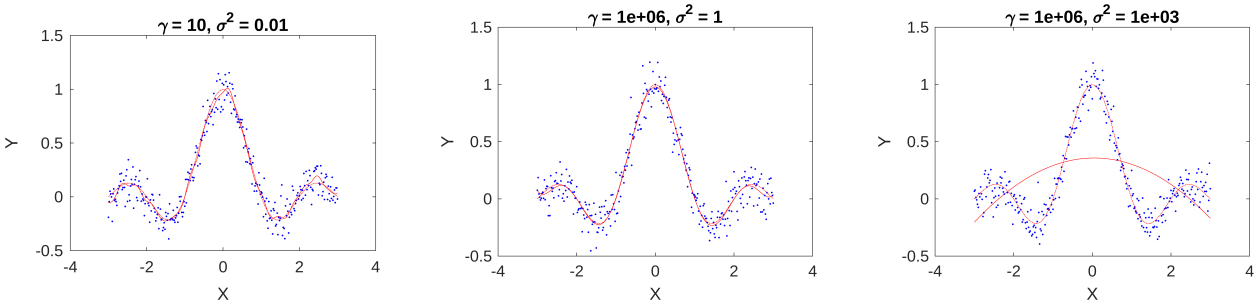

Figure 2.5: Predictions SVM models with RBF kernels (dashed line) trained on the sinc data.

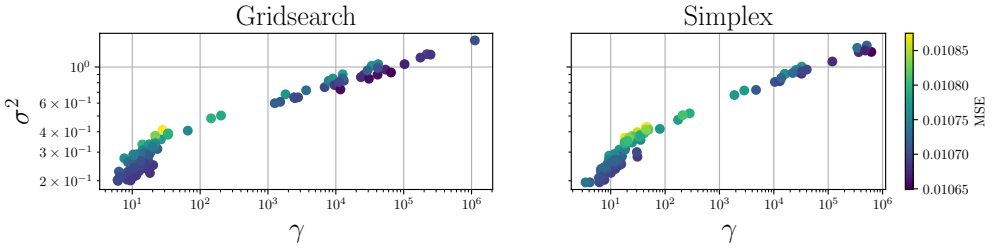

Figure 2.6: Hyperparameters found with `tunelssvm` after 100 repetitions for both algorithms.

2.3 Bayesian tuning and ARD

We consider the Bayesian framework for (hyper)parameter tuning and automatic relevance determination (ARD). In the Bayesian framework, we perform inference on three different levels: the level of the parameters (the bias and weights in the primal representation or Lagrange multipliers in the dual representation), the level of the hyperparameters (the regularization coefficients, called μ and ζ in the slides) and the level of the model parameters (*e.g.* the bandwidth of the Gaussian kernel). This is also reflected by the fact that the provided Matlab code is organized in three function calls which optimize each of these levels. On each level, we specify a prior probability distribution and, together with the data, we then consider the posterior distribution. Subsequent levels are interconnected, since the likelihood of level i is equal to the evidence at level $i-1$. As such, the Bayesian optimization framework can be interpreted as gradually integrating out (marginalizing out) parameters at each level. For instance, tuning the bandwidth of an RBF kernel in level 3 is an optimization problem which assumes that w, b, μ, ζ are fixed (as determined by the previous levels). The optimization procedure takes the log of the posterior as objective function and computes the MAP estimator. Due to the interconnection of adjacent levels, one often runs the Bayesian framework several times, updating (hyper)parameters of earlier levels again to take into account the changes made to other (hyper)parameters. We observed that this process often converges already after 3 – 5 iterations in the application considered here. Because of this probabilistic framework, one can also compute uncertainty estimates on the output and create a regressor which gives errorbars on its prediction. The results from the Bayesian tuning, after three iterations over the three levels, are shown in Figure 2.7.

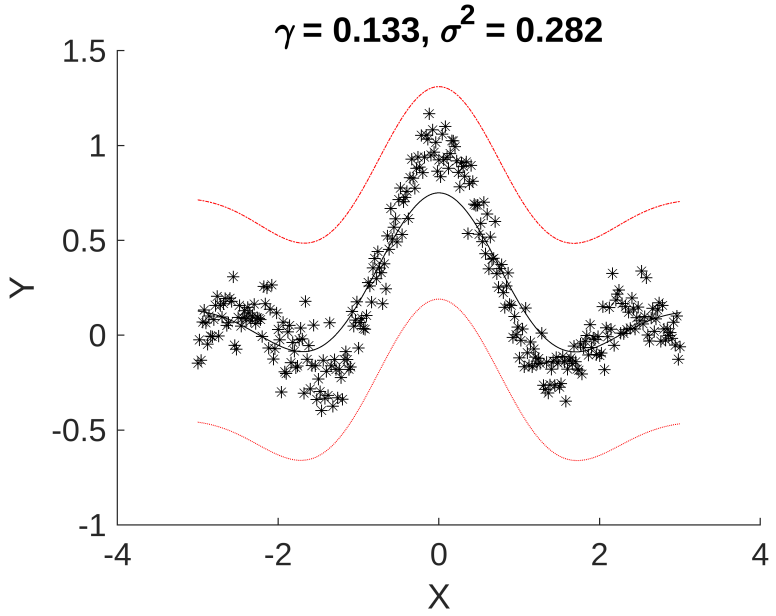

Figure 2.7: SVM regressor trained on sinc data with parameters and hyperparameters found after three iterations of the Bayesian optimization procedure.

In ARD, one slightly modifies the RBF kernel by adding an additional square matrix S:

$$K(x,z) = \exp\left(-(x-z)^T S^{-1}(x-z)\right) , \tag{2.4}$$

where $S = \text{diag}\left(s_1^2, \ldots, s_n^2\right)$. The values s_i^2 are then taken as the kernel hyperparameters, instead of the usual σ^2, and are therefore optimized in level 3 of the Bayesian framework outlined above. Small values of $1/s_i^2$ imply that the corresponding input values are not relevant for the model. The ARD algorithm iteratively removes such irrelevant inputs as long as it improves the results of the obtained SVM, and as such automatically determines the relevant inputs of a problem. In each step, the input with the lowest $1/s_i^2$ value is removed, which is known as backward selection. In the provided example code of the assignment, three-dimensional random input data is generated. The output of the function is computed using information from only the first component, such that the second and third input are noise and irrelevant for the regression. The ARD algorithm is able to correctly deduce that only the first input is of relevance.

Input selection can also be performed in a more brute-force fashion through cross-validation. For instance, we can compare results after tuning the s_i^2 hyperparameters from the above matrix S using cross-validation, and implement an iterative algorithm that deletes input features based on the obtained scores similar to the scheme outlined above. However, for this exercise, we implemented a second scheme which does not require any modification of the kernel function. In our scheme, we train LS-SVMs on a subset of the original input data obtained by leaving out one or two of the three input features. For each subset, we perform a 10-fold cross-validation to have an unbiased estimate for the MSE. The results are shown in Figure 2.8. As expected, subsets which contain the first input element have the best performance, with the overall best provided by the subset that only uses the first column as input and ignores the two other variables. The subsets that consist of noise have very similar MSE values which demonstrates that the SVM is not able to correctly learn

a true mapping. Note that this idea can easily be turned into an iterative algorithm that mimics the ARD algorithm by only removing a single input rather than enumerating all subsets as in this demonstration, which is only feasible for low dimensions.

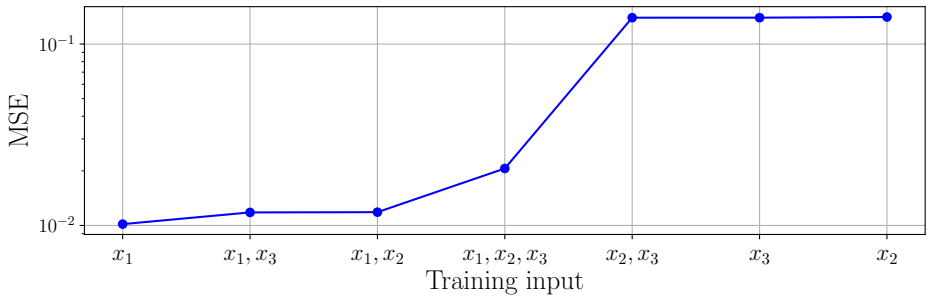

Figure 2.8: Relevance determination by cross-validation on subsets of inputs.

