Assignment 1: SVM classification

Daniele Giannuzzi r0876755

1 A simple example: two Gaussians

An artificial dataset is constructed from two classes of Gaussians with the same covariance matrices. In particular we generate two classes samples, each sample composed of two attributes:

- $X_1 = [N(0,1) + 1, N(0,1) + 1]\,, \quad Y_1 = 1$
- $X_2 = [N(0,1) - 1, N(0,1) - 1]\,, \quad Y_2 = -1$

Where X_1 and Y_1 are the samples and labels of the first class and X_2 and Y_2 are referred to the second class. N is the gaussian random noise with mean: $\mu_N = 0$ and standard deviation $\sigma_N = 1$ added to 1 or -1 depending on the class. Therefore we can state that the two classes have mean: $\mu_1 = (1,1)$, $\mu_2 = (-1,-1)$.

Knowing the data distributions, one line to optimally classify those data is exactly the line that passes from the point (0,0) and angle $\theta = -45°$ since the means of the two classes are (1,1) and (-1,-1) and the noise is distributed with the same standard deviation for both the two attributes. We can observe this in Figure 1. This separation line to classify the data is optimal in the sense that we separate the data in order to reduce the misclassification. However, the two classes can not be separated by a linear hyperplane without misclassifying some of the points in the training set.

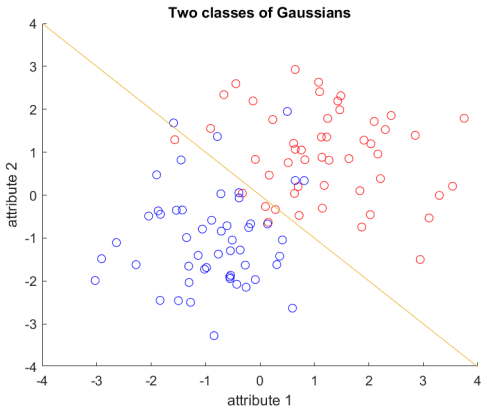


Figure 1: Two classes of Gaussians, separated by a classification line

2 Support vector machine classifier

We now create a dataset on the standford website: $https://cs.stanford.edu/people/karpathy/svmjs/demo/$ by adding more data points to the existing dataset, trying out both the linear and the RBF kernel for classification. We tried two points allocations: the second one differ from the first for the number of points on the other side of the linear hyperplane found with the first allocation attempt. It was done to study in depth how the linear hyperplane varies and how in this situations, when classes are not linear separable with good results, a rbf strategy could be better.

2.1 Linear kernel

Using linear kernel for the two points allocations, shows how linear separation is not always possible, since even using soft margin it could lead to not so good performance. In Figure 2 and 3 we can see the behaviour of SVM with linear kernel for the two points allocations and several values of the C parameter: $\{0.2, 1, 100\}$. The substantial difference when using different values of C is the amount of regularization we want in our classification. Lower values of C, make the SVM using more support vectors, trying to maximize the margin between the support vectors of the two classes allowing for a lower amount of mistakes, while using higher values for C let the

SVM try to misclassifying less. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. If we increase C, a greater penalty is put on violation of the constraint, the solution will change to reduce the size of the violations (and hence the number of violations) so the margin is made narrower, and less patterns will fall inside it, so there are fewer support vectors.

What we can notice in Figure 2 and 3 is that the amount of iterations increase dramatically with increasing values of C, from 43 iterations for the lowest value of C used to 4019 iterations for the highest one. We can also see how adding 3 data points for the red class on the other part of the linear hyperplane found in Figure 2 produce a considerable change in the hyperplane. We can also see how for both the datasets, the higher change in the hyperplane orientation is going from 0.2 value to 1, while using values so much higher than 1, for instance 100 as we used, does not produce in this case so different results.

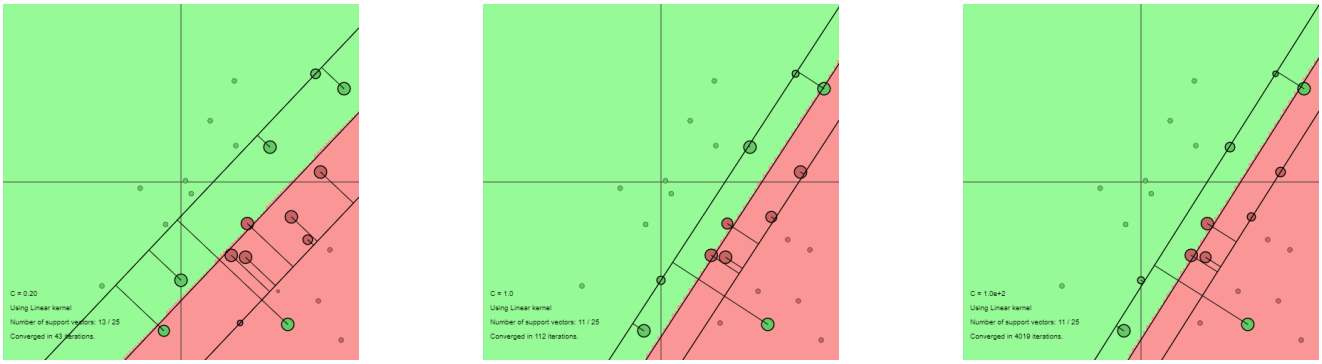


Figure 2: SVM linear kernel with $C \in \{0.2, 1, 100\}$, first points allocation

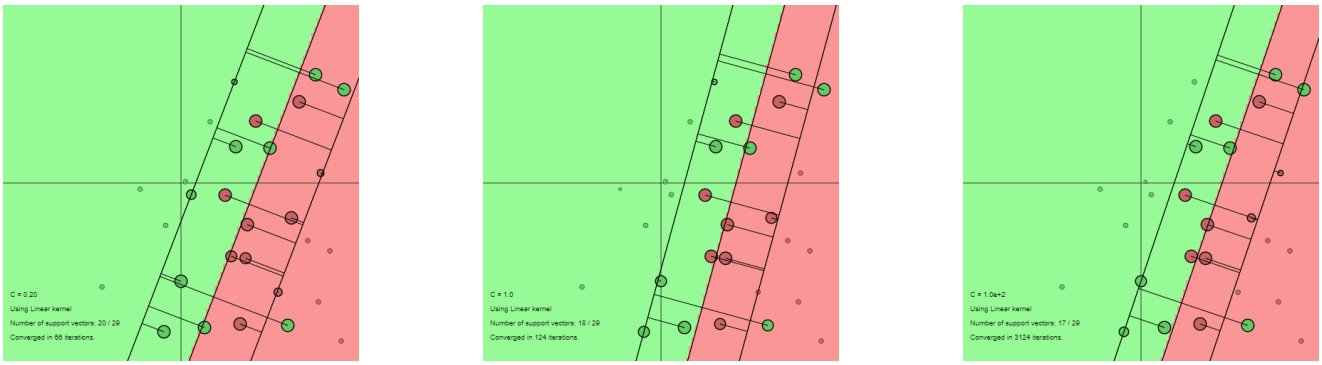


Figure 3: SVM linear kernel with $C \in \{0.2, 1, 100\}$, second points allocation

2.2 RBF kernel

Using the RBF kernel trick for SVM, we do not need the classes to be linearly separable. Indeed, the kernel trick allows for non-linear separation, therefore making the SVM a good classifier in a variety of situations.

Higher values of C parameter, as before, make sure that the SVM classify the training data points correctly. In this tradeoff, we must be careful since with higher values of C, the SVM tends also to overfit the training data, while using lower values we reduce this behaviour. As we can observe from Figure 4 and 5 for the two different datasets, using low values of C (i.e. 0.2) we do not fit the data properly, while increasing it too much (i.e. to 100) even if we reach in the red are also the most difficult points, we do not use more support vectors, and we are probably overfitting. Therefore, a good trade-off when setting the C parameter for the SVM need to be found.

In Figure 6 and 7 we also practiced in changing the values of $Sigma$ parameter, which we can use only for the RBF kernel, since it is related to the degree of non-linearity we want to introduce for our classification. Lower values of $Sigma$ introduce more non-linearity, while using higher values make the hyperplanes found by the SVM tend to be more linear. We tried different values for sigma for the first dataset: $Sigma \in \{0.2, 1.0, 5.0\}$ and for the second dataset: $Sigma \in \{0.2, 0.63, 5.0\}$. As we can clearly observe from the figures, using small values of sigma makes the SVM overfit the training data, while using much higher values makes it to generalize too much. Therefore for the two dataset a good value was found in 1.0 for the first and 0.63 for the second even if, depending on the test distributions, this last one could also tend to overfit a bit.

As a final consideration on the difference between linear and rbf kernel, we can state than the last one can be useful for many applications since problems are generally non-linear, even if good hyperparameters values need

to be found. For linear problems the linear kernel can be useful, also in that case with a proper tuning of the hyperparameter C.

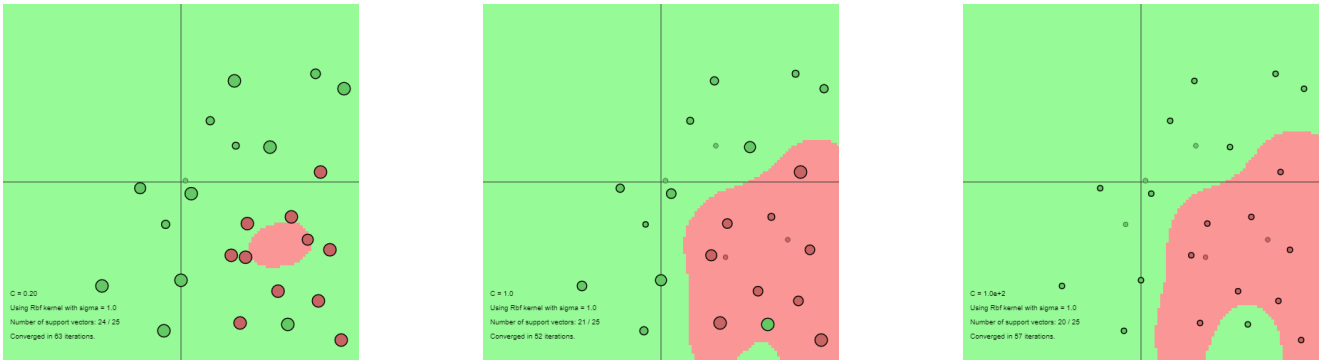


Figure 4: SVM rbf kernel with $C \in \{0.2, 1, 100\}$, first points allocation

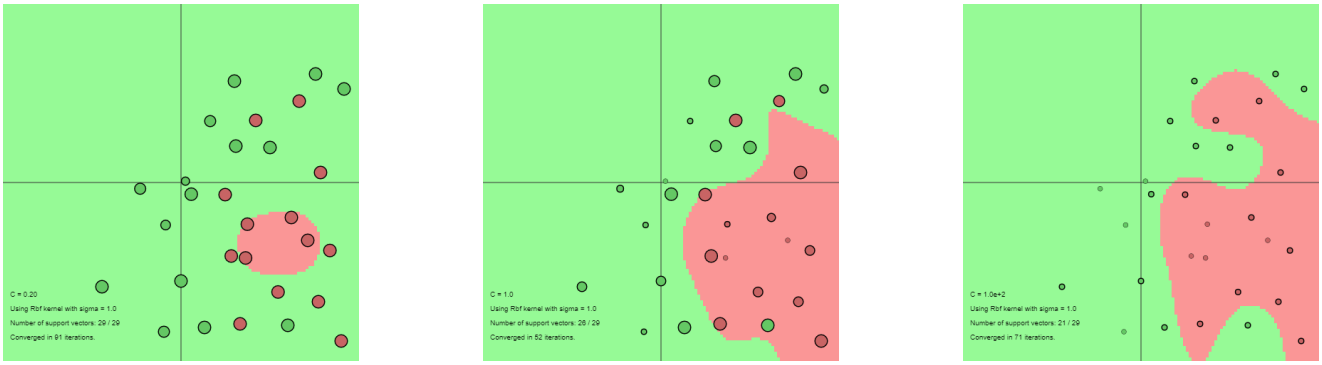


Figure 5: SVM rbf kernel with $C \in \{0.2, 1, 100\}$, second points allocation

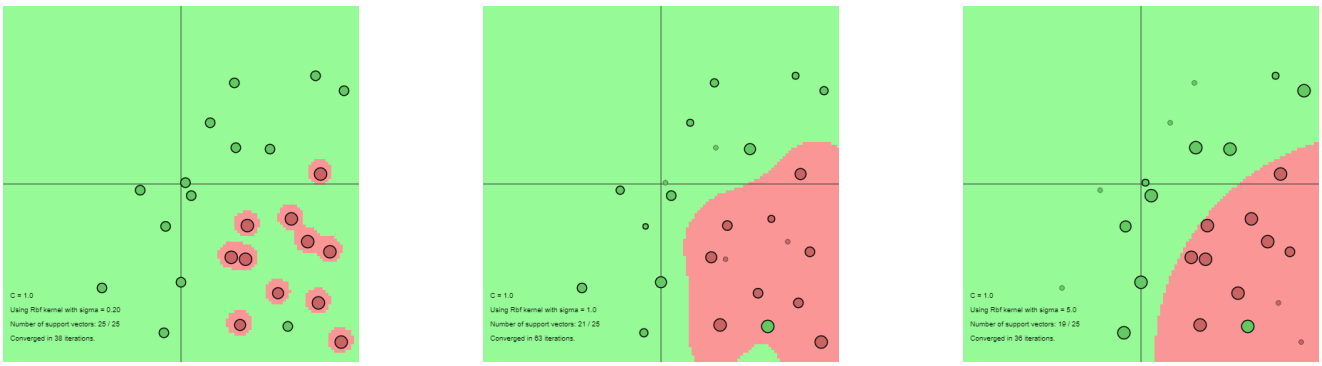


Figure 6: SVM rbf kernel with $Sigma \in \{0.2, 1.0, 5.0\}$, first points allocation

