KATHOLIEKE UNIVERSITEIT LEUVEN

SUPPORT VECTOR MACHINES

EXERCISE SESSIONS REPORT

Student:
Henri DE PLAEN r0681349

Professor: Johan SUYKENS

H02D3a
2017 - 2018

August 16, 2018

Contents

Exercise Session 1

Classification

1.1 A Simple Example: Two Gaussians

Figure 1.1 shows some empirical ways of separating two datasets. A classifier would be valid as it performs better than random assignation as for the three black lines. Using the green or blue line as a classifier boundary would just result in a 50% performance, like random classifier. This corresponds to want a classifier whose ROC curve is above the diagonal. An optimal classifier would be a classifier which makes no mistake, or equivalently that has a performance score of 100%. This is not possible in our case: the purple rectangle indicated a subset where no linear boundary can delimit the red circles from the blue circles.

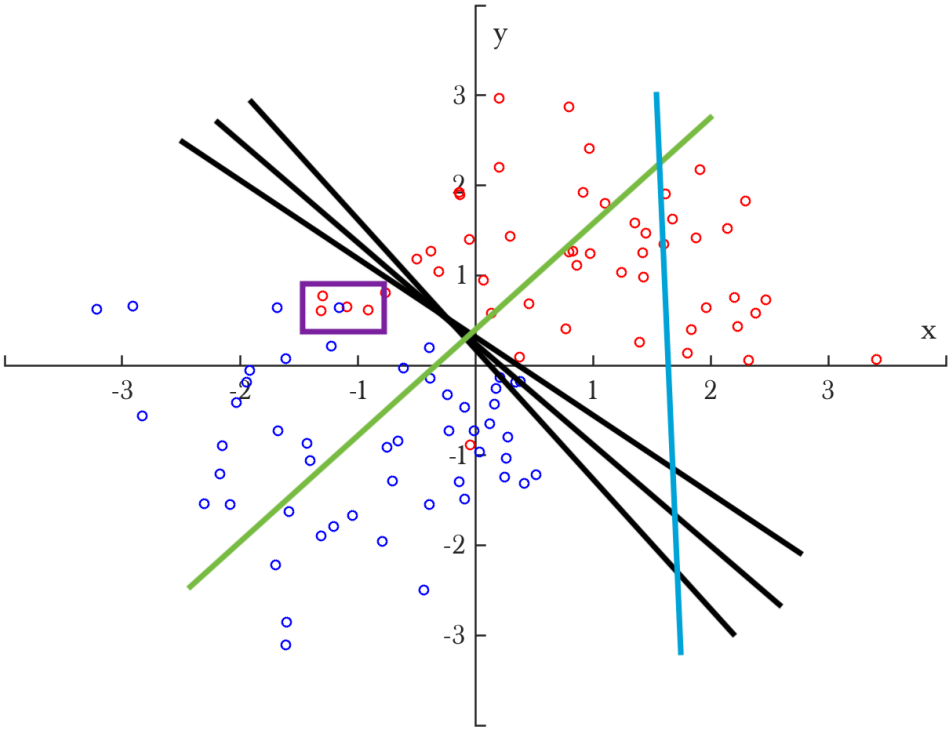


FIGURE 1.1: Empirical temptations of linearly separating two data-sets.

A way of computing a boundary is with SVM's (Vapnik). In its primal form, the support vector

machine with linear kernel corresponds to the following minimisation problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} w^T w + c \sum_{k=1}^{N} \xi_k \\ \text{subject to} \quad & y_k \left[w^T x_k + b \right] \geq 1 - \xi_k, \ k = 1, \ldots, N \\ & \xi_k \geq 0, \ k = 1, \ldots, N. \end{aligned} \tag{1.1}$$

Figure 1.2 tries to give a geometric interpretation of this equations. The final classifier will be given by the black line on the figure, with equation $y(x) = w^T x + b$ surrounded by the two lines, the red with equation $y_1(x) = y(x) + 1$ and the blue with equation $y_2(x) = y(x) - 1$. Lets first have a look at the two constraints

- $y_k \left[w^T x_k + b \right] \geq 1 - \xi_k, \ k = 1, \ldots, N$ is the core of the classification. It says that each data-point must be associated to its class, i.e. above the red line $y_1(x)$ for the red points and under the blue line $y_2(x)$ for the blue points: $y_k \left[w^T x_k + b \right] \geq 1$. Because the *Vapnik-Chervonenkis dimension* of this classifier is $h = 3$, it can be that it is impossible to classify correctly all data-points of the training set. We therefore tolerate this condition to be violated by the introduction of *slack variables* ξ_k which can be interpreted as penalties on the violation. They correspond to the distance of the points on the wrong side of their line with that same line. On the figure, this is represented by the length of the coloured perpendicular lines to $y_1(x)$ or $y_2(x)$.
- To ensure that the penalties are indeed penalties, they must be positive. $\xi_k = 0$ when its corresponding data-point is at the good side of its corresponding line $y_k y_i(x_k) \geq 1$. For points at the wrong side of their respective line, ξ_k is positive. If this condition of positiveness was not verified, we could have a point that is "rewarded" for not being well classified falsifying the whole idea of classification. Interesting is that points that are well classified, thus on the good side of the central line, but on the other side of their color line are also penalised. This is because they fall in the region between the two color lines which is to be maximised without points in it.

The objective function also consists of two parts:

- $\frac{1}{2} w^T w$ consists in maximising the normal distance $\frac{2}{\|w\|}$ between the red $y_1(x)$ and the blue line $y_2(x)$ surrounding $y(x)$. Ideally, this distance should be maximal. In the case where all data-points can be linearly separated, this would correspond to the case where the distance of the separation line $y(x)$ and the closest point of each class is maximal.
- $c \sum_{k=1}^{N} \xi_k$ tries to minimise the *slack variables* corresponding to the violation of the classification constraints. The more these violations are considered to be important, the greater c should be. By varying this parameter trade-off c, one can decide whether the violation should be absolutely minimised or the distance between the red $y_1(x)$ and the blue line $y_2(x)$ should be maximised.

The classifier can only be optimal if $\xi_k \leq 1$ for all k, which means that all data-points of a given colour are on the same side of the separating line $y(x)$. In our equations, this means solving the case for $c = \infty$, but is not possible here. Conversely, if we set $c = 0$, we allow the ξ_k taking arbitrarily high values and thus points being badly classified without it having any influence. The sole distance of the "buffer zone" would be maximised, having all its points in it. This would obviously lead to an incorrect model.

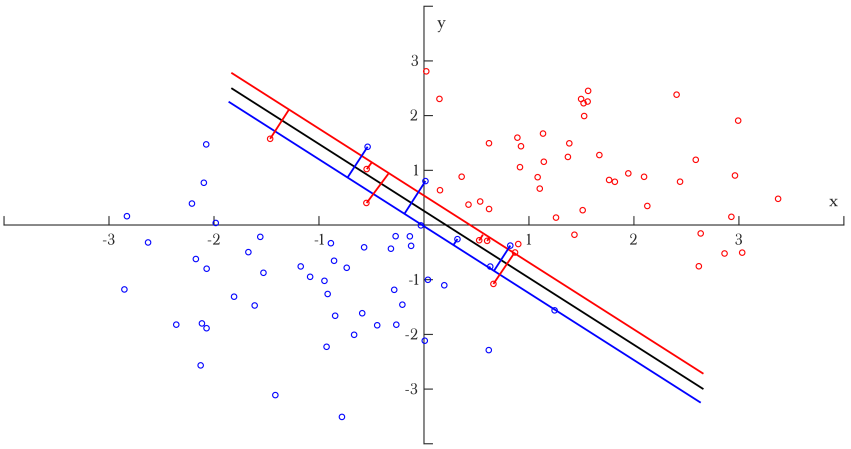


FIGURE 1.2: Geometric representation of a support vector machine with linear kernel where misclassification cannot be avoided.

1.2 The Support Vector Machine

1.2.1 Linear kernel

When data-points are added near the boundary, they have as significant impact on that boundary, as their corresponding slack variables have a weight on the objective function. When the data-points are added far from the boundary, outside band defined by $y_1(x) = y(x) + 1$ and $y_2(x) = y(x) - 1$, their slack variables are null and that addition has thus no effect on the boundary. This phenomenon can be seen at figure 1.3.

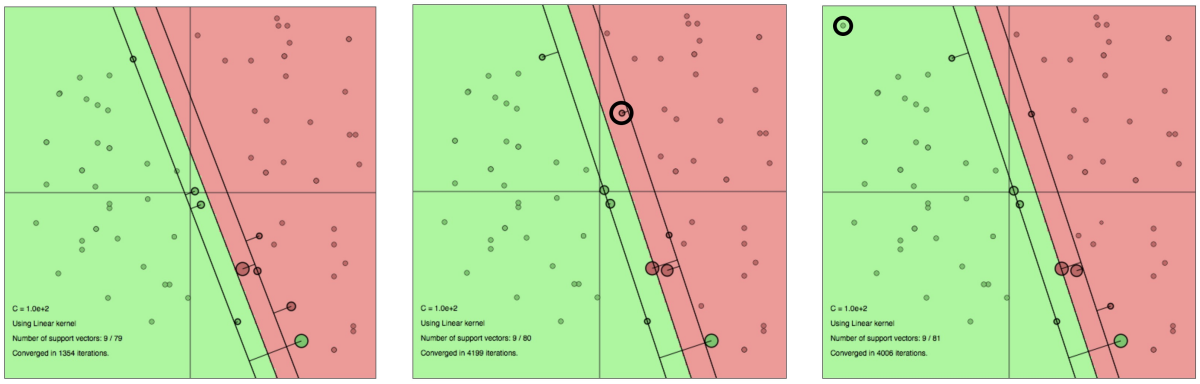


FIGURE 1.3: Influence of the addition of data-points on the boundaries. The parameter value is $c = 100$.

When data-points are added on the wrong side of the boundary, their corresponding slack variables are adding a huge wieght on the objective function. It thereby will profoundly affect the boundary. Indeed, the weight of the slack variables on the objective fonction grows quadratically with the distance that separates their corresponding data-point to the corresponding side of the boundary band. This boundary is thus re-ajusted. This can be seen on figure 1.4 where the re-ajustement of the boundary lets some improvement for minimising the other term $w^T w$ and widening the boundary band.

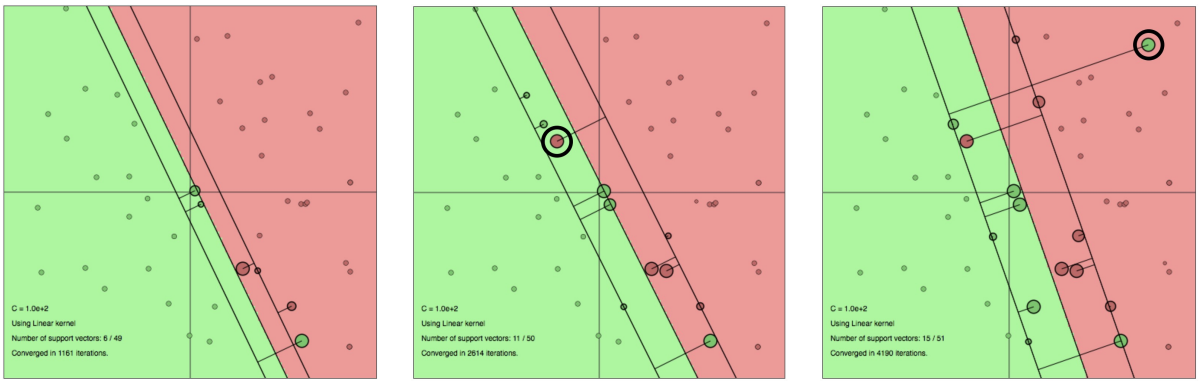


FIGURE 1.4: Influence of the addition of *outlying* data-points on the boundaries. The parameter value is $c = 100$.

As mentioned in the previous section, the regularisation parameter c controls a trade-off between the widening of the boundary band and the minimisation of the slack variables in the objective function. The lower it is, the less slack variables have a weight compared to the widening of the boundary band. This trade-off can be observed at figure 1.5

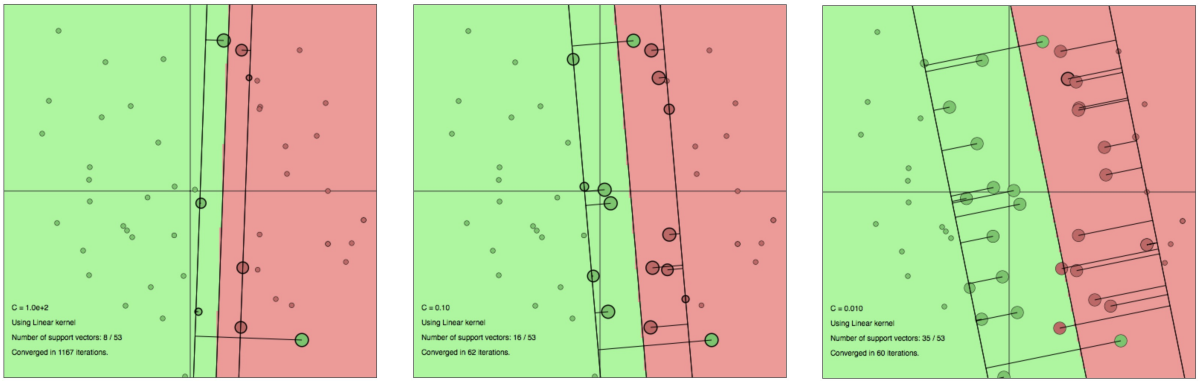


FIGURE 1.5: Influence of the parameter value c on the boundaries. The parameter values are respectively $c = 100$, $c = 0.10$ and $c = 0.01$.

1.2.2 RBF kernel

By changing to a RBF kernel, the boundary does not consist in a line anymore, but in a sum of circles around each datapoint. In a certain sense, each data-point has set an "influence zone" around itself. This can be seen at figure 1.6.

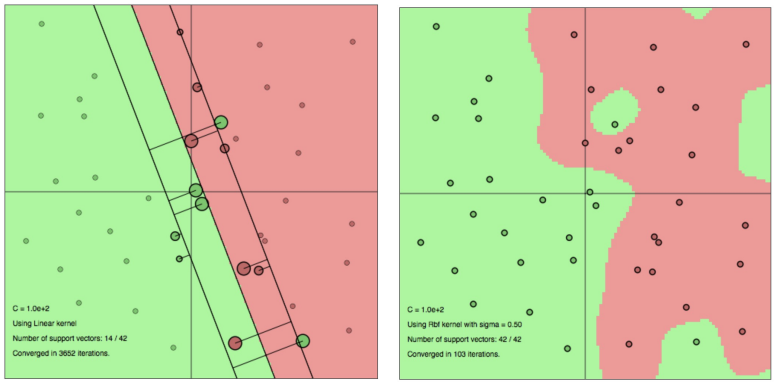


FIGURE 1.6: Influence of the kernel functions on the boundaries. A *linear kernel function* is used left with $c = 100$ and a *RBF kernel function* with $c = 100$ and $\sigma = 0.5$ is used right.

This "influence zone" can be controlled by a parameter σ. In the dual space, the classifier is given by $\hat{y} = \text{sign}\left(\sum_{k=1}^{N} \alpha_k K(x_k, x) + b\right)$ with $K(x_k, x) = \exp\left(-\frac{\|x_k - x\|^2}{2\sigma^2}\right)$. The classifier is thus a sum of "influence zones" or Gaussian distributions around each of the point of the training set. The bigger the α_k, the more its corresponding point has weight in this classifier. These "influence zone" have thus all the same size, defined by σ, but may very in weight. The influence of σ can be seen at figure 1.8.

Here again, the regularisation parameter c controls the weight of misclassification errors. In the dual problem, it can also be seen as an upper bound on the α_k. The smaller this value, the less the data-points matter compared to the smoothness of the classifier. For extreme low values, the data-points have almost no weight and the function is over-smoothed resulting in a uniform and thus unnecessary classifier. This phenomenon can be seen at figure 1.7.

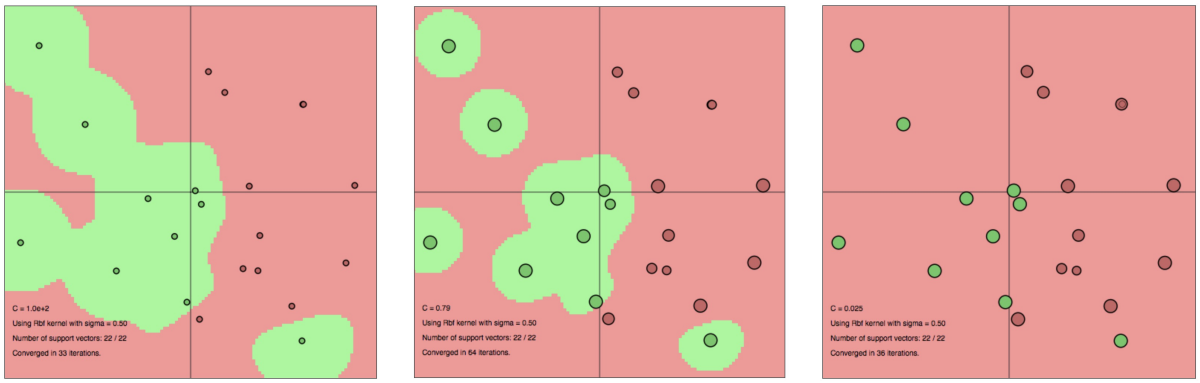


FIGURE 1.7: Influence of the parameter value c on the boundaries. The parameter values are respectively $c = 100$, $c = 0.79$ and $c = 0.025$ ($\sigma = 0.5$ everywhere).

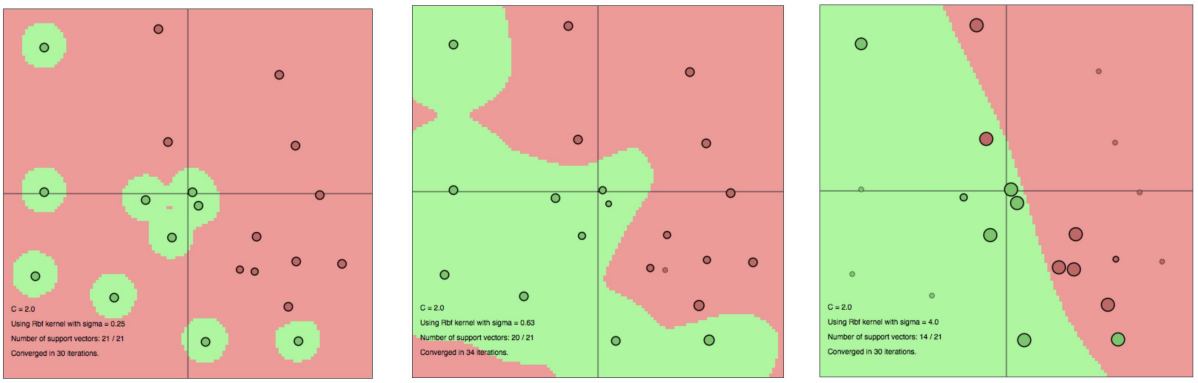


FIGURE 1.8: Influence of the parameter value σ on the boundaries. The parameter values are respectively $\sigma = 0.25$, $\sigma = 0.63$ and $\sigma = 4.0$ ($c = 2.0$ everywhere).

In the case of *non-separable datasets*, c controls the smoothness of the function. For high values, disconnected zones are not favoured, whereas as high values tend to result in a lot of disconnted zones as in figure 1.9.

The parameter σ still controls the "influence zone" of each data-point. For very low values of σ, the sole close neighbourhood of each data-point. This results in highly disconnected sets and low generalisation capability, a clear case of *overfitting*. Extreme high values leads to an addition of the "influence zone" of each data-point everywhere, leading to a clear break between both classes, very similar to the results of a linear kernel (figure 1.10).

The best parameter values choice is always a trade-off. Based on the previous experiments on the *non-separable dataset*, I would define them as being $c = 0.2$ and $\sigma = 1.0$.

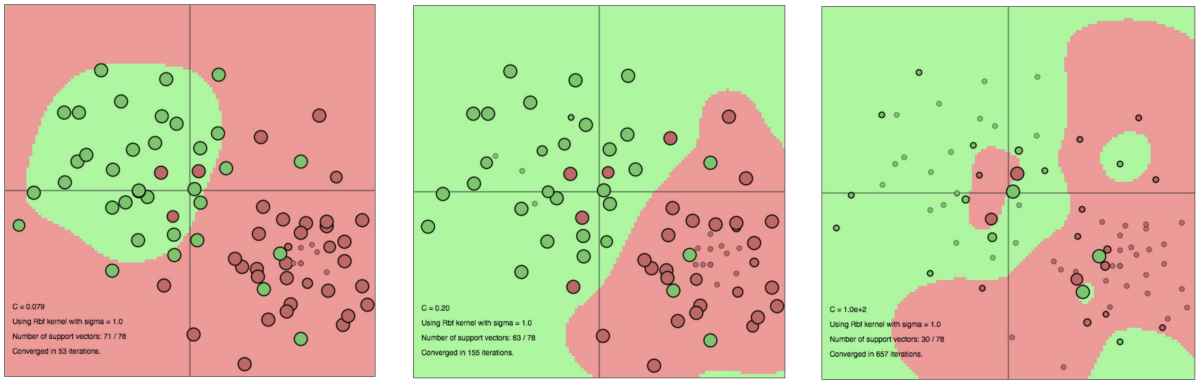


FIGURE 1.9: Influence of the parameter value c on the boundaries of *non-seperable datasets*. The parameter values are respectively $c = 0.079$, $c = 0.20$ and $c = 100$ ($\sigma = 1.0$ everywhere).

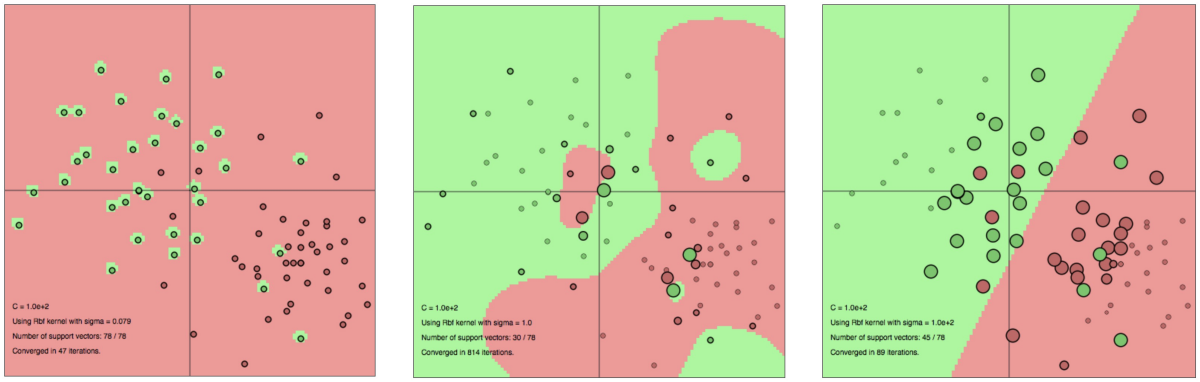


FIGURE 1.10: Influence of the parameter value σ on the boundaries of *non-seperable datasets*. The parameter values are respectively $\sigma = 0.079$, $\sigma = 1.0$ and $\sigma = 100$ ($c = 100$ everywhere).

The support vectors are the data-points x_k used in the classifier $\hat{y} = \text{sign}\left(\sum_{k=1}^{N} \alpha_k K(x_k, x) + b\right)$ with $K(x_k, x) = \exp\left(-\frac{\|x_k - x\|^2}{2\sigma^2}\right)$ in the case of RBF kernels. To be present in the summation, their respective support value must be $\alpha_k \neq 0$. When adding $K(x_k, x)$ doesn't add anything interesting enough to compensate the smoothness loss, its respective $\alpha_k = 0$ and x_k doesn't become a support vector. The addition of two data-points can be seen at figure 1.11, one for which adding its corresponding $K(x_k, x)$ adds a significant interest for the classifier and one without. Similarly, figure 1.12 shows different parameter values combinations: one where there is a significant disparity in the support vector values due to the phenomenon that has just been explained and another where overfitting results in all the support vectors just influencing their close neighborhood without being in competition with each other.

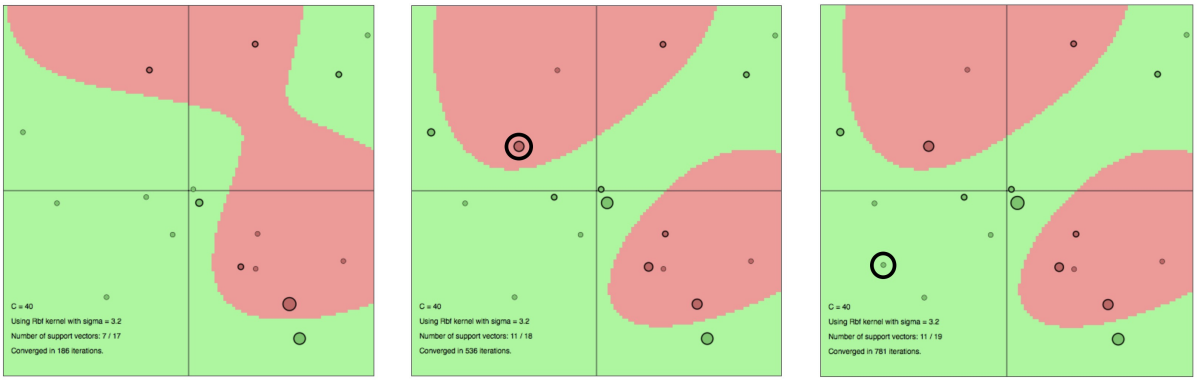


FIGURE 1.11: Influence of the addition of data-points on the *support vectors*. The parameter values are $c = 40$ and $\sigma = 3.2$.

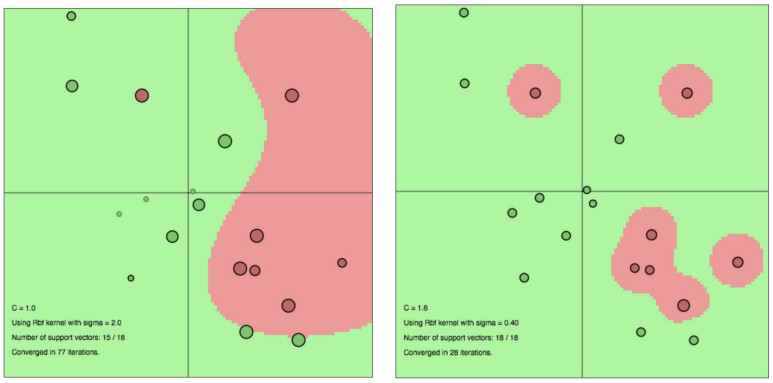
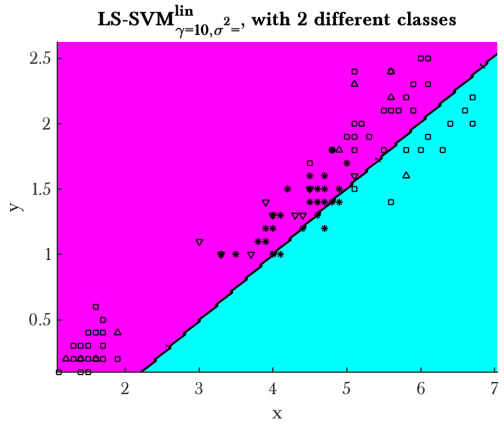


FIGURE 1.12: Influence of parameter values on the importance of the *support vectors*. Left is $c = 1.0$ and $\sigma = 2.0$. Right is $c = 1.5$ and $\sigma = 0.40$.

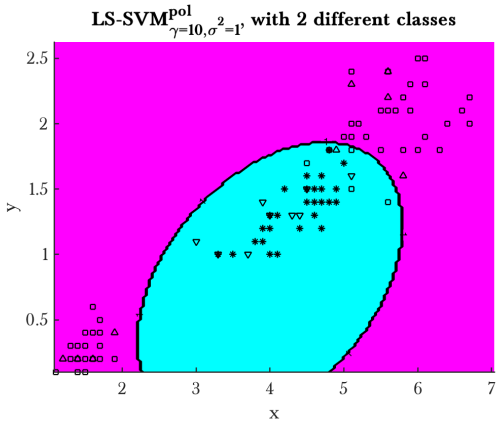
1.3 Using LS-SVMlab

The use of polynomial kernels allows the addition of higher order terms, and thus more than a linear interaction between the different support vectors. In other words, polynomial kernels represent the similarity of the support vectors in a vector space of higher dimensions than the support vector space, which allows the representation of a non linear classification model. An example is given at figure 1.13. Nevertheless, adding higher order interactions means augmenting the risk of *overfitting* and one must be very cautious in the choice of the degree. The main idea is to always respect *Occam's razor principle* which can be summarised by "less is better". The lowest degree that produces a good performance should be chosen. The third degree adds the possibility of cubic shaped curves where the second degree only allows quadratic shaped curves for example. In our example, the third and the fourth degree clearly bring nothing more than the second degree and this last one should be chosen. The better performance of the higher degrees may be a sign of overfitting.

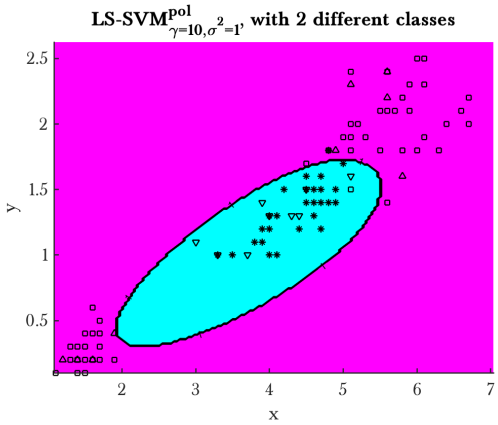
Even though both parameters, the degree of a polynomial kernel and the σ^2 of a RBF kernel, should be chosen with a lot of caution to avoid any overfitting while allowing specification of the model, the σ^2 doesn't add complexity to the model in a direct way. It is true that a low σ^2 will result in augmenting the number of support vectors, but the complexity gain is not as straightforward as for the degree of the polynomial kernel. Figure 1.14 shows how a too low choice of σ^2 can result in overfitting and thus a bad result on the test set and a too high choice may lead to bad specialisation of the model.