2.4 Robust regression

The standard LS-SVM regressor is not robust against outliers, as we can see from Figure 2.9. Intuitively speaking, outliers seem to 'drag' the SVM predictions towards them and seem to have a significant influence on the obtained regressor. This can be understood by recalling that LS-SVM regressors, as the name implies, rely on a least squares loss function. Hence, large training errors, such as caused by outliers, have a large impact on the model. We can solve this issue by relying on

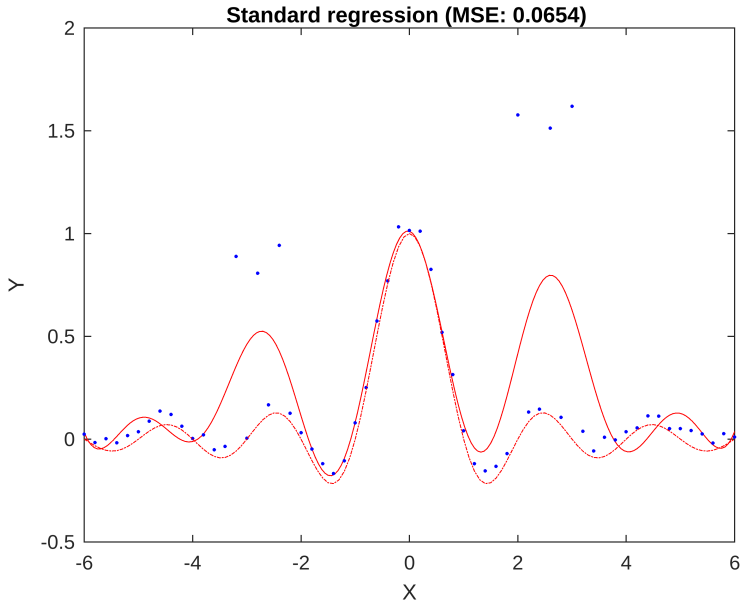

Figure 2.9: Standard LS-SVM regressor trained on data containing outliers.

reweighing schemes and adapting the loss function. The results of these robust regressors are shown in Figure 2.10. We compare the results of different weighing schemes by reporting the MSE on a

test set of 241 evenly spaced datapoints of the true sinc function in the same interval, to check which scheme is able to approximate the true target despite the outliers. Each regressor is tuned using `tunelssvm` with the mean absolute deviation (MAE) loss function rather than MSE. Since the MAE loss is linear rather than quadratic, outliers are less influential and hence this loss function is more suited for data containing outliers. Different reweighing schemes give slightly different regressors, but have overall the same shape and share the same robustness property. There are four possible weighing schemes implemented in the LS-SVM toolbox which further modify the cost function to increase the robustness: the Huber, Hampel, logistic and Myriad weighing schemes. There are no major differences between the different schemes and all schemes achieve very comparable values for the MSE on the test set.

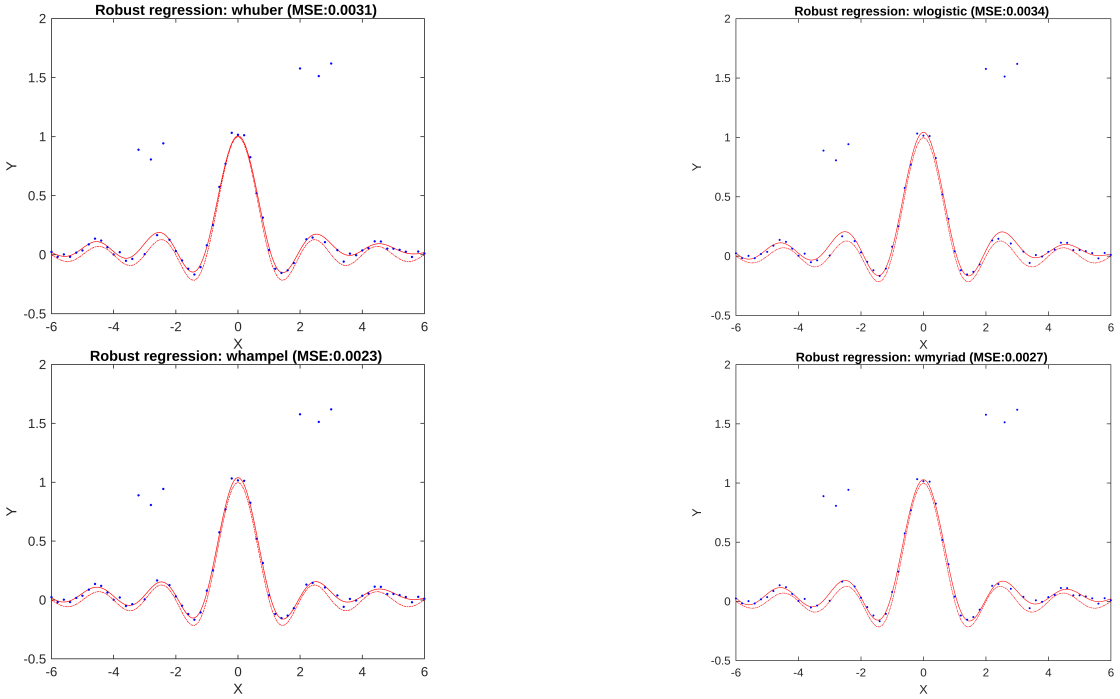

Figure 2.10: Robust regression with different weighing functions.

2.5 Time series prediction

We consider applications of SVM regressors to predicting two time series.

2.5.1 Logmap dataset

The first time series we approximate with an SVM regressor is the `logmap` dataset. One can tune the hyperparameters for this problem as follows. We perform a grid search over a range of interest for the order parameter. Each order parameter specifies how many previous timesteps are used as input to the SVM with the next timestep as output value. Therefore, at each value of the order

parameter, we are dealing with a standard regression problem. Hence, we can apply the tuning procedures discussed above to tune the hyperparameters for this specific order value. We can then compare the performance of the SVM for different order values and determine the order parameter that gives the best result. The results are shown in Figure 2.11. The optimal order parameter determined in this way is equal to 11. Inspecting the predictions, we notice that the SVM provides a good approximation at the beginning and end of the time series, where the test data is regular. However, the SVM fails to capture the irregular behaviour between $t = 20$ and $t = 40$.

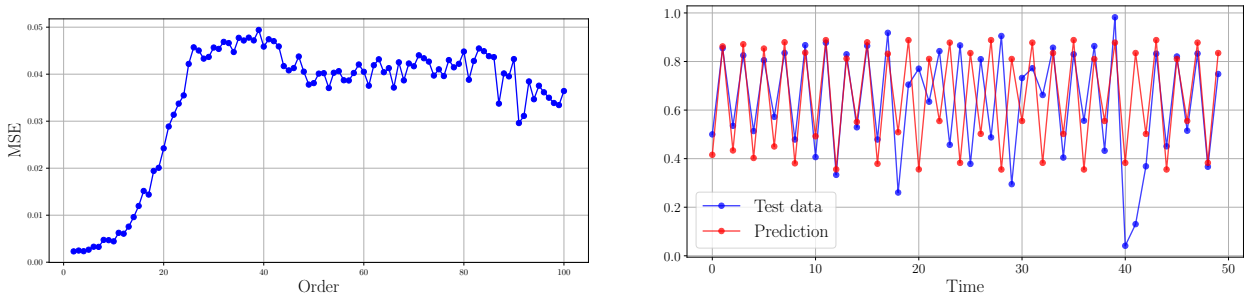

Figure 2.11: Left: Validation MSE for varying order parameters. *Right*: Predictions of SVM regressor with order equal to 11.