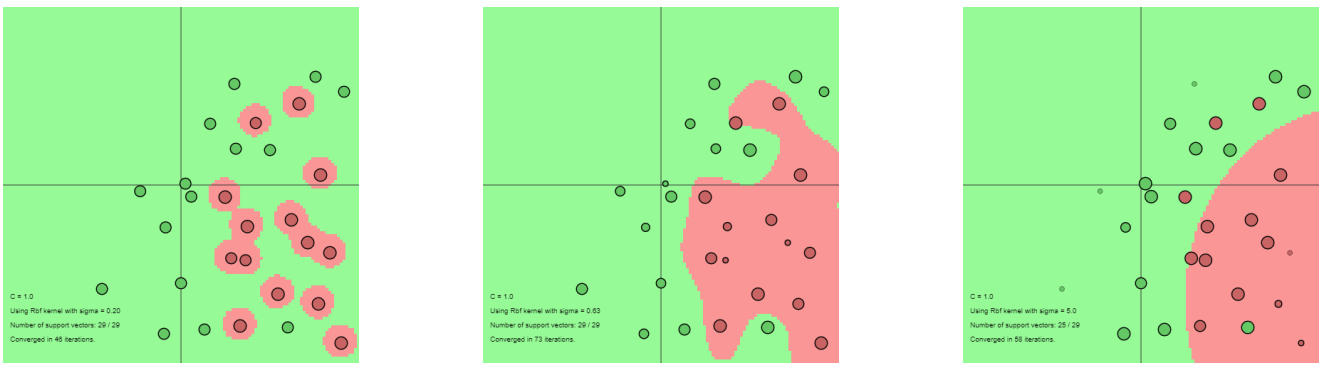


Figure 7: SVM rbf kernel with $Sigma \in \{0.2, 0.63, 5.0\}$, second points allocation

3 Least-squares support vector machine (LS-SVM) classifier

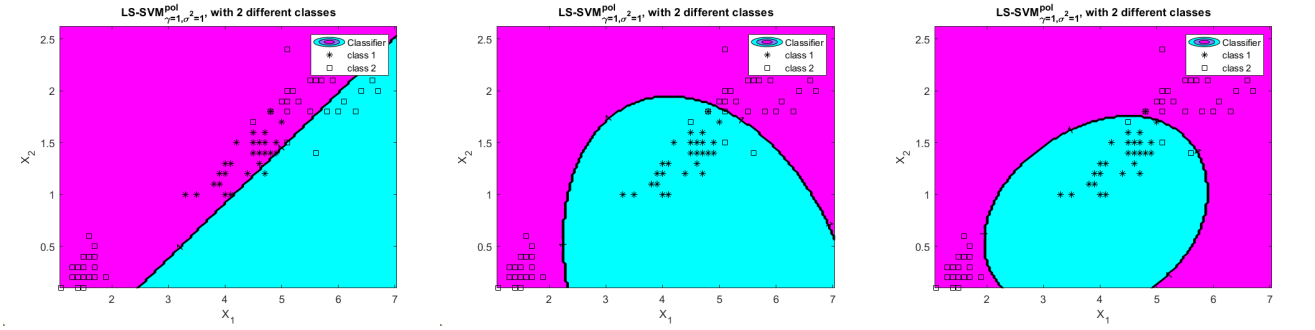
In this section we explore the matlab toolbox LS-SVMlab, the least squares based variant of the support vector machine (LS-SVM). We use several kernels: linear, polynomial, RBF. We use in this case the Iris dataset for the classification. It is composed of 100 samples in the training set and 20 in the test set.

3.1 Influence of hyperparameters and kernel parameters

The hyperparameter (regularization constant γ) and kernel parameters highly influence the classification model. Therefore we explore in this section what changes in the performances and decision boundaries of the classifier changing the hyperparameters and kernel parameters.

3.1.1 Polynomial kernel with different $degree$ values

We change here the kernel parameters, trying out a polynomial kernel with $degree = 1, 2, 3, ..., 10$ and keeping the hyperparameter $\gamma = 1$, assessing the performance on the test set. The results are shown in Figure 8. As we can observe for $degree = 1$ the decision boundary is linear, while it becomes curve and more complex as higher is the degree. In this case we should highlight how from $degree = 3$ until $degree = 10$ the misclassification Error Rate is 0. Even if performance did not lower for high values of $degree$ parameter, it is not good practice using a too high value of $degree$ for polynomial kernel, since it could easily lead to overfitting. For this reason we reported also the case for $d = 20$, for which we have 20% $ErrorRate$, corroborating what we said before. We can also notice in this case how the decision boundaries become super complex without a real need for it.



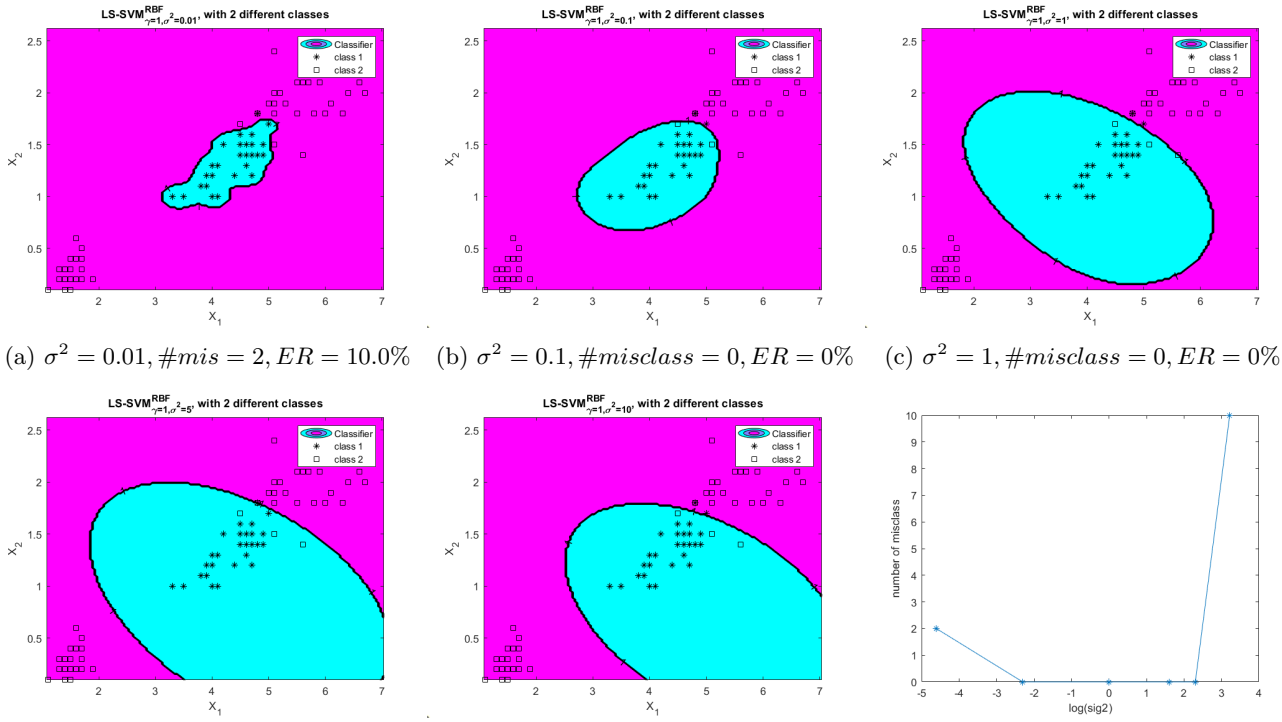
(a) $d = 1, \#misclass = 11, ER = 55\%$ (b) $d = 2, \#misclass = 1, ER = 5\%$ (c) $d = 3, \#misclass = 0, ER = 0\%$

(d) $d = 4, \#misclass = 0, ER = 0\%$ (e) $d = 5, \#misclass = 0, ER = 0\%$ (f) $d = 20, \#misclass = 4, ER = 20\%$

Figure 8: Iris Dataset. LS-SVM with polynomial kernel; d referring to the degree of the polynomial kernel, $\#misclass$ to the number of misclassifications on test set and ER to the Error Rate on the test set. The plots are showing the decision boundaries set on the training set.

3.1.2 RBF kernel with different σ^2 and γ values

σ^2 kernel parameter: we now focus on the RBF kernel with squared kernel bandwidth, trying out a good range of different σ^2 values as kernel parameters, fixing $\gamma = 1$. The σ^2 values tried were: $[0.01, 0.1, 1, 5, 10, 25]$. In Figure 9 we can see the results of our explorative study. For very low values: $\sigma^2 = 1$ the decision boundaries seem overfitting the training data, giving $10\%ER$ on the test set. For greater values the classifier seems to generalize better, loosing too much information for high values, such as for $\sigma^2 = 25$ which reaches $ER = 50\%$. Decision boundaries are not showed for this parameter value, since the simulation over the input space results in only one class. That is also why the Error Rate is exactly 50%, because we are misclassifying all the samples of one of the two classes, that are equally present in cardinality terms in the test set. Therefore a good range of values for the σ^2 parameter is: $[0.1, 10]$.



(a) $\sigma^2 = 0.01, \#mis = 2, ER = 10.0\%$ (b) $\sigma^2 = 0.1, \#misclass = 0, ER = 0\%$ (c) $\sigma^2 = 1, \#misclass = 0, ER = 0\%$

(d) $\sigma^2 = 5, \#misclass = 0, ER = 0\%$ (e) $\sigma^2 = 10, \#misclass = 0, ER = 0\%$ (f) $\#misclass$ depending on σ^2

Figure 9: Iris Dataset. LS-SVM with rbf kernel. Variable: σ^2. Fixed: $\gamma = 1$. $\#misclass$ refers to the number of misclassifications on test set and ER to the Error Rate on the test set. The plots are showing the decision boundaries on the training set

γ hyperparameter: having fixed $\sigma^2 = 1$ since we can notice from the image f) of Figure 9 that is a stable and good value to use, we now try to evaluate the performances of our classifier varying the γ regularization hyperparameter. The γ values tried were: $[0.01, 0.025, 0.05, 0.1, 1, 5, 10, 25, 50, 100, 250, 500, 1000]$. In Figure 10 we can observe, starting from low values of γ how little it is generalized one class compared to the other when this parameter is very small (i.e. $\gamma = 0.025$). The performance in this case is not good, with an $ER = 25\%$. Increasing its value, the $class1$ boundaries are more generalized, inducing also less Error Rate at test time. We have $ER = 5\%$ for $\gamma = 0.05$, while for higher values of γ (starting from $\gamma = 0.1$) the Error Rate on the test set for this dataset is stable on 0%. Therefore a good range of values for this parameter starts from $\gamma = 0.1$.