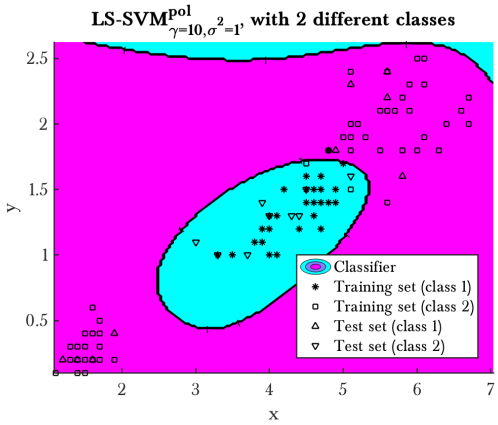
(A) *Linear* kernel function. Prediction error of 55%.



(B) Polynomial quadratic function of *degree 2*. Prediction error of 5%.



(C) Polynomial quadratic function of *degree 3*. Prediction error of 0%.



(D) Polynomial quadratic function of *degree 4*. Prediction error of 0%.

FIGURE 1.13: Influence of the degree of polynomial kernel functions on the classification. The other parameter values are $\gamma = 10$, $t = 1$ and $\sigma = 1$.

Similarly, the choice of γ is very important as is defines the chosen trade-off between smoothness of the classifier and performance on the training set. A too high value may here again lead to overfitting and a too low value to too much generalisation (figure 1.15).

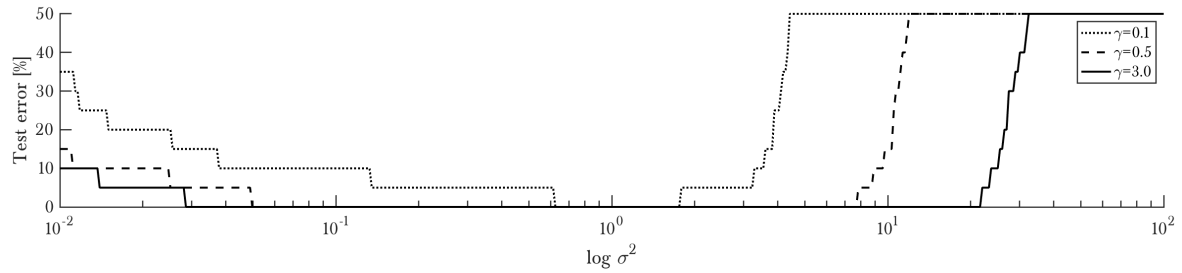


FIGURE 1.14: Influence of the σ^2 of the RBF kernel function on the performance for different values of γ.

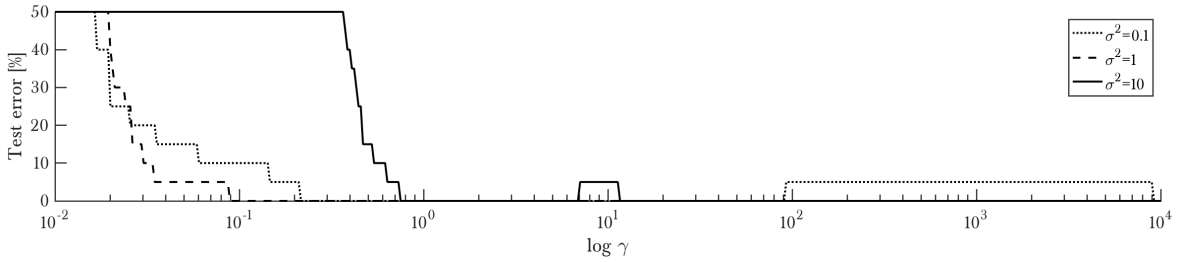


FIGURE 1.15: Influence of the γ of the RBF kernel function on the performance for different values of σ^2.

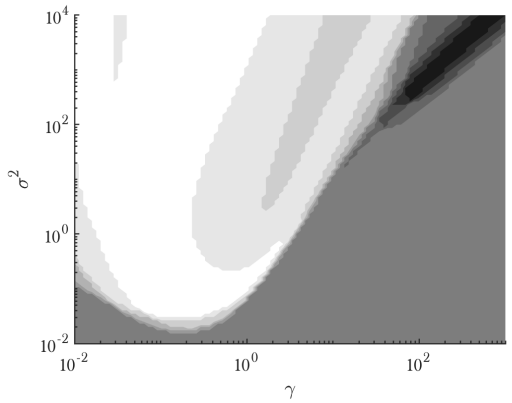
1.3.1 The choice of the Hyper-parameters

The recurring difficulty in the choice of the parameters is to avoid overfitting, i.e. the reproduction of the training set by the model and not the generalisation of its underlying relations. One way of detecting it is to use a *validation set* on which the results of the trained model can be evaluated. If the model reproduces the training set, it will not give a good performance on the validation set. Estimation of the performance by the validation set for a whole range of parameter values are given at figure 1.16a and 1.16b. The problem, especially is small datasets like the one used here is that the estimation is also very dependent on the validation set. Furthermore, is suppresses a significant part of the total data-set to be used: all the information is not exploited to train the model. Even though the main picture is the same, the details are not. The use of a validation set is thus better than no use at all, but still not perfect.

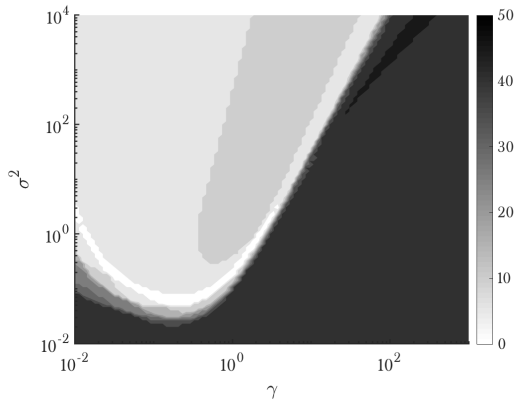
A way of exploiting all the data while keeping an estimate of the possible overfitting is to use cross-validation. This techniques comprises the separation of the whole data-set in different pairs of training and validation sets. The method above is then applied on each of these pairs and the performance is then defined my taking the mean of the performances on all the validation sets. This can be seen as a statistical generalisation of the classical validation set technique. Its drawback is the execution time, but its advantage is the use of the whole training set and thus the training with all available information. We now have an unbiased estimator of the performance for a given data-set and model (figure 1.16c).

We still have the problem of not letting the model being trained on the whose available information. This can be solved by using the *leave-one-out* method. It is very similar to cross-validation but each training-validation pair comprises one sole data-point in the validation set. The process if thus much longer, but almost all information is used for the training. It is a sort of extreme cross-validation and is much more precise (figure 1.16d). This gives an unbiased estimator, but with better overall performance. The sole disadvantage is the higher variance of the estimator.

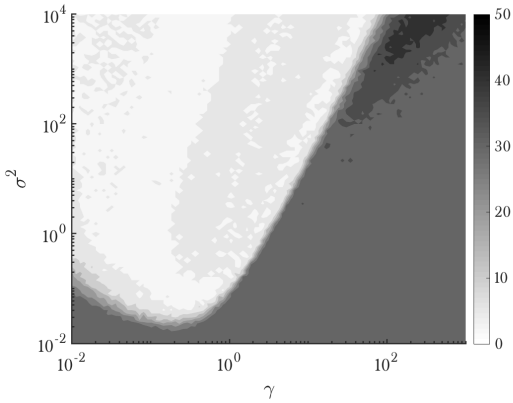
In the case of a problem with a small data-set as this one, the leave-one-out method should be prefered as the maximum information possible should be used for the training. Furthermore, execution time is certainly here not a bottleneck.



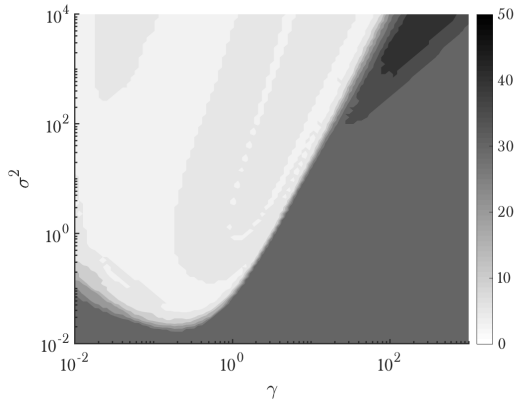
(A) Random training set (80%) and test set (20%) #1.



(B) Random training set (80%) and test set (20%) #2.



(C) 10-fold cross-validation.



(D) Leave-one-out.

FIGURE 1.16: Comparison of the misclassification performance in function of parameters σ^2 and γ for different validation schemes.

Using these "overfitting estimators", one can now tune the parameters by minimising the estimator. SVM's have this superb particularity of being convex problems. Unfortunately, the tuning of the parameters is often not convex at all as can be seen in our example (figure 1.16). Therefore, multiple methods can be used to find the local minima and hopefully, the global one. The grid search works similarly as the naive approach we used up to now: trying all different possibilities in a certain range. The Nelder-Mead method uses the gradient of the function to minimise it and is thus much more sensible to local minima. Furthermore, both of these methods can be used in their sequential or parallel variants. This is a first explanation to the differences one can observe at table 1.1. These methods are further discussed in the next exercise session. The other differences are explained by the fact that the region of optimal parameters, i.e. with the best score of the validation performance, is quite large: all found parameter combinations lie in it. This also explains the optimal ROC-curve for all these combinations (figure 1.17). Using the training set to determine that curve is not a good idea as it will represent the result of the specific training and thus be subjected to overfitting. Using the test set avoids that possibility. In summary, a ROC-curve computed with the training set will also show overfitting, phenomenon that we absolutely want to avoid when creating a model comparison measure.

Optimisation technique	Nelder-Mead method			Brute Force Gridsearch		
Coupled Simulated Annealing	$\gamma = 5.7541$	$\sigma^2 = 0.0998$	cost = 4%	$\gamma = 2.6964$	$\sigma^2 = 0.0166$	cost = 3%
Randomised Directional Search	$\gamma = 8.5303$	$\sigma^2 = 1.8921$	cost = 5%	$\gamma = 0.1741$	$\sigma^2 = 0.4091$	cost = 4%

TABLE 1.1: Comparison of the hyper-parameters found by different optimisation techniques.

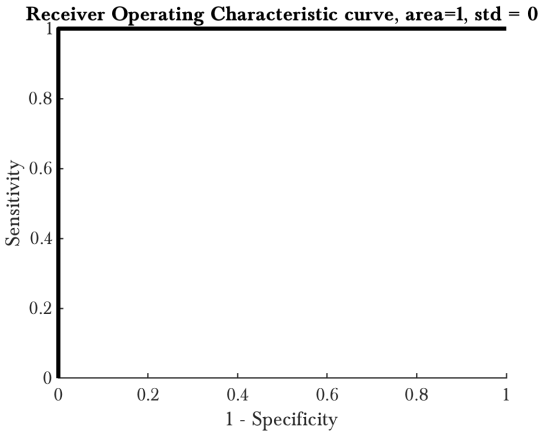


FIGURE 1.17: Optimal ROC curve for all parameter sets found at table 1.1.

1.4 Homework Problems

1.4.1 The Ripley Data-Set

The Ripley data-set consists of two classes where the data for each class have been generated by a mixture of two Gaussian distributions. The training set contains 250 data-points and the test set 1000. Both classes have the same number of data-points. A visualisation of the training set can be seen at figure 1.18). The two classes seem to not overlap much, especially in the left part of the plot. There is also a clear top-bottom distinction. At first sight, this problems seems to be well suited for an LS-SVM classification. Let's train the model now and test it on the test set (figure 1.19). The performance is 89.2%, which is quite good, but there still seems some room for improvement: a curved boundary would be more suited. Let's try RBF kernel functions.

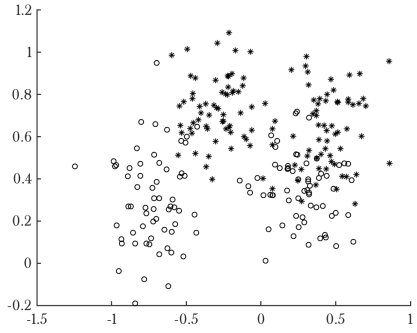
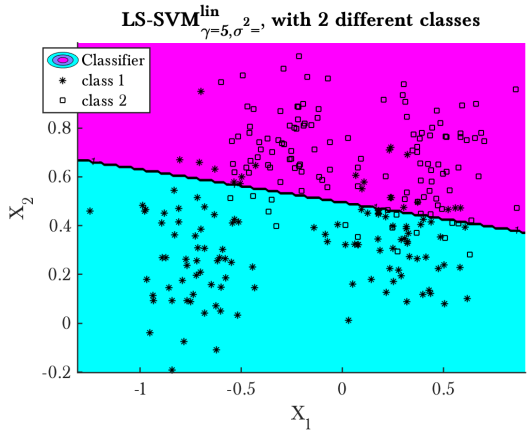


FIGURE 1.18: Training set of the Ripley data-set.

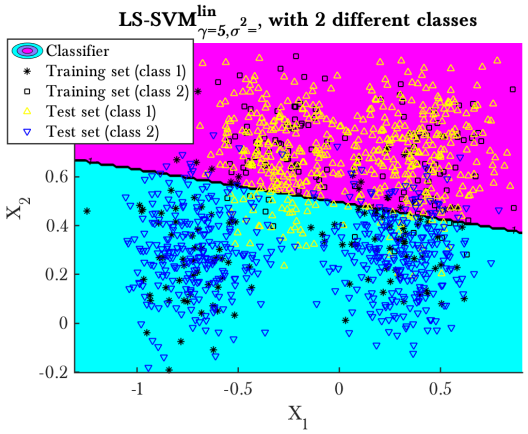
Figure 1.20 shows the results of tuned and trained RBF model. Repeated tunings reveal some small but significant variations for σ^2 which means that a less smooth boundary doesn't change much to the final results. There are more variance to the optimal γ which also means that the smoothness isn't very important. All different tunings resulted in very similar performance, here 90.8%.

The RBF kernel function seems to perform slightly better than the linear kernel function, but lets compare their respective RC-curves (figure 1.21). As predicted, the ROC-curve of the RBF kernel function performs better, but the difference is marginal.

Both of these models obtain good results even is the RBF is slightly better. This data-set is very well suited to be used with such LS-SVM's.

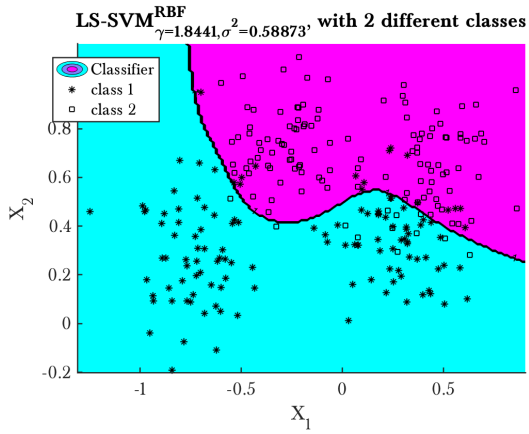


(A) Results on the training set.

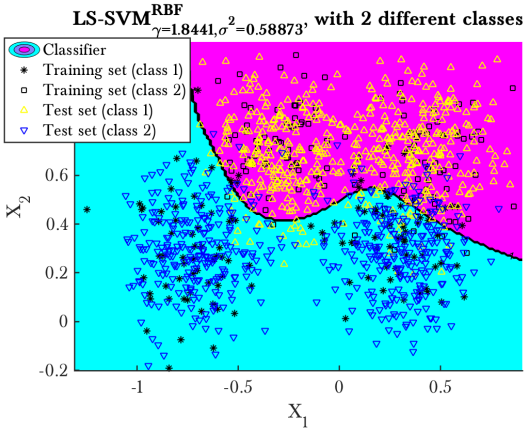


(B) Same, but test set added.

FIGURE 1.19: LS-SVM applied to the Ripley dataset with a linear kernel function. The test set performance is 89.2%.

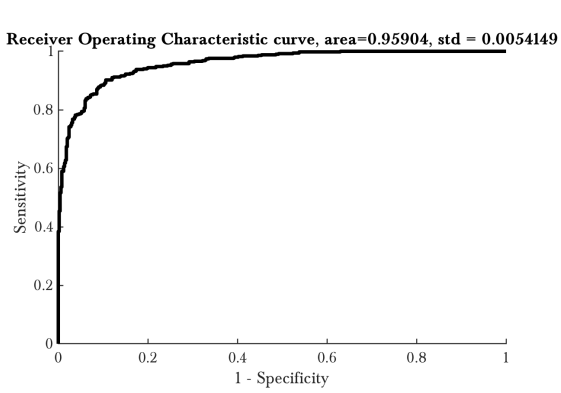


(A) Results on the training set.

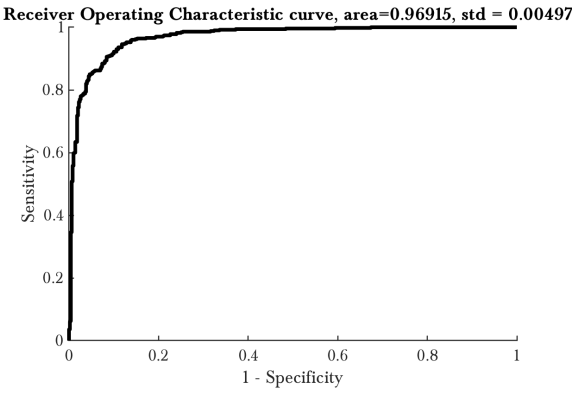


(B) Same, but test set added.

FIGURE 1.20: LS-SVM applied to the Ripley dataset with a RBF kernel function. The test set performance is 90.8%.



(A) Linear kernel function.



(B) RBF kernel function.

FIGURE 1.21: Comparison of the ROC-curves of the linear and RBF kernel functions applied to the Ripley data-set.

1.4.2 Breast Cancer Data-set

The Breast Cancer Data-Set consists of 569 data-points, a training set of 400 data-points and a test set of 169 data-points. The prior class distribution is 60%-40%. each data-point consists of 30 features, which are computed from a digitised image of a fine needle aspirate of a breast mass. They describe characteristics of the cell nuclei present in the image.

Visualising the data-set is more complicated as the input space is 30-dimensional. The disparity and the mean value of the normalised features of each data-point are plotted at figure 1.22. One can directly notice that the second class seams to have much greater feature values in general. But that's not all: the mean difference is not only true on all features, it seems to be true on almost every feature apart (figure 1.23). It seems that all 30 features are not needed and that a lot of redundancy is present. Let's try to use a linear classifier on the sole first two features (figure 1.24). The performance is 88.76% on the test set which is already a good results. We can now try with all 30 features: 95.27%. This score is very good and we could conclude that a linear classifier is sufficient according to *Occam's razor principle*. Nevertheless, let's try the RBF kernel functions.

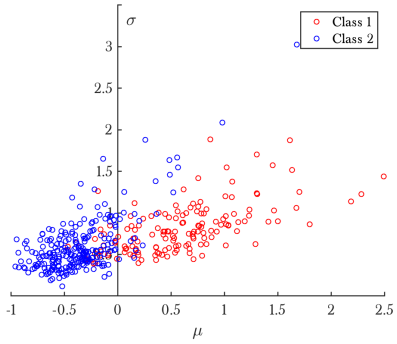
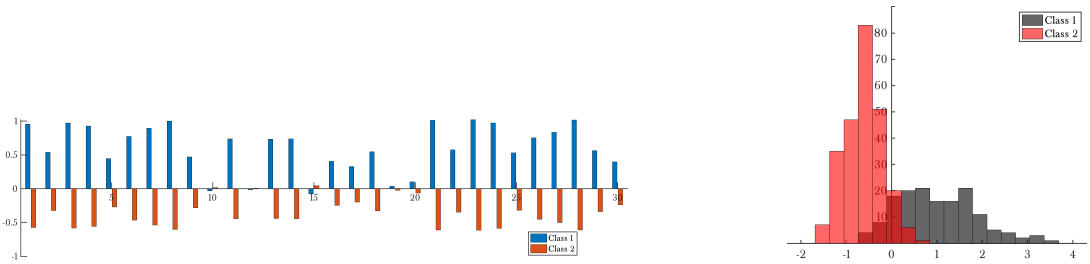


FIGURE 1.22: Disparity-Mean visualisation of the rraining set of the Breast Cancer data-set.



(A) Linear kernel function.

(B) Distribution of feature 21.

FIGURE 1.23: Comparison of the mean value of each normalised feature with the distribution of one.

Repeated tuning shows no big variation in the determination of the parameters γ and σ^2. This is due to the much bigger data-set and certainly the much higher input dimension. The found values seem to be close to the global maximum and we can thus judge them as acceptable. After training, the performance on the test set is 97.04%. This result is lightly better than the linear kernel model. After comparing the two ROC-curves (figure 1.24) and keeping *Occam's razor principle* in mind, one should conclude to keep the linear model.

The LS-SVM method works very well with this data-set, but this also due to the simplicity of the model: a lot of redundancy and almost linearly separable everywhere. Further improvements ideas could by dimensionality reduction, *e.g.* PCA, but this outside the scope of this homework.

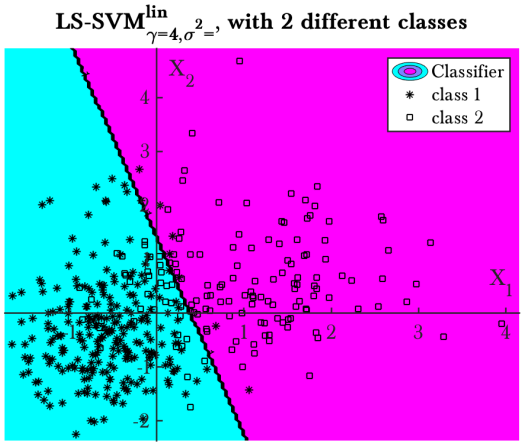
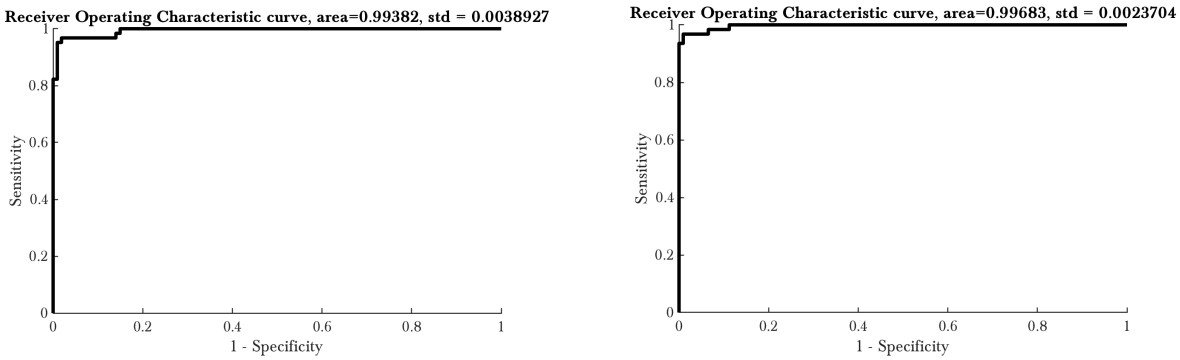


FIGURE 1.24: Disparity-Mean visualisation of the training set of the Breast Cancer data-set.



(A) Linear kernel function.

(B) RBF kernel function.

FIGURE 1.25: Comparison of the ROC-curves of the linear and RBF kernel functions applied to the Breast Cancer data-set.

1.4.3 Diabetes Database

The Pima Indian Diabetes database is a collection of 768 medical diagnosis reports of diabetes from a population living near Phoenix (Arizona, USA) which present a high incidence rate of the disease[1]. Each report consists of 8 attributes and a diagnose. The continuous type attributes or input vector represent medical measures on the individual and are presented in table 1.2. The diagnose or output vector is of binary type and is equal to -1 or +1 whether the individual has diabetes or not. The aim is to classify the data and to predict from the attributes whether an individual has diabetes or not. The original paper used a training set of 576 points and achieved a classification of 76% on the remaining 192 test points. In this homework, we will only work with a training set of 300 data-points and a test set of 168 data-points.

The distribution of each feature can be seen at figure 1.26. At first sight, some input features seem to have no effect like the number of times pregnant, the diastolic blood pressure or the 2-Hour serum insluin. Other parameters seems to have a more important influence as the plasma glucose concentration, the triceps skin fold thickness, the body mass index, the diabetes pedigree

[1]Smith, J.,W., Everhart, J.,E., Dickson, W.,C., Knowler, W.,C. and Johannes, R.,S., Using the ADAP learning algorithm to forecast the onset of diabetes mellitus, in *Proceedings of the Symposium on Computer Applications and Medical Care*, IEEE Computer Society Press, 261-265, 1988.

#	Attribute
1	Number of times pregnant
2	Plasma glucose concentration
3	Diastolic blood pressure
4	Triceps skin fold thickness
5	2-Hour serum insulin
6	Body mass index
7	Diabetes pedigree function
8	Age

TABLE 1.2: Value of each attribute in the input vectors of the Pima Indian Diabetes database.

function and certainly the age.

The linear model gives a score of 71.31%. Compared to the score obtained by the first paper, we can consider that there is room for improvement. Let's try a RBF kernel functions. The tuning here seems to deliver quite variable optimal solutions for γ and σ^2 but always in the same order of magnitude. Due to the fact that this model has more complicated intricate relations between the feature values and the diagnose, the parameter optimisation problem is less convex. The tuning is thus sensible to local minima, but the final cost is always similar. Let's thus use the model with one of the tuned values. The obtained performance on the test set is 77.98% which is better than the score of the original paper and can thus be considered as satisfying. The ROC-curve is also better (figure 1.27). As this model does comparable scores as what can be fonud in the litterature, it can be concluded that the LS-SVM model with RBF kernel functions is well suited for this classification. As much higher scores seem to never have been obtained, one can conclude that the data-set lacks information to perform better; maybe adding significant features could help. In other words, the information is exploited here near to its maximum.

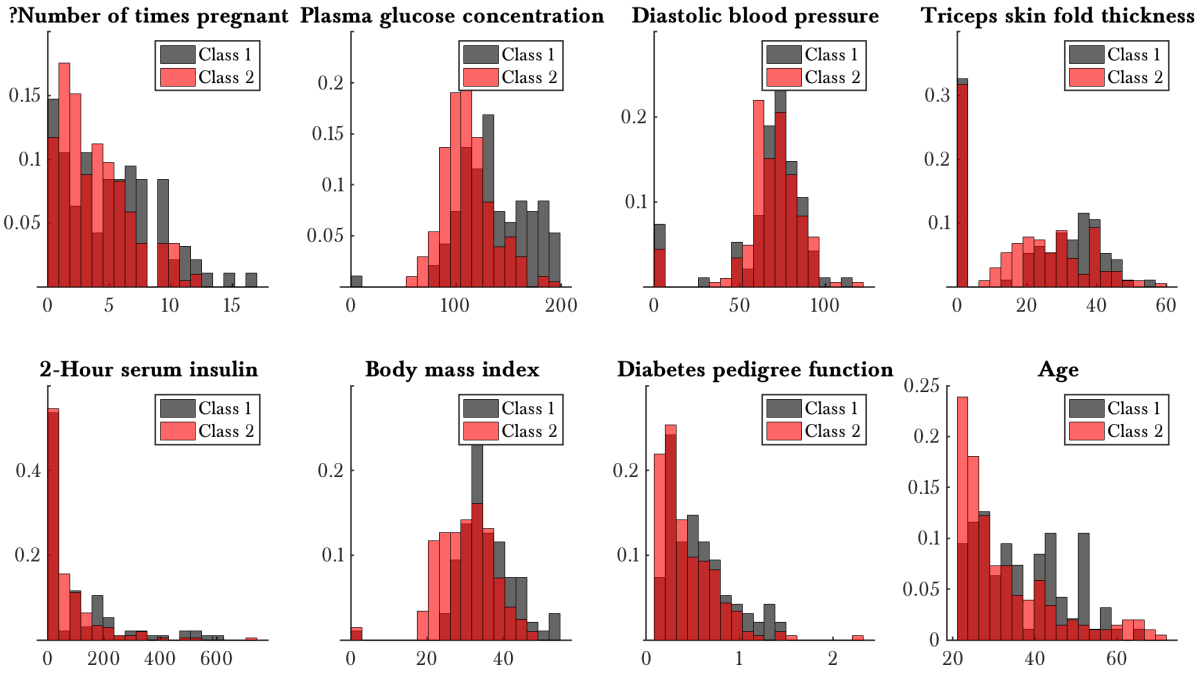
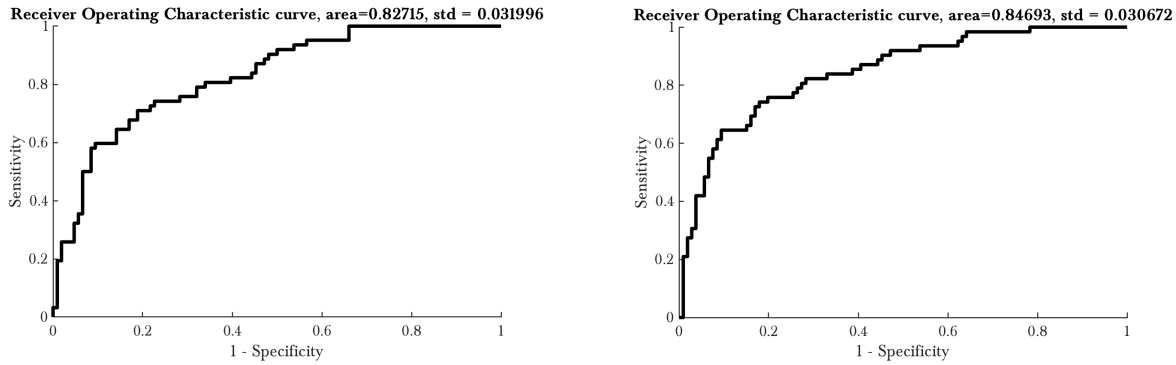


FIGURE 1.26: Distribution of each feature of the Pima Indian Diabetes data-set for each diagnose.



(A) Linear kernel function.

(B) RBF kernel function.

FIGURE 1.27: Comparison of the ROC-curves of the linear and RBF kernel functions applied to the Pima Indian Diabetes data-set.

Exercise Session 2

Function Estimation and Time-series Prediction

2.1 The Support Vector Machine for Regression

Let's consider a first example at figure 2.1. The linear kernel seems to not fit the data-points very well. It fails to approach the high values left and right of the plot, of the low values in the middle of the plot. Clearly, the data-points exhibit an underlying non-linearity that the linear kernel is unable to reproduce. Conversely, the quadratic kernel seems to reproduce that non-linearity very well, so as the cubic and the RBF kernels. Furhtermore, in comparison to the quadratic, the cubic and RBF kernels seem to add no further subtlety to the reproduction of the underlying relation. The cubic kernel allows the existence two "bends" in the final regression where the quadratic function only one, but this added complexity seems not to be of any need. Similarly, the RBF kernel allows for as many local "bends" as there exists data-points, having more difficulty to capture a global trend on the whole data-set in exchange. Nevertheless, this also seems not very useful here. Left of the plot, the RBF allows a second bend which doesn't look like having any gain at all. In other words, cubic and RBF kernel functions seem to have unnecessary added complexity compared to the quadratic kernel function.