2.5.2 Santa Fe dataset

Next, we turn our attention to a more challenging data set, the chaotic Santa Fe time series. Looking at the training data, we notice that a regressor ideally has to be able to capture the sudden drops in the time series and therefore has to have a sufficiently high order value.

Since this time series is more challenging, we design a new tuning procedure. One issue is the different goal in training the SVM and testing it. That is, we train the model as if designed for a standard regression problem while the network is actually applied in an auto-regressive manner. Since an auto-regressive model has to use its own predictions to predict subsequent values, it ideally has to be capable to handle slight deviations from the true values to ensure a stable time series. Therefore, we rely on a tuning procedure which measures the performance of the SVM on a time series. That is, we measure the MSE when the trained SVM is used as an auto-regressor on the time series in the window $[t_{544}, t_{644}]$ of the training data, shown in Figure 2.12.[4] We have chosen this window since it has a similar shape as the actual test set. The results of the standard cross-validation tuning as well as this new tuning procedure are shown in Figure 2.13. The results with the second tuning procedure seem more intuitive: higher order values give more robust results and generally give lower MSE values, since the chaotic nature of the time series requires the regressor to take long-term dependencies into account. The best order parameter is 8, respectively 33 for the first, respectively second tuning procedure. We show a few results of SVM regressors in Figure 2.14. While a low order parameter is more or less able to predict the locations of the extrema, a higher order parameter allows us to approximate the sudden drop to a reasonable extent. Due to the shape

[4]We were not able to fully separate this time series from the other training data. As such, the current implementation suffers from data leakage. However, we merely provide the second tuning procedure as a proof of concept.

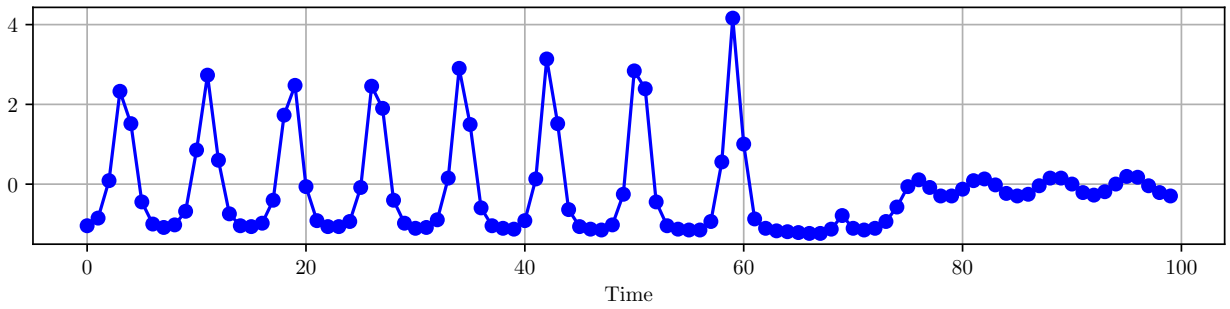

Figure 2.12: Validation set used for the second tuning procedure.

of the MSE values obtained with the second tuning procedure, we can conclude that higher order values generally have a better performance. This seems to be confirmed visually from Figure 2.14, where the shape of the SVM regressor with order equal to 100 seem to be able to approximate the shape at the end more accurately.

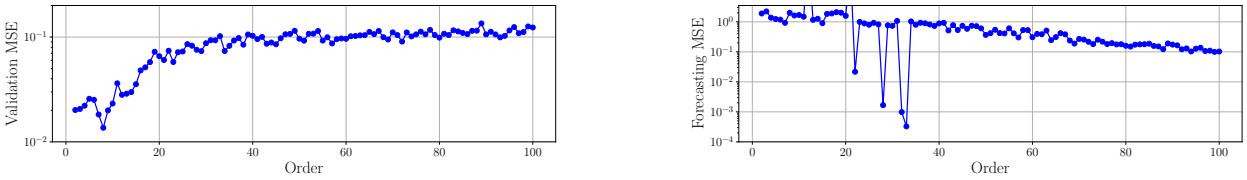

Figure 2.13: Left: Tuning the order parameter with cross-validation. *Right*: Tuning the order parameter by forecasting on a time series.

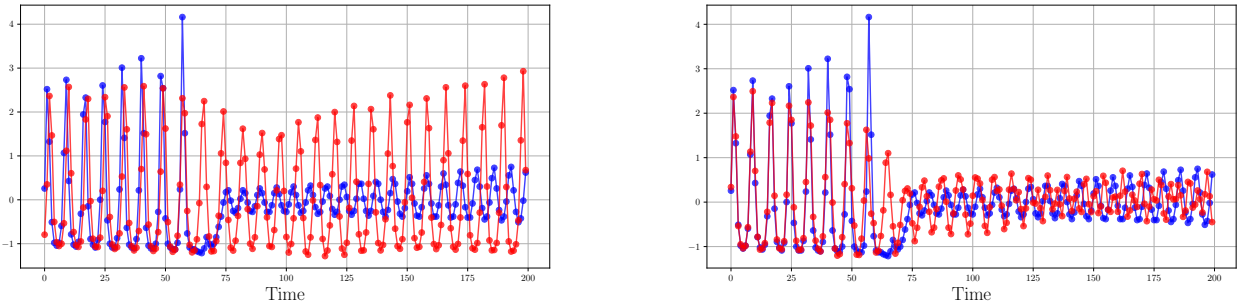

Figure 2.14: Predictions of SVM regressors on the Santa Fe test set for different order values. *Left*: Order equal to 39. *Right*: Order equal to 100.

Finally, the distribution of the Santa Fe training data is shown in Figure 2.15. We see that the data is not symmetrically distributed and has a few outliers. Therefore, we have considered tuning the order parameter by considering the MAE scores rather than the MSE scores. We report the grid search results, using both tuning procedures described above, in Figure 2.16. The best order parameters obtained were 9 and 39, respectively, which are comparable to the results obtained with the MSE scores. The results seem to indicate that regressors with an order parameter between 35 – 45 overall have a better MAE score.

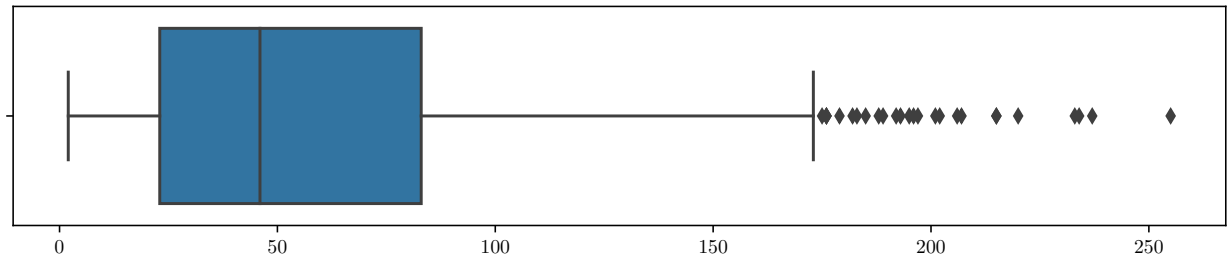

Figure 2.15: Boxplot showing the distribution of the Santa Fe training data.

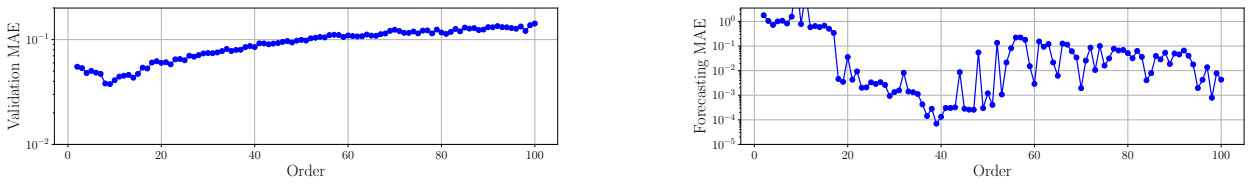

Figure 2.16: Left: Tuning the order parameter with cross-validation. *Right*: Tuning the order parameter by forecasting on a time series.