3.2 Tuning parameters using validation

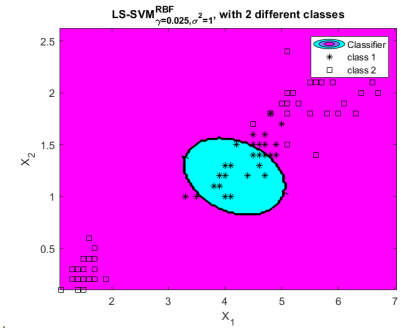
The intuitions about good hyperparameters values ranges developed in the previous section is now used to motivate automated tuning algorithms. In general, automatic tuning algorithms further split up the training data into a training and a validation part. This can be done in multiple ways: random split; K-Fold crossvalidation; Leave-One-Out validation.

Random split: One way of splitting up the training dataset into a training and a validation part is by randomly taking some data points for training. The remaining data points are consequently used for validation. We split the training set in 80% of samples for train and 20% for the validation set.

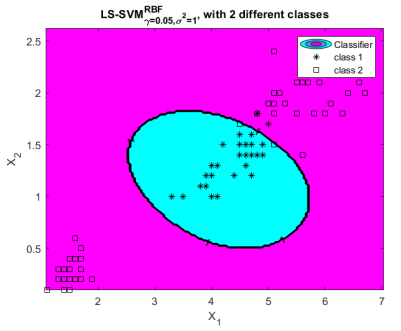
K-Fold crossvalidation: In K-Fold crossvalidation, K refers to the number of groups that a given data sample is to be split into. K is often taken to be equal to 10 as it happens also here. It means we split our training set in 10 Folds, each time each Fold is taken as the validation set and the remaining 9 as set of training.

Leave-One-Out (LOO) validation: Leave-one-out validation is a special case of K-fold crossvalidation, where K is taken equal as the number of data points. Therefore, each time we are taking out a sample, training on all the others. Obviously this is computational demanding, since we should train and validate $K = \#samples$ times.

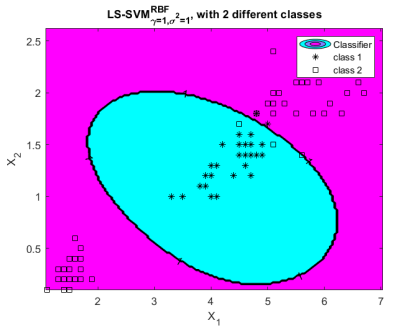
Given the nature of these 3 methods of tuning the parameter, evaluating the performances on the validation set in the case of K-Fold and LOO consists in computing the mean of the Error Rates on the several validation sets. This results also in removing the bias introduced by the data split. If we observe Figure 11, 12, 13 we can clearly notice the consequences of this. Using the random split method we are keeping all the data bias, hence having performances depending on the data splitting we are doing. Indeed we can observe frequently very low or very high ER (e.g. 0%, 40%) compared to the other 2 methods for which the ER varies between 3% and 35% circa.



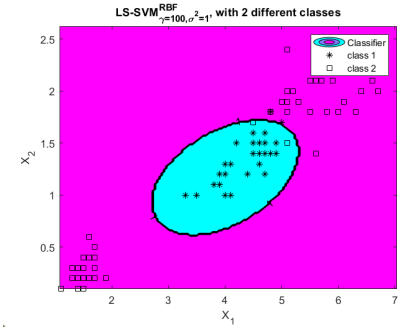
(a) $\gamma = 0.025, \#mis = 5, ER = 25\%$



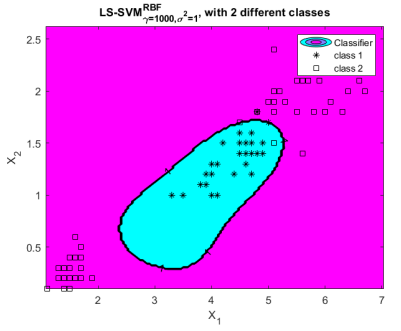
(b) $\gamma = 0.05, \#misclass = 1, ER = 5\%$



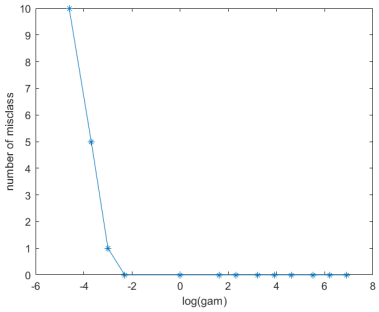
(c) $\gamma = 1, \#misclass = 0, ER = 0\%$



(d) $\gamma = 10^2, \#misclass = 0, ER = 0\%$

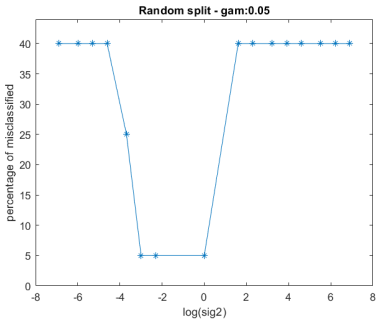


(e) $\gamma = 10^3, \#misclass = 0, ER = 0\%$

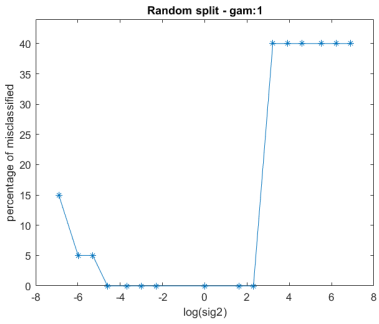


(f) $\#misclass$ depending on γ

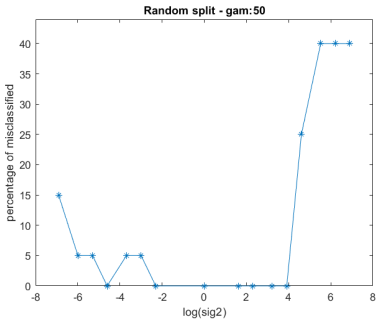
Figure 10: Iris Dataset. LS-SVM with rbf kernel. Variable: γ. Fixed: $\sigma^2 = 1$. $\#misclass$ refers to the number of misclassifications on test set and ER to the Error Rate on the test set. The plots are showing the decision boundaries on the training set



(a) Fixed:$\gamma = 0.05$, Variable: σ^2

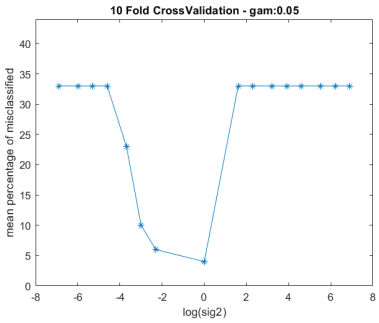


(b) Fixed:$\gamma = 1$, Variable: σ^2

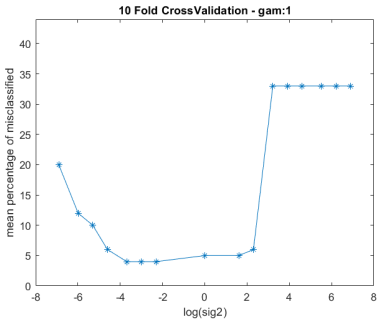


(c) Fixed:$\gamma = 50$, Variable: σ^2

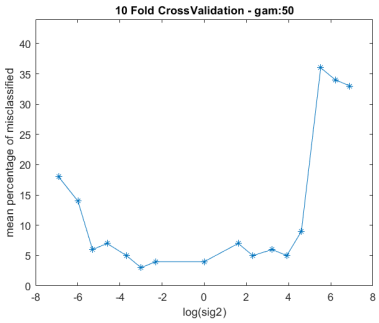
Figure 11: Random split 80-20, tuning γ and σ^2, evaluating the percentage of misclassified validation set samples



(a) Fixed:$\gamma = 0.05$, Variable: σ^2



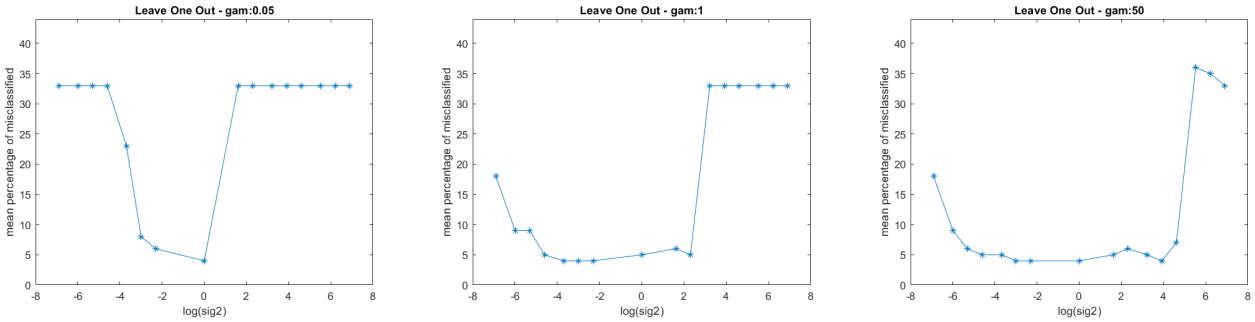
(b) Fixed:$\gamma = 1$, Variable: σ^2



(c) Fixed:$\gamma = 50$, Variable: σ^2

Figure 12: 10-Fold cross validation, tuning γ and σ^2, evaluating the mean percentage of misclassified validation set samples

Indeed, using K-Fold and LOO strategy the bias is instead minimized. Therefore we can see, how these strategies are to prefer, being more reliable than the previous one. We can observe more smoothed curves, from which we



(a) Fixed:$\gamma = 0.05$, Variable: σ^2 (b) Fixed:$\gamma = 1$, Variable: σ^2 (c) Fixed:$\gamma = 50$, Variable: σ^2

Figure 13: Leave-One-Out validation, tuning γ and σ^2, evaluating the mean percentage of misclassified validation set samples

can find better hyperparameters searching for the minima. We want to highlight how K-Fold is just in the middle of the other two strategies. It tends to the Random Split strategy for very low values of K, while it converges to LOO for high value. K-Fold it is preferable to LOO for the lower computational cost, while when you have a small dataset or when an accurate estimate of model performance is more important than the computational cost of the method LOO is to prefer.

We highlight that we show only some values of γ in the figures for space reasons. Therefore, having look at those reported values and graphs, to choose the proper parameters values in this case, we can simply observe the LOO performances. We can disregard $\gamma = 0.05$ since the performance is largely affected by little variations of σ^2. We can choose one of the local minima present for $\gamma = 1$ or $\gamma = 50$ in a stable region (where also varying σ^2 a bit, does not lower the performance too much) or simply choose the global minimum of these curves (which it could be not the global minimum for the problem).

3.3 Automatic parameters tuning

In the LS-SVMlab toolbox, the tuning procedure is fully automized by the function $tunelssvm$. We can choose 'Simplex' (Nelder-Mead method) or 'Gridsearch' strategies. The differences of this two methods are evaluated in Table 1, over 10 runs. The Gridsearch method seems to be the best performing, giving the lowest error rate (i.d. 'cost') while taking more computational time. Therefore, for large problems instances the Simplex method should be preferred. However, we should highlight that these two strategies give varying results each run, because of the initialization of the parameters due to Coupled Simulated Annealing results and the intrinsic randomness of these methods. While the error rate does not vary too much over different runs, the parameters γ and σ^2 can. This is consequence of multiple local minima present in the solution space, as we have seen also in the previous section.

Method	**Mean**	**Std**	**Max**	**Min**
Simplex	0.37	0.046	0.4	0.3
Gridsearch	0.35	0.067	0.4	0.2

Table 1: Error Rate statistics of the $tunelssvm$ method using Simplex or Gridsearch strategies

3.4 Using ROC curves

In practice, an alternative way to judge a classifier is by using the Receiver Operating Characteristic (ROC) curve of a binary classifier. The main performance measure derived from such a curve is the area under the curve: the better the classifier, the higher the area under the curve. We can tune the hyperparameters of LS-SVM with gridsearch and then show the results on the ROC curve computed on the test set. We use the test set since we use ROC to evaluate the performance of a model and using the training set will give a biased estimate on the performances as the classifier used those data to tune the hyperparameters. Doing this we evaluate the performance of the classifier on a variety of False Positive and True Positive Rates. Nevertheless, in this case the performance is always the same as we can observe in Figure 14. The $Area\ Under\ the\ ROC\ curve = 1$, with $Standard\ Error = 0$. These are the properties of a perfect classifier.