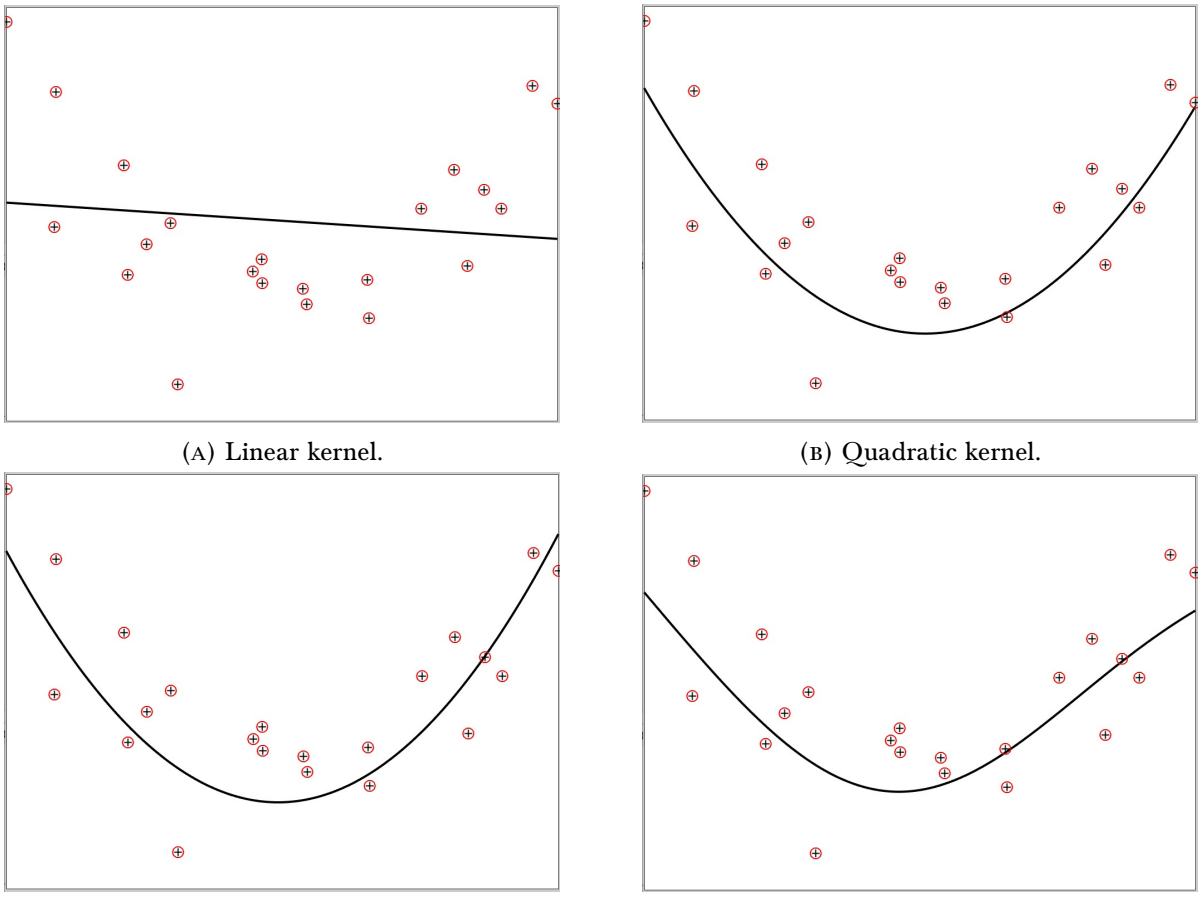
The main task in statistical learning is to avoid any *overfitting*: any useless addition of complexity may allow the regression to capture the variance of the specific data-set on which it is trained on top of the underlying relation is it supposed to capture. The regression would then reproduce the training set itself instead of generalising it. I therefore would like to cite John von Neumann

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

In other words, one parameter is sufficient to add a lot of complexity to the model and increase a lot the risk of overfitting. The general rule of thumb is thus to always reduce as much as possible the complexity of the model and always stick to the lowest one able to reproduce the general scheme of the data. This principle is also known as *Occam's razor.* In conclusion, the linear kernel shows some underfitting[1] whereas the cubic and RBF kernels exhibit clear overfitting. In our case, we may therefore conclude that the *quadratic kernel* is the best one for our model.

With the same reasoning, we may also conclude that the *linear kernel* is the best suited one for the example given at figure 2.2.

[1] In opposition to overfitting, *underfitting* appears when the chosen parameters choice hasn't got enough complexity to model the underlying relations of the data.



(A) Linear kernel.

(B) Quadratic kernel.

(C) Cubic kernel.

(D) RBF kernel with $\sigma = 1.5$.

FIGURE 2.1: Comparison of different kernels for a SVM regression. The parameter values are Bound = ∞ and e = 0. The best kernel seems to be the *quadratic* one.

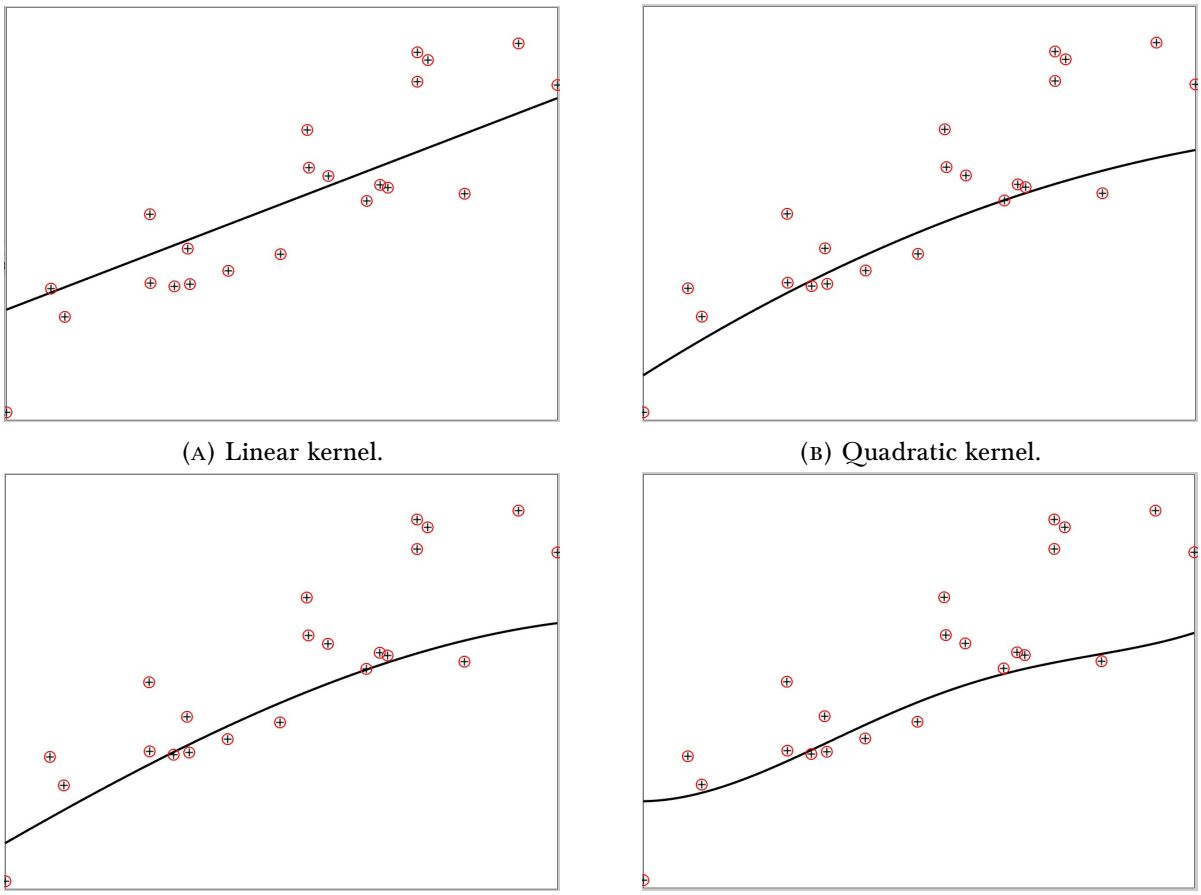
The general optimisation problem to be solved for a SVM regression is the following one

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} w^T w + \text{Bound} \sum_{k=1}^{N} (\xi_k + \xi_k^*) \\ \text{subject to} \quad & y_k - w^T \phi(x_k) - b \leq \mathsf{e} + \xi_k \\ & w^T \phi(x_k) + b - y_k \leq \mathsf{e} + \xi_k^* \\ & \xi_k, \xi_k^* \geq 0. \end{aligned} \tag{2.1}$$

which can be rewritten in dual form, with $K(x_k, x_l) = \phi(x_k)^T \phi(x_l)$, as

$$\begin{aligned} \underset{\alpha, \alpha^*}{\text{maximise}} \quad & -\frac{1}{2} \sum_{k,l=1}^{N} (\alpha_k - \alpha_k^*)(\alpha_l - \alpha_l^*) K(x_k, x_l) - \mathsf{e} \sum_{k=1}^{N} (\alpha_k + \alpha_k^*) + \sum_{k=1}^{N} y_k (\alpha_k - \alpha_k^*) \\ \text{subject to} \quad & \sum_{k=1}^{N} (\alpha_k - \alpha_k^*) = 0 \\ & \alpha_k, \alpha_k^* \in [0, \text{Bound}]. \end{aligned} \tag{2.2}$$

This justifies the denomination for Bound. This optimisation problem can be interpreted as follows the minimisation of the norm of the coefficient $\|w\|_2$ as well as the slack variables representing the



(A) Linear kernel.

(B) Quadratic kernel.

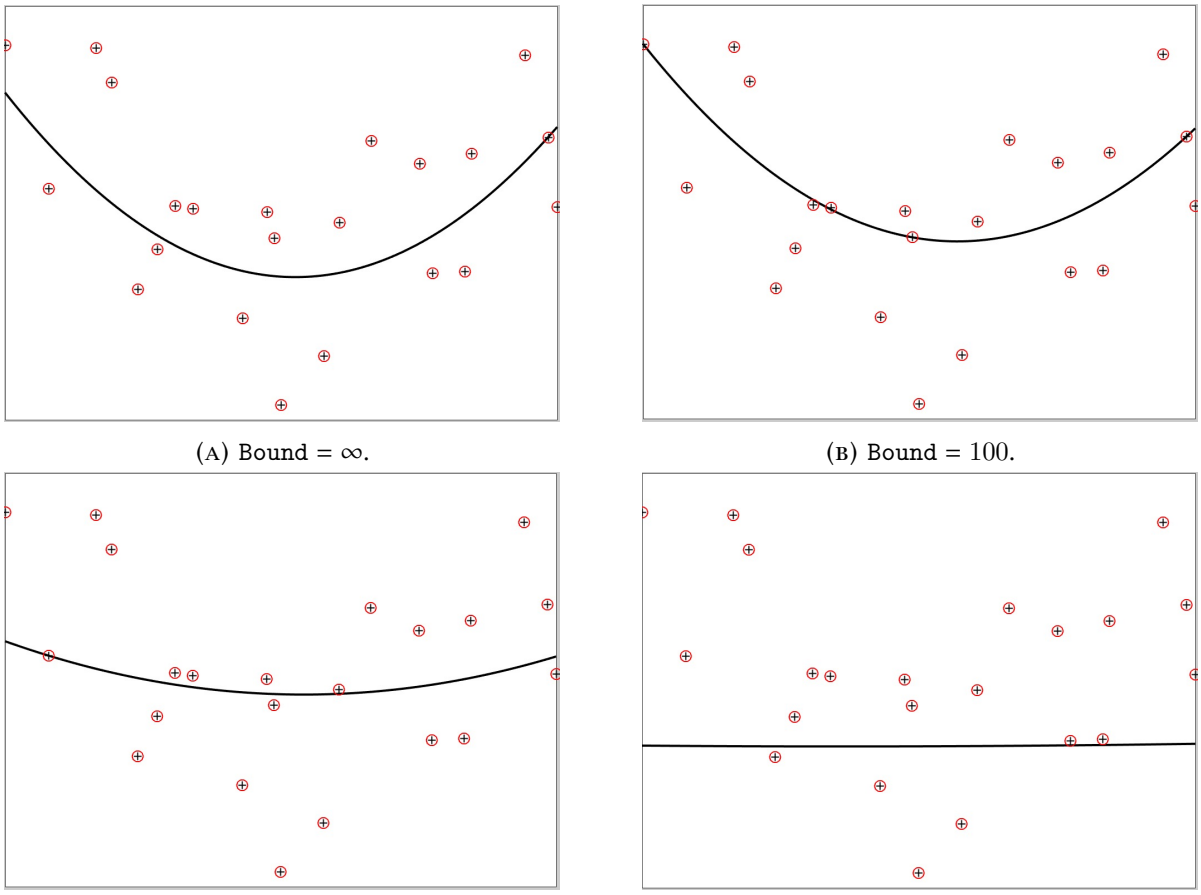
(C) Cubic kernel.

(D) RBF kernel with $\sigma = 2$.

FIGURE 2.2: Comparison of different kernels for a SVM regression. The parameter values are Bound = ∞ and e = 0. The best kernel seems to be the *linear* one.

error between the function approximation and the real value. Similarly as before, the choice of the trade-off between those two minimisation is quantified by a parameter, here Bound. This value is a sort of trade-off between the smoothness of the approximation and fitness error. When this value is infinite or when sole the approximation error counts, this corresponds to a *classical least squares fit*. Conversely, when the value is very low, the sole minimisation of the norm of coefficient $\|w\|$ is important, resulting in it being null. As $y_k = w^T \phi(x_k) + b + \xi_k$, this corresponds to reducing the equation to $y_k = b + \xi_k$ and letting arbitrary errors for the approximation. For a quadratic kernel function, this corresponds to flattening the parabola and can be seen at figure 2.3. When errors for certain points are very higher, due to their quadratic and thus convex presence in the objective function, they count for almost all the weight. This weight is represented by the α_k, α_k^* in the dual function. The effect of the Bound variable becomes avoiding that some data-points have a too high influence, hence its denomination.

Another parameter is e introducing a margin in which the errors are not counted. The higher this value, the more the sole far away data-points will count in the final approximation. This a sort of built-in *pruning* method. The points within that range will be changed from a low weight α_k to a null one. Setting a higher e will thus augment the sparsity of the method and thus the speed, but will also reduce its correctness as the more data-points used, the better the aproximation. This phenomenon can be seen at figure 2.4.



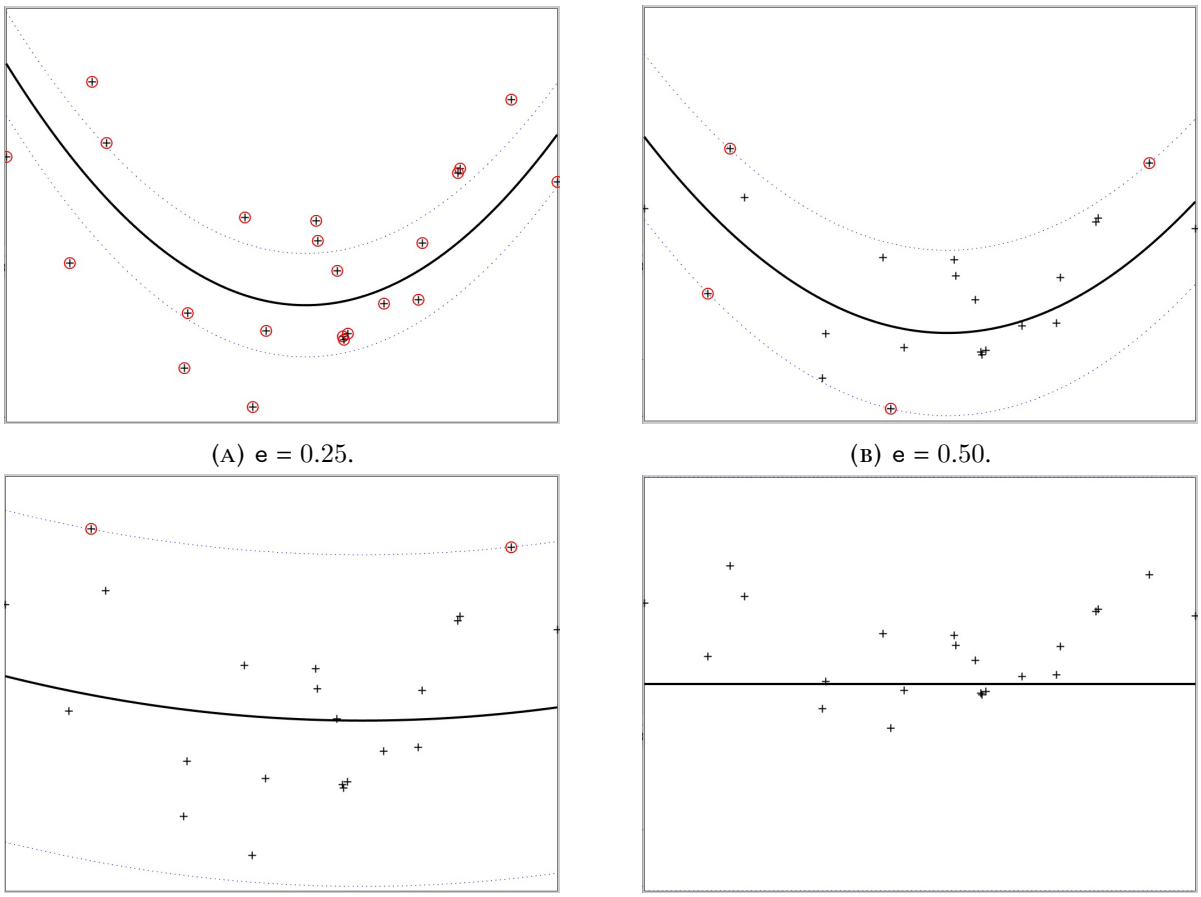
(A) Bound = ∞.

(B) Bound = 100.

(C) Bound = 0.1.

(D) Bound = 0.001.

FIGURE 2.3: Comparison of different values of Bound for a SVM regression with quadratic kernel and e = 0.



(A) e = 0.25.

(B) e = 0.50.

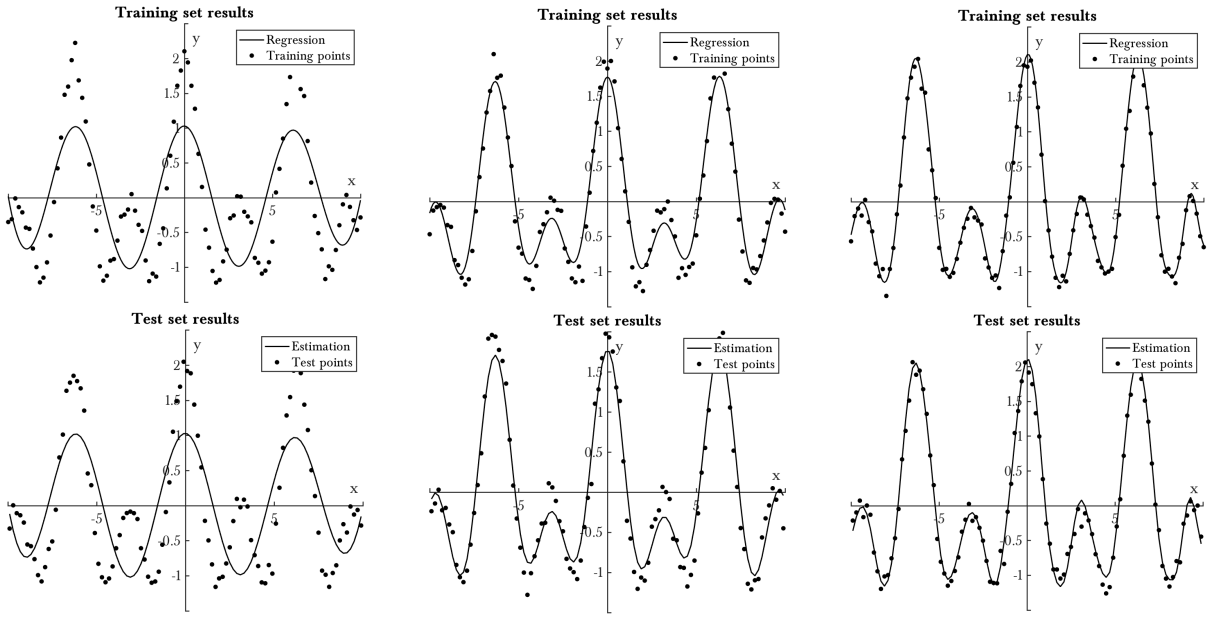
(C) e = 1.00.

(D) e = 2.00.

FIGURE 2.4: Comparison of different values of e for a SVM regression with quadratic kernel and Bound = ∞. The support vectors are encircled in red.

2.2 A Simple Example: Sum of Cosines

The use of RBF kernel functions adds a new parameter σ^2 which represents the diameter of the "influence zone" of each data point in the training set. In certain sense, it measures how much "detail" of the training set the approximation is able to capture. Increasing too much will result in overfitting and reducing it too much will lead to a too "gross" approximation. To avoid overfitting, one should select the biggest value that captures enough details or in other words, an approximation that is fine enough. Figure 2.5 shows an approximation for different values of σ^2. By applying the just cited principle, one can conclude that $\sigma^2 = 0.1$ is the best choice. The sole problem is that the scale isn't perfect, but luckily, there is a second parameter for that.

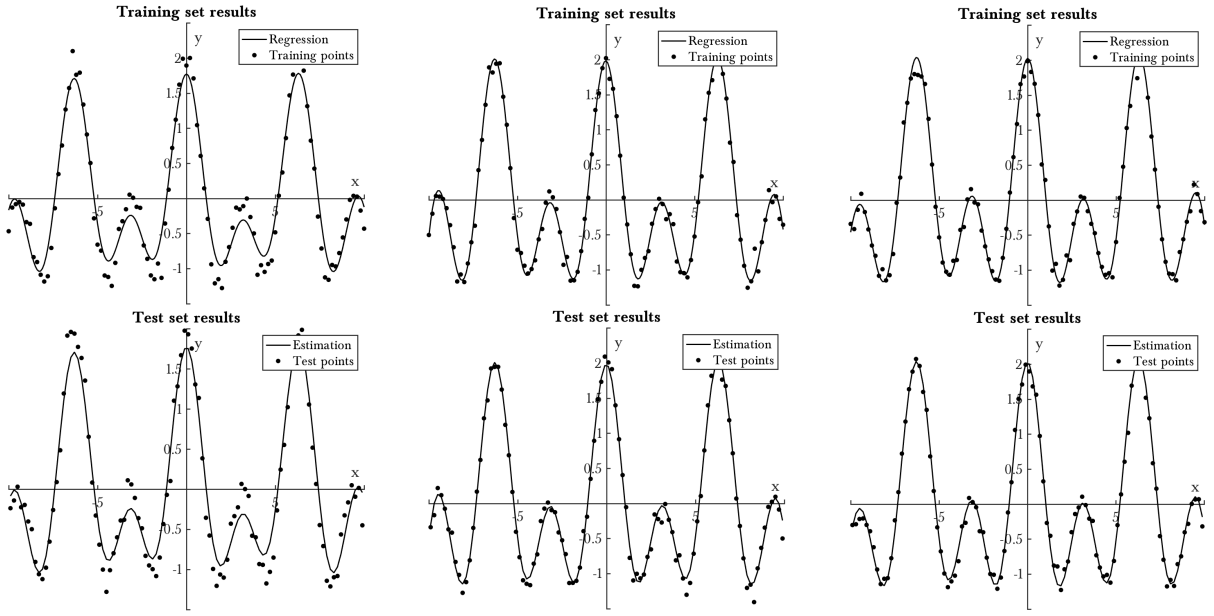


(A) $\sigma^2 = 0.30$. (B) $\sigma^2 = 0.10$. (C) $\sigma^2 = 0.01$.

FIGURE 2.5: Comparison of different values of σ^2 for a SVM regression of $f(x) = \cos(x) + \cos(2x)$ with RBF kernel functions. The other fixed parameter value is $\gamma = 100$.

As seen in the previous section, the γ value (previously `Bound`) represents a trade-off between the smoothness of the approximation and the fitness error. For a too low value, the SVM wouldn't be able to generalise the underlying relations well as for too high values, it would try to minimise the fitness error as much as possible and as we know that in this example the training data is noisy, sticking the approximation to the noise and thus resulting in typical overfitting. In other words, one must reduce the γ parameter as much as possible as long as it captures the general scheme. Figure 2.6 shows different values of γ and, as before, by applying the same principle, one can conclude that the optimal value is $\gamma = 100$.

Now, one may think that σ^2 and γ have the same influence. Nevertheless, they still are quite different in their definitions and thereby also in their results. As mentioned before, the σ^2 controls the diameter of the "influence zone" and therefore have an influence in the x and in the y-directions. Conversely, the γ is defined as the influence of the square error, which only has an influence in the y-direction. By trying to give a sort of analytical equivalent, one could argue that the σ^2 controls the local Lipschitz constant of the approximation, whereas γ control its scale, or more precisely its vertically sticking to the training data-points. One should thus choose the biggest possible value for σ^2 to allow the approximation to be shaggy enough to avoid capturing the noise without loosing the needed details as the smallest possible value for γ to stick to the data to avoid



(A) $\gamma = 100$. (B) $\gamma = 1000$. (C) $\gamma = 10000$.

FIGURE 2.6: Comparison of different values of γ for a SVM regression of $f(x) = \cos(x) + \cos(2x)$ with RBF kernel functions. The other fixed parameter value is $\sigma^2 = 0.1$.

overfitting without loosing of the underlying generalisation.

In conclusion, the optimal values here are $\sigma^2 = 0.1$ and $\gamma = 1000$ though more tests could certainly refine these values.

2.3 Hyper-parameter Tuning

By tuning the parameter values using a 10-fold cross-validation, one obtains similar results as what we projected in the previous section, i.e. $\gamma = 1.2647\text{e+}3 \approx 1\text{e+}3$ and $\sigma^2 = 0.0824 \approx 1$. The results can be seen at figure 2.7. By testing in a more general way different optimisation methods as summarised at table 2.1, we obtain the same values for σ^2, but much more varying values for γ. As could be seen at figure 2.1 of the previous section, the γ hasn't got a lot of influence as long as it is big enough. The choice of a big enough σ^2 prevents overfitting and the choice of γ thus doesn't matter much. Nevertheless, in a more general way, it should always be chosen as small as possible.

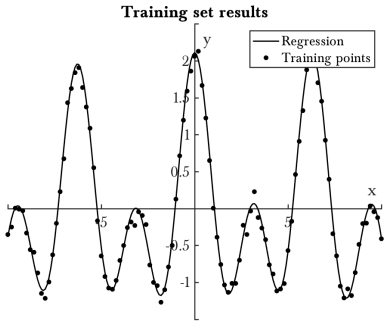


FIGURE 2.7: Approximation of the training set with the tuned parameters.

Looking at the rest of the table shows some differences between the methods. *Grid search* is an exhaustive searching through a manually specified subset of the hyperparameters. This ressembles much an automated and more precise search of what we manually did in the previous section. The *Nelder-Mead method* is a gradient-based optimisation algorithm. It is therefore much more sensitive to local minima explaining the higher variance of the tuned hyperparameters. The upside of this method is its execution time compared to the exhaustive search. One can also speed up both of these

algorithms by parallelising their execution. This is done by using the *randomised directional search* instead of the *coupled simulated annealing*. The grid search algorithm is well suited for parallelisation compared to the Nelder-Mead algorithm which is sequential and where parallelisation hasn't any effect.

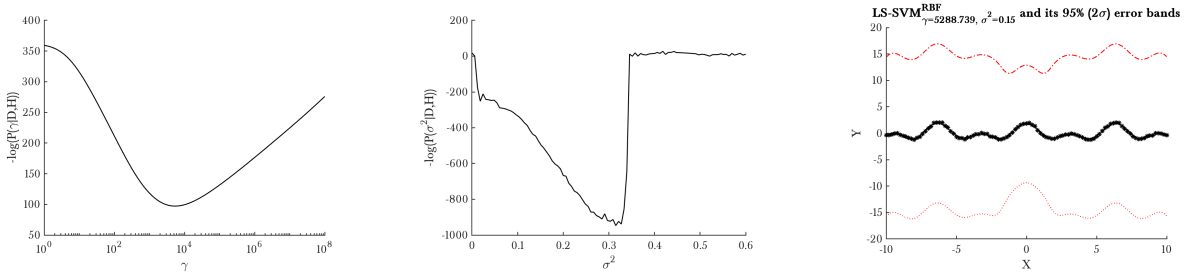
Global optimisation technique	Optimisation technique				
Coupled Simulated Annealing	Nelder-Mead method	μ_γ = 2.1e+4 σ_γ = 3.2e+4	$\mu_{\sigma^2} = 0.1074$ $\sigma_{\sigma^2} = 0.0261$	$\mu_{\texttt{cost}} = 0.96\%$ $\sigma_{\texttt{cost}} \approx 0$	$\mu_t = 8.9s$ $\sigma_t = 1.00s$
	Brute Force Gridsearch	μ_γ = 2.2e+5 σ_γ = 6.4e+5	$\mu_{\sigma^2} = 0.1235$ $\sigma_{\sigma^2} = 0.0409$	$\mu_{\texttt{cost}} = 1.01\%$ $\sigma_{\texttt{cost}} \approx 0$	$\mu_t = 13.8s$ $\sigma_t = 0.95s$
Randomised Directional Search	Nelder-Mead method	μ_γ = 4.5e+6 σ_γ = 3.1e+7	$\mu_{\sigma^2} = 0.1022$ $\sigma_{\sigma^2} = 0.0437$	$\mu_{\texttt{cost}} = 1.03\%$ $\sigma_{\texttt{cost}} \approx 0$	$\mu_t = 9.0s$ $\sigma_t = 1.24s$
	Brute Force Gridsearch	μ_γ = 1.13e+5 σ_γ = 3.35e+5	$\mu_{\sigma^2} = 0.1008$ $\sigma_{\sigma^2} = 0.0489$	$\mu_{\texttt{cost}} = 1.02\%$ $\sigma_{\texttt{cost}} = 0.04\%$	$\mu_t = 13.5s$ $\sigma_t = 1.43s$

TABLE 2.1: Comparison of the tuning by different optimisation techniques. These are based on 50 experiments of each 1000 data-points in the training set.