3 Unsupervised Learning and Large Scale Problems

In this final assignment, we consider the application of SVMs in unsupervised learning settings and large scale problems.

3.1 Kernel PCA

In kernel PCA, we apply linear PCA in the feature space and rely on the kernel trick to obtain non-linear features of the data. We can use PCA for denoising data. We will first discuss linear PCA. As a simple illustration, consider the case where we draw samples from a linear 1D function, and artificially add small Gaussian noise to it. In such a case, we would have one large principal component, explaining a large variance in the direction along the line, and a smaller one, due to noise present in the data, in a direction perpendicular to the line. Retaining less principal components than the original input dimensionality can achieve denoising: in this case, a single principal component likely captures the relationship $y = 3x$ to high accuracy, and would not use the information induced by the noise. PCA provides transformations matrices between the original space and the feature space (in linear PCA, this is the lower-dimensional space spanned by the principal components, also called the "target space" in the lectures). By transforming a datapoint to the feature space and applying the inverse mapping, we can create a reconstruction of the original datapoint, which (if the number of components is tuned correctly) provides a denoised version of the original datapoint. This idea is demonstrated in Figure 3.1. In this case, a single principal component is able to explain more than 99% of the variance. The idea is easily extended

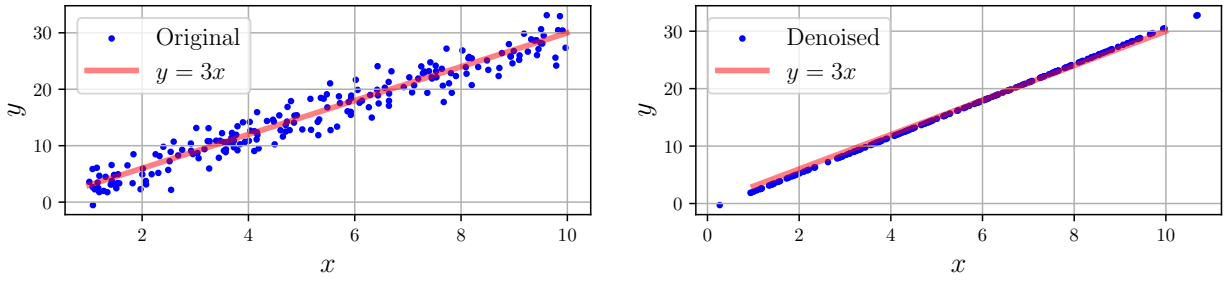

Figure 3.1: Illustration of denoising with linear PCA.

to high-dimensional input spaces, such as images. Indeed, we can interpret images as vectors by scanning the pixel values column-wise for each channel, which essentially corresponds to flattening the image array. By retaining a certain fraction of the principal components, ordered by their magnitude, we can capture most of the information in a smaller dimensional space and rely on the reconstruction to move data points to a lower-dimensional hyperplane of the original input space which, hopefully, does not include the noise. In such high-dimensional spaces, increasing the number of components captures more information and is therefore certainly beneficial at the start. However, at a certain point, retaining too many principal components starts to include meaningless principal components which correspond to small eigenvalues, which hence include the information from the noise. Therefore, a balance has to be found between simplicity and minimal reconstruction

error to achieve the best results.

Comparing kernel PCA against linear PCA, it is clear that linear PCA is an inadequate solution in case the datapoints have non-linear dependencies, such as the yin-yang problem we will consider below. Hence, kernel PCA offers more flexibility due to the kernel trick. Because of this more powerful framework, kernel PCA can also be more robust against noise, as seen in the lectures. Besides, using abstract features constructed from applying non-linear mappings to the data can allow us to potentially use principal components which are richer in information. Since the feature space (obtained by applying the kernel trick) can have more dimensions than the original space, applying linear PCA in the feature space therefore allows us to obtain a higher-dimensional representation of the original data (provided there are sufficient samples in the dataset), which can be advantageous in some applications. However, linear PCA can still have advantages over kernel PCA. First, kernel PCA does not offer improvements in case the data has purely linear relationships, as in the example we considered above. However, it is likely that most real-life problems involve complicated relationships, such that linear PCA is not a suitable option while kernel PCA is still applicable. Besides, kernel PCA is computationally more expensive, since we often have to try different kernel functions and have to tune additional parameters such as the bandwidth of the RBF kernel. While the decomposition of linear PCA has a fixed computational cost, performing cross-validation to determine the kernel and its hyperparameters can be computationally more expensive and depends on *e.g.* the size of the grid if we employ a grid search. Third, since the kernel transforms the data to a high-dimensional feature space, the principal components become abstract. On the other hand, the principal components obtained in linear PCA are still interpretable (*e.g.* in face recognition using PCA, we often talk about "eigenfaces", since the principal components resemble faces of the dataset).

As an example, consider the yin-yang dataset. We conclude from Figure 3.2 that using too many principal components is not advised, as this starts to introduce noise. An optimal number of principal components captures the clusters that are present in the dataset without overfitting on the noise. There are several ways to tune the denoising hyperparameters. For instance, one can take the kernel to be fixed (for the RBF, we hence fix σ^2) and choose the optimal number of components. Similar to linear PCA, this choice can be made by considering the explained variance, which is the fraction of the cumulative sum of the sorted eigenvalues over the total sum of the eigenvalues. One can choose a certain threshold of amount of variance that should be explained and determine the number of principal components based on this criterion. This criterion is based on the observation that highly correlated data have only a few relevant features which correspond to large eigenvalues, while the other principal components, likely corresponding to noise, have low values and a negligible contribution to the explained variance. Fixing $\sigma^2 = 0.4$, the results are shown in Figure 3.3 and the optimal number of components, explaining 90% of the variance, is equal to 11. Another method would be to tune both the kernel hyperparameters as well as the number of components through cross-validation or a train-validation split. We can then tune the set-up by minimizing the reconstruction loss (such as MSE) measured on the validation set. We have performed such a tuning procedure and found the optimal hyperparameters to be $\sigma^2 = 1$ with 15 principal components.

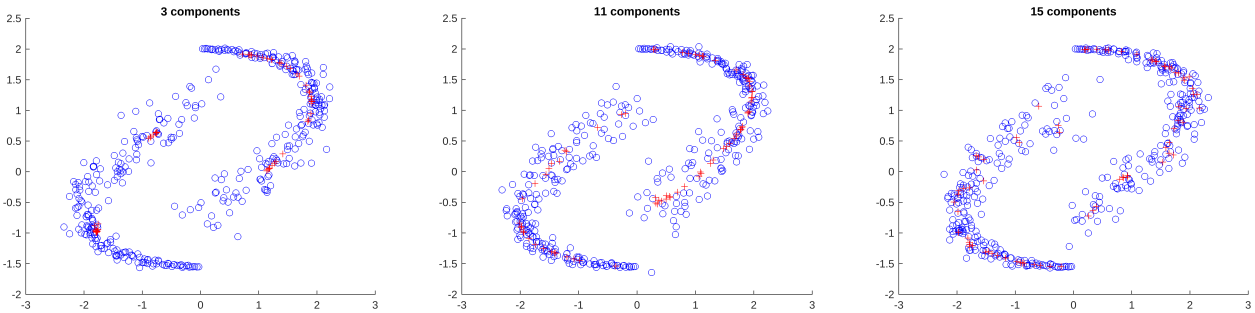

Figure 3.2: Denoising the yin-yang dataset with varying number of principal components.