3.5 Bayesian framework

Using the Bayesian framework, it is possible to get probability estimates. In Figure 15 we can observe the results of using this. The probability that a certain data point belongs to the positive class given the model is represented

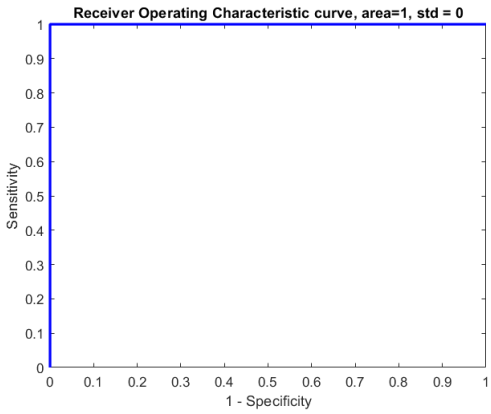
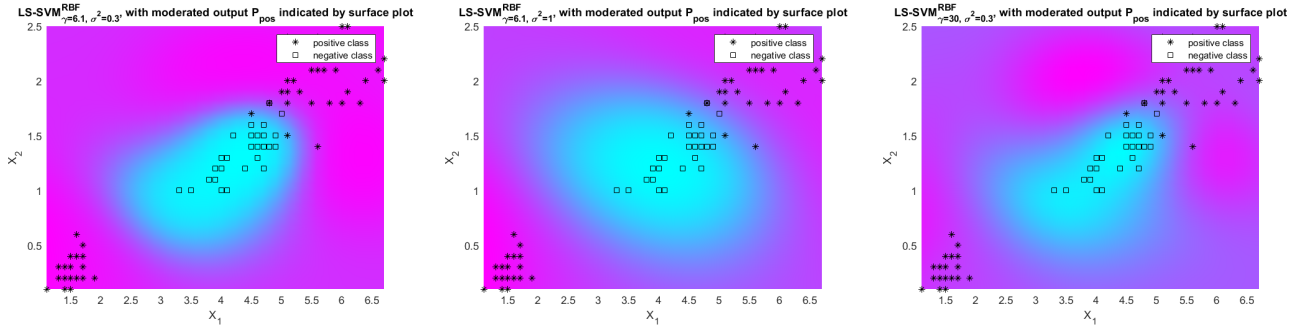


Figure 14: ROC curve on iris dataset using LS-SVM tuned with gridsearch

by the color, ranging from blue to purple. We start from the tuned hyperparameters values in image a), increasing the value of σ^2 as we can observe from image b) extend the probability of one class prediction smoothing the boundaries and the associated probabilities generalizing more, while increasing the value of γ seems to unbalance the probabilities to the negative class.



(a) Tuned: $\gamma = 6.1$, $\sigma^2 = 0.3$ (b) Fixed: $\gamma = 6.1$, Variable: $\sigma^2 = 1$ (c) Fixed: $\sigma^2 = 0.3$, Increasing: $\gamma = 30$

Figure 15: Bayesian framework: varying parameters

4 Homework problems

4.1 Ripley dataset

Ripley is a synthetic dataset. It consists of features from a mixture of two bidimensional normal distributions with different means and the same variance. It consists of 250 samples for training and 1000 samples for test. Each sample consists of 2 attributes. This makes this dataset easy to visualize in 2 dimensions. Labels are binary.

Given the two classes data distributions in Figure 16, we can observe that the two classes distributions are very similar to gaussian ones, even if the positive class has a more large variance. Being distributed in this way, with a small overlapping region and highly separable, all the linear, polynomial and RBF kernels for the LS-SVM could work in a good manner. Indeed, given the distributions, also linear decision boundaries could work in this case.

After hyperparameters tuning exploiting the *tunelssvm* method making use of the *gridsearch* strategy, we find good hyperparameters values for all the three kernels approaches as reported in Table 2, in order to evaluate their performances with the ROC curves on the test set. Since the dataset has only 2 attributes, we decided also to show the decision boundaries of the three LS-SVMs based on the different kernels in Figure 17.

Kernel	γ	σ^2	**Degree**
Linear	0.010569	-	1
Polynomial	241.6413	-	5
RBF	4.903	0.8508	-

Table 2: Ripley dataset, LS-SVM classifier, hyperparameters found for the three different kernels

The performances on the test set can be observed in Figure 18. As one can notice, the RBF outperforms the

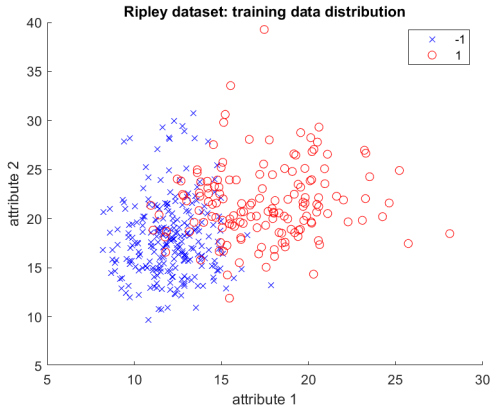
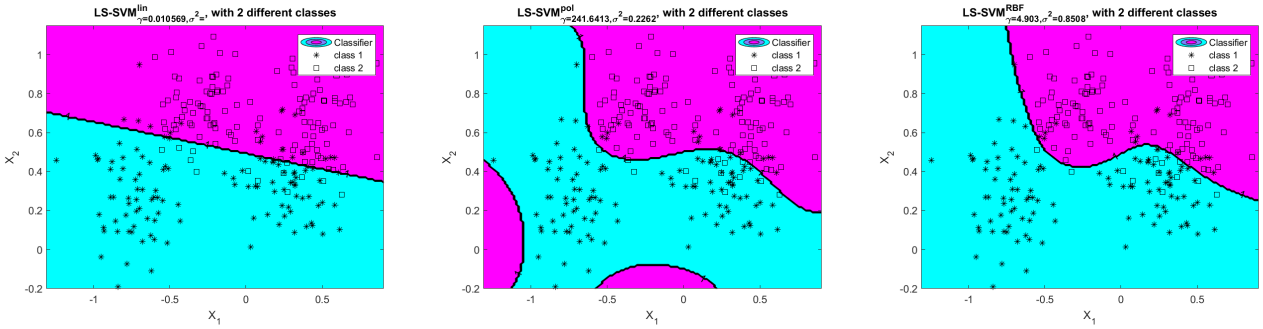


Figure 16: Ripley dataset: training data distributions of the two classes



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

Figure 17: Ripley dataset, LS-SVM classifier, decision boundaries on the training data for the kernel types: a) linear; b) polynomial; c) RBF

other strategies for almost every threshold value. The worst performing is the Linear kernel LS-SVM even if it seems to be better than the others for very low false positive rates, while the polynomial curve is just in the middle of the others. From Table 3 we can observe the values of the Area Under the ROC curve and the standard error which is the approximate standard deviation of a statistical sample population. This values of performance corroborate what we stated before, with RBF kernel which outperforms the polynomial and linear kernels.

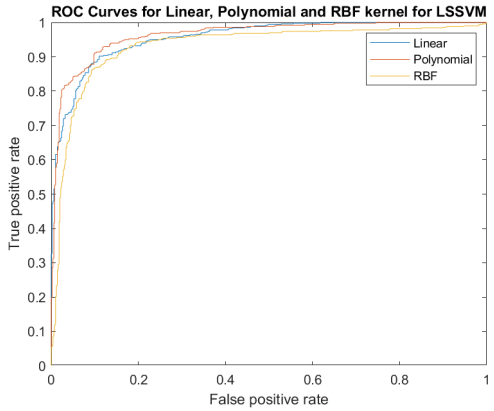


Figure 18: Ripley dataset, LS-SVM classifier, ROC curves for the kernel types: linear; polynomial; RBF

Kernel	Area under ROC	Standard Error
Linear	0.95799	0.0054944
Polynomial	0.96312	0.0054316
RBF	0.96885	0.0049502

Table 3: Ripley dataset, AUROC and Standard Error for the kernel types: linear; polynomial; RBF

4.2 Wisconsin Breast Cancer dataset

Each record represents follow-up data for one breast cancer case. It consists of 400 samples for training and 169 samples for test. Each sample consists of 30 attributes. Therefore we can not visualize this dataset in 2D as we made for the Ripley dataset. Labels are binary. We use again $tunelssvm$ with the $gridsearch$ option, to find the hyperparameters for the various kernel strategies. The values found are reported in Table 4. We should highlight that with this method also a hyperparameter t is found for the polynomial kernel, nevertheless we do not report it since it is not a main component of our study. As we can observe the polynomial kernel has degree 3, and the RBF has high values for γ and σ^2. This results in curve decision boundaries which tend to be linear, thus probably the classes are linearly separable.

Kernel	γ	σ^2	**Degree**
Linear	0.061215	-	1
Polynomial	0.000312	-	3
RBF	19.0253	47.5423	-

Table 4: Breast cancer dataset, LS-SVM classifier, hyperparameters found for the three different kernels

Setting the tuned hyperparameters for the 3 methods, the resulting performances had on the test set using the ROC curve, are shown in Figure 19. As one can observe, for low values of false positive rate, the RBF is the best strategy, while for values greater than 0.05 circa, the linear strategy becomes the one with the best performance. Both these 2 strategies outperform the one based on the polynomial kernel. Given these statements and the Area Under the ROC curve and Standard Error reported in Table 5, we can conclude that the samples space is linearly separable, as we supposed when we tuned the hyperparameters. Moreover, we can observe how the Standard Error for the polynomial kernel is 10 times higher compared with the ones of the other two strategies, with a much lower value of AUROC. Therefore for an application we only can choose a system based on RBF or Linear kernel for this specific dataset.

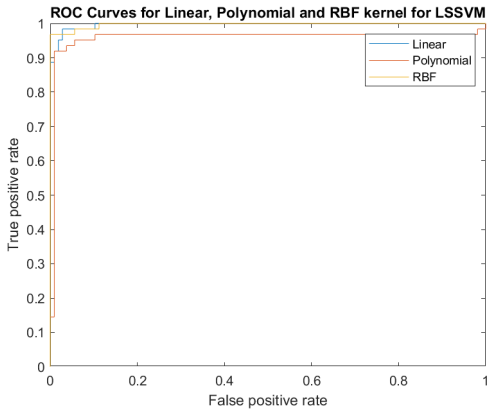


Figure 19: Breast dataset, LS-SVM classifier, ROC curves for the kernel types: linear; polynomial; RBF

Kernel	**Area under ROC**	**Standard Error**
Linear	0.99653	0.0023678
Polynomial	0.95764	0.023376
RBF	0.99729	0.0022237

Table 5: Breast dataset, AUROC and Standard Error for the kernel types: linear; polynomial; RBF

4.3 Diabetes dataset

Each record represents a case of diabetes or not. It consists of 300 samples for training and 168 samples for test. Each sample consists of 8 attributes, for this reason we can not show the 2D distributions. Labels are binary.

As before, we proceed with hyperparameter tuning. The results of this process can be seen in Table 6. Also in this case we can notice a very low degree for the polynomial kernel and high values for RBF parameters, making probable that also in this case the space is approximately linearly separable.

From Figure 20 and Table 7 we can observe how Linear and RBF kernel strategies, which show very similar performances, outperform the polynomial one. As we can see performances are not so high as the Breast dataset, showing probably a more overlap in the two classes in the samples space.

Kernel	γ	σ^2	Degree
Linear	0.24789	-	1
Polynomial	0.00033	-	3
RBF	3802.71	81394.3	-

Table 6: Diabetes dataset, LS-SVM classifier, hyperparameters found for the three different kernels

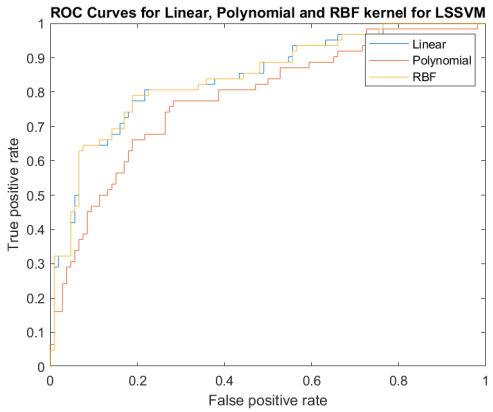


Figure 20: Diabetes dataset, LS-SVM classifier, ROC curves for the kernel types: linear; polynomial; RBF

Kernel	Area under ROC	Standard Error
Linear	0.84327	0.031891
Polynomial	0.78378	0.037040
RBF	0.84419	0.031912

Table 7: Diabetes dataset, AUROC and Standard Error for the kernel types: linear; polynomial; RBF

4.4 Conclusions

The LS-SVM with different kernels showed satisfying performances on all the dataset with the exception for the diabetes one, on which it seems to not perform well, even if this is probably responsability of a dataset with much overlap of attributes values for the two classes. We can conclude also that given different kernel strategies, the SVM is very flexible to all the scenarios showing good performances. This constitutes in our opinion the greatest advantage of this machine learning technique.