2.4 Application of the Bayesian Framework

2.4.1 Regression

The Bayesian framework is a three-level principle for infering the different parameters of a model. Each level consists in maximising the posterior probability of the parameters. The Bayesian framework works from specific to general: first level infers w and b, results of the SVM optimisation itself, the second level γ which defines the trade-off between smoothness and fitness and the third level infers σ which defines a part of the kernel function. This technique is shown at figure 2.8.



(A) Level 2. (B) Level 3. (C) Results.

FIGURE 2.8: Level 2 and 3 of the Bayesian framework applied to a function regression with its results.

2.4.2 Classification

This idea of posterior probability can also be used in classification problems where we define the *posterior class probability* as

$$p(y|x, D, \mathcal{H}) = \frac{p(y)p(x|y, D, \mathcal{H})}{\sum_{y'=\pm 1} p(y')p(x|y', D, \mathcal{H})} \tag{2.3}$$

which can be interpreted as the probability of an event after some evidence has been taken into account. In our case, the evidence is based on the distribution of the elements of the training set and the parameters. This posterior class distribution ban be seen at figure 2.9 and represents the

probability of being the positive class. In other words, 1 corresponds to a certainty of the element situated there being in the positive class. As only classes exist, 0 means a certainty of being in the negative class.

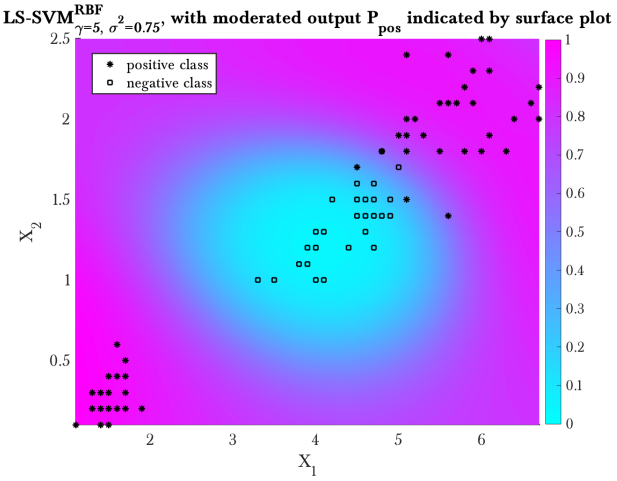
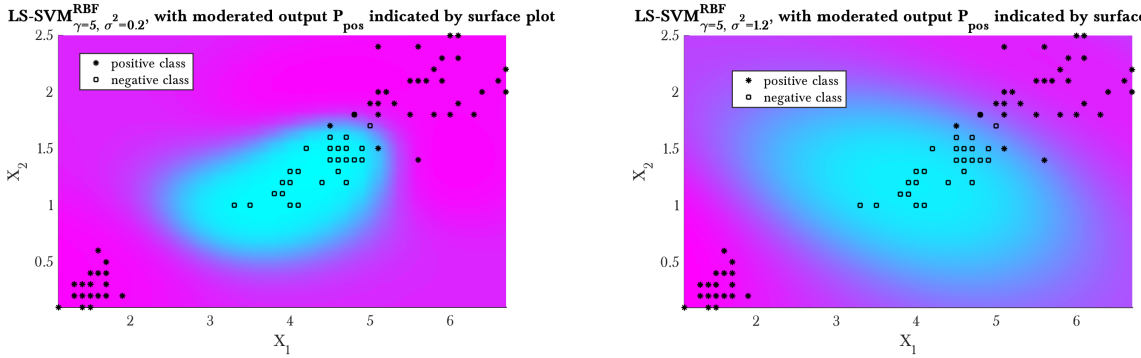


FIGURE 2.9: Posterior class probability of the iris data-set with $\sigma^2 = 0.75$ and $\gamma = 5$.

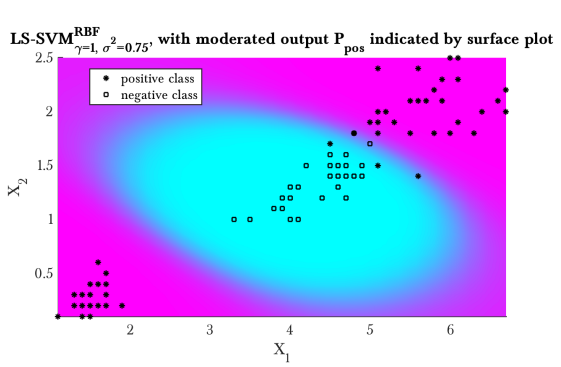
Similarly as before, the hyper-parameters γ and σ^2 are used to control how the classifier will be shaped. σ^2 controls the size of the "influence zone" of each data-point and thus the general size of an area dominated by a class. This phenomenon can be well seen at figure **??**. Similarly, γ is the trade-off between the minimisation of $\|w\|$ and the minimisation of the slack variables or the trade-off between the smoothness ans the fitness of the function. Its concrete effect will be to control the smoothness of the frontier between both classes. This phenomenon can be well seen at figure **??**.



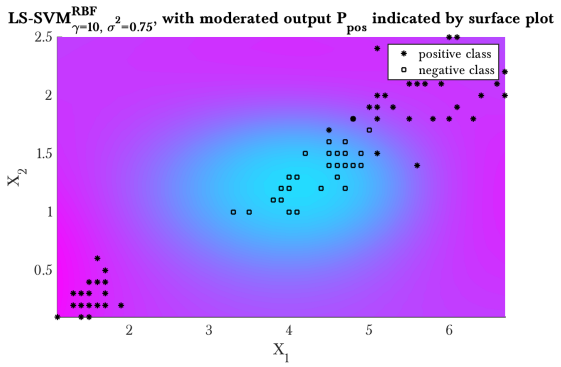
(A) $\sigma^2 = 0.2$.

(B) $\sigma^2 = 1.2$.

FIGURE 2.10: Influence of the σ^2 parameter on the posterior class probability. The other fixed parameter value is $\gamma = 5$.



(A) $\gamma = 1$.



(B) $\gamma = 10$.

FIGURE 2.11: Influence of the γ parameter on the posterior class probability. The other fixed parameter value is $\sigma^2 = 0.75$.

Automatic Relevance Determination

This posterior probability can also be used to remove noisy or non-relevant dimensions of the input. This is done by assigning a weight to each input dimension and optimising it using the third level of inference. The biggest or the smallest values can be removed. In other words, the input dimensions for which σ^2 is very small, thus subject to overfitting can be removed. Similarly, too high values of σ^2 represent a lack of specification of the model to the given problem. Concretely, the ARD starts with all input dimensions and prunes them one at a time by first determining the σ^2 corresponding to the minimal posterior probability and then looks at which removed combination obtain a better score. The score of each input dimension combination has been calculated at table 2.2. Indeed, we clearly see that the sole the first dimension correspond to a meaningful input, the rest being solely noise (figure 2.12).

Sets	$-\log$(posterior class distribution)
$\{1\}$	-132.3954
$\{2\}$	2.4306
$\{3\}$	35.0754
$\{1, 2\}$	89.7848
$\{2, 3\}$	237.8481
$\{1, 3\}$	35.9556
$\{1, 2, 3\}$	-20.4035

TABLE 2.2: Posterior class distribution in function of the kept dimensions.

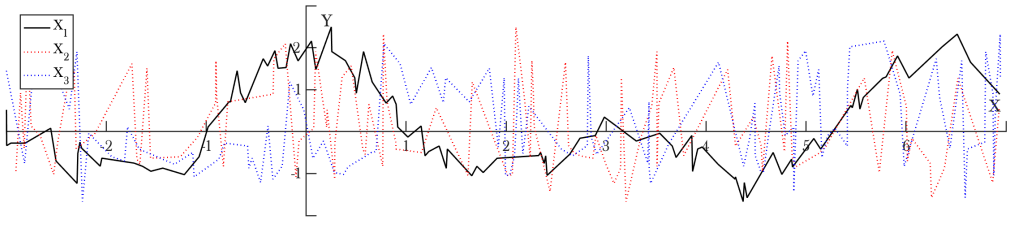


FIGURE 2.12: Sole dimension X_1 is relevant, the rest is just noise.

This algorithm uses the a cost function to determine which input to prune. In this case, that cost function was the third level of inference of the Bayesian framework. One could also use another cost function as with a 10-fold cross-validation as summarised at table 2.3. It is even more

clear that all the information is in the first input dimension: the results with the first dimension are always better than without and the sole first dimension does the best score.

Sets	Mean MSE
{1}	0.0857
{2}	1.5128
{3}	1.6877
{1, 2}	1.0110
{2, 3}	1.5050
{1, 3}	1.1479
{1, 2, 3}	1.2079

TABLE 2.3: 10-fold cross-validation in function of the kept dimensions.

2.5 Robust Regression

Figure 2.13 shows how outlying point can significantly perturb the final approximation. Indeed, their corresponding fitness error ξ_k will have a greater value and thus a huge weight in the objective function, due to the *convexity of the least squares*. In the first series of the three outliers, we notice that the first and the second one are a little bit on the path, their $|\alpha_k|$ are thus smaller than the third one which is clearly not on the path. The important weight of these three outliers is sufficient to pull the approximation downward in the first peak, making the data-points in this region of the abcissa relatively more important. The second series of three outliers shows this phenomenon even bigger: the further vertically the outlier is from the path, the more it pulls the approximation towards it and the more it makes the other data-points in the same region of the abcissa more important. In this region, the approximation curve is clearly deviated. Normally, the more important data-points are in the bends of the approximation function.

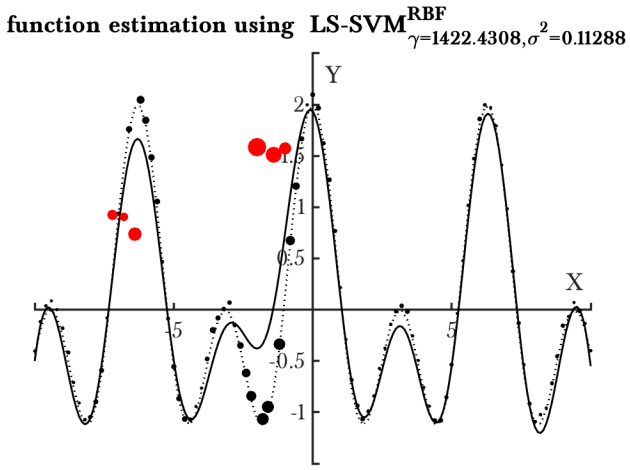


FIGURE 2.13: Result of the training with some outliers (red). The size of the data-points is proportional to their $|\alpha_k|$. The approximation corresponds to the continuous line and the exact function to the dotted line.

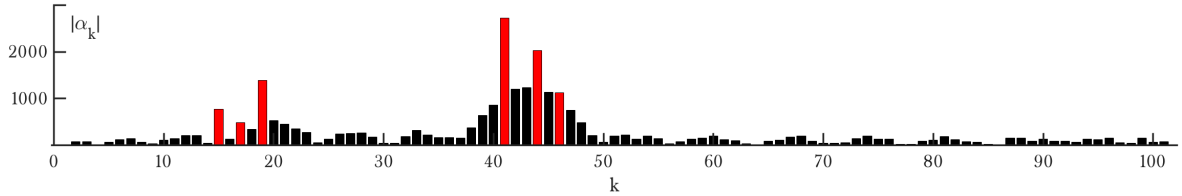
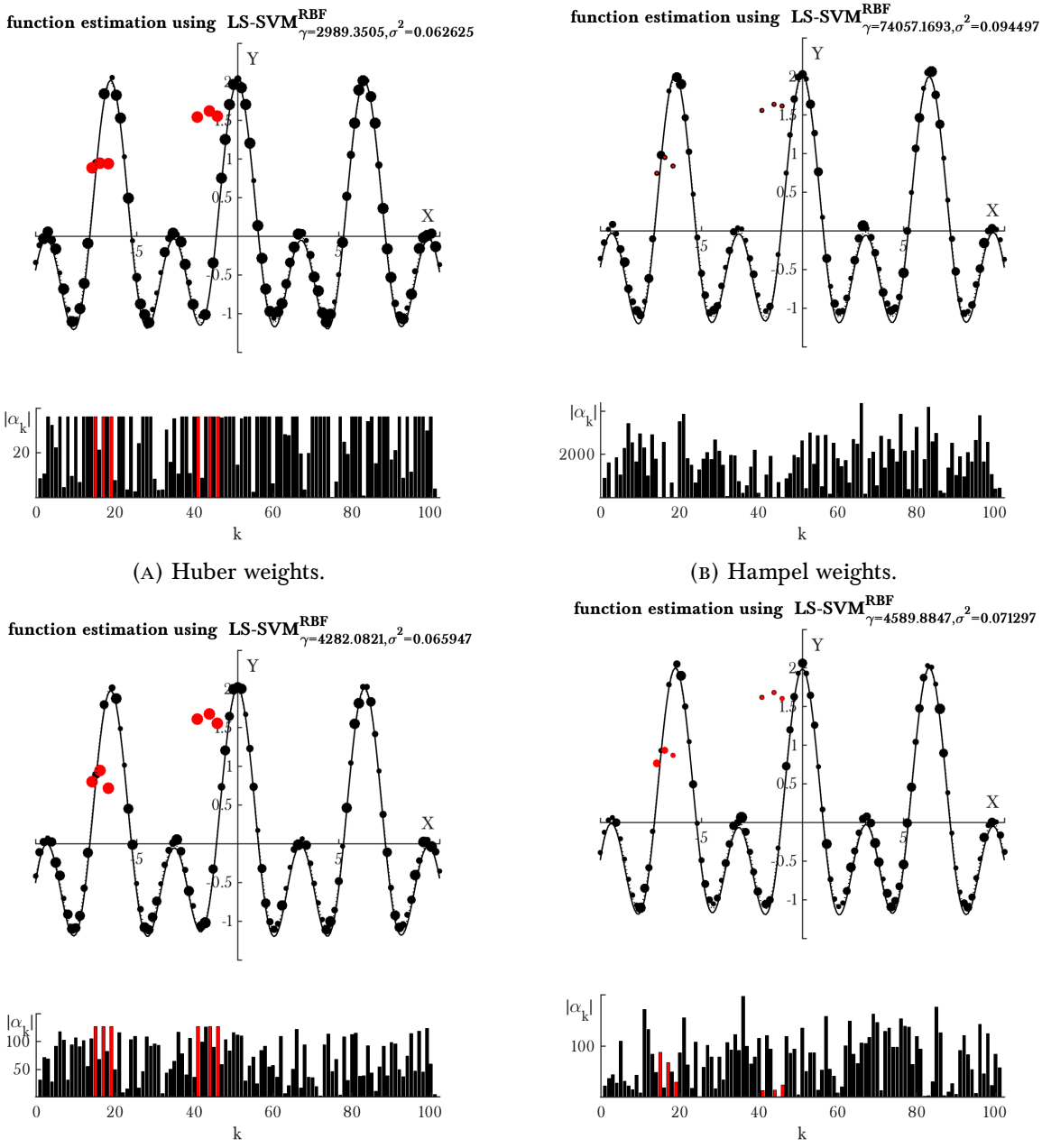


FIGURE 2.14: Comparison of the $|\alpha_k|$ of each data-point. Outliers are marked in red.

Notice on figure 2.14 how the addition of the three outliers pulled the whole region around them much higher as their fitness errors augment as well. The minimisation of the fitness errors

least squares results in finding a middle ground between much more spaced point and leads to greater errors. To conclude, SVM with classical least square fitness errors minimisation are very sensible to outliers.



(A) Huber weights.

(B) Hampel weights.

(C) Logistic weights.

(D) Myriad weights.

FIGURE 2.15: Comparison of different weight functions in a robust regression with outliers (red).

This problem of high sensibility to outliers can be solved by giving less weight to extreme values of the fitness errors.,Outliers will thus have less influence in the least squares part of the objective function. An example is using Huber weights that linearise the convex quadratic term after some value

$$e_k = L_\delta(y_k, f(x_k)) = \begin{cases} \frac{1}{2}(y_k - f(x_k))^2 & \text{for } |y_k - f(x_k)| \leq \delta, \\ \delta\,|y_k - f(x_k)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases} \tag{2.4}$$

An alternative which decreases after some value is the Hampel weight function

$$e_k = L_{\delta_1,\delta_2,\delta_3}(y_k, f(x_k)) = \text{sgn}(x) \begin{cases} |x_k| & \text{for } 0 \leq |x_k| \leq \delta_1, \\ \delta_1 & \text{for } \delta_1 < |x_k| \leq \delta_2, \\ \delta_1 \frac{(\delta_3 - |x_k|)}{(\delta_3 - \delta_2)} & \text{for } \delta_2 < |x_k| \leq \delta_3, \\ 0 & \text{for } \delta_3 < |x_k|, \end{cases} \tag{2.5}$$

Due to its decrease after δ_2, the function will give almost no weight to outliers. A trade-off between no weight after a certain value and a linear growth can be the logistic function which stabilises after some value. Another function is the myriad weight, which behaves similarly to the Hampel weights (figure 2.15).

During the tuning, the minimisation objective is the *mean absolute error* instead of the *mean square error*. This is due to the convexity of the function already discussed before, which favours the weight of the outliers. Using the mean absolute error changes it to linearity to give less weight to the outliers.

2.6 Homework Problem

2.6.1 Introduction: Time-series Prediction

In this exercise, we are going to try to predict a chaotic function: the logistic map

$$x_{n+1} = r x_n (1 - x_n) \tag{2.6}$$

with a chaotic choixe of r. The data-set consists of a training set of 150 data-points and a test set of 50. A LS-SVM with RBF kernel functions will be used as an *Auto Regressive model*.

By applying the code given in the exercise guidelines, one obtains the following results (figure 2.16) Of course, this result is not satisfying: it seems that the model predicts the output in a totally random way. A chaotic model is stastically noisy and a extreme generalisation is of course random. Nevertheless, there exists some underlying relations which should be reproduced. The model lacks specification, which is a typical case of underfitting[2]. We should therefore augment the order to allow the model to have more complexity. To determine to optimal order, one should use a different set to detect overfitting. Ideally, a validation set should be used or other alternatives which we discussed before like cross-validation or the leave-one-out method. Here, the test set is being used. Of course, before each score computation, the other parameters are tuned again (with cross-validation) to let the model being exploited at its full capacity. Figure 2.17 shows a sudden drop after `order` > 20. *Occam's razor principle* will tell us to keep the order as low as possible and

[2]In opposition to overfitting, *underfitting* appears when the chosen parameters choice hasn't got enough complexity to model the underlying relations of the data.

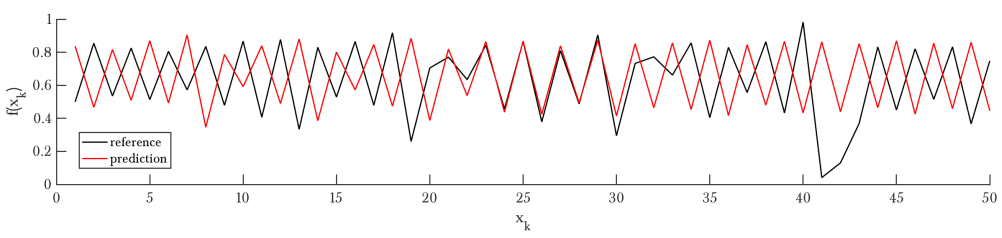


FIGURE 2.16: Results on the test set with `order` $= 10$, $\gamma = 10$ and $\sigma^2 = 10$.

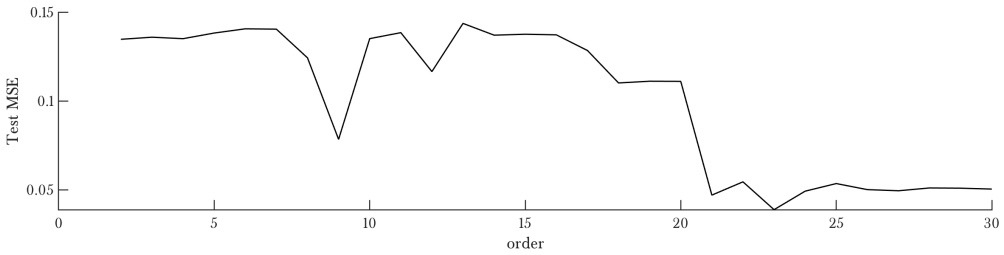


FIGURE 2.17: MSE of the test set in function of the order.

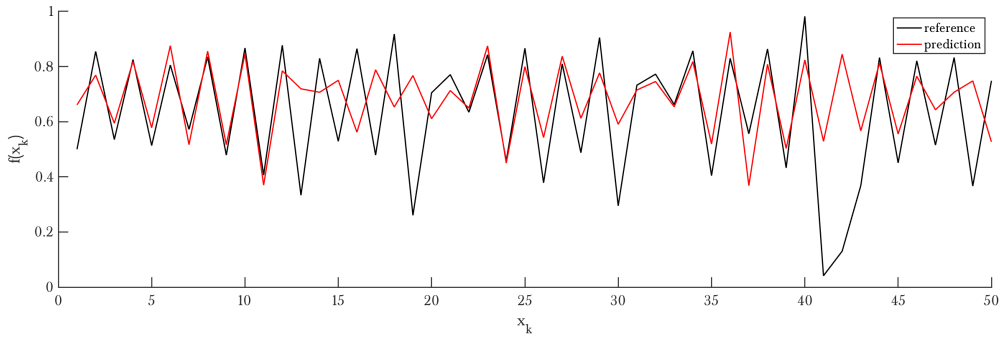


FIGURE 2.18: Results on the test set with `order` = 23, γ = `1.69e+5` and $\sigma^2 = 9176$.

`order` = 23 is therefore chosen. The corresponding results are displayed at figure 2.18, but hte results are still not very satisfying. An observation made during the γ and σ^2 tuning is that their optimal value seems to grow with the order. This makes sense as distances become proportionally much smaller in higher dimensions: data-points are more isolated as the dimension grows. This phenomenon is known as the *curse of dimensionality*. In our example, it can be seen at figure 2.19.

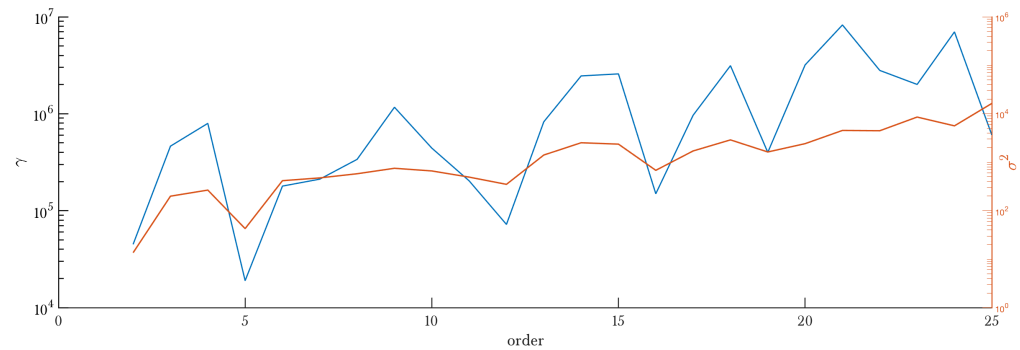


FIGURE 2.19: Example of the consequences of the *curse of dimensionality*.

2.6.2 Application: Santa Fe Laser Dataset

Here again, the function that is going to be modelled behaves chaotically. The Santa Fe data-set is a univariate time series derived from laser-generated data recorded from a far-infrared laser in Santa Fe, New Mexico (USA).

Now, we have to determine the optimal hyper-parameters. A straightforward way to do it is to evaluate the results of the training for each combination. The problem of this technique is that it will not detect any overfitting and eventually lead to an overfitted solution. This is why we have to test the performance on an individual data-set: the *validation set.* Other alternatives are to use cross-validation or the leave-one-out method which have as advantage of training the model on all the available information and being less sensitive to the chosen training-validation seperation. This method has been up to now to find the optimal γ and σ^2. To find the optimal model order, cross-validation may still be used. This method has a smaller variance can thus easily show the optimal order for the model. In the previous section, we determined the optimal order of the model with the scores on the test set. That is normally a bad idea as the test set is only there to give a final score to our model, one that is not contaminated bu the training, nor the tuning (validation set). Figure 2.20 shows the performance of the cross-validation in function of the order. By applying *Occam's razor principle*, the optimal order would be . Figure 2.21 shows the corresponding results on the test set. The MSE score is 430.9351.

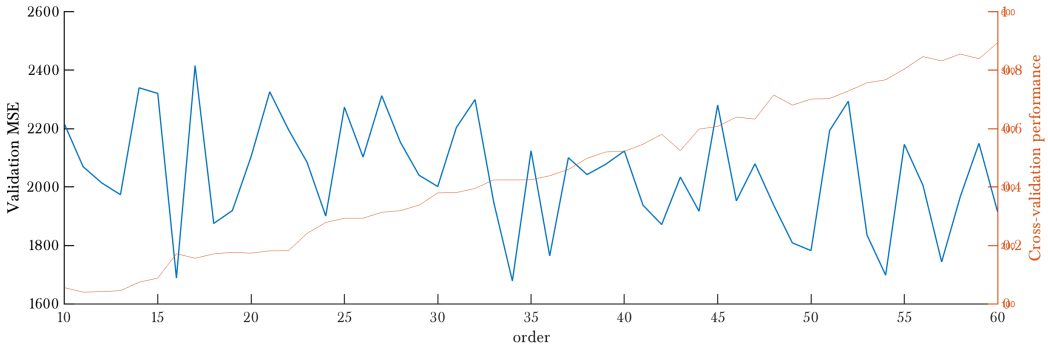


FIGURE 2.20: Validation of the order with a validation set and a cross-validation method.

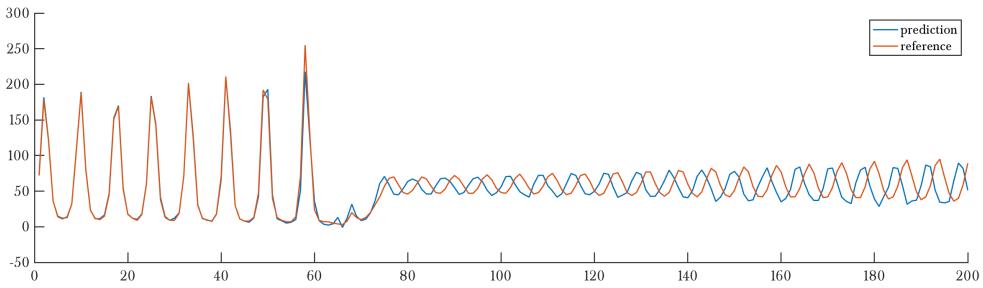


FIGURE 2.21: Results on the test set with `order` = 50, γ = `2e+4` and $\sigma^2 = 10$.

Exercise Session 3

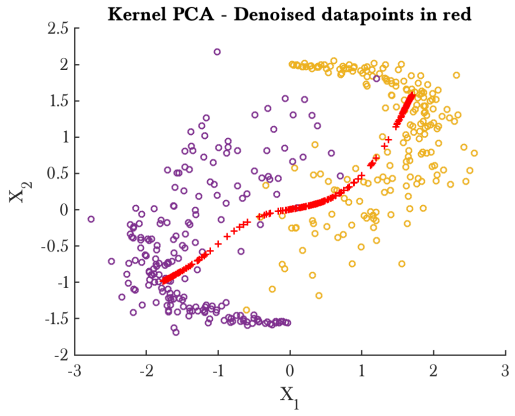
Unsupervised Learning

3.1 Kernel Principal Component Analysis

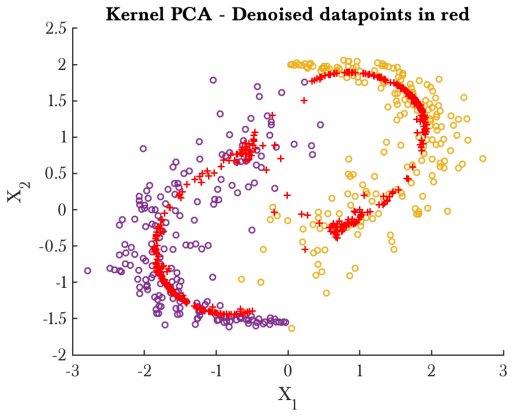
When applying a kernel component analysis to the data-set of figure 3.1, we are able to reduce the dimension using non-linear relations. When denoising a data-set, the main goal is to generalise the underlying relations well without capturing the noise. After reconstruction, the data-set well then appear without noise. This is done by reducing the dimension of the input data-set and by this mean keeping the sole important information and not the noise. Augmenting the number of principal components n_C will thus allow to retain more and more information of the input space. When a too large number of principal components n_C is chosen, the noise will also be captured and a too low will lead to bad specification. In other words, denoising with a too low number of principal components will lead to a bad reconstruction of the data and thus an unusable output data-set and a too high n_C will lead to the capture of noise and thus a bad denoising. This phenomenon can well be seen on figure 3.1.

By transforming the data-set using the Mercer's trick, we are able to perform a reduction based on non-linear relations. The original data-set that has non-linear relations between the elements will be first mapped onto a set where the linear relations are more obvious. This allows for reduction of data-sets where classes are based on non-linear relations (figure 3.2). The other main difference between classical linear PCA and Kernel PCA is that the reduction — which is a spectral analysis of a matrix — is based on the covariance matrix of all the transformed elements and not of the covariance of each input variable. By consequence, the maximum number of principal components n_C is equal to the number of elements in the training set and not the number of dimensions of the input space as in the classical PCA (in our case, up to 400 against 2).

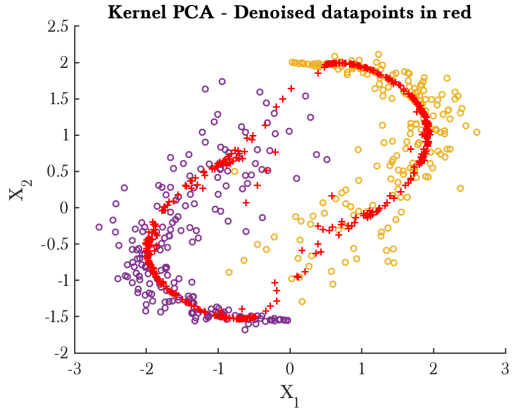
An idea for the tuning of the parameters of the KPCA would be crossvalidation on the errors of preimages in the original space. In other words, by minimising the mean square error of the inverse KPCA on a validation set: $\sum_{k=1} \left[x_k - \text{KPCA}\left(\text{KPCA}^{-1}\left(x_k\right)\right)\right]^2$. Cross-validation or the leave-one-out method could be used as validation set.