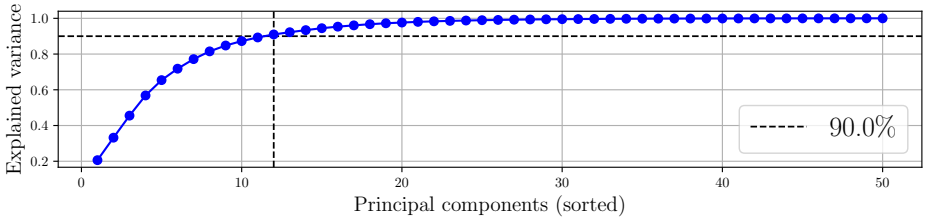

Figure 3.3: Explained variance for the yin-yang dataset, with an RBF kernel with $\sigma^2 = 0.4$.

3.2 Spectral clustering

Spectral clustering is a technique originating from graph theory. There, we can create optimal clusters of a graph by the min-cut, which is the cut that removes the edges of the graph with the lowest interconnection weights. These weights are encoded in the adjacency matrix or similarity matrix, encoding the pair-wise similarities. The problem can be reformulated and solved by considering the Laplacian matrix. The solution can then be expressed in terms of the eigensystem of this Laplacian matrix. The idea underlying the clustering technique is essentially as follows. If we have a dataset, we determine this Laplacian matrix and its eigenvectors. These eigenvectors, possibly considering only the largest among them as in PCA analysis, determine an embedding into the eigenspace. Instead of performing a clustering algorithm, such as the k-means algorithm, in the original input space, we can perform a clustering in the eigenspace of the Laplacian instead. For some specific structures of datasets, this method can give better results than k-means clustering in the original input space. For instance, if the data is distributed according to ring-shaped clusters, standard k-means clustering algorithms will likely fail, because the rings can have a large radius such that datapoints on opposite sides of the ring have a large distance between them and are likely not clustered together because of this. On the other hand, ring-shaped clusters likely correspond to isolated clusters in the eigenspace, such that they are still clustered together by a k-means algorithm performed in the eigenspace. This core idea can be reformulated in an LS-SVM context, as explained in more detail in the slides.

The main difference between spectral clustering and classification is that the former is an unsupervised learning problem while the latter is a supervised problem. Hence, in spectral clustering, the datapoints do not have labels attached to them, and the goal is for the model to find the appropriate clusters within the data. In classification, the goal is instead to be able to predict the

correct label when provided the features. Spectral clustering takes the similarity matrix as an input besides the dataset, while classification takes the class labels as additional input.

We consider an application of spectral clustering to a dataset having two intertwined ring-shaped clusters and tune the bandwidth of the RBF kernel. In Figure 3.4 and Figure 3.5, we show two examples corresponding to two choices of σ^2. We notice that a larger bandwidth does not make a clear separation in the projected eigenspace, and as a result, the algorithm is unable to properly identify the ring-shaped clusters. If we lower the bandwidth, the projections are clearly separated into two distinct lines and the clusters are correctly identified.

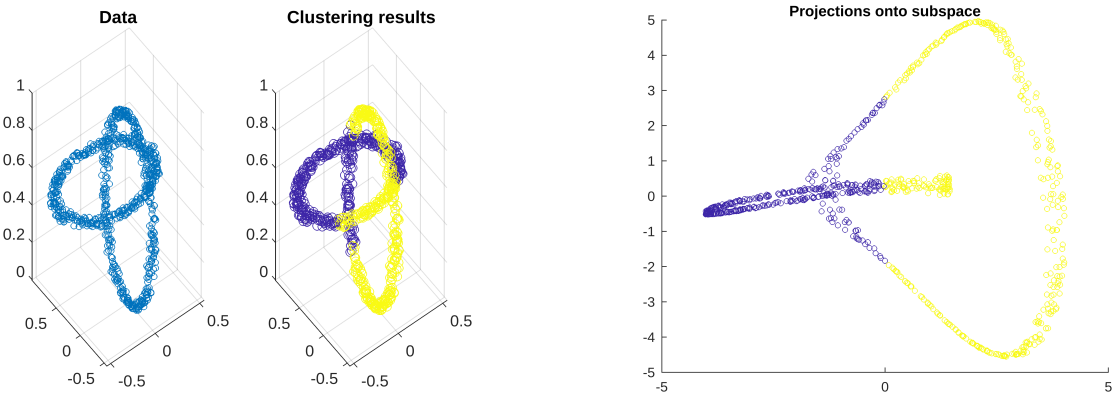

Figure 3.4: Spectal clustering application on ring-shaped clusters, with $\sigma^2 = 0.05$.

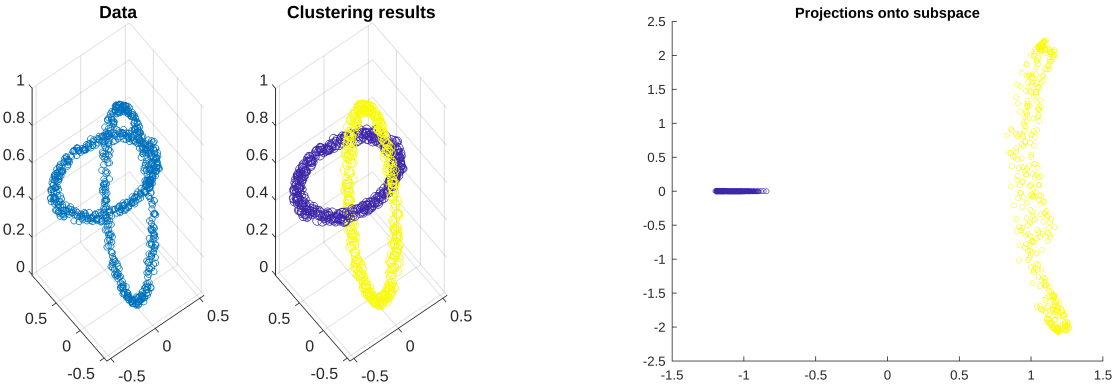

Figure 3.5: Spectal clustering application on ring-shaped clusters, with $\sigma^2 = 0.01$.

3.3 Fixed-size LS-SVM

We now turn our attention to large scale problems, where we have to deal with a $N \times N$ kernel matrix, where N is "large". We consider the fixed-size LS-SVM (FS-LSSVM) using the Nyström method which provides an approximation of the feature map. Here, we approximate the original kernel matrix with a "smaller" kernel matrix of size $M \times M$ which is based on a subset of the original dataset. Using this approximation of the feature map, we can then solve the primal problem as well. In this case, one is faced with the problem of deciding to solve the primal problem or the

dual problem for a specific application. As a general rule of thumb, it is advised to solve the dual problem in case we have a relatively small dataset from a high-dimensional space. This is because the dual problem has to be solved using the kernel matrix, which has size $N \times N$ with N the number of datapoints. Hence, in case the primal problem is computationally hard to solve due to the curse of dimensionality, the dual problem might be easier to solve. It is more effective to solve the primal problem in the opposite case, *i.e.* when the dataset is large and the number of input features is quite limited. In this case, one could work with the Nyström approximation of the feature map and directly solve the primal problem.

While the subset used in the approximation can in principle be sampled randomly, one can also create more representative subsets based on some criterion, such as a subset that maximizes the quadratic Renyi entropy. As seen in the lectures, the quadratic Renyi entropy H_R can be computed in terms of the smaller kernel matrix $\Omega_{(M,M)}$:

$$H_R = -\log\left(\frac{1}{M^2}\sum_{ij}\left(\Omega_{(M,M)}\right)_{ij}\right) . \tag{3.1}$$

Taking the Gaussian RBF as an example, we study the influence of the bandwidth of the RBF kernel on the obtained subset with the quadratic Renyi entropy condition in Figure 3.6. From the figure, we conclude that the best results are obtained with an intermediate value of σ^2. Indeed, the obtained subset is spread out across the cluster, taking datapoints at the border into account as well as several datapoints closer to the centre of the cluster. A low value of σ^2, on the other hand, results in a subset with more datapoints around the centre of the cluster, with generally smaller distances among the points. On the other hand, a large bandwidth results in a subset that is mostly located around the edges of the cluster, with large distances between the points. This can intuitively be explained by considering that, in order to maximize H_R, one has to minimize the argument of the logarithm in Eq. (3.6) and hence wants to minimize the sum of the kernel matrix entries. Each kernel matrix entry depends on the distance between the datapoints (see Eq. (1.9)) and hence we want to maximize the distances. However, since the bandwidth determines the fall-off of the exponential function, for small σ, we can reach a low value in the kernel function with smaller distances, whereas large distances are required to obtain low values of the kernel function for larger σ.