Assignment 2: Function Estimation and Time Series Prediction

Daniele Giannuzzi r0876755

1 Support vector machine for function estimation

The support vector machine (SVM) can be also used for regression.

First, making use of the $uiregress$ demo present in the toolbox, we construct a dataset where a linear kernel is better than any other kernel, using 20 data points, trying to understand the influence of the parameters ϵ and $Bound$. From the theory, we can consider regression in the set of linear functions $f(x) = w^T x + b$, with N training data $x_k \in \mathbb{R}^m$ and output values $y_k \in \mathbb{R}$. In finding the regression function, one wants to minimize the empirical risk R_{emp} (training error), that is proved to converge as well as the expected risk $R(\theta)$ (generalization error) to $\inf_\theta R(\theta)$ for $N \to \infty$ when the learning process is consistent. We should highlight that without restricting the set of admissible functions, empirical risk minimization is not consistent. The expected risk $R(\theta)$ is reported in Equation 1, where $p(x, y)$ refers to the probability that sample is generated, y is the ground truth function, and $f(x; \theta)$ the function of the classifier. The objective function, which corresponds in minimizing the empirical risk R_{emp} is reported in Equation 2, where w^T and b are the trained parameters of the classifier function. The structure to which the objective function is constrained is described in Equation 3, with the C parameter that is the regularization term. In Equation 4 we have the Vapnik's ϵ-insensitive loss function useful for the computation of the objective function. In practice, if the prediction of a sample x differ from the ground truth y at least of ϵ there is a penalization for the objective function (also using the Lagrangian method and slack variables not reported here).

$$R(\theta) = \int \frac{1}{2} |y - f(x; \theta)| p(x, y) dx dy \tag{1}$$

$$min \quad R_{\text{emp}} = \frac{1}{N} \sum_{k=1}^{N} \left| y_k - w^T x_k - b \right|_\epsilon \tag{2}$$

$$s.t. \; \|w\|_2^2 \leq C \tag{3}$$

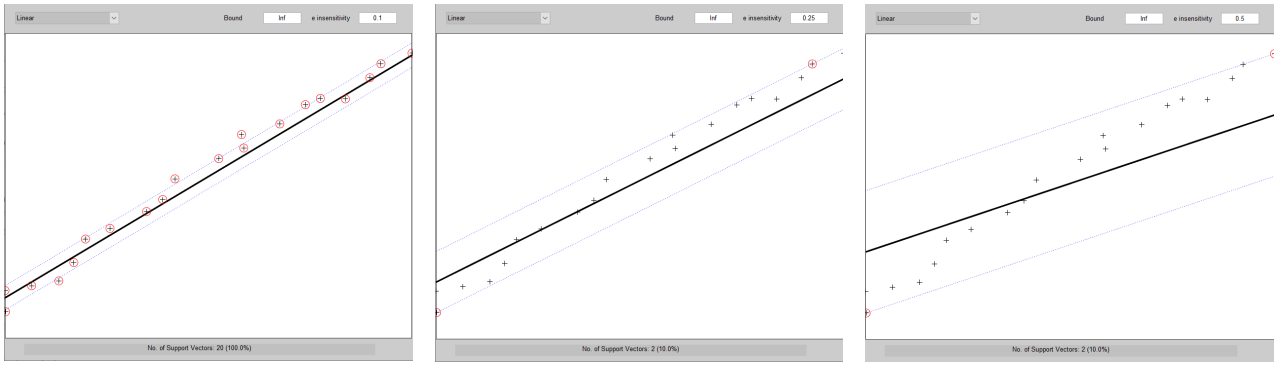
$$|y - f(x)|_\epsilon = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & \text{, otherwise} \end{cases} \tag{4}$$

Therefore, the parameter C, also named $Bound$ in the toolbox, regulates the tradeoff between the achieving a low training error and the ability to generalize the classifier to unseen data. It must be $C > 0$, with the strength of the regularization that is inversely proportional to C. We can notice the influence of C on regression keeping $\epsilon = 0$ in Figure 2. If $C = \infty$, as by default in the demo, then the objective function aim is to minimize the empirical risk only, without a term for a better generalization. On the other hand if it is too small then the empirical risk is not minimized at all, leading to weights that are not trained enough.

On the other hand, ϵ parameter controls the width of the ϵ-insensitive zone. It specifies the ϵ-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. As we can notice in Figure 1, the bigger the ϵ value, the fewer support vector are selected, therefore having sparsness property, which means that decision boundary can be expressed in terms of a limited number of support vectors, subset of given training data.

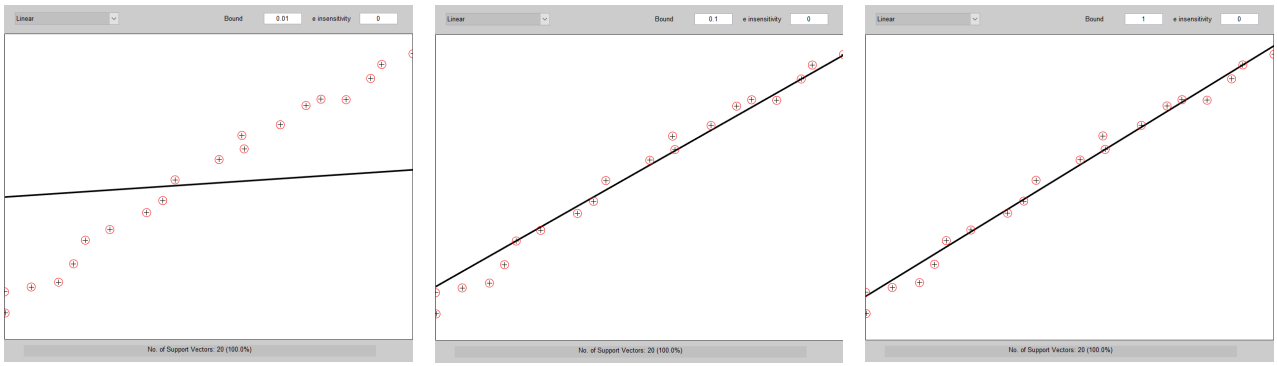
We now construct a non-linear more challenging dataset again with 20 data points, in order to analyze the differences in using different kernels. As we can observe from Figure 3 for non-linearity polynomial and rbf kernels are well suited, even if the optimization of the $degree$ parameter is easier than σ^2 one, since we work with integer values, both the kernels achieve comparable good performances.

SVM regression is different from a Ordinary Least Squares (OLS) fit because in OLS case the empirical risk function is made of a squared loss function, without regularization terms which lead to a better generalization at test time.



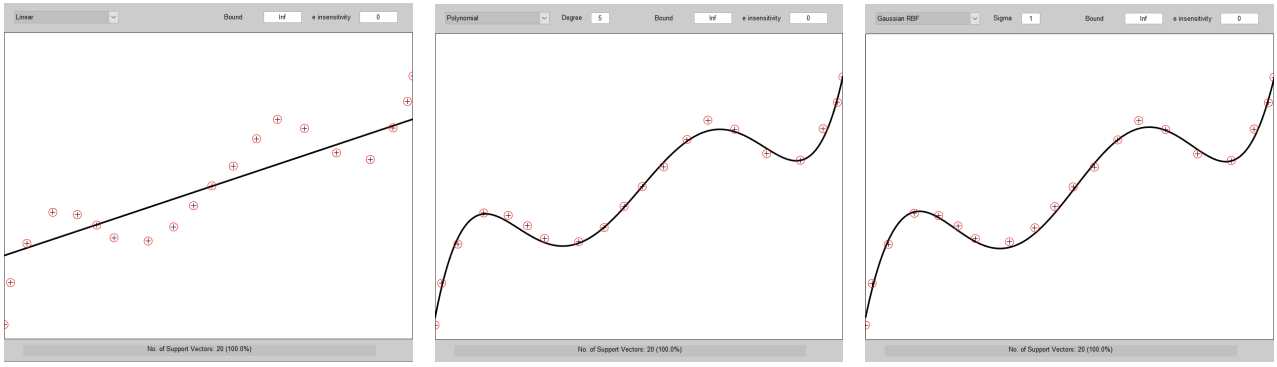
(a) $\epsilon = 0.1$ (b) $\epsilon = 0.25$ (c) $\epsilon = 0.50$

Figure 1: SVM for regression, demo uiregress. Linear kernel: trying incrementing ϵ values, fixed $Bound = Inf$



(a) $Bound = 0.01$ (b) $Bound = 0.10$ (c) $Bound = 1.00$

Figure 2: SVM for regression, demo uiregress. Linear kernel: trying incrementing $Bound$ values, fixed $\epsilon = 0$



(a) Kernel: linear (b) Kernel: polynomial ($degree = 5$) (c) Kernel: rbf ($\sigma^2 = 1$)

Figure 3: SVM for regression, demo uiregress. Trying to use different kernels

2 A simple example: the sinc function

2.1 Regression of the sinc function

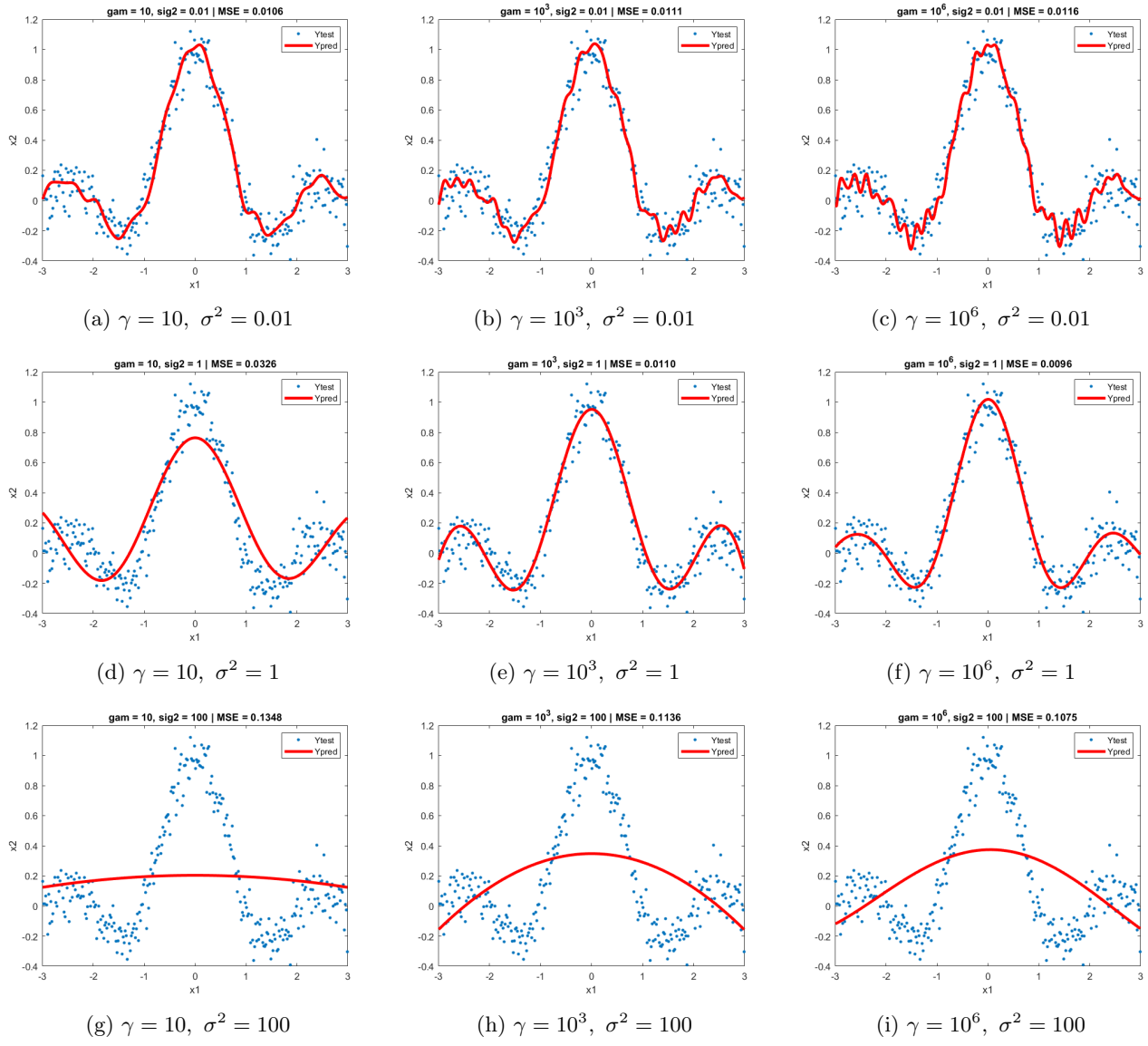
We proceed with the least squares based variant of the support vector machine (LS-SVM), using the LS-SVMlab toolbox. SVM has been widely used in classification and regression problems. However, the major drawback of SVM is its higher computational cost for the constrained optimization programming. This disadvantage is overcome by LS-SVM, which solves linear equations instead of a quadratic programming (QP) problem. The only drawback in LS-SVM case is the lack of sparseness, but it can be imposed using pruning techniques from the neural networks area (e.g. optimal brain damage, optimal brain surgeon). Therefore, LS-SVM is preferred especially for large scale problem, because its solution procedure is high efficiency and after pruning both sparseness and performance of LS-SVM are comparable with those of SVM.