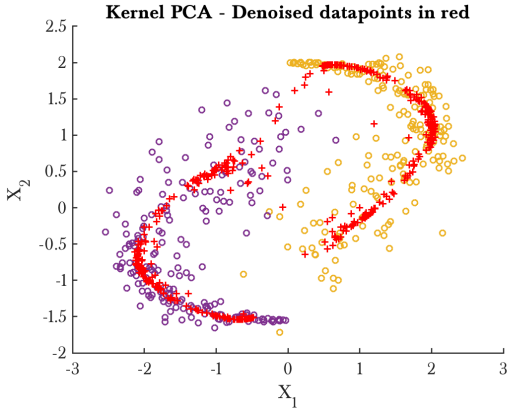
(A) $n_C = 2$.



(B) $n_C = 3$.

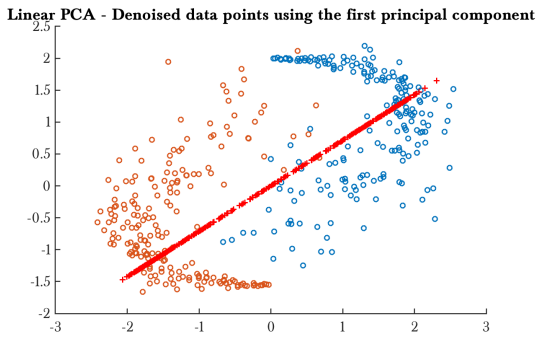


(C) $n_C = 4$.

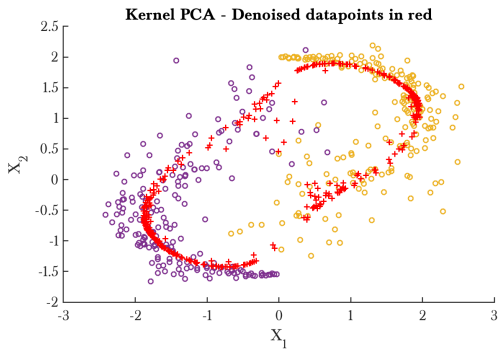


(D) $n_C = 5$.

FIGURE 3.1: Influence of the number n_C of components in a Kernel Principal Component Analysis (PCA) with RBF kernel functions.



(A) PCA ($n_C = 2$).

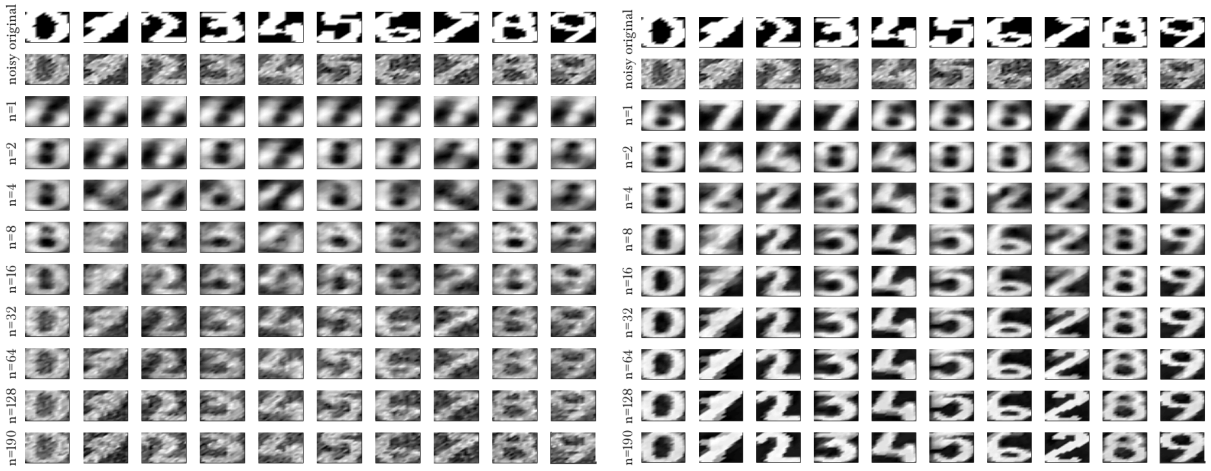


(B) KPCA ($n_C = 3$).

FIGURE 3.2: Comparison of a classical (linear) PCA with a Kernel PCA.

3.2 Handwritten Digit Denoising

Kernel Principal Component Analysis can also be used to denoise data: as it extracts the main features of the data, it suffices to calculate the pre-image to reconstruct it without the less important features. These less important features are often white noise as this information is decollerated across the data-set. The corresponding relation will thus result in a low eigenvalue after diagonalisation. In this way, the noise can be dropped off and the data denoised.



(A) Linear PCA. (B) Kernel PCA.

FIGURE 3.3: Comparison of desnoising using a linear and a kernel PCA method.

This of course is only possible if the correlations of the other relations are relatively much more significant that the white noise. If the significant underlying relations show high correlation, they will not appear important after diagonlisation and not being captured. Figure 3.3 compares the denoising of digits using a (classical) linear PCA and a kernel PCA on a test set where the denoising relations have been deduced from a training set. Lets compare both methods

- **Linear PCA**. Digits are complex symbols that are not well represented by linear relations. The PCA consists in diagonalising the (normalised) covariance matrix of the *input variables*. Applied to the whole training set of all different digits, no component — or eigenvector of input variables, "building blocks" of the digits — seems to be particularly present (the corresponding input variables show a high correlation, *i.e.* a high eigenvalue). In this way, the linear PCA fails to discern the underlying components (eigenvectors) of the digits, except a group in form of an 8 and a 7 which seem to be present in all digits. Using a few principal let these two eigenvectors show up, leading up in some blurry images. The linear relation fails to find other meaningfull components and captures noisy eigenvectors. Those ones are the next to be added leading to a noisy reconstruction of the digits. In other words, the linear relations cannot capture sufficient meaningful components (non-noisy eigenvectors of input variables) to reconstruct the digits without noise and thus denoising them. This method is thus not well suited and fails. One advantage of this method though is that augmenting the training set will not increase much the computation size as the correlation matrix to be diagonalised will not increase in size.

- **Kernel PCA**. There are two main differences with the linear (classical) PCA: (1) the diagonalised correlation matrix is based on the data-points and not the input variables and (2) Mercer's trick is used to transform the data and can thus capture non-linear relations. The first difference leads to reconstructing the digits based on a sum of other digits and not of

sum of input variables. When digits are summed up, the digit shape will show up as the white noise will flatten as it is decorrelated by definition. Furthermore, the transformation (here a RBF function) allows a better measure of the distance between two digits than a pure scalar product of their normalised input values. For these two reasons, the KPCA is able to find the components of the digits quite well. As for the linear PCA, the first ones to show up for a low number of components are a 8 and a 7-shaped form, but afterwards, other meaningful component are added instead of noise. It is also clear that the main information is contained in the first components and that $n_c = 190$ doesn't show much better results that $n_c = 32$, but takes more time to compute. Adding all of them will of course reconstruct all the data including the noise. Overall, this method shows significantly better results than its classical linear counterpart and even with some imperfections for the 7 of the test set, we can conclude that this method successfully denoises the digits. Augmenting the data-set could of course improve the results but the counterpart is that it will augment the size of the matrix to be diagonalised and thus the computation time quadratically with classical algorithms.

3.3 Spectral Clustering

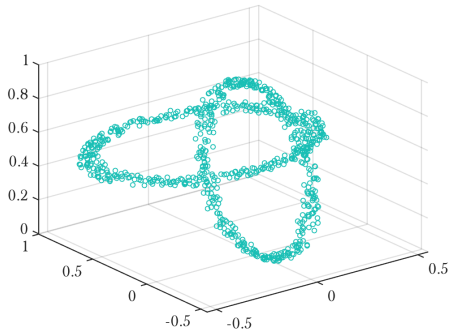
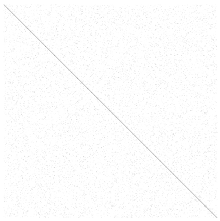
Spectral clustering is the the result of *size reduction* in the feature space. Similarities are computed with RBF and act as correlations between the data-points. This is a way of computing non-linear correlations in a matrix Ω_c of *full dimension.* These similarities are then reduced to a *mono-dimensional* vector to determine their class, or cluster in this case. This method is thus based on the sole data-points and needs *no training set with prior information on the classes.* Only the test set is needed. The only extra set it may need is a validation set to tune the fewer parameters than a classifier (prior information for this set could be useful though).

In comparison, LS-SVM classifiers are a *multidimensional boundary* in the same feature space. This boundary has *full dimension of the feature space - 1* and is spanned by the data-points of the training set to minimise misclassification in addition to be smooth. The main difference though is that it needs *a training set with prior information on the classes.* Ideally it needs a training set to learn, a validation set to tune the parameters and a test set to see the unbiased results.

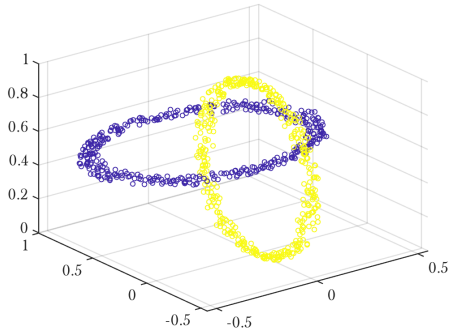
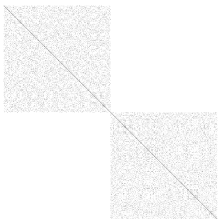
The influence of the hyperparameter σ^2 is based on the same principles as discussed in section 3.5.1 where the development of the dissimilarity matrix in the feature space is elaborated. Here again, a very small value of σ^2 will impose that two points are very close to each other so that there dissimiliarity is significantly greater than with the other points. A too small value of σ^2 will thus lead to an overall constant Ω_c matrix. The largest eigenvectors will be globally very similar putting all the data-points in the same cluster. Clusters with too much specifications are asked to the classifier, which he fails to find.

Conversely, a very high value of σ^2 will make no big difference in the dissimilarity between a close point and a relatively close point. This will lead to too much generalisation making too gross clusters. The classifier is not sensitive to details and fails to find the right clusters.

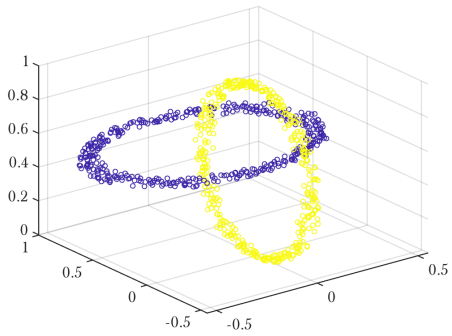
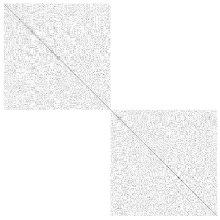
As always the good value of σ^2 is a trade-off between generalisation and specification. Here, it takes even a little more practical meaning: the σ^2 has to be in adequacy with the distance between the points in the cluster. Let's illustrate this very simplistically. In the toy example `two3drings`, neighbouring points of a ring are separated by an approximate distance of ≈ 0.05. Let's take three times this value as standard deviation as it corresponds to approximately 100% of a RBF area. This corresponds to $\sigma^2 \approx 0.008$. This is confirmed by the results for different values of σ^2 that can be seen at figure 3.4. The ordered dissimilarity matrix in the feature space — which is represented vy the kernel matrix, hence the link with KPCA — also gives a good estimation of the clustering results. Ideally, there should be no dissimilarity between data-points of a same cluster and thus sole the two diagonal blocks should contain non-zero elements.

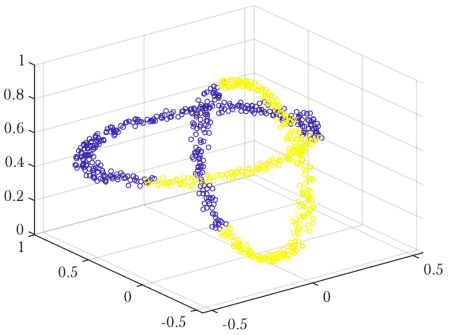
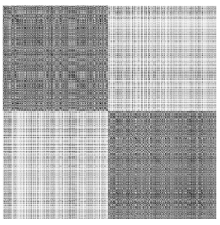
(A) $\sigma^2 = 0.001$.

(B) $\sigma^2 = 0.005$.

(C) $\sigma^2 = 0.01$.

(D) $\sigma^2 = 0.2$.

FIGURE 3.4: Influence of the hyperparameter σ^2 of the RBF on spectral clustering. The results are left and the corresponding ordered dissimilarity matrix in the feature space right (white corresponds to 0 and black to 1)

3.4 Fixed-size LS-SVM

As the eigenvalue decomposition increases dramatically with the feature space size, one must find a way to reduce it without loosing to much information. The idea developed in this exercise is to select a fixed amount M of points based on the maximisation of an entropy criterion, in this case the quadratic Rényi entropy (which is also called collision entropy)

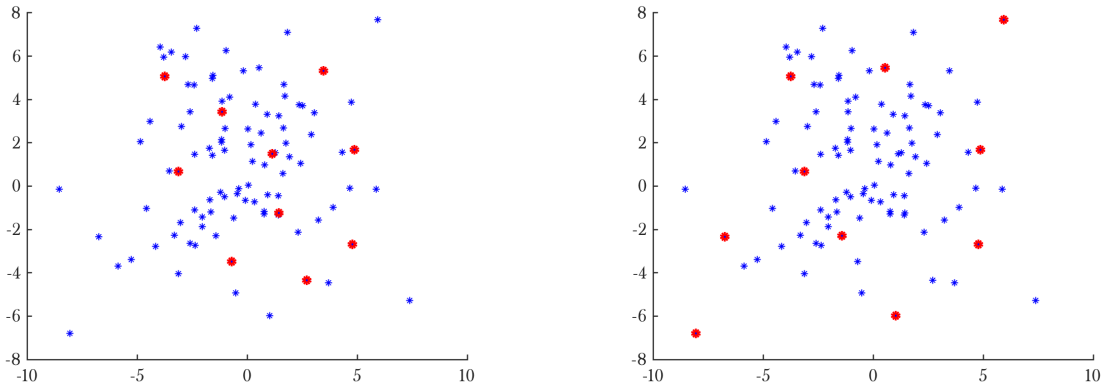
$$\begin{aligned} H &= -\log\left(\sum_{k=1}^{M}\left(\sum_{i=1}^{M} u_{i,k}\right)^2 \tilde{\lambda}_k\right) \\ &= -\log\left(\sum_{k=1}^{M} p(u_k)^2\right) \end{aligned}$$

with $\Omega_c U = U\tilde{\Lambda}$ and Ω_c the kernel matrix of the sub-feature space of the Nyrström approximation. If an eigenvector of normalised data has a lot of disparity and thus contains a lot of information, the sum of its elements will be low and the corresponding probability also. If a eigenvector is constant for example, it doesn't contain much information on the dissimilarity between the data-points and the sum of its elements will be high. By consequence, it will weigh a lot on the entropy. All these probabilities are quadratically added and weighted by their eigenvalue, to make large eigenvectors more important. The goal is minimise the weighted quadratic sum of these eigenvectors, or to maximise the entropy. In other words, the higher the entropy, the less probable the eigenvectors are and thus the more information they contain. The maximisation of the entropy corresponds to the minimisation of the lost information lost during the subspace selection. This is attained my maximising the dissimilarities between the subset points which corresponds to maximising the distances between the selected data-points in the input space.

Though, one has to choose a right mapping onto the feature space, *i.e.* the right hyper-parameter σ^2 and measures the sensitivity to the distance needed to define the dissimilarities as discussed in section 3.5.1. A low value for σ^2 will lead to considering close points as very distant and thus very dissimilar and a too high value will lead to considering all the data-points as close as each other. The σ^2-value acts thus as a sort of scale factor and one must find the ideal one, that considers distant points with a high dissimilarity and close points with a low dissimilarity. Figure 3.5 shows the influence of this hyper-parameter on the subspace selection. We see that $\sigma^2 = 0.1$, the selected points can be very close as very far from each other as dissimilarities will always be similar, this happens also for a too high value as for $\sigma^2 = 50$. The points seem to be more equally spaced for $\sigma^2 = 1$ and $\sigma^2 = 5$.

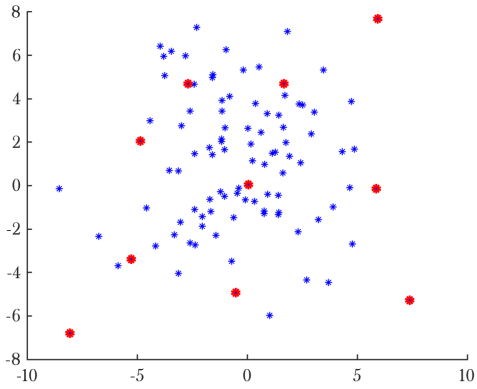
Figure 3.6 shows the evolution and the maximum entropy for different values of σ^2. Very low values consider points as very distant from each other leading to an overall very low dissimilarity whatever subset is chosen. This leads to an almost maximal entropy since the begininning with of course no possible amelioration. Very high values of σ^2 consider all points as close as each other, with a very high dissimilarity. This results in a possible amelioration as the subset still influences the dissimilarity matrix, but the maximum possible entropy is very low as all subset will have a relative high dissimilarity. The optimal values for σ^2 occur when the maximum possible entropy is very high, but when amelioration is also possible, meaning that the chosen subset has an influence. On figure 3.6, this corresponds — starting from the left or small values of σ^2 — when the dashed line begins to separate from the optimal line, *i.e.* the region $1 \lessapprox \sigma^2 \lessapprox 10$.

Intuitively, we can see that the subset converges to the subspace with the most separated points. If the points were very close, they would have a high dissimilarity causing that a change of one of the two points for another point would lead to a entropy increase. The selection converges thus to the *subset with the most separated points.*

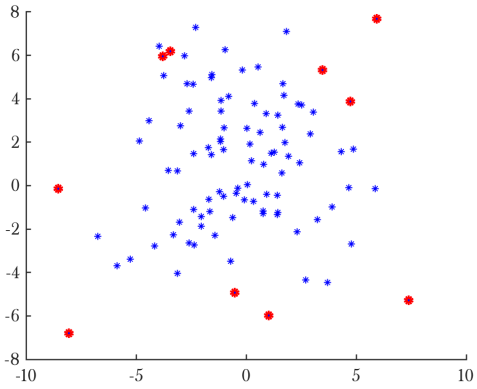


(A) $\sigma^2 = 0.1$.

(B) $\sigma^2 = 1$.



(C) $\sigma^2 = 5$.



(D) $\sigma^2 = 50$.

FIGURE 3.5: Influence of the hyper-parameter σ^2 on subset selected by maximising the quadratic Rényi entropy.

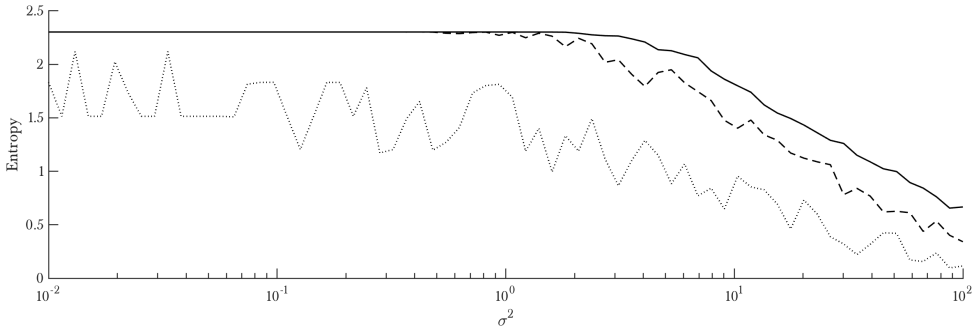
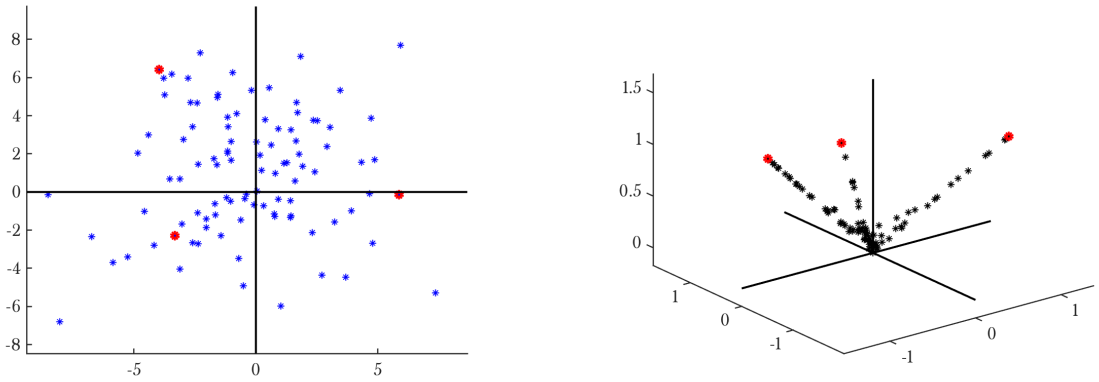


FIGURE 3.6: Influence of σ^2 on the maximum entropy (continuous line), the found entropy at the beginning of the search (dotted lines) and in the middle of the search (dashed line).

By reconstructing the feature we can notice that the resulting data-points are just a factor of one of the element of the subset. Ideally, each data-point corresponds to a factor of one sole subset element, the other factors being null. In this way, sparsity is achieved. This can be seen at figure 3.7.

Another alternative exists for specifying even more. This is done by minimising the l_0-norm of the support vector values, *i.e.* the number of non-zero α_i which is the sparsity criterion. Concretely,

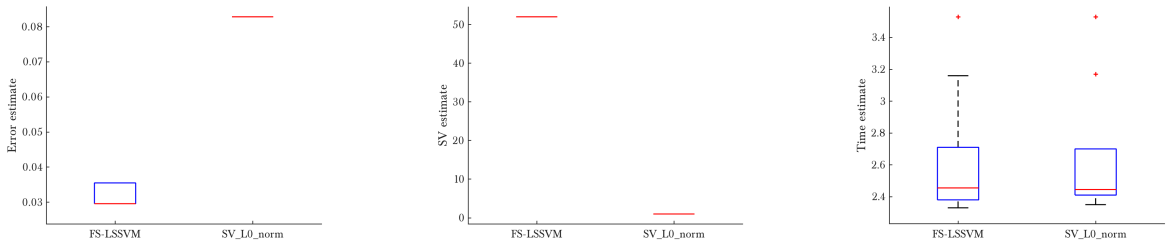


(A) Original space. (B) Feature space.

FIGURE 3.7: Mapping of the all the elements in the feature space on elements of the subset with maximal entropy ($\sigma^2 = 5$).

instead of minimising the norm of the weight vector $\|\omega\|_2^2$, one minimises $\|\alpha\|_2^2$, which becomes a minimisation of the l_0-norm of the α-vector during the iterative process. In other words, the smoothness criterion is replaced by a sparsity criterion. In the following case, the minimisation is directly done on the Nyström approximated support vectors and not on the full dimensional space. This helps to increase the sparsity even more. Of course, the algorithm will converge to a local minimum as the l_0-norm problem is NP-hard, but this suffices for our goals.

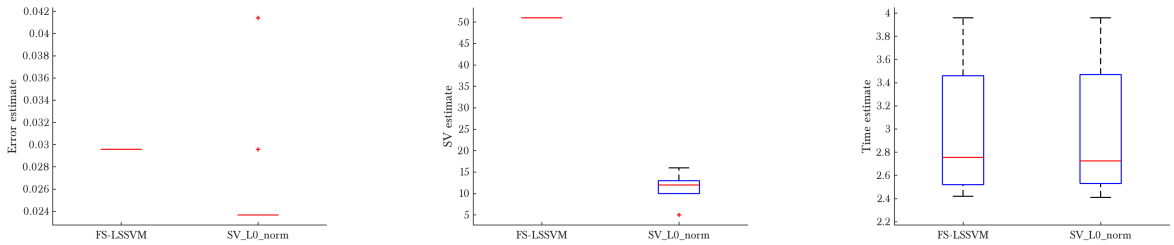
Results of this sparsifying approaches on the *Breast Cancer Dataset* are compared to the previously fixed size LS-SVM at figure 3.8. Firstly, the number of support vectors in much lower for the l_0 approximation. This makes sense as the l_0-approximation has been designed for big data-sets and to increase sparsity. By looking at both algorithms, we can notice that they have similar time executions. Indeed, the fixed size LS-SVM has complexity $\mathcal{O}(NM^2) = \mathcal{O}(k^2N^2)$ where N is the data-set size and $M = k\sqrt{N}$ the dimension of the Nyström approximations with k a factor for a heuristic approximation for selecting representative points and also $\mathcal{O}(k^2N^2)$ for the l_0-approximation[1]. The performance of the FS LS-SVM algorithm stays stable with the reduced Nyrtsröm dimension as it increases the performance for the l_0-approximation (figure 3.9) as well as the execution time. Though, after a certain value, it doesn't get significantly better (figure 3.10). Nevertheless, the execution time continues to increase. This is thus the optimal value for k.



(A) Error. (B) Number of support vectors. (C) Execution time.

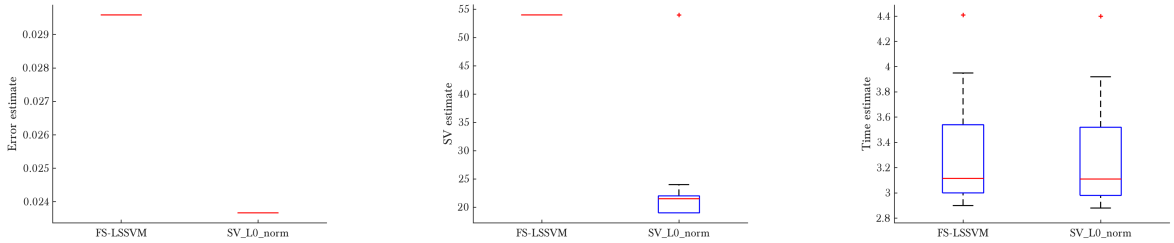
FIGURE 3.8: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 6$.

[1]Raghvendra Mall and J.A.K. Suykens. *"Sparse Reductions for Fixed-Size Least Squares Support Vector Machines on Large Scale Data"*



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.9: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 8$.



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.10: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 10$.

3.5 Homework Problems

3.5.1 Handwritten Digit Denoising

Influence of σ^2

Densoising consists in the extraction of the main components of a pattern and suppressing the less significant ones. More technically, this corresponds to the diagonalisation of a covariance matrix. In the case of a LS-SVM approach to a kernel PCA, this is the correlation matrix to be diagonalised

$$\Omega_c = \begin{bmatrix} (\phi(x_1) - \hat{\mu}_\phi)^T(\phi(x_1) - \hat{\mu}_\phi) & \cdots & (\phi(x_1) - \hat{\mu}_\phi)^T(\phi(x_N) - \hat{\mu}_\phi) \\ \vdots & \ddots & \vdots \\ (\phi(x_1) - \hat{\mu}_\phi)^T(\phi(x_N) - \hat{\mu}_\phi) & \cdots & (\phi(x_N) - \hat{\mu}_\phi)^T(\phi(x_N) - \hat{\mu}_\phi) \end{bmatrix} \tag{3.1}$$

with (using Mercer's trick)

$$\begin{aligned} \Omega_{c,kl} &= (\phi(x_k) - \hat{\mu}_\phi)^T(\phi(x_l) - \hat{\mu}_\phi) \\ &= K(x_k, x_l) - \frac{1}{N}\sum_{r=0}^{N} K(x_r, x_k) - \frac{1}{N}\sum_{r=0}^{N} K(x_r, x_l) + \frac{1}{N^2}\sum_{r=0}^{N}\sum_{s=0}^{N} K(x_r, x_s) \\ &= \exp\left(-\frac{\|x_k - x_l\|^2}{2\sigma^2}\right) - \frac{1}{N}\sum_{r=0}^{N}\exp\left(-\frac{\|x_r - x_k\|^2}{2\sigma^2}\right) - \frac{1}{N}\sum_{r=0}^{N}\exp\left(-\frac{\|x_r - x_l\|^2}{2\sigma^2}\right) \\ &\quad + \frac{1}{N^2}\sum_{r=0}^{N}\sum_{s=0}^{N}\exp\left(-\frac{\|x_r - x_s\|^2}{2\sigma^2}\right) \end{aligned}$$

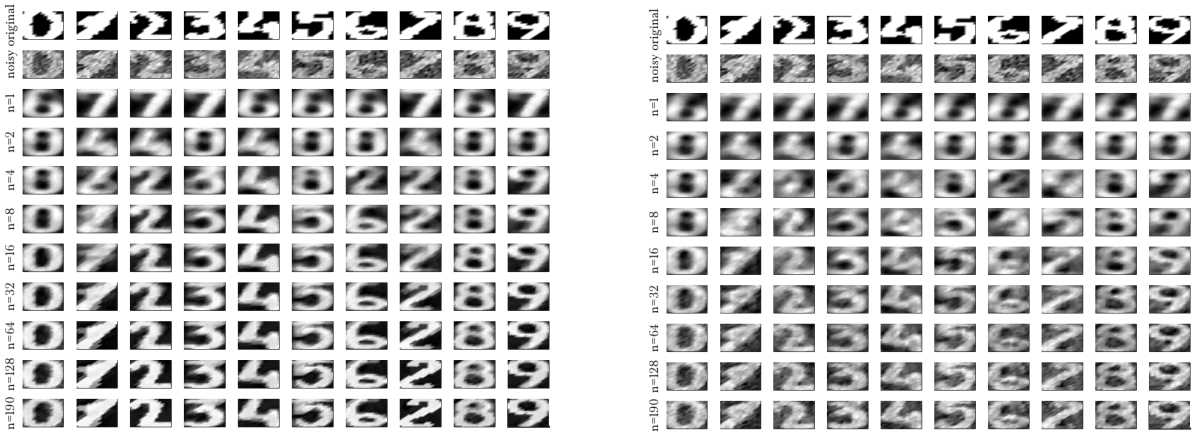
which can be written in english:

$$
\begin{aligned}
\Omega_{c,kl} \;=\;& \text{distance between } x_k \text{ and } x_l \\
&- \text{mean distance between } x_k \text{ and all data-points} \\
&- \text{mean distance between } x_l \text{ and all data-points} \\
&+ \text{mean distance between of all data-points}
\end{aligned}
$$

In other words, this is a relative measure of the distance between both data-points compared to the rest of the data-set. This value will thus be high if x_k and x_l are closer to each other. Ω_c is thus an enhanced covariance matrix of the data-points.