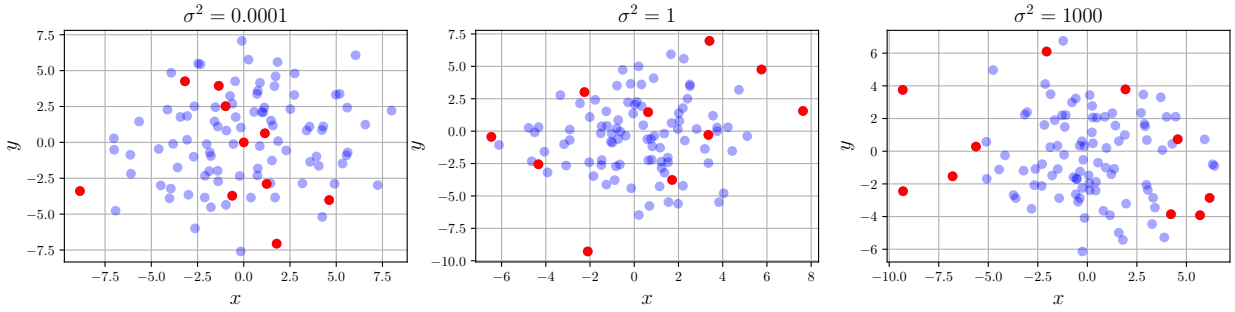

Figure 3.6: Influence of the bandwidth of the RBF kernel on the obtained subset of FS-LSSVM based on the quadratic Renyi entropy condition.

We can iteratively apply an ℓ_0-type of penalty to a pretrained FS-LSSVM to obtain sparser solutions. In the provided source code, the value of M is chosen as $M = \lceil k\sqrt{N} \rceil$, where k is a tunable hyperparameter which is set to 4 by default. We consider the first 600 datapoints of the shuttle dataset, such that M is equal to 106 for the FS-LSSVM. The results are shown in Figure 3.7. We can conclude that both methods have a very similar computational complexity. While slightly increasing the error, the ℓ_0-penalty reduces the number of support vectors and hence results in a sparser solution. For some applications, one might prefer sparsity over accuracy (for instance, if interpretability of the model is important). For such applications, applying this ℓ_0 penalty term might lead to better models.

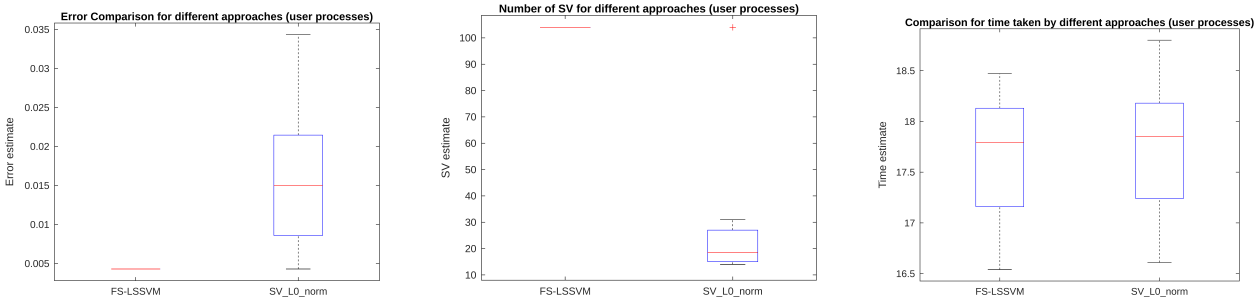

Figure 3.7: Comparison between FS-LSSVM with and without an ℓ_0-penalty.

3.4 Denoising digits

We study denoising handwritten digits using kernel PCA. Figure 3.8 shows a comparison between kernel PCA with a Gaussian RBF (with `sigmafactor`, as defined in the `digitsdn` script, equal to 0.7) and linear PCA. It is clear that the kernel PCA results in a higher quality for the denoised images. Whereas increasing the number of components for the linear PCA case can easily start including information from the noise and make the images more blurry, this is less easily an issue for kernel PCA. This is likely due to the fact that kernel PCA learns principal components obtained from abstract, non-linear features obtained from the original, raw data, which can be more informative if the bandwidth is chosen properly. Put differently, the feature map can transform the data in such a way that the noise and the actual data are easier to separate.

In Figure 3.9, we study the influence of the bandwidth of the RBF kernel on the denoising process. A first remarkable difference with the default parameter setting of Figure 3.8 is that the blurriness of the digits is affected by the bandwidth. That is, a tuned bandwidth, such as shown above, results in clearly defined images. On the other hand, extremely high or low values for the bandwidth result in blurry digits. Moreover, increasing the number of principal components for a large bandwidth results in more noisy images compared to a lower number of components.

To understand this, we analyze the magnitude of the eigenvalues of the kernel matrix for varying values of σ^2 in Figure 3.10. In Figure 3.10, we notice that for low values of σ^2, the eigenvalues are almost all identical to 1 with only a few of the last principal components (not shown here) having a low magnitude. This can be understood by considering how the kernel matrix, given by Eq. (1.9),

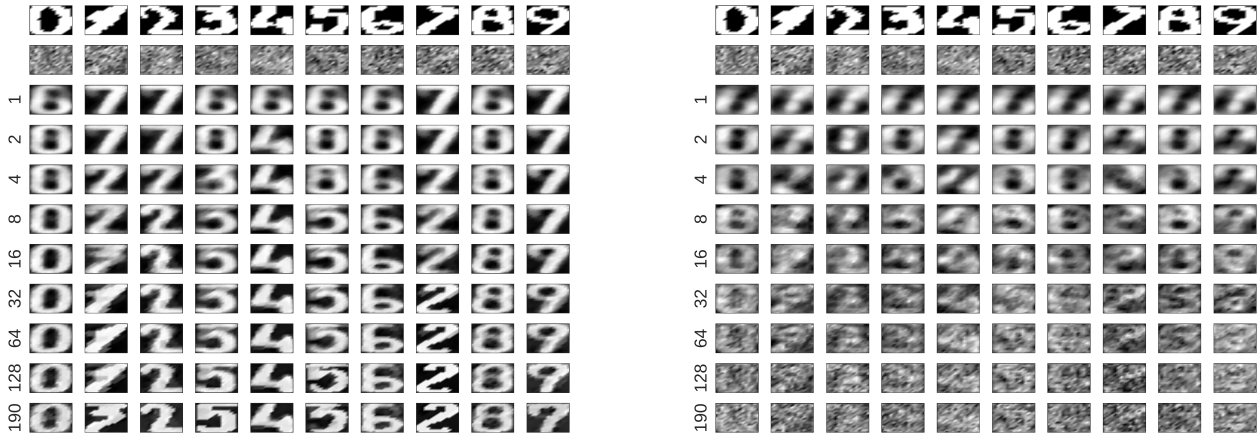

Figure 3.8: Comparison between kernel PCA (left) and linear PCA (right) for denoising handwritten digits. The kernel PCA uses an RBF kernel with `sigmafactor` $= 0.7$.