To get an intuitive idea what function estimation is about, we use an artificial dataset constructed from the sinc function with white noise. In particular X is generated as a sequence of 601 points equally distant, ranging from

-3 to 3 with an increment of 0.01. Then $Y = sinc(X) + 0.1\dot{N}(0,1)$, where N is the added normally distributed noise, scaled by a factor of 0.1. Consequently, we create the training and the test sets, taking the odd generated points as training samples and even generated points as testing samples.

Now, we train the LS-SVM model using a RBF kernel with the generated training data, evaluating its performance on the test set, using several combinations for the σ^2 kernel parameter and γ regularization parameter. In particular: $\sigma^2 \in \{0.01, 1, 100\}$, $\gamma \in \{10, 10^3, 10^6\}$.

The results on the test set are shown in Figure 4. As we can observe, for low values of σ^2 (last row of the figure) too much linearity is induced in the function estimation, while the contrary happens for high values which introduce too much non-linearity, which brings to overfitting (first row of the figure). The γ parameter introduce much regularization for low values (e.g. 10) and almost no regularization for higher values (e.g. 10^6) as we can observe in rispectively in Figure 6a and 4c. Observing the figures, good parameters are found for the combination $\gamma = 10^6$, $\sigma^2 = 1$, which leads to the $MSE = 0.0096$, since a good trade-off in non-linearity and regularization seems to be reached.



(a) $\gamma = 10,\ \sigma^2 = 0.01$ (b) $\gamma = 10^3,\ \sigma^2 = 0.01$ (c) $\gamma = 10^6,\ \sigma^2 = 0.01$

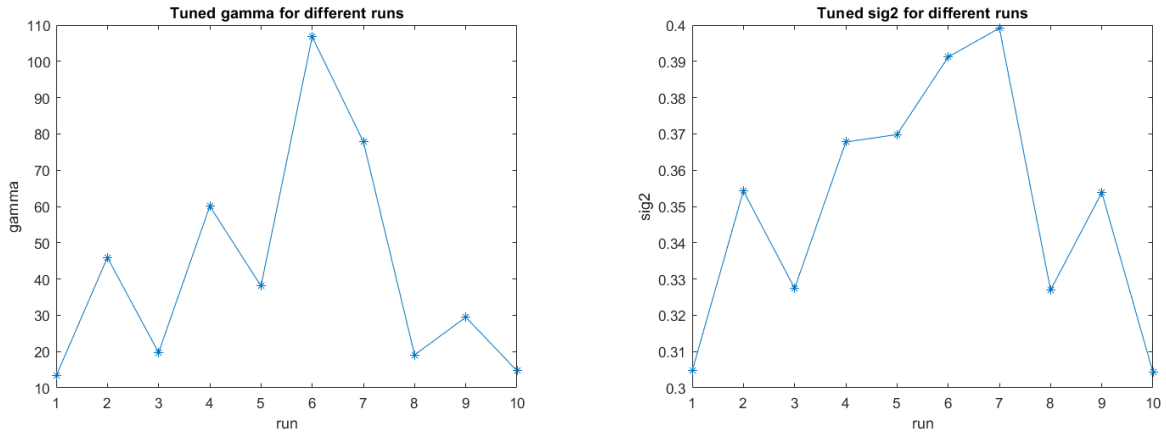
(d) $\gamma = 10,\ \sigma^2 = 1$ (e) $\gamma = 10^3,\ \sigma^2 = 1$ (f) $\gamma = 10^6,\ \sigma^2 = 1$

(g) $\gamma = 10,\ \sigma^2 = 100$ (h) $\gamma = 10^3,\ \sigma^2 = 100$ (i) $\gamma = 10^6,\ \sigma^2 = 100$

Figure 4: LS-SVM with rbf kernel. Variable: σ^2 and γ. MSE refers to the Mean Squared Error on test set. The blue points are the test data points, the red line is the estimated regression function.

We also used the $trainlssvm$ function to tune these parameters, repeating the experiment 10 times using both the $gridsearch$ and $simplex$ methods. As we can observe from Figure 5, using the $gridsearch$ method, the values of σ^2 are very low, ranging from 0.3 to 0.4, while the values of γ are spread in a range that is larger, from 10 to 110. We can observe that when one is higher, say σ^2, also the other parameter is higher. This is because if we introduce less non-linearity increasing the value of σ^2, we need a lower regularization. On the other hand using the $simplex$ method, as one can observe in Figure 6 the ranges for the values of both parameters are larger, showing that many combinations of these two parameters lead to good results in MSE performance, as we can

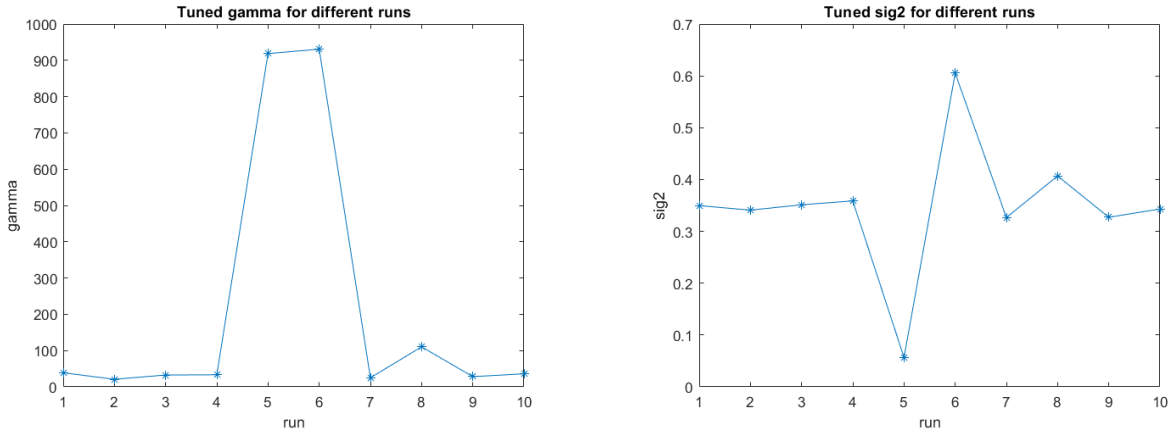
notice from Table 1. In particular we can notice from the MSE standard deviation that in practice the same value of MSE is found on the test set after optimization of parameters. Another consideration we can make looking the values for Elapsed time mean and standard deviation is that the $simplex$ method is approximatively 30% faster than the $gridsearch$ one.



(a) γ values over different runs

(b) σ^2 values over different runs

Figure 5: Tuned hyperparameters γ and σ^2 using $tunelssvm$ function with $gridsearch$ over 10 runs



(a) γ values over different runs

(b) σ^2 values over different runs

Figure 6: Tuned hyperparameters γ and σ^2 using $tunelssvm$ function with $simplex$ over 10 runs

Method	MSE mean	MSE std	Elapsed time mean (sec)	Elapsed time std
Gridsearch	0.009568	7.542e-05	1.2472	0.09642
Simplex	0.009595	7.177e-05	0.7965	0.09937

Table 1: LS-SVM using $tunelssvm$ function over 10 different runs for hyperparameters tuning

2.2 Application of the Bayesian framework

In addition to the approach used before, the Bayesian framework can also be used to tune and to analyze the LS-SVM regressor. The basic result from the Bayesian framework for the LS-SVM is the derivation of the probability that the data points are generated by the given model. This is called the posterior probability. This probability criterion is expressed as a number. Bayesian inference is made of 3 levels of inference, in each level trying to estimate the optimal posterior. The bay_lssvm function estimates the posterior probabilities of model parameters and hyperparameters on the different inference levels. In the first level one optimizes the support values α's and the bias b. In the second level one optimizes the regularization parameter γ. In the third level one optimizes the kernel parameter, that in case of RBF kernel is σ^2 At each of these levels one has, as in Equation 5:

$$\text{Posterior} = \frac{\text{Likelihood}}{\text{Evidence}} \times \text{Prior} \tag{5}$$

The likelihood at a certain level equals the evidence at the previous level. In this way, by gradually integrating out the parameters at different levels, the subsequent levels are linked to each other. The cost (negative logarithm of the posteriors) is computed as output by the bay_lssvm function for each level.

Using the $bay\_optimize$ function, one want to maximize the posterior probabilities (minimizing the computed cost) of model parameters and hyperparameters with respect to the different levels in Bayesian inference.

An important aspect of Bayesian methods is that the assumptions are made very explicit in the prior. In this case we use as a prior $\gamma = 10$ and $\sigma^2 = 0.4$ since we use LS-SVM with RBF kernel. As we can observe in Figure 7 the regression function generated by the Bayesian method is delimited by the 95% error bands. Indeed, the computation of the $bay_errorbar$ method takes into account the estimated noise variance and the uncertainty of the model parameters, estimated by Bayesian inference. The found parameter values after maximizing the posterior are $\gamma = 0.46588$ and $\sigma^2 = 0.26706$.

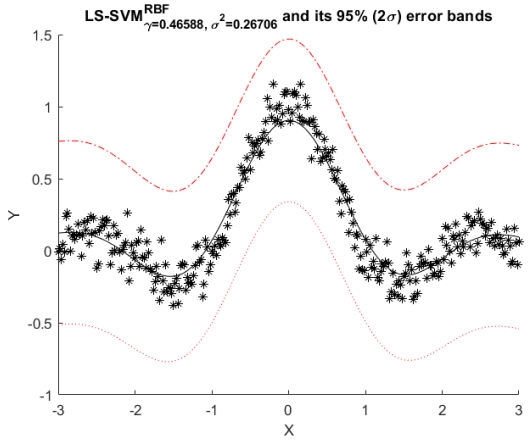


Figure 7: Bayesian error bar and function estimation

3 Automatic Relevance Determination

In addition to parameter tuning, the Bayesian framework can also be used to select the most relevant inputs by Automatic Relevance Determination (ARD), with the $bay\_lssvmARD$ function. We must note that different models can lead to different conclusions about relevance of inputs (attributes of the dataset), then the obtained relevance is not valid in an absolute sense, but only relative with respect to the considered model.

In practice, in each step, the input with the largest optimal σ^2 is removed (backward selection), since using the RBF kernel a large σ^2 value translates into irrelevance of the input to the model. Indeed, this method only works with the RBF kernel with a σ^2 per input. For every step, the generalization performance is approximated by the cost associated with the third level of Bayesian inference.

Using a dataset X composed of 3 attributes, each one varying in range $[-3, 3]$ with normal distribution, imposing $Y = sinc(x_1) + 0.1\dot{N}(0, 1)$ where N(0,1) is the added white noise. Using the ARD function, one should find that x_1 is the most relevant input, as it actually happens since the found relevance ranking for the attributes of X is: $1, 2, 3$ or $1, 3, 2$ depending on the run. In Figure 8 one can observe the regression function using the σ and γ found by the method, using only the first attribute, that is the meaningful one.