If the σ^2 is very low, the resulting measure of distance will be much stricter. Data-points will have to be very close to show some correlation. This will result in very few correlated point and an overall constant Ω_c. After diagonalisation the few meaningful principal components will correspond to very specialised features not always shared by all the digits of a same class. This is a form of *overfitting*.

Conversely, when the σ^2 is too high, all point will be considered relatively close to each other, without much disparity. The resulting Ω_c will thus similarly also be overall constant. The resulting principal components will correspond to very broad features often shared by many classes of digits. This will lead to a too strong generalisation without enough specification. This will also lead to the model not being able to make the distinction between the meaningful features and the noise as these meaningful features won't appear very correlated in Ω_c. By consequence a KPCA with a too high σ^2 will not be able to denoise correctly. This phenomenon can be seen at figure 3.11. One advantage though is that the lack of specialisation avoids to classify the digit 7 of the test set as a 2, but overall, a too high σ^2 leads to a bad denoising and thus a failure of the method with this parameter values choice.



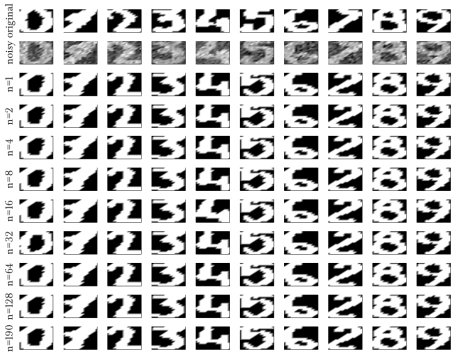
(A) $\sigma^2 \approx 36$ (rule of thumb).

(B) $\sigma^2 \approx 280$ (too high value).

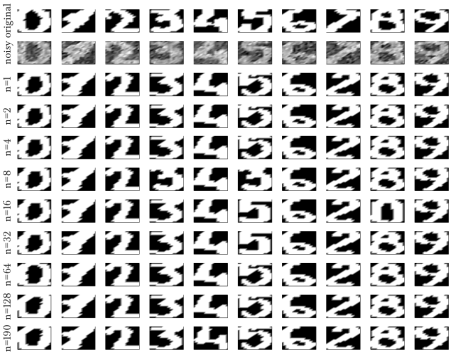
FIGURE 3.11: Influence of a too big parameter value for σ^2 in a KPCA denoising.

This overall phenomenon of overfitting depending on σ^2 and underfitting can be seen at figure 3.12. For very low values of σ^2, sole very clear components of the digits are found and everything is considered as noise. This results in almost no increase of performance as the number of components n_c increase, but also in a very rough reconstruction corresponding to the principal components of elements very close to each other. As the σ^2-value increase, the digits are decomposed in smoother components. This also lead to more components being needed to reconstruct

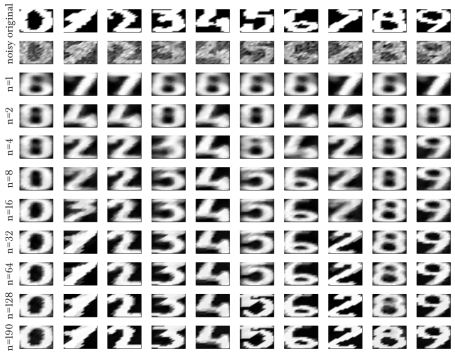
the digits correctly. For extreme values of σ^2, the same phenomenon as for figure 3.11 appears: too much generalisation leading to incorporation of the noise in the principal components. The overfitting and over-generalisation are to be seen at figure 3.13: the performance on the validation set first decreases with σ^2, and then increases drastically as the KPCA doesn't succeed in removing the noise anymore. As noticed before, the optimal σ^2 tends to increase with the number of components n_c. Furthermore, the results of the training and the validation set cannot be absolutely compared as the RMSE is sensitive to the set size. Quite logically, the training set only increases its RMSE with σ^2 as it overfits for small values, and the validation set is u-shaped showing the optimum.



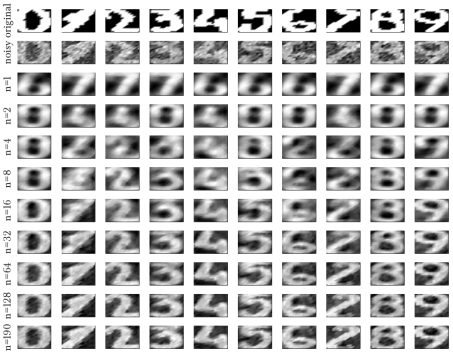
(A) $\sigma^2 \approx$ 5.13e-1.



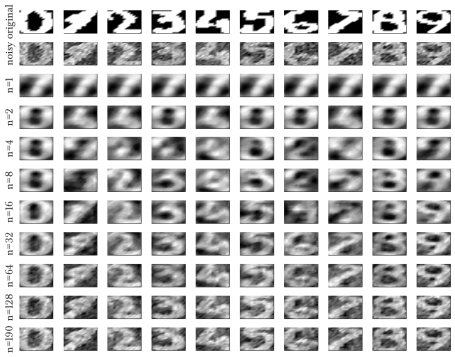
(B) $\sigma^2 \approx$ 3.24e+0.



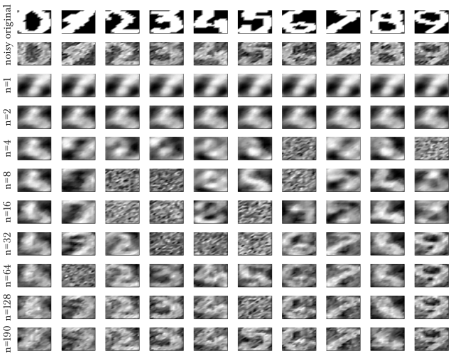
(C) $\sigma^2 \approx$ 2.04e+1.



(D) $\sigma^2 \approx$ 1.29e+2.

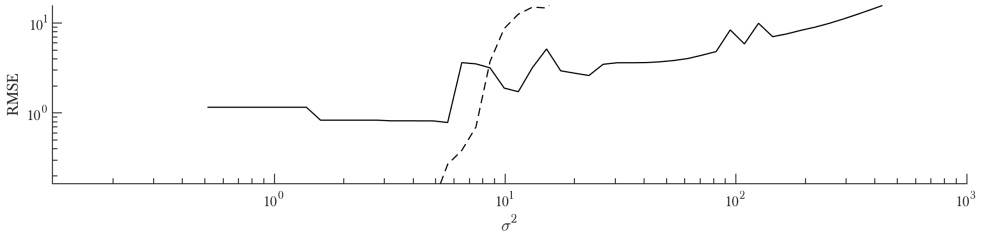


(E) $\sigma^2 \approx$ 8.13e+2.

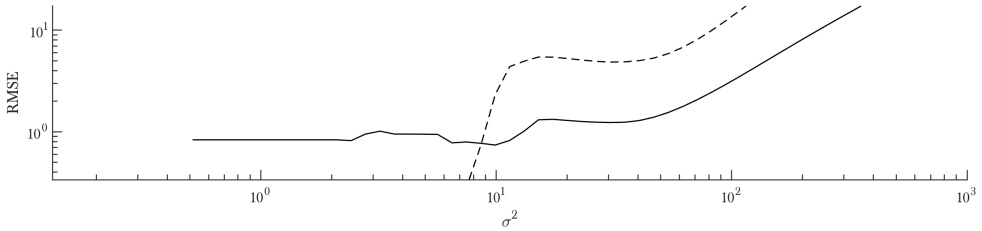


(F) $\sigma^2 \approx$ 5.13e+3.

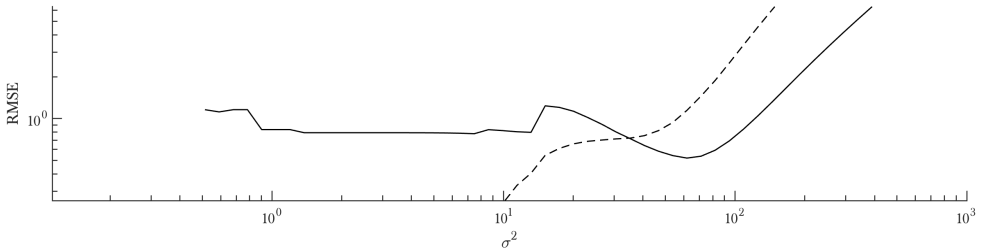
FIGURE 3.12: Overall influence of the parameter value for σ^2 in a KPCA denoising.



(A) $n_c = 2$.



(B) $n_c = 16$.

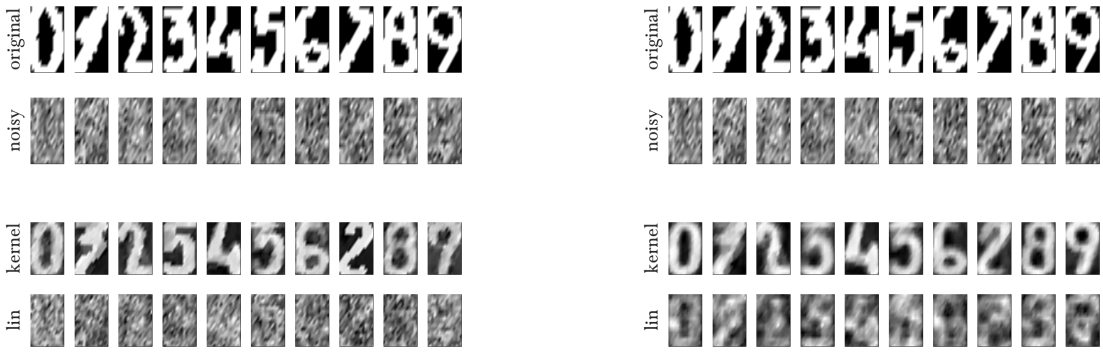


(C) $n_c = 64$.

FIGURE 3.13: Performance of the validation (continuous) and training (dotted) set in function of σ^2 for a different number of components n_c. The focus is done on the validation set.

Comparison of linear and kernel PCA

A general trend observed at figure 3.13 is the overall optimum performance on the validation set increase with the number of components: this suggests that all the noise is captured in the last components. This makes sense as if the noise is perfectly decorrelated — which should be the case in an ideal infinite training set where all digits have the same shape with additional perfectly white noise —, the corresponding components should have a corresponding eigenvalue almost null. As shown before, (classical) linear PCA is much more sensitive to noise and much more components tend to carry the noise. Therefore, it is not possible to choose to kas meep uch components as for the KPCA. To summarise, KPCA works ideally with keeping all components, the only last ones being dropped contain the major part of the noise and (classical) linear PCA ideally works by keeping as few components as possible as the noise is comprised in the most part of them. Overall, KPCA denoising is able to restore much more information and is thus a clear winner. This phenomenon can well be seen at figure 3.14. For a broader discussion on the difference between (classical) linear and kernel PCA in the specific case of denoising, please refer to section 3.2.



(A) $n_c = 190$.

(B) $n_c = 20$.

FIGURE 3.14: Comparison of (classical) linear PCA and KPCA on the *validation set* in the case of heavy noise for a different number of components n_c.

Trying these two methods with their optimal number of components on the test set leads interestingly to better results that on the validation set (figure 3.15). It is true that no parameter optimisation has been optimised on the validation set except the number of components very simplistically (it was actually more the illustration of a general trend). In this sense, the validation set still acts as a test here and should obtain similar results. The difference observed should therefore more be a consequence of the data-points of the set itself. The data-points of the test set must have shapes corresponding more to those of the training set, those of the validation set are more "extravagant".

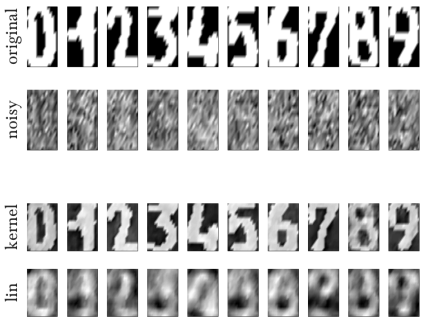


FIGURE 3.15: Comparison of (classical) linear PCA and KPCA on the *test set* with heavy noise. The KPCA has $n_c = 190$ and the (classical) linear PCA $n_c = 20$.

Let us now focus or on KPCA. We set $n_c = 190$ and calculate the optimal value for σ^2 (figure 3.16). We notice that the optimal range of the training set

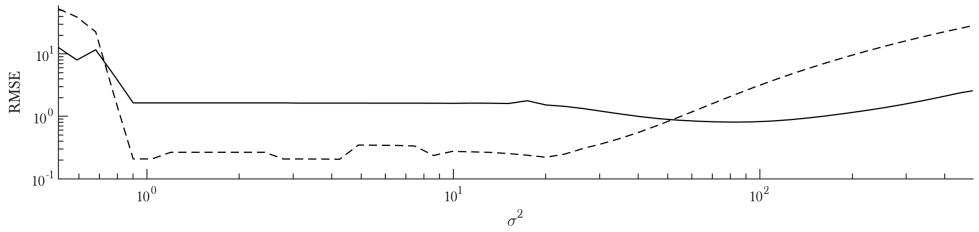


FIGURE 3.16: Performance of the validation (continuous) and training (dotted) set in function of σ^2 for $n_c = 190$ in case of a heavy noise. The focus is done on the validation set.

occurs for lower σ^2-values than the validation set. This is of course due to the fact that the KPCA is optimised for the training set and is allowed to verfit, what is not the case for the validation set. Another difference is that very low values of σ^2 aren't performing well on the training set and don't lead to verfitting: this is caused by the important noise that cannot be overfitted. The optimal value is given by the validation set and is worth $\sigma^2 \approx 82$. This value is very close to the one given by the rule of thumb $\sigma^2 \approx 51$. Applying it to the test set shows very similar results, indicating that the rule thumb is well chosen (figure 3.17).

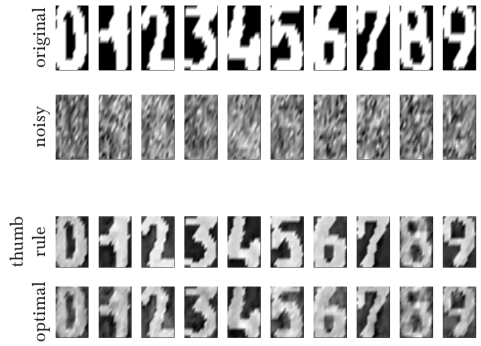


FIGURE 3.17: Comparison of the optimal value for σ^2 determined on the validation set to the rule of thumb.

3.5.2 Shuttle (statlog)

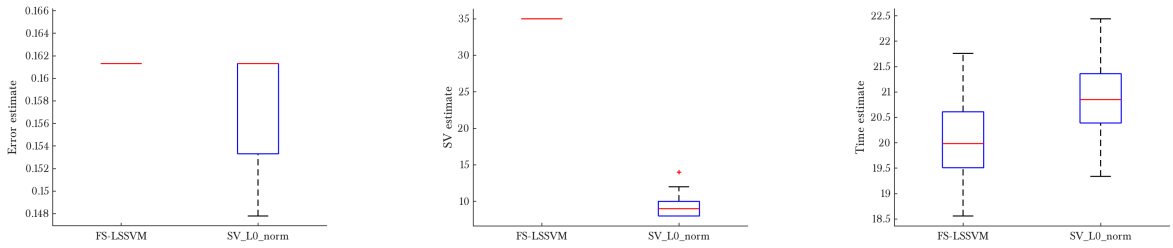
The StatLog shuttle data-set (NASA) contains 57000 data-points with 9 features. Each data-point is assigned to one of the seven classes, where 80% of the data-points are assigned to the first class. Therefore a algorithm which assigns class 1 by default to all data-points should have a performance of 80%. On the contrary, an algorithm that assigns a class randomly to each data-points should have a performance of $\frac{1}{7} = 14\%$. According to the authors of the data-set, the ideal goal is to have a performance of 99-99.9%. For the purpose of this homework, the data-set has been randomly divided into a test set of 13 500 data-points and a training set of 43 500 data-points.

The toolbox as given only was written for binary classification. An adaptation to the code was needed to determine multiclass classification. The estimated classes were defined as

```
testYh = max(round(testYh/max(testYh)*7),1)
```

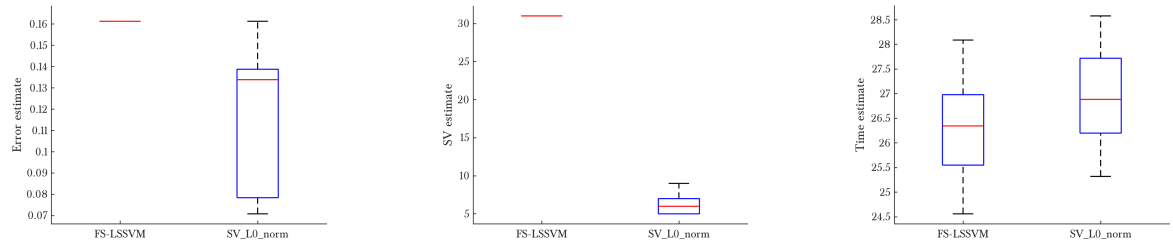
and matched to the corresponding classes of the training set by using the highest match for each class, starting with class 1, the biggest. This had to be adapted for the test and training set of the methods, but also for the tuning of the hyperparameters.

On the other side, the size of the training set is quite important which obliges us to use a fast algorithm. For this reason, the Randomised Directional search, which can run in parallel, to the Coupled Simulated Annealing which only runs is sequential. For execution time purposes, the sole linear kernel will be used. Results for different reduction dimensions can be found at figures 3.18, 3.19 and 3.20. The optimal mean score is approximately 87% and is obtained for $k = 8$, increasing it only results in an execution time increase. This is significantly better than the default score but still lower than an almost perfect classification as the authors ideally want. The better score is in overall obtained by the l_0 method, but this it takes also much more time. The multiclass error as implemented here is a relative greedy program which increases the execution time of the program that uses it the most, in this case, the l_0 method. This explains why this algorithm has a greater execution time.



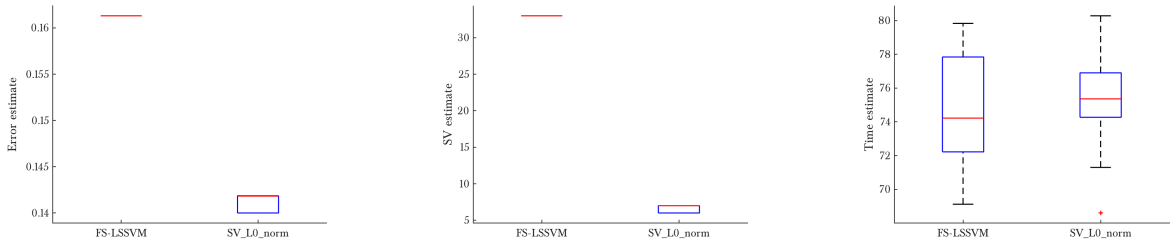
(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.18: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 6$.



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.19: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 8$.



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.20: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 15$.

3.5.3 California

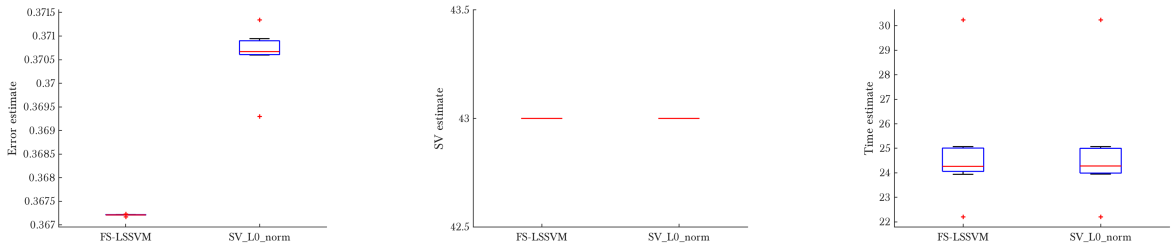
According to the authors[2], the data pertains to the houses found in a given California district and some summary stats about them based on the 1990 census data. The columns are as follows, their names are pretty self explanatory:

- longitude
- latitude
- housing median age
- total number of rooms
- total number of bedrooms
- population
- households
- median income
- median house value
- ocean proximity

In the data-set given for this homework, the only binary variable (ocean proximity) was not included. The goal is to determine the median house value.

For the propose of this algorithm, the data is first preprocessed. A classical normalisation is applied to each of the input and output variables. Theoretically, this doesn't change the performance of the algorithm as each support vector is weighted by its respective support vector value. Nevertheless, in addition to better preventing numerical rounding errors due to the magnitude difference, the optimisation of the subset in the Nyrström approximation as well as the l_0-norm minimisation aren't convex anymore. After normalisation of the output data, a random sequence of the same variance should have a RMSE on the output data of approximately $\sqrt{2} \approx 1.41$. This is thus the base result for a totally random regressor. Furthermore, a default regressor which should give the mean value (0) to all data should have an RMSE of approximately $\sigma^2 = 1$. This is thus our default comparison score: any regressor that has statistically learned something should perform with an RMSE < 1.

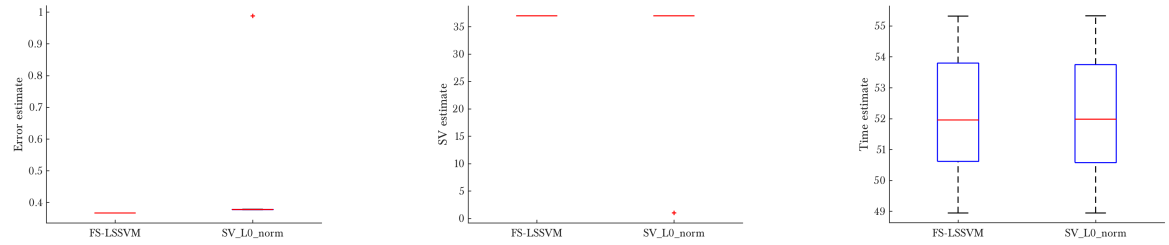
Trying a linear kernel doesn't give much results (figure 3.21). Even by increasing the dimension size, there is a stagnation of the performance (figure 3.22); only the execution size increases. The results are still better than a default regressor, but the stagnation suggests we could do better. It looks like the linear classifier is too general to specify on the training data. With a smaller k, the results are already much better (figure 3.23 and 3.23). In the case of polynomial kernels, $k = 2$ seems to be the optimal choice. The sole problem is the execution time. As always, the designer's choice is to decide wether precision is wanted or rapidity.



(A) Error. (B) Number of support vectors. (C) Execution time.

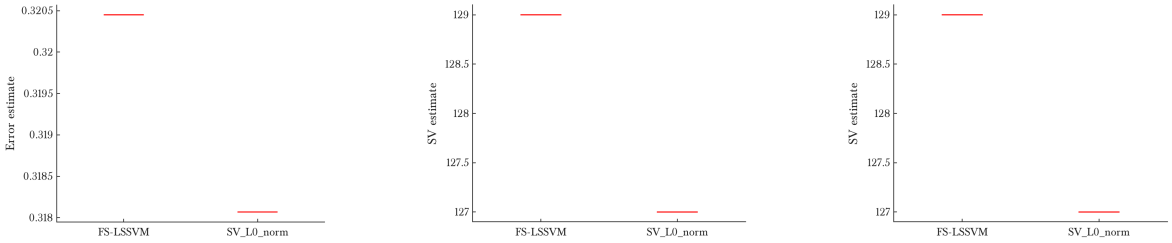
FIGURE 3.21: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 4$ with a linear kernel function.

[2]Pace, R. Kelley, and Ronald Barry. *"Sparse spatial autoregressions."* Statistics & Probability Letters 33.3 (1997): 291-297.



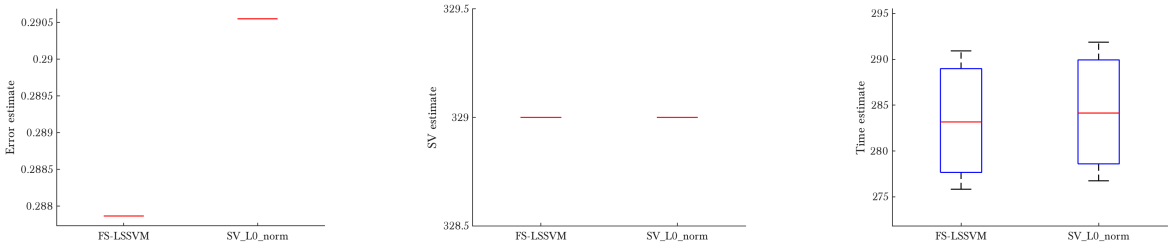
(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.22: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 8$ with a linear kernel function.



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.23: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 1$ with a polynomial kernel function.



(A) Error. (B) Number of support vectors. (C) Execution time.

FIGURE 3.24: Comparison of a fixed size LS-SVM approach to a l_0-approximation with $k = 4$ with a polynomial kernel function.

Appendix

Appendix A

MATLAB

A.1 Classification

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 1
% Question 1
% Author: Henri DE PLAEN
% Date: 3/8/2018

clear all ; close all ; clc ;
addpath('../docs');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
X1 = 1 + randn(50,2);
X2 = -1 + randn(51,2);

Y1 = ones(50,1);
Y2 = -ones(51,1);

X = [X1;X2];
Y = [Y1;Y2];

hold on ;
plot(X1(:,1),X1(:,2),'ro');
plot(X2(:,1),X2(:,2),'bo');

axis([-4 4 -4 4]) ;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
hold on ;
ax = gca ;
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
%set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
```

```
set(lin,'LineWidth',1.2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('x') ; ylabel('y') ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 3
% Question 1
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;
xt_1 = Xt(Yt==1,:) ;
xt_2 = Xt(Yt==-1,:) ;

% params
gam = 1e+1 ;
degree_max = 4 ;
t = 1 ;

% lin
disp('lin') ;
[alpha,b] = trainlssvm({X,Y,'c',gam,[],'lin_kernel'});

figure(1) ;
plotlssvm({X,Y,'c',gam,[],'lin_kernel'},{alpha,b});

hold on ;

plot(xt_1(:,1),xt_1(:,2),'^k') ;
plot(xt_2(:,1),xt_2(:,2),'vk') ;


legend({'Classifier','Training set (class 1)','Training set (class 2)','Test ...
    set (class 1)','Test set (class 2)'}) ;


[Yht, ~] = simlssvm({X,Y,'c',gam,[],'lin_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;

disp(pc_err) ;

% poly
disp('poly') ;
degree_span = 1:degree_max ;

for idx = 2:length(degree_span)
    % calc
    [alpha,b] = trainlssvm({X,Y,'c',gam,[t,degree_span(idx)],'poly_kernel'});

    % plot
    figure(idx) ;
    plotlssvm({X,Y,'c',gam,[t,degree_span(idx)],'poly_kernel'},{alpha,b});
```

```
    hold on ;

    plot(xt_1(:,1),xt_1(:,2),'^k') ;
    plot(xt_2(:,1),xt_2(:,2),'vk') ;

    h_l = legend({'Classifier','Training set (class 1)','Training set (class ...
        2)','Test set (class 1)','Test set (class 2)'}) ;

    if idx == degree_max
        set(h_l,'visible','on') ;
    else
        set(h_l,'visible','off') ;
    end

    % error
    [Yht, ~] = simlssvm({X,Y,'c',gam,[t,degree_span(idx)],'poly_kernel'}, ...
        {alpha,b}, Xt);
    err = sum(Yht~=Yt);
    pc_err = err/length(Yt) ;
    disp(pc_err) ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx_fig = 1:degree_max
    figure(idx_fig) ;
    hold on ;
    ax = gca ;
    lin = findobj(gca, 'Type', 'Line') ;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    %set(0,'DefaultLineColor','k');
    set(gca,'box','off') ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18) ;
    set(gca,'LineWidth',1.2) ;
    set(lin,'LineWidth',1.2) ;
    %set(lin,'MarkerFaceColor','k') ;
    %set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
    xlabel('x') ; ylabel('y') ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 1
% Question 1
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;
xt_1 = Xt(Yt==1,:) ;
xt_2 = Xt(Yt==-1,:) ;

% params
type = 'c' ;
```

```

% min_gam = .5 ;
% max_gam = 3 ;
% n_gam = 6 ;
gam_span = [.1 .5 3] ;
line_sp = {':k','--k','-k'} ;

min_sig = -2 ;
max_sig = 2 ;
n_sig = 500 ;

% rbf
% gam_span = linspace(min_gam,max_gam,n_gam) ;
sig_span = linspace(min_sig,max_sig,n_sig) ;
sig_span = 10.^sig_span ;
pc_err = zeros(1,length(sig_span)) ;

for idx1 = 1:length(gam_span)
    gam_loc = gam_span(idx1) ;
    for idx2 = 1:length(sig_span)
        % calc
        [alpha,b] = trainlssvm({X,Y,type,gam_loc,sig_span(idx2),'RBF_kernel'});

        % error
        [Yht, ~] = simlssvm({X,Y,type,gam_loc,sig_span(idx2),'RBF_kernel'}, ...
            {alpha,b}, Xt);
        err = sum(Yht~=Yt);
        pc_err(idx2) = err/length(Yt) ;
    end
    semilogx(sig_span,pc_err*100,line_sp{idx1}) ;
    hold on ;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
ax = gca ;
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
set(lin,'LineWidth',2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('log \sigma^2') ; ylabel('Test error [%]') ;

legend({'\gamma=0.1', '\gamma=0.5', '\gamma=3.0'}) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 1
% Question 1
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
```

```
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;
xt_1 = Xt(Yt==1,:) ;
xt_2 = Xt(Yt==-1,:) ;

% params
type = 'c' ;

min_gam = -2 ;
max_gam = 4 ;
n_gam = 500 ;

line_sp = {':k','--k','-k'} ;

% min_sig = -2 ;
% max_sig = 2 ;
% n_sig = 500 ;
sig_span = [.1 1 10] ;

% rbf
gam_span = linspace(min_gam,max_gam,n_gam) ;
gam_span = 10.^gam_span ;
%sig_span = linspace(min_sig,max_sig,n_sig) ;
pc_err = zeros(1,length(gam_span)) ;

for idx1 = 1:length(sig_span)
    sig_loc = sig_span(idx1) ;
    for idx2 = 1:length(gam_span)
        % calc
        [alpha,b] = trainlssvm({X,Y,type,gam_span(idx2),sig_loc,'RBF_kernel'});

        % error
        [Yht, ~] = simlssvm({X,Y,type,gam_span(idx2),sig_loc,'RBF_kernel'}, ...
            {alpha,b}, Xt);
        err = sum(Yht~=Yt);
        pc_err(idx2) = err/length(Yt) ;
    end
    semilogx(gam_span,pc_err*100,line_sp{idx1}) ;
    hold on ;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
ax = gca ;
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
set(lin,'LineWidth',2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('log \gamma') ; ylabel('Test error [%]') ;
```