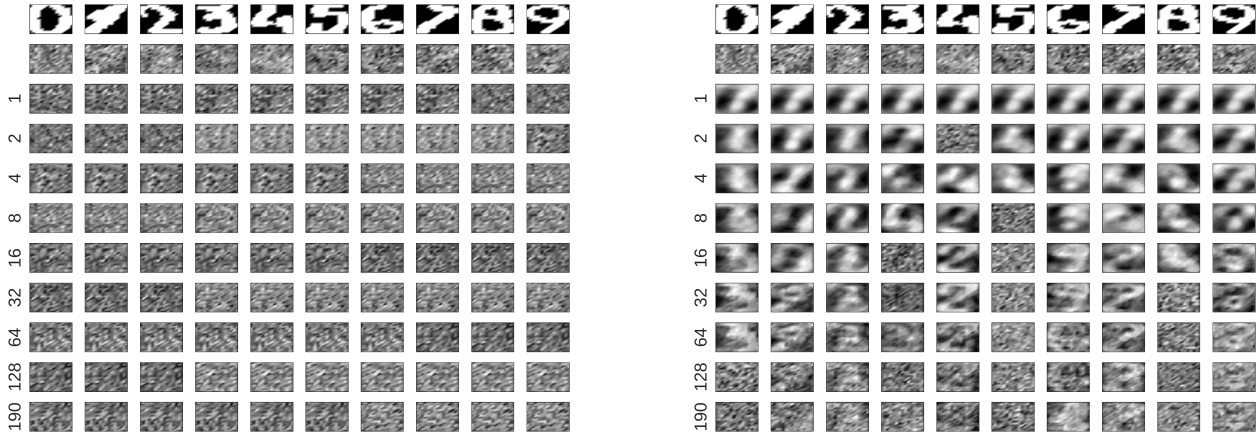

Figure 3.9: Influence of bandwidth of the RBF kernel on denoising. *Left*: $\sigma^2 = 0.0001$. *Right*: $\sigma^2 = 100$.

looks like for low σ^2. To discuss low bandwidth, let us consider the extreme case of $\sigma \to 0$, such that the RBF kernel reduces to a Dirac delta distribution. In this case, the kernel matrix is diagonal with all entries equal to 1. As a result, all the eigenvalues of the kernel matrix are equal to 1. This is to a good approximation the case for the digits denoising example considered here. As a consequence, each principal component is rather uninformative, and we need many components to explain a large amount of the variance of the dataset, such that the denoising quality is poor. On the other hand, if the bandwidth is large, then we can again gain understanding in the situation by considering an extreme case. We notice that the RBF kernel for $\sigma \to +\infty$ reduces to a constant function with a value of 1. As such, all the entries of the kernel matrix would be equal to 1 and the matrix has only one non-zero eigenvalue, *i.e.* only one principal component. In a more realistic situation, we expect that a large bandwidth gives an RBF kernel which, as discussed more extensively in Section 1, is a close approximation of the linear kernel. Therefore, we expect similar results as the linear PCA. In particular, there are likely only a few large eigenvalues. We indeed observe such a distribution for $\sigma^2 = 100$, as Figure 3.10 shows. We also observe this in the denoising procedure: the reconstruction with a single component is largely dominated by a principal component resembling a 7. As in the

linear case, increasing the number of principal components starts including components with a small eigenvalue, making the denoised digits blurry. We notice from Figure 3.10 that further increasing the bandwidth causes the magnitude of the eigenvalues overall to drop, indicating that the principal components become less informative.

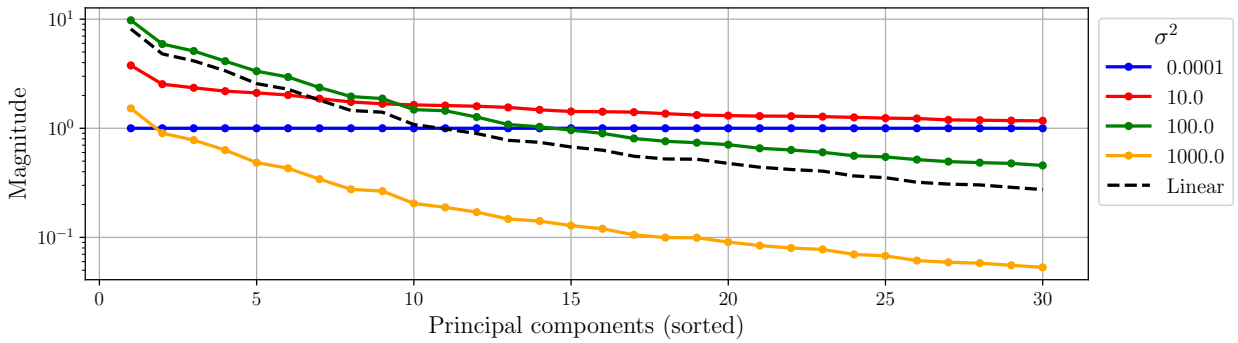

Figure 3.10: Magnitude of eigenvalues for the largest principal components for linear and kernel PCA with an RBF kernel.

We investigate the reconstruction errors on the training set in Figure 3.11. We report the sum of the reconstruction errors rather than the mean, to compare the performance of the denoising procedure across the entire datasets. We notice that the best results are obtained with a lower value of the kernel bandwidth parameter. Intuitively, this agrees with the discussion made above that lower values of the bandwidth parameter are more likely to result in informative principal components. Generally, the reconstruction error decreases with increasing number of principal components. This is as expected, since taking into account more principal components accounts for more information of the dataset. The reconstruction errors on the test sets are shown in Figure 3.12. We clearly notice that the error is overall larger than on the training set. This is to be expected: since the transformation matrices are learned from the training data, the reconstruction error will typically be lower on the training set. While both test sets seem to have different values for the errors, they overall have the same dependence on the number of components and the bandwidth, which moreover seems to agree with the observations made on the test set. However, larger bandwidths are even more disfavoured than the reconstruction on the training set. Likely, this means that these principal components are overfitting on the noise, and again agrees with our earlier observation that the principal components for large σ^2 are smaller in magnitude and hence less informative.

3.5 FS-LSSVM applications

We consider applications of FS-LSSVM to a classification and regression problem. A random sample of the datasets under consideration are shown in Figure 3.13, using t-SNE plots. For the FS-LSSVM applications, we restrict ourselves to 5 000 datapoints, since training becomes computationally infeasible otherwise. As such, we tune and test FS-LSSVMs with the size of the smaller kernel matrix determined by $M = \lceil k\sqrt{N} \rceil = 283$.

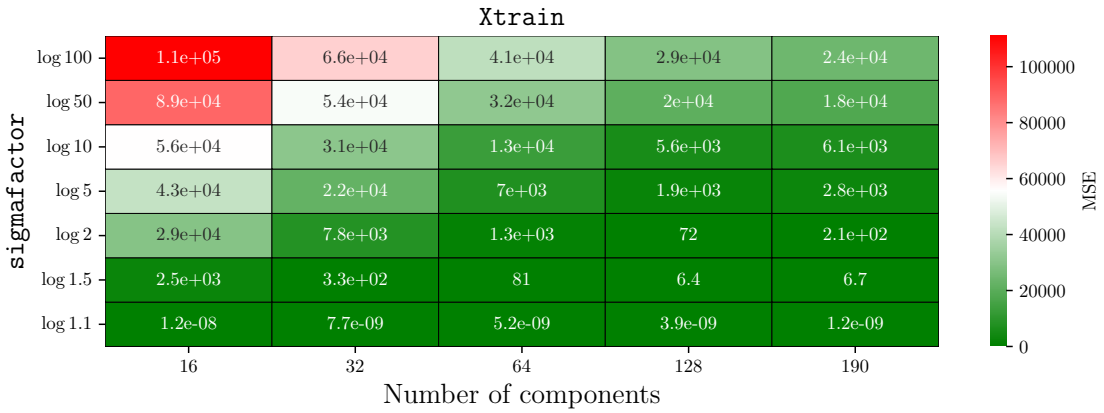

Figure 3.11: Reconstruction errors for denoising digits on the training set.