We can use crossvalidation for attributes ranking and selection, but this could be inefficient. We can start crossvalidating one model for each feature, choosing the best feature id est the feature for which the MSE is the lowest. Then we crossvalidate each model on the combination of the already chosen feature in combination with every other feature, selecting then the second meaningful feature as before. In this way we proceed the opposite of before: instead of a backword selection we apply a forward selection.

4 Robust regression

In situations where the data is corrupted with non-Gaussian noise or outliers, it becomes important to incorporate robustness into the estimation. We than consider a dataset X with values ranging from -6 to 6 with 61 points equally spaced of 0.2, with $Y = sinc(X) + 0.1\dot{U}(0, 1)$ where $U(0, 1)$ is uniformly distributed noise. We then modify 6 of those points, making them outliers, observable as red points in Figure 9.

If we train a LS-SVM regressor model, without giving special attention to the outliers, its performance is highly biased in their favour as we can notice in Figure 10a. On the other hand if we use a version of this model robust against outliers, as we can observe in Figure 10b the bias is no more present and the estimated function

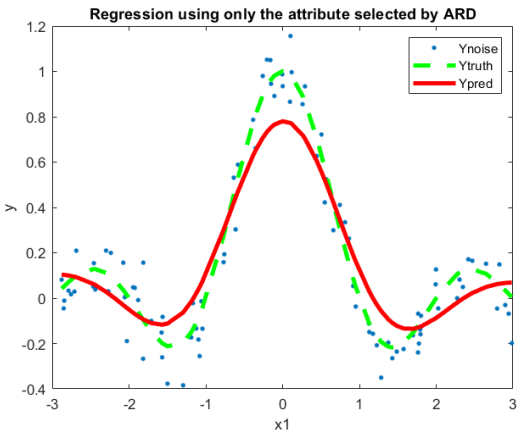


Figure 8: Function estimation using ARD method

is a correct regression of the available points. In case of the robust version the MAE (Mean Absolute Error) is used instead of the MSE (Mean Squared Error). This because a drawback of least squares (L_2 estimator) is that it is less robust against outliers and non-Gaussian noise. The L_1-estimator reduces the influence of outliers, while outliers are too much emphasized in the cost function of the L_2 estimator, since one wants to minimize the error made also on outliers that is much larger if we consider the squared difference between the real value and the predicted one.

To be robust against outliers, we use a weighted version of LS-SVM, with a modified cost function (in specific the Hampel weighting function). However, several weighting functions exist. In Figure 11a, 11b, 11c rispectevely

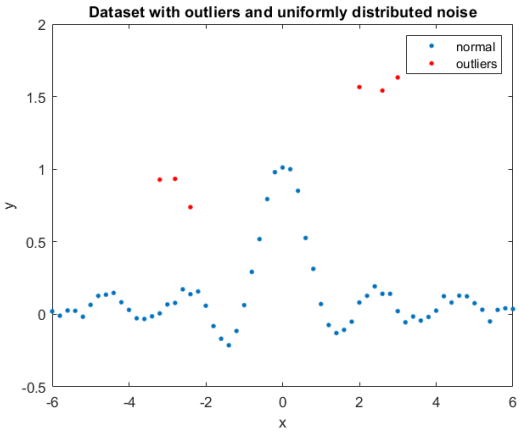
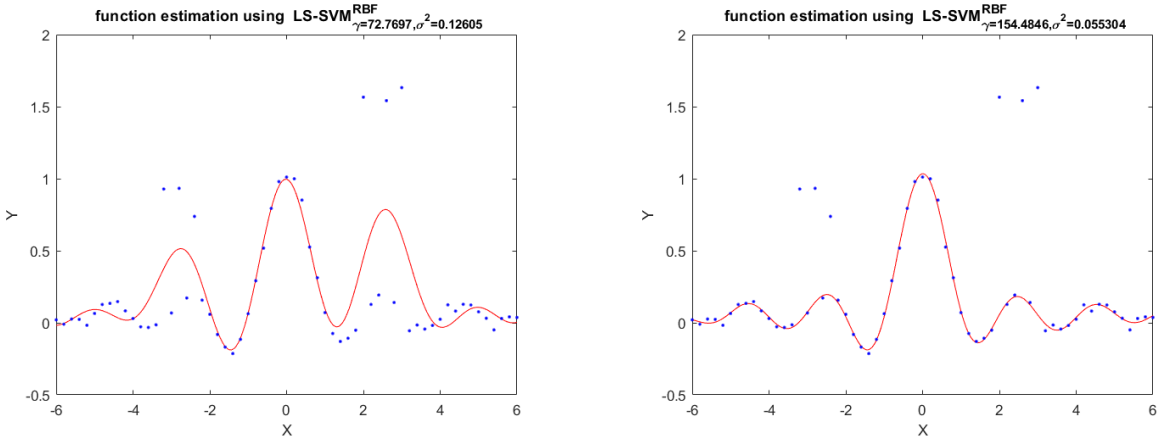


Figure 9: Dataset with outliers (red) and uniformly distributed noise on normal points (blue)

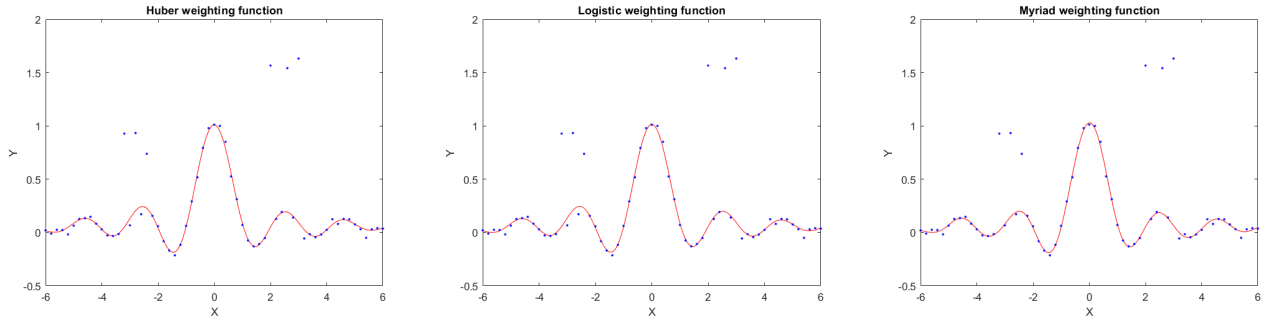


(a) Non-robust LS-SVM

(b) Robust LS-SVM using Hampel weighting function

Figure 10: Tuned hyperparameters γ and σ^2 using *tunelssvm* function with *gridsearch* over 10 runs

the Huber, Logistic and Myriad weighting functions are used. No substantial differences can be observed in using one or the other weighting function for the sinc function estimation.



(a) Huber weighting function (b) Logistic weighting function (c) Myriad weighting function

Figure 11: Several weighting functions for a robust LS-SVM

5 Homework problems: Time Series prediction

5.1 Logmap dataset

We now apply time series prediction on the logmap dataset. It consists of 150 points of training and 50 of test. A representation of the dataset is shown in Figure 12.

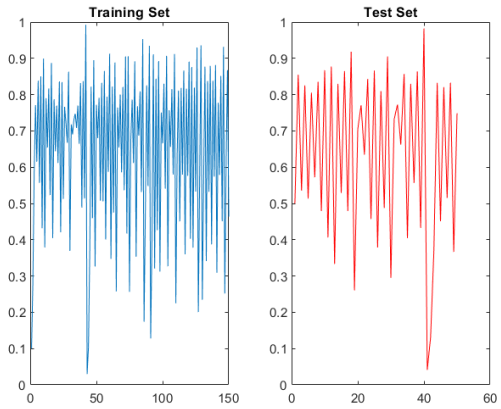
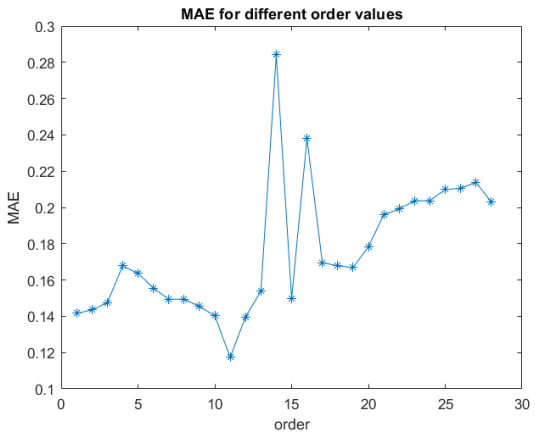


Figure 12: Logmap dataset with train (blue) and test (red) set

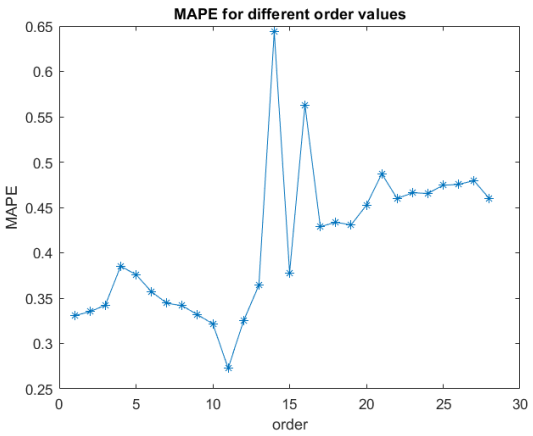
As indicated numerous times before, the parameters γ and σ^2 can be optimized using crossvalidation. In the same way, one can optimize $order$ as a parameter, also named lag. One should find a trade-off in optimizing this parameter. High values lead to short windows for the model to properly learn from the past, while small values results in low amount of windows to train. A strategy to optimize these 3 parameters could be using a rolling validation set of fixed size, since we can not use folds as in other cases as the order here matters, or we can simply take the last points of the training set as the validation ones. We adopted this strategy since having a rolling validation window on a small dataset could lead to misleading results. Therefore 120 point are actually used for the training of the model, while the last 30 points (20%) are used as the validation set. Then, cross-validation is based upon feedforward simulation on the validation set using the feedforwardly trained model. We use $tunelssvm$ method to tune γ and σ^2 with a 10-CV strategy. One could try to optimize the tuned LS-SVM for a range of values of $order \in \{1, 2, 3, ..., 28\}$. In doing this, we try to minimize the MAPE (Mean Absolute Percentage Error) that is one of the most popular used error metrics in time series forecasting. The formula is reported in Equation 6.

$$MAPE = \frac{1}{n} \times \sum \left| \frac{\text{actual value} - \text{forecast value}}{\text{actual value}} \right| \tag{6}$$

As a result, as we can observe in Figure 13b the MAPE is minimized for $order = 11$, as well as the MAE as we can notice in Figure 13a.



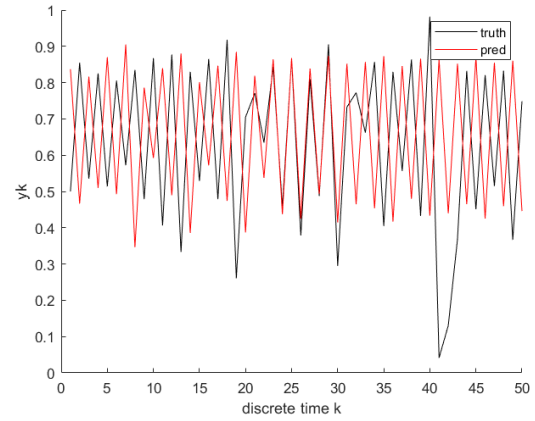
(a) MAE



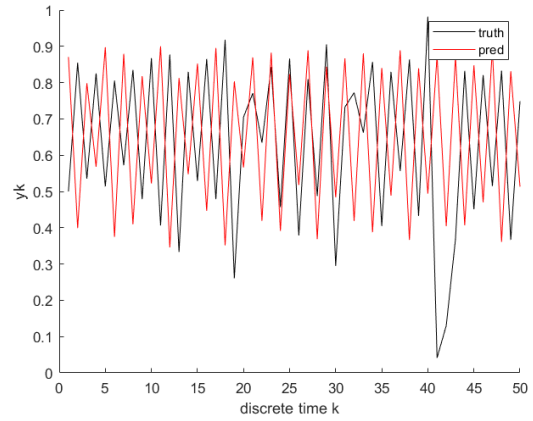
(b) MAPE

Figure 13: MAE and MAPE on the logmap validation set, using different order values

Therefore using $order = 11$ and tuning γ and σ^2, we try to predict the test set values. The result of the prediction is shown in Figure 14b. It is slightly better than using the $order = 10$ as by default, as reported in Figure 14a. The main problem we can observe in this time series prediction is that the prediction on the test start with a different phase, which leads to bad results. Indeed, the signal is not always going up and down, sometimes staying more time than expected in high or low values. This is what affect the performance the most in the prediction of this time series.