```
legend({'\sigma^2=0.1', '\sigma^2=1', '\sigma^2=10'}) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 3
% Question 4
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;

% params
type = 'c' ;
n = 100 ;

min_gam = -2 ;
max_gam = 3 ;
n_gam = n ;
gam_span = linspace(min_gam,max_gam,n_gam) ;
gam_span = 10.^gam_span ;

min_sig = -2 ;
max_sig = 4 ;
n_sig = n ;
sig_span = linspace(min_sig,max_sig,n_sig) ;
sig_span = 10.^sig_span ;

misclass = zeros(length(gam_span),length(sig_span)) ;

% generate random indices
idx = randperm(size(X,1));

% create the training and validation sets
% using the randomized indices
Xtrain = X(idx(1:80),:);
Ytrain = Y(idx(1:80));
Xval = X(idx(81:100),:);
Yval = Y(idx(81:100));

for idx1 = 1:length(gam_span)
    gam_loc = gam_span(idx1) ;
    for idx2 = 1:length(sig_span)


        % train lssvm model
        [alpha,b] = ...
            trainlssvm({Xtrain,Ytrain,'c',gam_loc,sig_span(idx2),'RBF_kernel'});

        % evaluate
        estYval = ...
            simlssvm({Xtrain,Ytrain,'c',gam_loc,sig_span(idx2),'RBF_kernel'}, ...
            {alpha,b},Xval);

        misclass(idx1,idx2) = sum(estYval~=Yval)/length(Yval);
```

```
    end
end

[C,h] = contourf(gam_span,sig_span,misclass*100) ;
set(h,'LineColor','none') ;
colormap(flipud(gray)) ;
caxis([0 50]) ;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
hold on ;
ax = gca ;
set(ax,'xscale','log','yscale','log');
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
set(lin,'LineWidth',2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('\gamma') ; ylabel('\sigma^2') ;
zlabel('Test error [%]') ;

%legend({'\gamma=1', '\gamma=10', '\gamma=100'}) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 3
% Question 5
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;

% params
type = 'c' ;
n = 100 ;

min_gam = -2 ;
max_gam = 3 ;
n_gam = n ;
gam_span = linspace(min_gam,max_gam,n_gam) ;
gam_span = 10.^gam_span ;

min_sig = -2 ;
max_sig = 4 ;
n_sig = n ;
sig_span = linspace(min_sig,max_sig,n_sig) ;
sig_span = 10.^sig_span ;

```

```
performance = zeros(length(gam_span),length(sig_span)) ;

for idx1 = 1:length(gam_span)
    gam_loc = gam_span(idx1) ;
    for idx2 = 1:length(sig_span)
        % train lssvm model
        performance(idx1,idx2) = ...
            crossvalidate({X,Y,'c',gam_span(idx1),sig_span(idx2),'RBF_kernel'}, ...
            10,'misclass');
    end
end

[C,h] = contourf(gam_span,sig_span,performance*100) ;
set(h,'LineColor','none') ;
colormap(flipud(gray)) ;
caxis([0 50]) ;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
hold on ;
ax = gca ;
set(ax,'xscale','log','yscale','log');
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
set(lin,'LineWidth',2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('\gamma') ; ylabel('\sigma^2') ;
zlabel('Test error [%]') ;

%legend({'\gamma=1', '\gamma=10', '\gamma=100'}) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Part 3
% Question 5bis
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load iris ;

% params
type = 'c' ;
n = 100 ;

min_gam = -2 ;
max_gam = 3 ;
n_gam = n ;
```

```
gam_span = linspace(min_gam,max_gam,n_gam) ;
gam_span = 10.^gam_span ;

min_sig = -2 ;
max_sig = 4 ;
n_sig = n ;
sig_span = linspace(min_sig,max_sig,n_sig) ;
sig_span = 10.^sig_span ;

performance = zeros(length(gam_span),length(sig_span)) ;

for idx1 = 1:length(gam_span)
    gam_loc = gam_span(idx1) ;
    for idx2 = 1:length(sig_span)
        % train lssvm model
        performance(idx1,idx2) = ...
            leaveoneout({X,Y,'c',gam_span(idx1),sig_span(idx2),'RBF_kernel'},'misclass');
    end
end

[C,h] = contourf(gam_span,sig_span,performance*100) ;
set(h,'LineColor','none') ;
colormap(flipud(gray)) ;
caxis([0 50]) ;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
hold on ;
ax = gca ;
set(ax,'xscale','log','yscale','log');
lin = findobj(gca, 'Type', 'Line') ;
ax.XAxisLocation = 'origin';
ax.YAxisLocation = 'origin';
set(0,'DefaultLineColor','k');
set(gca,'box','off') ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18) ;
set(gca,'LineWidth',1.2) ;
set(lin,'LineWidth',2) ;
%set(lin,'MarkerFaceColor','k') ;
%set(gca,'XTickLabel',[]) ; set(gca,'YTickLabel',[])
xlabel('\gamma') ; ylabel('\sigma^2') ;
zlabel('Test error [%]') ;

%legend({'\gamma=1', '\gamma=10', '\gamma=100'}) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Homework 1
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load ripley.mat ;

```

```
X1 = X(Y==1,:) ;
X2 = X(Y==-1,:) ;
Xt1 = Xt(Yt==1,:) ;
Xt2 = Xt(Yt==-1,:) ;

% visualise
figure(1) ;
hold on ;
plot(X1(:,1),X1(:,2),'*k') ;
plot(X2(:,1),X2(:,2),'ok') ;

% lin
figure(2) ;
gam = 5e+0 ;
[alpha,b] = trainlssvm({X,Y,'c',gam,[],'lin_kernel'});
plotlssvm({X,Y,'c',gam,[],'lin_kernel'},{alpha,b});

figure(3) ;
plotlssvm({X,Y,'c',gam,[],'lin_kernel'},{alpha,b});
hold on ;
plot(Xt1(:,1),Xt1(:,2),'^y') ;
plot(Xt2(:,1),Xt2(:,2),'vb') ;
legend({'Classifier','Training set (class 1)','Training set (class 2)','Test ...
    set (class 1)','Test set (class 2)'}) ;
[Yht, Yl] = simlssvm({X,Y,'c',gam,[],'lin_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;

% rbf
model = {X,Y,'c',[],[],'RBF_kernel','csa'};
[gam,sig2,cost] = tunelssvm(model,'simplex', ...
    'crossvalidatelssvm',{10,'misclass'});

figure(5) ;
[alpha,b] = trainlssvm({X,Y,'c',gam,sig2,'RBF_kernel'});
plotlssvm({X,Y,'c',gam,sig2,'RBF_kernel'},{alpha,b});

figure(6) ;
plotlssvm({X,Y,'c',gam,sig2,'RBF_kernel'},{alpha,b});
hold on ;
plot(Xt1(:,1),Xt1(:,2),'^y') ;
plot(Xt2(:,1),Xt2(:,2),'vb') ;
legend({'Classifier','Training set (class 1)','Training set (class 2)','Test ...
    set (class 1)','Test set (class 2)'}) ;
[Yht, Yl] = simlssvm({X,Y,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp('perf') ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
for idx = 1:7
    figure(idx) ;
    ax = gca ;
    lin = findobj(gca, 'Type', 'Line') ;
    %ax.XAxisLocation = 'origin';
    %ax.YAxisLocation = 'origin';
    set(0,'DefaultLineColor','k');
    set(gca,'box','off') ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18) ;
    set(gca,'LineWidth',1.2) ;
    %set(lin,'LineWidth',4) ;
    %set(lin,'Color','k') ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Homework 2
% Author: Henri DE PLAEN
% Date: 8/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load breast.mat ;

X = trainset ;
Y = labels_train ;
Xt = testset ;
Yt = labels_test ;

% normalise
X_mean = mean(X,1) ;
X_std = std(X,1) ;
X = X-repmat(X_mean,size(X,1),1) ;
X = X./repmat(X_std,size(X,1),1) ;

Xt = Xt-repmat(X_mean,size(Xt,1),1) ;
Xt = Xt./repmat(X_std,size(Xt,1),1) ;

X1 = X(Y==1,:) ;
X2 = X(Y==-1,:) ;
Xt1 = Xt(Yt==1,:) ;
Xt2 = Xt(Yt==-1,:) ;

% visualise
figure(1) ;
m1 = mean(X1,1) ;
m2 = mean(X2,1) ;
bar([m1;m2]');
legend('Class 1','Class 2') ;
hold on ;

figure(2) ;
hold on ;
```

```
edges = linspace(-2,4,20) ;
histogram(X1(:,21),edges,'FaceColor','k') ;
histogram(X2(:,21),edges,'FaceColor','r') ;
legend('Class 1','Class 2') ;

figure(3) ;
plot(mean(X1,2),std(X1,1,2),'or') ;
hold on ;
plot(mean(X2,2),std(X2,1,2),'ob') ;
legend('Class 1','Class 2') ;
xlabel('\mu') ; ylabel('\sigma') ;

% figure(4) ; [~,score1] = pca(X1) ; [~,score2] = pca(X2) ;
% plot(score1(:,2),score1(:,3),'or') ; hold on ;
% plot(score2(:,2),score2(:,3),'ob') ; legend('Class 1','Class 2') ;
% xlabel('PCA1') ; ylabel('PCA2') ;

% lin
gam = 4e+0 ;
[alpha,b] = trainlssvm({trainset,labels_train,'c',gam,[],'lin_kernel'});

[Yht, Yl] = simlssvm({X,Y,'c',gam,[],'lin_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;

[alpha,b] = trainlssvm({X(:,1:2),Y,'c',gam,[],'lin_kernel'});
[Yht, ~] = simlssvm({X(:,1:2),Y,'c',gam,[],'lin_kernel'}, {alpha,b}, Xt(:,1:2));
plotlssvm({X(:,1:2),Y,'c',gam,[],'lin_kernel'}, {alpha,b}) ;
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp(pc_err) ;


% rbf
model = {X,Y,'c',[],[],'RBF_kernel','csa'};
[gam,sig2,cost] = tunelssvm(model,'simplex', ...
    'crossvalidatelssvm',{10,'misclass'});

[alpha,b] = trainlssvm({X,Y,'c',gam,sig2,'RBF_kernel'});

[Yht, Yl] = simlssvm({X,Y,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp('perf') ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
for idx = 1:6
```

```
    figure(idx) ;
    ax = gca ;
    lin = findobj(gca, 'Type', 'Line') ;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    set(0,'DefaultLineColor','k');
    set(gca,'box','off') ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18) ;
    set(gca,'LineWidth',1.2) ;
    %set(lin,'LineWidth',4) ;
    %set(lin,'Color','k') ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 1
% Homework 3
% Author: Henri DE PLAEN
% Date: 8/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
load diabetes.mat ;

X = trainset ;
Y = labels_train ;
Xt = testset ;
Yt = labels_test ;

% normalise
X_mean = mean(X,1) ;
X_std = std(X,1) ;
X = X-repmat(X_mean,size(X,1),1) ;
X = X./repmat(X_std,size(X,1),1) ;

Xt = Xt-repmat(X_mean,size(Xt,1),1) ;
Xt = Xt./repmat(X_std,size(Xt,1),1) ;

X1 = X(Y==1,:) ;
X2 = X(Y==-1,:) ;
Xt1 = Xt(Yt==1,:) ;
Xt2 = Xt(Yt==-1,:) ;

% titles = {'?Number of times pregnant ';'Plasma glucose concentration ...
    ';'Diastolic blood pressure ';'Triceps skin fold thickness ';'2-Hour ...
    serum insulin ';'Body mass index ';'Diabetes pedigree function ';'Age '};
%
% % visualise
% figure(1) ;
% for idx = 1:8
%     subplot(2,4,idx) ; hold on ;
%     edges = linspace(min(X(:,idx)),max(X(:,idx)),20) ;
%     ...
    histogram(X1(:,idx),edges,'FaceColor','k','Normalization','probability') ;
%     ...
    histogram(X2(:,idx),edges,'FaceColor','r','Normalization','probability') ;
%     legend('Class 1','Class 2') ;
```

```
%     title(titles{idx}) ;
%
%     ax = gca ;
%     lin = findobj(gca, 'Type', 'Line') ;
%     ax.XAxisLocation = 'origin';
%     ax.YAxisLocation = 'origin';
%     set(0,'DefaultLineColor','k');
%     set(gca,'box','off') ;
%     set(gca, 'FontName', 'Baskervald ADF Std')
%     set(gca, 'FontSize', 18) ;
%     set(gca,'LineWidth',1.2) ;
% end

figure(1) ;
plot(mean(X1,2),std(X1,1,2),'or') ;
hold on ;
plot(mean(X2,2),std(X2,1,2),'ob') ;
legend('Class 1','Class 2') ;
xlabel('\mu') ; ylabel('\sigma') ;

% figure(4) ; [~,score1] = pca(X1) ; [~,score2] = pca(X2) ;
% plot(score1(:,2),score1(:,3),'or') ; hold on ;
% plot(score2(:,2),score2(:,3),'ob') ; legend('Class 1','Class 2') ;
% xlabel('PCA1') ; ylabel('PCA2') ;

% lin
gam = 4e+0 ;
[alpha,b] = trainlssvm({trainset,labels_train,'c',gam,[],'lin_kernel'});

[Yht, Yl] = simlssvm({X,Y,'c',gam,[],'lin_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;

% rbf
model = {X,Y,'c',[],[],'RBF_kernel','csa'};
[gam,sig2,cost] = tunelssvm(model,'simplex', ...
    'crossvalidatelssvm',{10,'misclass'});

[alpha,b] = trainlssvm({X,Y,'c',gam,sig2,'RBF_kernel'});

[Yht, Yl] = simlssvm({X,Y,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, Xt);
err = sum(Yht~=Yt);
pc_err = err/length(Yt) ;
disp('perf') ;
disp(pc_err) ;

roc(Yl,Yt);
lin = findobj(gca, 'Type', 'Line') ;
set(lin,'Color','k') ;
set(lin,'LineWidth',4) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
for idx = 1:3
```

```
    figure(idx) ;
    ax = gca ;
    lin = findobj(gca, 'Type', 'Line') ;
    ax.XAxisLocation = 'origin';
    ax.YAxisLocation = 'origin';
    set(0,'DefaultLineColor','k');
    set(gca,'box','off') ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18) ;
    set(gca,'LineWidth',1.2) ;
    %set(lin,'LineWidth',4) ;
    %set(lin,'Color','k') ;
end
```

A.2 Function Estimation and Time-series Prediction

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 1
% Question 1
% Author: Henri DE PLAEN
% Date: 5/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs'));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% f = @(x) 2.*x ;
f = @(x) x.^2 ;
n = 22 ;

X = linspace(-1,1,n) ;
Y = f(X) + .3*randn(size(X)) ;
X = X + .2*randn(size(X)) ;
X = X(:) ;
Y = Y(:) ;

save('ass2p1q1','X','Y') ;

uiregress ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 2
% Question 1
% Author: Henri DE PLAEN
% Date: 6/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
gam = 1e+4  ;
sig2 = .1  ;
```

```

X = (-10:0.1:10)' ;
Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1) ;

Xtrain = X(1:2:length(X)) ;
Ytrain = Y(1:2:length(Y)) ;
Xtest = X(2:2:length(X)) ;
Ytest = Y(2:2:length(Y)) ;

[alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}) ;

figure(1)  ;
plotlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'},{alpha,b}) ;
title('Training set results')  ;
legend('Regression','Training points') ;

YtestEst = ...
    simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'},{alpha,b},Xtest) ;

figure(2)  ; hold on ;
plot(Xtest,YtestEst,'-k') ;
plot(Xtest,Ytest,'.k') ;
legend('Estimation','Test points') ;
title('Test set results') ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
for idx = 1:2
    figure(idx)  ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    ax.XAxisLocation = 'origin' ;
    ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.2)  ;
    set(lin,'LineWidth',1.5)  ;
    set(lin,'Color','k')  ;
    set(lin,'MarkerSize', 15)  ;
    %set(lin,'MarkerFaceColor','k')  ;
    %set(gca,'XTickLabel',[])  ; set(gca,'YTickLabel',[])
    xlabel('x')  ; ylabel('y')  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 3
% Question 1
% Author: Henri DE PLAEN
% Date: 6/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
```

```
optFun = 'gridsearch';
globalOptFun = 'csa';

X = (-10:0.1:10)' ;
Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1) ;

Xtrain = X(1:2:length(X)) ;
Ytrain = Y(1:2:length(Y)) ;
Xtest = X(2:2:length(X)) ;
Ytest = Y(2:2:length(Y)) ;

[gam,sig2,cost] = tunelssvm({X,Y,'f',[],[],'RBF_kernel', ...
    globalOptFun},optFun,'crossvalidatelssvm',{10,'mse'}) ;

[alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'}) ;

figure(1)  ;
plotlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'},{alpha,b}) ;
title('Training set results')  ;
legend('Regression','Training points') ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
for idx = 1:2
    figure(idx)  ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    ax.XAxisLocation = 'origin' ;
    ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.2)  ;
    set(lin,'LineWidth',1.5)  ;
    set(lin,'Color','k')  ;
    set(lin,'MarkerSize', 15)  ;
    %set(lin,'MarkerFaceColor','k')  ;
    %set(gca,'XTickLabel',[])  ; set(gca,'YTickLabel',[])
    xlabel('x')  ; ylabel('y')  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 3
% Question 2
% Author: Henri DE PLAEN
% Date: 6/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
optFun = 'gridsearch';
globalOptFun = 'ds';
n = 1e+3 ;
m = 5e+1 ;
```

```

Time = zeros(1,m) ;
Gam = zeros(1,m) ;
Sig2 = zeros(1,m) ;
Cost = zeros(1,m) ;

hw = waitbar(0) ;

for idx = 1:m
    X = linspace(-10,10,n)' ;
    Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1) ;

    tic ;
    [gam,sig2,cost] = tunelssvm({X,Y,'f',[],[],'RBF_kernel', ...
        globalOptFun},optFun,'crossvalidatelssvm',{10,'mse'}) ;
    Time(idx) = toc ;
    Gam(idx) = gam ;
    Sig2(idx) = sig2 ;
    Cost(idx) = cost ;
    waitbar(idx/m,hw) ;
end

delete(hw) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% DISP RESULTS
clc ;

disp('time') ;
disp(mean(Time)) ;
disp(std(Time)) ;

disp('gamma') ;
disp(mean(Gam)) ;
disp(std(Gam)) ;

disp('sig2') ;
disp(mean(Sig2)) ;
disp(std(Sig2)) ;

disp('cost') ;
disp(mean(Cost)) ;
disp(std(Cost)) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 4
% Question 1
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
gam = 1e+4 ;
sig2 = .15 ;
```

```
18
19 n = 100 ;
20 gam_span = 10.^linspace(0,8,n) ;
21 sig2_span = linspace(0,.6,n) ;
22
23 % data-sets
24 X = (-10:0.1:10)' ;
25 Y = cos(X) + cos(2*X) + 0.1.*randn(length(X),1) ;
26
27 Xtrain = X(1:2:length(X)) ;
28 Ytrain = Y(1:2:length(Y)) ;
29 Xtest = X(2:2:length(X)) ;
30 Ytest = Y(2:2:length(Y)) ;
31
32 %prealloc
33 gam_post = zeros(size(gam_span)) ;
34 sig2_post = zeros(size(sig2_span)) ;
35
36 % Bayesian framework
37 % L1
38 criterion_L1 = bay_lssvm({Xtrain,Ytrain,'f',gam,sig2},1) ;
39
40 % L2
41 for idx=1:length(gam_post)
42     gam_post(idx) = bay_lssvm({Xtrain,Ytrain,'f',gam_span(idx),sig2},2) ;
43 end
44
45 [~,idx_gam] = min(gam_post) ;
46 gam = gam_span(idx_gam) ;
47
48 figure(1) ;
49 semilogx(gam_span,gam_post,'-k') ;
50 xlabel('\gamma') ;
51 ylabel('-log(P(\gamma|D,H))') ;
52
53 % L3
54 for idx = 1:length(sig2_span)
55     sig2_post(idx) = bay_lssvm({Xtrain,Ytrain,'f',gam,sig2_span(idx)},3) ;
56 end
57
58 [~,sig2_idx] = min(sig2_post) ;
59 sig2 = sig2_span(sig2_idx) ;
60
61 figure(2) ;
62 plot(sig2_span,sig2_post,'-k') ;
63 xlabel('\sigma^2') ;
64 ylabel('-log(P(\sigma^2|D,H))') ;
65
66
67 % BAY OPTIMISE
68 % params
69 gam = 1e+4 ;
70 sig2 = .15 ;
71
72 [~,alpha,b] = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},1) ;
73 [~,gam]     = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},2) ;
74 [~,sig2]    = bay_optimize({Xtrain,Ytrain,'f',gam,sig2},3) ;
75
76 sig2e = bay_errorbar({Xtrain,Ytrain,'f',gam,sig2},'figure');
77
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:3
    figure(idx) ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    %ax.XAxisLocation = 'origin' ;
    %ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.2)  ;
    set(lin,'LineWidth',1.5)  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 4
% Question 2
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

load iris ;

% params
gam = 1 ;
sig2 = 0.75 ;

% Bayesian
bay_modoutClass({X,Y,'c',gam,sig2},'figure') ;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
ax = gca  ;
lin = findobj(gca, 'Type', 'Line')  ;
ax.XAxisLocation = 'origin' ;
ax.YAxisLocation = 'origin' ;
set(0,'DefaultLineColor','k') ;
set(gca,'box','off')  ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18)  ;
set(gca,'LineWidth',1.2)  ;
set(lin,'LineWidth',1.5)  ;
caxis([0 1]) ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 4
% Question 3
% Author: Henri DE PLAEN
% Date: 7/8/2018

```

```
clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
type = {'-k',':r',':b'} ;
sets = {1,2,3,1:3,1:2,2:3,[1 3]} ;

% params
gam = 1e+3 ;
sig2 = 0.1 ;

% ARD
X = 10.*rand(100,3)-3;
Y = cos(X(:,1)) + cos(2*(X(:,1))) + 0.3.*randn(100,1) ;

[selected, ranking] = bay_lssvmARD({X,Y,'class',gam,sig2},'discrete','svd') ;

for idx=1:length(sets)
    %cost(idx) = bay_lssvm({X(:,sets{idx}),Y,'f',gam,sig2},3) ;

    Xbis = X(:,sets{idx}) ; Xbis = Xbis(:) ;
    Ybis = repmat(Y,length(sets{idx})) ;
    [Xbis,s_idx] = sort(Xbis) ;
    Ybis = Ybis(s_idx) ;
    cost(idx) = crossvalidate({Xbis,Ybis,'f',gam,sig2,'RBF_kernel'}, ...
        10,'mse','mean');
end

figure(1) ;
for idx = 1:3
    [~,s_idx] = sort(X(:,idx)) ;
    hold on ;
    X(s_idx,idx)
    plot(X(s_idx,idx),Y(s_idx,1),type{idx}) ;
end

xlabel('X') ; ylabel('Y') ;
legend({'X_1','X_2','X_3'}) ;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:1
    figure(idx) ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    ax.XAxisLocation = 'origin' ;
    ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.2)  ;
    set(lin,'LineWidth',1.5)  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
```

```
% Part 5
% Question 1
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
gam = 100 ;
sig2 = 0.1 ;

% exec
X = (-10:0.2:10)';
Y = cos(X) + cos(2*X) + 0.1.*rand(size(X));

out1 = [15 17 19];
Y(out1) = 0.7+0.3*rand(size(out1));
out2 = [41 44 46];
Y(out2) = 1.5+0.2*rand(size(out2));

% non robust
model = initlssvm(X,Y,'f',[],[],'RBF_kernel');
costFun = 'crossvalidatelssvm';
[gam,sig2,cost] = tunelssvm(model,'simplex',costFun,{10,'mse'});
[alpha,b] = trainlssvm({X,Y,'f',gam,sig2,'RBF_kernel'}) ;

figure(1) ;
plotlssvm({X,Y,'f',gam,sig2,'RBF_kernel'},{alpha,b});

hold on ;
marker_values = ...
    round((abs(alpha)-min(abs(alpha)))/(max(abs(alpha))-min(abs(alpha)))*200)+1 ...
    ;
scatter(X,Y,marker_values+1,'filled','k') ;
scatter(X([out1 out2]),Y([out1 out2]),marker_values([out1 ...
    out2])+10,'filled','r') ;

hold on ;
plot(X,cos(X) + cos(2*X),':k') ;

figure(2) ;
b = bar(abs(alpha),'FaceColor','flat') ;
b.CData(:,:) = repmat([0 0 0],length(X),1) ;
b.CData([out1 out2],:) = repmat([1 0 0],6,1) ;
xlabel('k') ; ylabel('|\alpha_k|') ;

%Questions
% What is the influence of the outlying data points?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:2
    figure(idx) ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    ax.XAxisLocation = 'origin' ;
```

```
    ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.7)  ;
    set(lin,'Color','k') ;
    %set(lin,'MarkerSize','k') ;
    set(lin,'LineWidth',1.5)  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Part 5
% Question 1
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all ; close all ; clc ;
addpath(genpath('../docs')) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

% params
gam = 100 ;
sig2 = 0.1 ;

% exec
X = (-10:0.2:10)';
Y = cos(X) + cos(2*X) + 0.1.*rand(size(X));

out1 = [15 17 19];
Y(out1) = 0.7+0.3*rand(size(out1));
out2 = [41 44 46];
Y(out2) = 1.5+0.2*rand(size(out2));

% non robust
model = initlssvm(X,Y,'f',[],[],'RBF_kernel');
costFun = 'crossvalidatelssvm';
[gam,sig2,cost] = tunelssvm(model,'simplex',costFun,{10,'mse'});
[alpha,b] = trainlssvm({X,Y,'f',gam,sig2,'RBF_kernel'}) ;

figure(1) ;
plotlssvm({X,Y,'f',gam,sig2,'RBF_kernel'},{alpha,b});

hold on ;
marker_values = ...
    round((abs(alpha)-min(abs(alpha)))/(max(abs(alpha))-min(abs(alpha)))*200)+1 ...
    ;
scatter(X,Y,marker_values+1,'filled','k') ;
scatter(X([out1 out2]),Y([out1 out2]),marker_values([out1 ...
    out2])+10,'filled','r') ;

hold on ;
plot(X,cos(X) + cos(2*X),':k') ;

figure(2) ;
b = bar(abs(alpha),'FaceColor','flat') ;
b.CData(:,:) = repmat([0 0 0],length(X),1) ;
```

```
b.CData([out1 out2],:) = repmat([1 0 0],6,1) ;
xlabel('k') ; ylabel('|\alpha_k|') ;

%Questions
% What is the influence of the outlying data points?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:2
    figure(idx) ;
    ax = gca  ;
    lin = findobj(gca, 'Type', 'Line')  ;
    ax.XAxisLocation = 'origin' ;
    ax.YAxisLocation = 'origin' ;
    set(0,'DefaultLineColor','k') ;
    set(gca,'box','off')  ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)  ;
    set(gca,'LineWidth',1.7)  ;
    set(lin,'Color','k') ;
    %set(lin,'MarkerSize','k') ;
    set(lin,'LineWidth',1.5)  ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Homework 2
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs2'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

load logmap ;

% prealloc
orders = 2:1:25 ;
mse_score = zeros(size(orders)) ;
gams = zeros(size(orders)) ;
sig2s = zeros(size(orders)) ;

n_expe = 10 ;
mse_score_loc = zeros(1,n_expe) ;
gam_loc = zeros(1,n_expe) ;
sig2_loc = zeros(1,n_expe) ;

hw = waitbar(0) ;

for idx = 1:length(orders)
    waitbar(idx/length(orders),hw) ;

    order  = orders(idx);
    W = windowize(Z,1:order+1);
    X = W(:,1:order);
    Y = W(:,end);