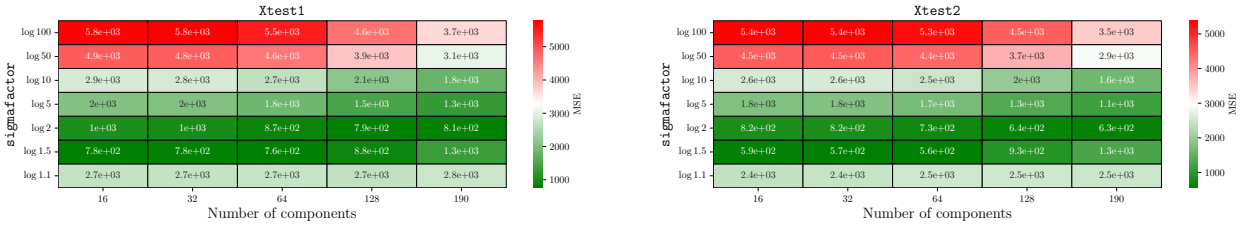

Figure 3.12: Reconstruction errors for denoising digits on the two test sets.

3.5.1 Classification: shuttle dataset

We consider the shuttle dataset from Ref. [8]. The data consists of 9 input features and has 7 different classes, of which around 80% corresponds to the first class. Hence, in this classification problem, one has to at least be able to achieve more than 80% accuracy in order to beat a classifier that predicts the majority class. There are 58 000 datapoints in the dataset. The input features are all numerical. The 7 output classes encode the following levels: Rad.Flow, Fpv.Close, Fpv.Open, High, Bypass, Bpv.Close and Bpv.Open. The labels are turned into binary attributes: class label 1 becomes the positive class while all other class labels are grouped together into the negative class. The FS-LSSVMs are able to get accurate predictions as shown by Figure 3.14. In particular, the sparse SVM has a comparable accuracy compared to the original SVM, while requiring on average only around 200 support vectors.

3.5.2 Regression: California dataset

Finally, we consider the California dataset of housing prices. In this regression problem, one has to predict the median house value based on the following 8 features: longitude, latitude, median age of a house within a block, total number of rooms, number of bedrooms, population within a block, number of households and median income for households within a block. There are 20 640 datapoints in the dataset. Given the output range of the housing prices, given in Figure 3.13, both

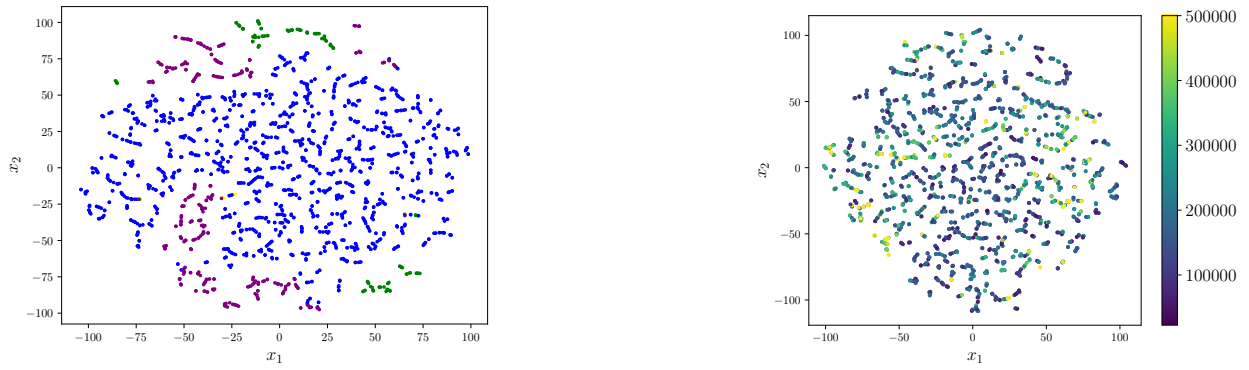

Figure 3.13: Left: t-SNE plot of the shuttle dataset. *Right*: t-SNE plot of the California dataset.

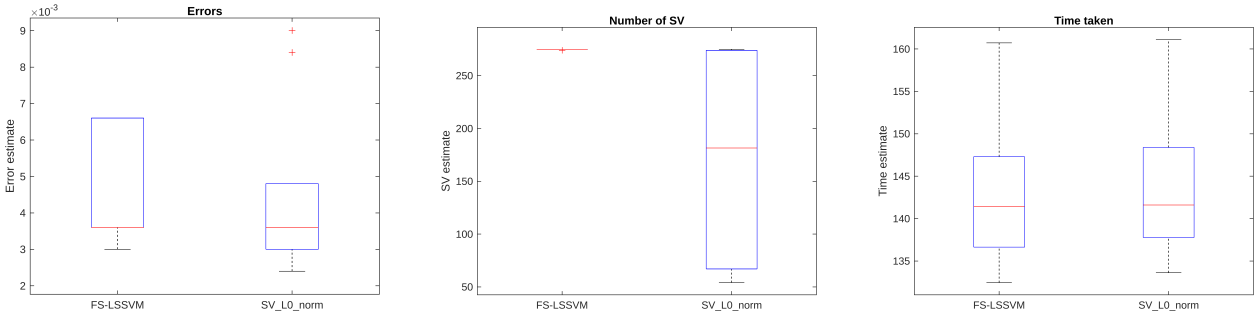

Figure 3.14: Performance of FS-LSSVM, with and without an ℓ_0-penalty, on the shuttle dataset.

LS-SVMs have good performance. The performance of the sparse SVM is slightly worse, although the difference is negligible. Instead of 283 support vectors, the sparse model requires only around 100 support vectors on average, providing a huge simplification of the original model.

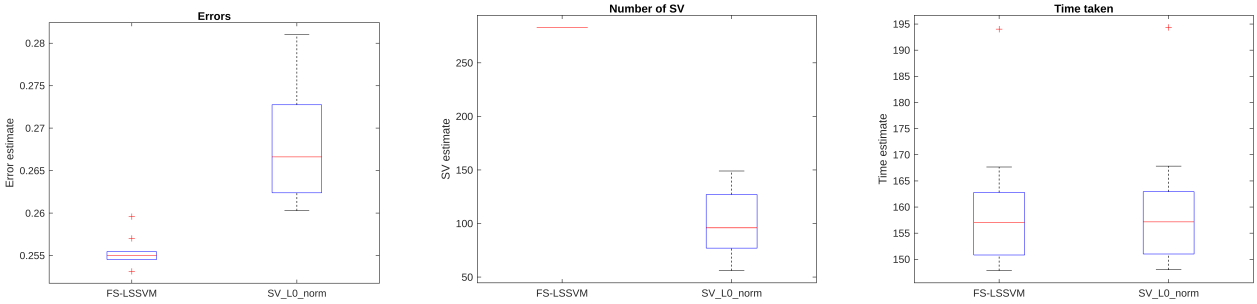

Figure 3.15: Performance of FS-LSSVM, with and without an ℓ_0-penalty, on the California dataset.

References

[1] Andrej Karpathy. *Support Vector Machine in Javascript: demo*. https://cs.stanford.edu/people/karpathy/svmjs/demo/. Accessed: 2023-05-03.

[2] Kris De Brabanter et al. *LS-SVMlab toolbox user's guide: version 1.7*. Katholieke Universiteit Leuven, 2010.

[3] *What Is Simulated Annealing?* https://nl.mathworks.com/help/gads/what-is-simulated-annealing.html. Accessed: 2023-05-03.

[4] Neptune.ai. *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?* https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc. Accessed: 2023-05-03.

[5] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. https://archive.ics.uci.edu/ml/index.php. Accessed: 2023-05-03.

[6] Scikit-learn. *sklearn.manifold.TSNE*. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. Accessed: 2023-05-20.

[7] Scikit-learn. *2.2. Manifold learning*. https://scikit-learn.org/stable/modules/manifold.html. Accessed: 2023-05-20.

[8] Jason Catlett. *Statlog (Shuttle) Data Set*. https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle). Accessed: 2023-05-20. 2023.