```

```
38      gam = 10 ;
39      sig2 = 10 ;
40
41      for idx2 = 1:n_expe
42          [gam,sig2] = ...
                tunelssvm({X,Y,'f',gam,sig2,'RBF_kernel'},'gridsearch','crossvalidatelssvm',{10,'ms
                ;
43          model = {X,Y,'f',gam,sig2,'RBF_kernel'} ;
44          [alpha,b] = trainlssvm(model);
45
46          Xs = Z(end-order+1:end,1);
47          prediction = predict(model,Xs,50);
48
49          gam_loc(idx2) = gam ;
50          sig2_loc(idx2) = sig2 ;
51
52          mse_score_loc(idx2) = sum((prediction-Ztest).^2)/length(prediction) ;
53      end
54
55      mse_score(idx) = mean(mse_score_loc) ;
56      gams(idx) = mean(gam_loc) ;
57      sig2s(idx) = mean(sig2_loc) ;
58  end
59
60  delete(hw) ;
61
62  figure(1) ;
63  plot(orders,mse_score,'-k') ;
64  xlabel('order') ; ylabel('Test MSE') ;
65
66  figure(2) ;
67  [hAx,hL1,hL2] = plotyy(orders,gams,orders,sig2s,@semilogy,@semilogy);
68  xlabel('order') ;
69  yyaxis left
70  ylabel('\gamma') ;
71  yyaxis right
72  ylabel('\sigma^2') ;
73  set(hL1,'LineWidth',1.7) ;
74  set(hL2,'LineWidth',1.7) ;
75  set(findall(0,'YAxisLocation','right'),'Yscale','log');
76  set(findall(0,'YAxisLocation','left'),'Yscale','log');
77  set(findall(0,'Type','text'), 'FontName', 'Baskervald ADF Std') ;
78
79  %Questions ?
80  % What are the observed results?
81  % Try to change the order of the model and evaluate the mse (mean square ...
        error) on the test set.
82  % How does the order of the model affect the obtained performance?
83
84  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
85  %% PLOT PARAMS
86  for idx = 1:2
87      figure(idx)  ;
88      ax = gca   ;
89      lin = findobj(gca, 'Type', 'Line')   ;
90      %ax.XAxisLocation = 'origin'  ;
91      %ax.YAxisLocation = 'origin'  ;
92      set(0,'DefaultLineColor','k')  ;
93      set(gca,'box','off')   ;
94      set(gca, 'FontName', 'Baskervald ADF Std') ;
```

```
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    set(lin,'LineWidth',1.5)   ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Homework 2
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs2'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

load santafe ;


order = 40;
gam = 2e+5 ;
sig2 = .5;
prop_val = .2 ;

X = windowize(Z,1:order+1);

% data-sets
size_train = round(size(X,1)*(1-prop_val)) ;
idx = randperm(size(X,1),size_train) ;
Xtrain = X(idx,1:order) ;
Ytrain = X(idx,end) ;

Xval = X ;
Xval(idx,:) = [] ;
Xval = Xval(:,1:order) ;
Yval = Xval(:,end) ;

%[gam,sig2] = ...
    tunelssvm({X(:,1:order),X(:,end),'f',gam,sig2,'RBF_kernel'},'simplex','crossvalidatelssvm',

gam = 2e+5 ;
sig2 = 12 ;

[alpha,b] = trainlssvm({X(:,1:order),X(:,end),'f',gam,sig2,'RBF_kernel'});

Xs = Z(end-order+1:end,1);
prediction = ...
    predict({X(:,1:order),X(:,end),'f',gam,sig2,'RBF_kernel'},Xs,length(Ztest));
plot([prediction Ztest]);
legend('prediction','reference') ;

MSE(prediction,Ztest)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:1
    figure(idx)  ;
    ax = gca   ;
    lin = findobj(gca, 'Type', 'Line')   ;
```

```
    %ax.XAxisLocation = 'origin'  ;
    %ax.YAxisLocation = 'origin'  ;
    set(0,'DefaultLineColor','k')  ;
    set(gca,'box','off')   ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    set(lin,'LineWidth',1.5)   ;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% USEFULL FUNCTIONS
function mse_score = MSE(x,y)

assert(length(x)==length(y)) ;
mse_score = sum((x-y).^2)/length(y) ;

end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 2
% Homework 2.5
% Author: Henri DE PLAEN
% Date: 7/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs2'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

load santafe ;


orders = 10:1:60;
gam = 2e+5 ;
sig2 = 2 ;
prop_val = .3 ;

performance1 = zeros(size(orders)) ;
performance2 = zeros(size(orders)) ;

hw = waitbar(0) ;

for idx = 1:length(orders)
    loc_order = orders(idx) ;
    X = windowize(Z,1:loc_order+1);

    size_train = round(size(X,1)*(1-prop_val)) ;
    idx_perm = randperm(size(X,1),size_train) ;
    Xtrain = X(idx_perm,1:loc_order) ;
    Ytrain = X(idx_perm,end) ;

    Xval = X ;
    Xval(idx_perm,:) = [] ;
    Xval = Xval(:,1:loc_order) ;
    Yval = Xval(:,end) ;

    waitbar(idx/length(orders),hw) ;
```

```
42
43      %[gam,sig2] = ...
            tunelssvm({X(:,1:loc_order),X(:,end),'f',gam,sig2,'RBF_kernel'},'simplex','crossvalidat
44
45      [alpha,b] = trainlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'});
46      pred = zeros(size(Xval,1),1) ;
47
48      for idx2 = 1:size(Xval,1)
49          pred(idx2) = ...
                simlssvm({Xtrain,Ytrain,'f',gam,sig2,'RBF_kernel'},{alpha,b},Xval(idx2,:)) ...
                ;
50      end
51
52      performance1(idx) = MSE(pred,Yval) ;
53      performance2(idx) = ...
            crossvalidate({X(:,1:loc_order),X(:,end),'f',gam,sig2,'RBF_kernel'},20,'mse','mean') ..
            ;
54  end
55  delete(hw) ;
56
57  figure(1) ;
58  plotyy(orders,performance1,orders,performance2) ;
59  xlabel('order') ;
60  yyaxis left
61  ylabel('Validation MSE') ;
62  yyaxis right
63  ylabel('Cross-validation performance') ;
64
65  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
66  %% PLOT PARAMS
67  for idx = 1:1
68      figure(idx)  ;
69      ax = gca   ;
70      lin = findobj(gca, 'Type', 'Line')   ;
71      %ax.XAxisLocation = 'origin'  ;
72      %ax.YAxisLocation = 'origin'  ;
73      set(0,'DefaultLineColor','k')  ;
74      set(gca,'box','off')   ;
75      set(gca, 'FontName', 'Baskervald ADF Std')
76      set(gca, 'FontSize', 18)   ;
77      set(gca,'LineWidth',1.2)   ;
78      set(lin,'LineWidth',1.5)   ;
79  end
80
81
82  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
83  %% USEFULL FUNCTIONS
84  function mse_score = MSE(x,y)
85
86  assert(length(x)==length(y)) ;
87  mse_score = sum((x-y).^2)/length(y) ;
88
89  end
```

A.3 Unsupervised learning

```
1  % SUPPORT VECTOR MACHINES
2  % KULeuven
3  % Exercise Session 3
4  % Part 1
```

```
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
for idx=3
    nb = 400 ;
    sig = 0.5 ;
    nc = idx ;
    sig2 = 0.8 ;
    approx = 1 ;
    kpca_script_bis ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT PARAMS
for idx = 1:2
    figure(idx)  ;
    ax = gca   ;
    lin = findobj(gca, 'Type', 'Line')   ;
    %ax.XAxisLocation = 'origin'  ;
    %ax.YAxisLocation = 'origin'  ;
    set(0,'DefaultLineColor','k')  ;
    set(gca,'box','off')   ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    set(lin,'LineWidth',1.5)   ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Part 2
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

digitsdn ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Part 3
% Author: Henri DE PLAEN
% Date: 14/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% ASSIGNMENT
load two3drings;        % load the toy example

[N,d]=size(X);

perm=randperm(N);   % shuffle the data
X=X(perm,:);

sig2=0.2;              % set the kernel parameters


K=kernel_matrix(X,'RBF_kernel',sig2);   %compute the RBF kernel (affinity) ...
    matrix

D=diag(sum(K));         % compute the degree matrix (sum of the columns of K)

[U,lambda]=eigs(inv(D)*K,3);  % Compute the 3 largest eigenvalues/vectors ...
    using Lanczos
                              % The largest eigenvector does not contain
                              % clustering information. For binary clustering,
                              % the solution is the second largest eigenvector.

clust=sign(U(:,2)); % Threshold the eigenvector solution to obtain binary ...
    cluster indicators

[y,order]=sort(clust,'descend');    % Sort the data using the cluster ...
    information
Xsorted=X(order,:);

Ksorted=kernel_matrix(Xsorted,'RBF_kernel',sig2);   % Compute the kernel ...
    matrix of the
                                                    % sorted data.


proj=K*U(:,2:3);    % Compute the projections onto the subspace spanned by ...
    the second,
                    % and third largest eigenvectors.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOTS
figure(1) ;
scatter3(X(:,1),X(:,2),X(:,3),30,clust);

figure(2) ;
imshow(1-Ksorted);
colormap('gray') ;

for idx_fig = 1:2
    figure(idx_fig)  ;
    ax = gca   ;
    lin = findobj(gca, 'Type', 'Line')   ;
    %ax.XAxisLocation = 'origin'  ;
    %ax.YAxisLocation = 'origin'  ;
    set(0,'DefaultLineColor','k')  ;
    set(gca,'box','off')   ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    set(lin,'LineWidth',1.5)   ;
```

```
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Part 4
% Author: Henri DE PLAEN
% Date: 15/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
rng(0681349) ; % reproducability
X = 3.*randn(100,2);

sig2_span = 10.^linspace(-2,2,70) ;
en = zeros(size(sig2_span)) ;
en1 = zeros(size(sig2_span)) ;
en2 = zeros(size(sig2_span)) ;

hw = waitbar(0) ;
for idx=1:length(sig2_span)
    ssize = 10;
    sig2 = sig2_span(idx) ;
    subset = zeros(ssize,2);

    idx_perm = randperm(size(X,1)) ;
    X = X(idx_perm,:) ;
    for t = 1:400

        %
        % new candidate subset
        %
        r = ceil(rand*ssize);
        candidate = [subset([1:r-1 r+1:end],:); X(mod(t,100)+1,:)]; % add ...
            point X(t,:) to subset in random place r

        %
        % is this candidate better than the previous?
        %
        if kentropy(candidate, 'RBF_kernel',sig2) > kentropy(subset, ...
            'RBF_kernel',sig2)
            subset = candidate;
        end
        if t==10
            en1(idx) = kentropy(subset, 'RBF_kernel',sig2) ;
        elseif t==50
            en2(idx) = kentropy(subset, 'RBF_kernel',sig2) ;
        end
    end

    en(idx) = kentropy(subset, 'RBF_kernel',sig2) ;
   waitbar(idx/length(sig2_span),hw) ;
end
delete(hw) ;

figure(1)  ;
semilogx(sig2_span,en,'-k','LineWidth',1.7) ; hold on ;
```

```
semilogx(sig2_span,en1,':k','LineWidth',1.7) ;
semilogx(sig2_span,en2,'--k','LineWidth',1.7) ;
xlabel('\sigma^2') ; ylabel('Entropy') ;
ax = gca   ;
lin = findobj(gca, 'Type', 'Line')   ;
%ax.XAxisLocation = 'origin'  ;
%ax.YAxisLocation = 'origin'  ;
set(0,'DefaultLineColor','k')  ;
set(gca,'box','off')   ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18)   ;
set(gca,'LineWidth',1.2)   ;
%set(lin,'LineWidth',1)   ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Part 4 bis
% Author: Henri DE PLAEN
% Date: 15/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
rng(0681349) ; % reproducability
clear
close all

X = 3.*randn(100,2);
ssize = 10;
sig2 = 50;
subset = zeros(ssize,2);
for t = 1:100

    %
    % new candidate subset
    %
    r = ceil(rand*ssize);
    candidate = [subset([1:r-1 r+1:end],:); X(t,:)]; % add point X(t,:) to ...
        subset in random place r

    %
    % is this candidate better than the previous?
    %
    if kentropy(candidate, 'RBF_kernel',sig2) > kentropy(subset, ...
        'RBF_kernel',sig2)
        subset = candidate;
    end
end
%
% make a figure
%
plot(X(:,1),X(:,2),'b*'); hold on;
plot(subset(:,1),subset(:,2),'ro','linewidth',6); hold off;

figure(1)  ;
ax = gca   ;
lin = findobj(gca, 'Type', 'Line')   ;
```

```
set(0,'DefaultLineColor','k')  ;
set(gca,'box','off')   ;
set(gca, 'FontName', 'Baskervald ADF Std')
set(gca, 'FontSize', 18)   ;
set(gca,'LineWidth',1.2)   ;
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Part 4 tris
% Author: Henri DE PLAEN
% Date: 15/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
rng(0681349,'twister') ; % reproducability
clear
close all

X = 3.*randn(100,2);
ssize = 3;
sig2 = 5;
subset = zeros(ssize,2);
subset_indices = ones(ssize,1);
for t = 1:300

    %
    % new candidate subset
    %
    r = ceil(rand*ssize);
    candidate = [subset([1:r-1 r+1:end],:); X(mod(t,100)+1,:)] ; % add point ...
        X(t,:) to subset in random place r
    ind_candidate = [subset_indices([1:r-1 r+1:end],:); mod(t,100)+1] ;

    %
    % is this candidate better than the previous?
    %
    if kentropy(candidate, 'RBF_kernel',sig2) > kentropy(subset, ...
        'RBF_kernel',sig2)
        subset = candidate;
        subset_indices = ind_candidate ;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOT
%
% make a figure
%
figure(1) ;
plot(X(:,1),             X(:,2),'b*'); hold on;
plot(X(subset_indices,1),X(subset_indices,2),'ro','linewidth',6); hold off;
title('original space')

%
% transform the data in feature space
%
```

```
figure(2) ;
features = AFEm(X(subset_indices,:),'RBF_kernel',sig2,X);
plot3(features(:,1),             features(:,2),             ...
    features(:,3),'k*'); hold on;
plot3(features(subset_indices,1),features(subset_indices,2),features(subset_indices,3),'ro','l
    hold off;
title('feature space')

for idx_fig = 1:2
    figure(idx_fig)  ;
    ax = gca   ;
    lin = findobj(gca, 'Type', 'Line')   ;
    set(0,'DefaultLineColor','k')  ;
    set(gca,'box','off')   ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    xl = xlim() ;
    yl = ylim() ;
    zl = zlim() ;
    hold on;
    line(2*xl, [0,0], [0,0], 'LineWidth', 2, 'Color', 'k');
    line([0,0], 2*yl, [0,0], 'LineWidth', 2, 'Color', 'k');
    line([0,0], [0,0], 2*zl, 'LineWidth', 2, 'Color', 'k');
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 1
% Question 1
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PRELIMINARIES
load digits;
clear size ;

[N, dim]=size(X);
Ntest=size(Xtest1,1);
minx=min(min(X));
maxx=max(max(X));

noisefactor =0.3;
noise = noisefactor*maxx; % sd for Gaussian noise

Xn = X;
for i=1:N
    randn('state', i);
    Xn(i,:) = X(i,:) + noise*randn(1, dim);
end

Xnt = Xtest1;
for i=1:size(Xtest1,1)
    randn('state', N+i);
    Xnt(i,:) = Xtest1(i,:) + noise*randn(1,dim);
```

```
end

Xtr = X(1:1:end,:);

n = 6 ;
sig2 = dim*mean(var(Xtr));
sigmafactor = 10.^linspace(-2,2,n);
sig2_span=sig2.*sigmafactor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

for idx_sig2 = 1:length(sig2_span)
    sig2 = sig2_span(idx_sig2) ;

    disp('Kernel PCA: extract the principal eigenvectors in feature space');
    disp(['sig2 = ', num2str(sig2)]);

    [lam,U] = kpca(Xtr,'RBF_kernel',sig2,[],'eig',240);
    [lam, ids]=sort(-lam); lam = -lam; U=U(:,ids);

    % KPCA
    disp(' ');
    disp(' Denoise using the first PCs');
    % choose the digits for test
    digs=[0:9]; ndig=length(digs);
    m=2; % Choose the mth data for each digit

    Xdt=zeros(ndig,dim);

    % which number of eigenvalues of kpca
    npcs = [2.^(0:7) 190];
    lpcs = length(npcs);

    % FIGURE
    figure(idx_sig2);
    colormap('gray');
    title('Denosing using kernel PCA');

    for k=1:lpcs
        nb_pcs=npcs(k);
        disp(['nb_pcs = ', num2str(nb_pcs)]);
        Ud=U(:,(1:nb_pcs)); lamd=lam(1:nb_pcs);

        for i=1:ndig
            dig=digs(i);
            fprintf('digit %d : ', dig)
            xt=Xnt(i,:);
            if k==1
                % plot the original clean digits
                %
                subplot(2+lpcs, ndig, i);
                pcolor(1:15,16:-1:1,reshape(Xtest1(i,:), 15, 16)'); shading ...
                    interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                %set(gca, 'FontSize', 18)   ;

                if i==1, ylabel('original'), end

```

```
                % plot the noisy digits
                %
                subplot(2+lpcs, ndig, i+ndig);
                pcolor(1:15,16:-1:1,reshape(xt, 15, 16)'); shading interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                %set(gca, 'FontSize', 18)   ;
                if i==1, ylabel('noisy'), end
                drawnow
            end
            Xdt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
            subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
            pcolor(1:15,16:-1:1,reshape(Xdt(i,:), 15, 16)'); shading interp;
            set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
            set(gca, 'FontName', 'Baskervald ADF Std')
            %set(gca, 'FontSize', 18)   ;
            if i==1, ylabel(['n=',num2str(nb_pcs)]); end
            drawnow
        end % for i
    end % for k
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 1
% Question 2
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PRELIMINARIES
load digits;
clear size ;

[N, dim]=size(X);
Ntest=size(Xtest1,1);
minx=min(min(X));
maxx=max(max(X));

noisefactor =0.3;
noise = noisefactor*maxx; % sd for Gaussian noise

Xn = X;
for i=1:N
    randn('state', i);
    Xn(i,:) = X(i,:) + noise*randn(1, dim);
end

Xnt = Xtest1;
for i=1:size(Xtest1,1)
    randn('state', N+i);
    Xnt(i,:) = Xtest1(i,:) + noise*randn(1,dim);
end

Xtr = X(1:1:end,:);

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
sig2_span(1) = dim*mean(var(Xtr)); %%%%
sig2_span(2) = 400 ;               %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


sigmafactor = 0.7;
sig2_span=sig2_span.*sigmafactor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT

for idx_sig2 = 1:length(sig2_span)
    sig2 = sig2_span(idx_sig2) ;

    disp('Kernel PCA: extract the principal eigenvectors in feature space');
    disp(['sig2 = ', num2str(sig2)]);

    [lam,U] = kpca(Xtr,'RBF_kernel',sig2,[],'eig',240);
    [lam, ids]=sort(-lam); lam = -lam; U=U(:,ids);

    % KPCA
    disp(' ');
    disp(' Denoise using the first PCs');
    % choose the digits for test
    digs=[0:9]; ndig=length(digs);
    m=2; % Choose the mth data for each digit

    Xdt=zeros(ndig,dim);

    % which number of eigenvalues of kpca
    npcs = [2.^(0:7) 190];
    lpcs = length(npcs);

    % FIGURE
    figure(idx_sig2);
    colormap('gray');
    title('Denosing using kernel PCA');

    for k=1:lpcs
        nb_pcs=npcs(k);
        disp(['nb_pcs = ', num2str(nb_pcs)]);
        Ud=U(:,(1:nb_pcs)); lamd=lam(1:nb_pcs);

        for i=1:ndig
            dig=digs(i);
            fprintf('digit %d : ', dig)
            xt=Xnt(i,:);
            if k==1
                % plot the original clean digits
                %
                subplot(2+lpcs, ndig, i);
                pcolor(1:15,16:-1:1,reshape(Xtest1(i,:), 15, 16)'); shading ...
                    interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                %set(gca, 'FontSize', 18)   ;

                if i==1, ylabel('original'), end

```

```
                % plot the noisy digits
                %
                subplot(2+lpcs, ndig, i+ndig);
                pcolor(1:15,16:-1:1,reshape(xt, 15, 16)'); shading interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                %set(gca, 'FontSize', 18)   ;
                if i==1, ylabel('noisy'), end
                drawnow
            end
            Xdt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
            subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
            pcolor(1:15,16:-1:1,reshape(Xdt(i,:), 15, 16)'); shading interp;
            set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
            set(gca, 'FontName', 'Baskervald ADF Std')
            %set(gca, 'FontSize', 18)   ;
            if i==1, ylabel(['n=',num2str(nb_pcs)]); end
            drawnow
        end % for i
    end % for k
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 1
% Question 3
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PRELIMINARIES
load digits ;
clear size ;

[N, dim]=size(X);
Ntest=size(Xtest1,1);
minx=min(min(X));
maxx=max(max(X));

noisefactor = 1.0;
noise = noisefactor*maxx; % sd for Gaussian noise

Xn = X;
for i=1:N;
    randn('state', i);
    Xn(i,:) = X(i,:) + noise*randn(1, dim);
end

Xnt = Xtest1;
for i=1:size(Xtest1,1)
    randn('state', N+i);
    Xnt(i,:) = Xtest1(i,:) + noise*randn(1,dim);
end

Xtr = X(1:1:end,:);

```

```
sig2 =dim*mean(var(Xtr)); % rule of thumb
sigmafactor = 0.7;
sig2=sig2*sigmafactor;

nb_pcs_k = 20 ;
nb_pcs_l = 20 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
% linear PCA
[lam_lin,U_lin] = pca(Xtr);

% kernel PCA
[lam,U] = kpca(Xtr,'RBF_kernel',sig2,[],'eig',240) ;
[lam, ids] = sort(-lam) ;
lam = -lam ;
U = U(:,ids) ;

digs = 0:9 ;
ndig = length(digs) ;
m = 2 ;

% prealloc
Xdt = zeros(ndig,dim) ;
Xdt_lin = zeros(ndig,dim) ;

figure;
colormap('gray');

Ud=U(:,(1:nb_pcs_k)); lamd=lam(1:nb_pcs_k);
Ud_lin=U_lin(:,(1:nb_pcs_l)); lamd_lin=lam_lin(1:nb_pcs_l);

lpcs = 3 ;
for k=1:lpcs
    for i=1:ndig
        dig=digs(i);
        fprintf('digit %d : ', dig)
        xt=Xnt(i,:);
        proj_lin=xt*Ud_lin; % projections of linear PCA
        switch k
            case 1
                % plot the original clean digits
                %
                subplot(lpcs+1, ndig, i);
                pcolor(1:15,16:-1:1,reshape(Xtest1(i,:), 15, 16)'); shading ...
                    interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                set(gca, 'FontSize', 18)   ;

                if i==1, ylabel('original'), end

                % plot the noisy digits
                %
                subplot(lpcs+1, ndig, i+ndig);
                pcolor(1:15,16:-1:1,reshape(xt, 15, 16)'); shading interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                set(gca, 'FontSize', 18)   ;
                if i==1, ylabel('noisy'), end
```

```
                drawnow
            case 2
                Xdt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
                subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
                pcolor(1:15,16:-1:1,reshape(Xdt(i,:), 15, 16)'); shading interp;

                if i==1, ylabel('kernel'), end
            case 3
                Xdt_lin(i,:) = proj_lin*Ud_lin';
                subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
                pcolor(1:15,16:-1:1,reshape(Xdt_lin(i,:), 15, 16)'); shading ...
                    interp;

                if i==1, ylabel('lin'), end
        end

        set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
        set(gca, 'FontName', 'Baskervald ADF Std') ;
        set(gca, 'FontSize', 18)   ;
    end % for i
end % for k
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 1
% Question 3
% Author: Henri DE PLAEN
% Date: 9/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PRELIMINARIES
load digits ;
clear size ;

[N, dim]=size(X);
Ntest=size(Xtest1,1);
minx=min(min(X));
maxx=max(max(X));

noisefactor = 1.0;
noise = noisefactor*maxx; % sd for Gaussian noise

Xn = X;
for i=1:N;
    randn('state', i);
    Xn(i,:) = X(i,:) + noise*randn(1, dim);
end

Xnt = Xtest2;
for i=1:size(Xtest2,1)
    randn('state', N+i);
    Xnt(i,:) = Xtest2(i,:) + noise*randn(1,dim);
end

Xtr = X(1:1:end,:);

```

```
sig2 =dim*mean(var(Xtr)); % rule of thumb
sigmafactor = 1;
sig2=sig2*sigmafactor;

nb_pcs_k = 190 ;
nb_pcs_l = 10 ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
% linear PCA
[lam_lin,U_lin] = pca(Xtr);

% kernel PCA
[lam,U] = kpca(Xtr,'RBF_kernel',sig2,[],'eig',240) ;
[lam, ids] = sort(-lam) ;
lam = -lam ;
U = U(:,ids) ;

digs = 0:9 ;
ndig = length(digs) ;
m = 2 ;

% prealloc
Xdt = zeros(ndig,dim) ;
Xdt_lin = zeros(ndig,dim) ;

figure;
colormap('gray');

Ud=U(:,(1:nb_pcs_k)); lamd=lam(1:nb_pcs_k);
Ud_lin=U_lin(:,(1:nb_pcs_l)); lamd_lin=lam_lin(1:nb_pcs_l);

lpcs = 3 ;
for k=1:lpcs
    for i=1:ndig
        dig=digs(i);
        fprintf('digit %d : ', dig)
        xt=Xnt(i,:);
        proj_lin=xt*Ud_lin; % projections of linear PCA
        switch k
            case 1
                % plot the original clean digits
                %
                subplot(lpcs+1, ndig, i);
                pcolor(1:15,16:-1:1,reshape(Xtest2(i,:), 15, 16)'); shading ...
                    interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                set(gca, 'FontSize', 18)   ;

                if i==1, ylabel('original'), end

                % plot the noisy digits
                %
                subplot(lpcs+1, ndig, i+ndig);
                pcolor(1:15,16:-1:1,reshape(xt, 15, 16)'); shading interp;
                set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
                set(gca, 'FontName', 'Baskervald ADF Std')
                set(gca, 'FontSize', 18)   ;
                if i==1, ylabel('noisy'), end
```

```
                drawnow
            case 2
                Xdt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
                subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
                pcolor(1:15,16:-1:1,reshape(Xdt(i,:), 15, 16)'); shading interp;

                if i==1, ylabel('kernel'), end
            case 3
                Xdt_lin(i,:) = proj_lin*Ud_lin';
                subplot(2+lpcs, ndig, i+(2+k-1)*ndig);
                pcolor(1:15,16:-1:1,reshape(Xdt_lin(i,:), 15, 16)'); shading ...
                    interp;

                if i==1, ylabel('lin'), end
        end

        set(gca,'xticklabel',[]);set(gca,'yticklabel',[]);
        set(gca, 'FontName', 'Baskervald ADF Std') ;
        set(gca, 'FontSize', 18)   ;
    end % for i
end % for k
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 1
% Question 3 tris
% Author: Henri DE PLAEN
% Date: 14/8/2018

clear all  ; close all  ; clc  ;
addpath(genpath('../docs_old'))  ;
addpath(genpath('./'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PRELIMINARIES
load digits;
clear size ;

[N, dim]=size(X);
Ntest=size(Xtest1,1);
minx=min(min(X));
maxx=max(max(X));

noisefactor = 1.0;
noise = noisefactor*maxx; % sd for Gaussian noise

Xn = X;
for i=1:N
    randn('state', i);
    Xn(i,:) = X(i,:) + noise*randn(1, dim);
end

Xnt = Xtest1;
for i=1:size(Xtest1,1)
    randn('state', N+i);
    Xnt(i,:) = Xtest1(i,:) + noise*randn(1,dim);
end

Xtr = X(1:1:end,:);

```

```
n = 50 ;
sig2 = dim*mean(var(Xtr));
sigmafactor = 10.^linspace(-2,1,n);
sig2_span=sig2.*sigmafactor;

% which number of eigenvalues of kpca
npcs = [190];
lpcs = length(npcs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
score1 = zeros(length(sig2_span),length(npcs)) ;
score2 = zeros(length(sig2_span),length(npcs)) ;

hw = waitbar(0) ;
for idx_sig2 = 1:length(sig2_span)
    sig2 = sig2_span(idx_sig2) ;

    disp('Kernel PCA: extract the principal eigenvectors in feature space');
    disp(['sig2 = ', num2str(sig2)]);

    [lam,U] = kpca(Xtr,'RBF_kernel',sig2,[],'eig',240);
    [lam, ids]=sort(-lam); lam = -lam; U=U(:,ids);

    ker = kernel_matrix(Xtr,'RBF_kernel',sig2,eye(dim))';

    % KPCA
    disp(' ');
    disp(' Denoise using the first PCs');
    % choose the digits for test
    digs=[0:9]; ndig=length(digs);
    m=2; % Choose the mth data for each digit

    Xdt=zeros(ndig,dim);
    Xtt=zeros(size(X,1),dim);

    for k=1:lpcs
        nb_pcs=npcs(k);
        disp(['nb_pcs = ', num2str(nb_pcs)]);
        Ud=U(:,(1:nb_pcs)); lamd=lam(1:nb_pcs);

        for i=1:ndig
            dig=digs(i);
            fprintf('digit %d : ', dig)
            xt=Xnt(i,:);
            Xdt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
        end % for i
        score1(idx_sig2,k) = ...
            sqrt((sum((sum(Xdt-Xtest1,2).^2)./size(Xtest1,2)).^2)/size(Xtest1,1)) ...
            ;

        for i=1:size(X,1)
            xt=Xn(i,:);
            Xtt(i,:) = preimage_rbf(Xtr,sig2,Ud,xt,'denoise');
        end % for i
        proj_e=U(1:nb_pcs,:)'*ker(:,1:nb_pcs)' ;
        score2(idx_sig2,k) = ...
            sqrt((sum((sum(Xtt-Xtr,2).^2)./size(Xtr,2)).^2)/size(Xtr,1)) ;
    end % for k

```

```
    waitbar(idx_sig2/length(sig2_span),hw) ;
end

delete(hw) ;

[~,idx_min] = min(score1) ;
sig2_opt = sig2_span(idx_min) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% PLOTS
for idx_fig = 1:length(npcs)
    figure(idx_fig)  ;
    loglog(sig2_span,score1(:,idx_fig),'-k',sig2_span,score2(:,idx_fig),'--k') ...
        ;
    %loglog(sig2_span,score1(:,idx_fig),'-k') ;
    xlabel('\sigma^2') ; ylabel('RMSE') ;
    ax = gca   ;
    lin = findobj(gca, 'Type', 'Line')   ;
    %ax.XAxisLocation = 'origin'  ;
    %ax.YAxisLocation = 'origin'  ;
    set(0,'DefaultLineColor','k')  ;
    set(gca,'box','off')   ;
    set(gca, 'FontName', 'Baskervald ADF Std')
    set(gca, 'FontSize', 18)   ;
    set(gca,'LineWidth',1.2)   ;
    set(lin,'LineWidth',1.5)   ;
end
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 2
% Author: Henri DE PLAEN
% Date: 15/8/2018

clear all  ; close all  ; clc  ;
%addpath(genpath('../docs_old'))  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
data = load('shuttle.dat','-ascii');

rng(0681349,'twister') ; % reproducability
idx_tr = randperm(size(data,1),43500) ;
X = data(idx_tr,1:end-1);
Y = data(idx_tr,end);
testX = data(:,1:end-1);
testY = data(:,end);
testX(idx_tr,:) = [] ;
testY(idx_tr) = [] ;

%Parameter for input space selection
%Please type >> help fsoperations; to get more information

k = 15 ;
function_type = 'c';
kernel_type = 'lin_kernel'; % or 'lin_kernel', 'poly_kernel'
global_opt = 'ds'; % 'csa' or 'ds'

%Process to be performed
user_process={'FS-LSSVM', 'SV_L0_norm'};
```

```
window = [15,20,25];

[e,s,t] = ...
    fslssvm(X,Y,k,function_type,kernel_type,global_opt,user_process,window,testX,testY);
```

```
% SUPPORT VECTOR MACHINES
% KULeuven
% Exercise Session 3
% Homework 3
% Author: Henri DE PLAEN
% Date: 15/8/2018

clear all  ; close all  ; clc  ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ASSIGNMENT
data = load('california.dat','-ascii');
tr_set_prop = 4/5 ;

data = normalize(data,1) ;

rng(0681349,'twister') ; % reproducability
idx_tr = randperm(size(data,1),size(data,1)*tr_set_prop) ;
X = data(idx_tr,1:end-1);
Y = data(idx_tr,end);
testX = data(:,1:end-1);
testY = data(:,end);
testX(idx_tr,:) = [] ;
testY(idx_tr) = [] ;

%Parameter for input space selection
%Please type >> help fsoperations; to get more information

k = 1 ;
function_type = 'f';
kernel_type = 'poly_kernel'; % or 'lin_kernel', 'poly_kernel'
global_opt = 'csa'; % 'csa' or 'ds'

%Process to be performed
user_process={'FS-LSSVM', 'SV_L0_norm'};
window = [15,20,25];

[e,s,t] = ...
    fslssvm(X,Y,k,function_type,kernel_type,global_opt,user_process,window,testX,testY);
```