(a) Order=10: $MAE = 0.3180, MAPE = 0.9715$



(b) Order=11: $MAE = 0.3061, MAPE = 0.9484$

Figure 14: Prediction on the logmap test set using differet $order$ values and consequently different γ and σ^2 values

5.2 Santa Fe dataset

Santa Fe dataset is composed of 1000 points of training and 200 of test. We can observe a representation of this dataset in Figure 15.

For this dataset a $order$ value with which we can start is 50. Therefore, we can analyze lower and higher values. We choose to analyze $order$ values from 20 to 80. The validation strategy is the same as the one used for logmap dataset. As we can observe from Figure 16a and 16b $order = 43$ seems to be the most eligible value, since it is a local minimum and varying it a bit does not affect so much the performance in a negative way, as happens instead for the $order = 56$.

For this reason we use $order = 43$ and tuning the γ and σ^2 on the entire training set using $tunelssvm$ method with 10 CV strategy and $simplex$ method. The resulting prediction on the test set are reported in Figure 17b. As we can observe, also in this case the performance using an optimized $order$ value is slightly better than the one using the suggested starting value as shown in Figure 17a. This means that the starting value was already a good trade-off value. We can observe in this case that the majority of error is made after time 60, when there is a change in amplitude of the signal. Also in this case the main problem is the different phase that our prediction has after that amplitude change.

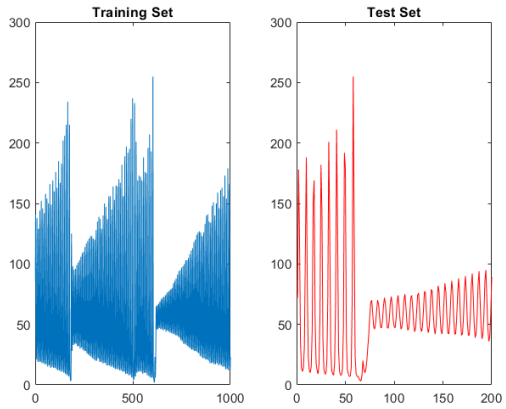
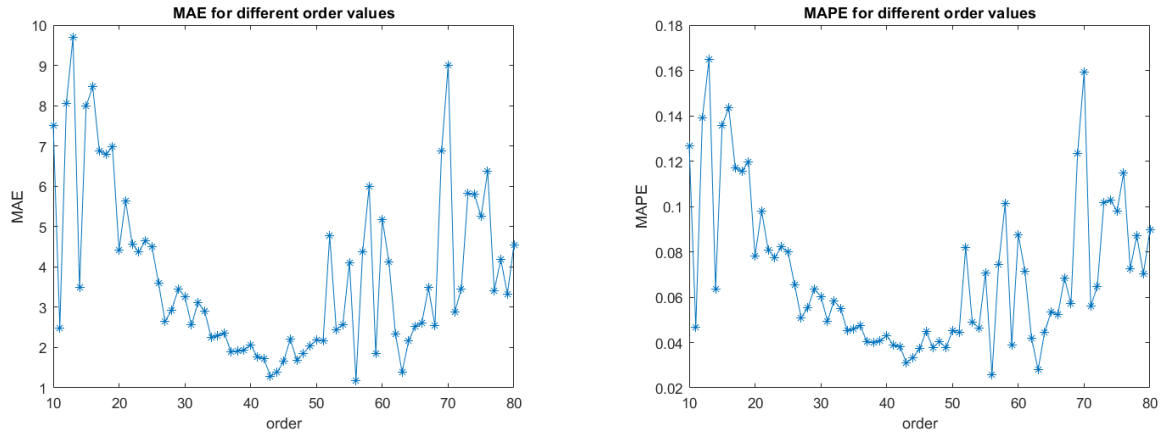
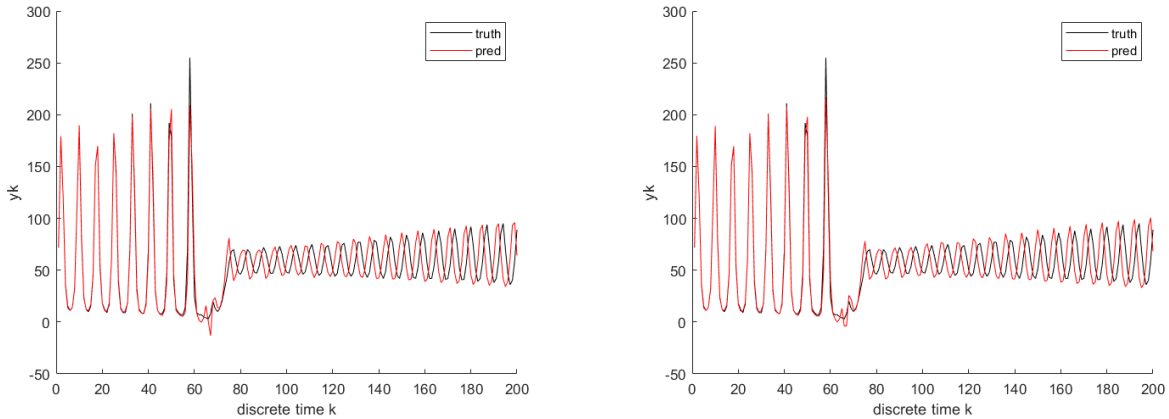


Figure 15: Santa Fe dataset with train (blue) and test (red) set



(a) MAE (b) MAPE

Figure 16: MAE and MAPE on the Santa Fe validation set, using different order values



(a) Order=50: $MAE = 15.8265, MAPE = 0.3253$ (b) Order=43: $MAE = 15.6787, MAPE = 0.3226$

Figure 17: Prediction on the Santa Fe test set using differet *order* values and consequently different γ and σ^2 values

Assignment 3: Unsupervised Learning and Large Scale Problems

Daniele Giannuzzi r0876755

1 Kernel principal component analysis

Kernel principal component analysis (KPCA) corresponds to linear PCA in a kernel-induced feature space, which is non-linearly related to the original input space. For the nonlinear PCA case the number of score variables can be larger than the dimension of the input space. In other words, the number of features in the kernel-induced feature space can be larger than the original one. Then, one selects as few score variables as possible and minimize the reconstruction error. However, the number of score variables is still limited by the number of input samples. The mapping from the score variables z to the reconstructed input variables x is done as:

$$\tilde{x} = h(z)$$

such that one minimizes the reconstruction error as in Equation 1, where N is the dimension of the input space:

$$\min \sum_{k=1}^{N} \|x_k - h\left(z_k\right)\|_2^2 \tag{1}$$

In practice, denoising can be done by moving the point in input space so that its corresponding map to feature space is optimized. This means that the data point in feature space is as close as possible with its corresponding reconstructed points using the principal components.

Kernel PCA has numerous applications: it can be used for feature extraction, denoising, dimensionality reduction and density estimation, among others. In this section, we will explore the use of kernel PCA for denoising. In order to illustrate the power of kernel PCA for denoising, this exercise focuses on an artificial example: a dataset which resembles the yin yang pattern. This example is mainly used to study the choice of the kernel and the number of components. We consider a dataset composed of 400 points, 200 per class. Noise was added to the generated data. We use Kernel PCA in order to reconstruct the original dataset without noise, trying several number of components $1, 2, 4, 6, 8, 10, 14, 18, 24$. The kernel hyperparameter σ^2 is kept fixed to 0.4 during all the experiments, as well as the data point dispersion (noise parameter) set to 0.3.

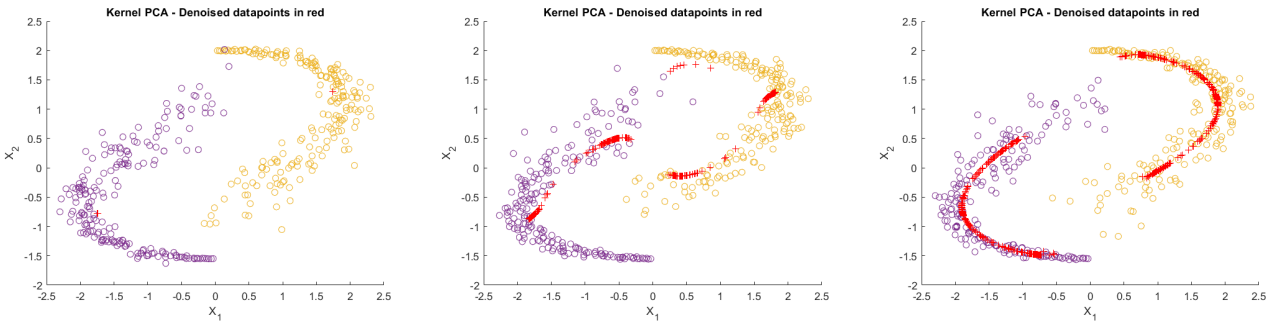
As we can observe from Figure 1 finding a good number of components is not trivial. Using too few components (such as 1 or 2) leads to a too strong reconstruction in which the reconstructed input space is very different from the original one, with many points concentrated in dense regions of the space. The extreme case is using 1 component, for which we have both classes condensed in 2 different points. From 4 to 8 components, the denoising performance is good, probably optimal with 4 components. We can notice how increasing the number of components gradually reduce the denoising. As a result, the reconstructed points are very close to the original ones. The extreme case is reported for 128 principal components selected, for which the reconstructed points overlap the original ones.

With linear PCA where the maximum is the number of score variables is as large as the dimension of the input space, therefore the number of principal components can be at most equal to the number of features. Furthermore, in case of linear PCA, the data points are projected onto linear axes with maximal variance. This means, as we can notice in Figure 2 that the data points reconstruction follows a line. Since the input space is not linear, the linear PCA does not show good performance.

In order to tune the number of components, being in unsupervised setting, one strategy could consist in plotting the explained variance percentage of individual components and the percentage of total (cumulative) variance captured by all principal components. By looking at this plot, we can easily decide how many components should be kept, which should be the ones that capture almost all the variance in the dataset. Then for kernel parameters and hyperparameters tuning, since we do not have labels, we should rely on a gridsearch, trying to minimize for example the MSE between the reconstructed and original data points.

2 Spectral clustering

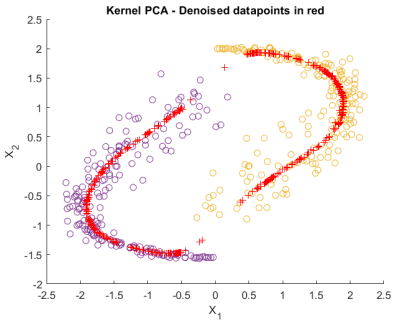
Spectral Clustering is a clustering algorithm which has performed better than many traditional clustering algorithms in many cases. It treats each data point as a graph-node and thus transforms the clustering problem into



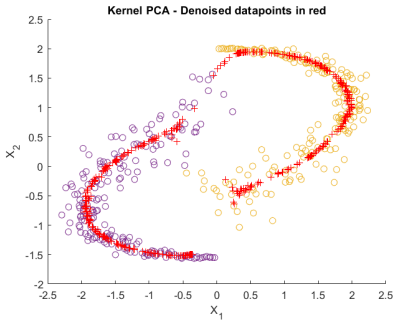
(a) 1 component

(b) 2 components

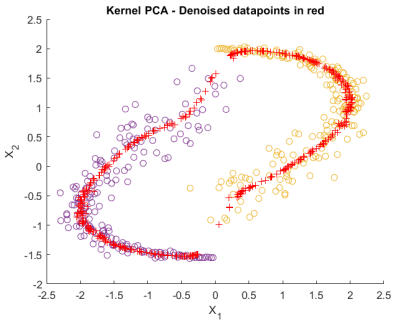
(c) 4 components



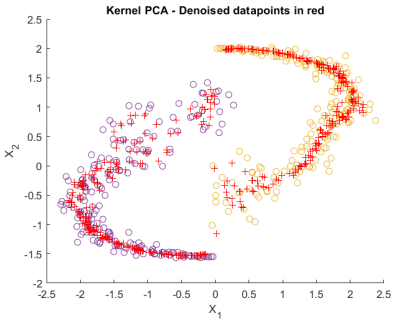
(d) 6 components



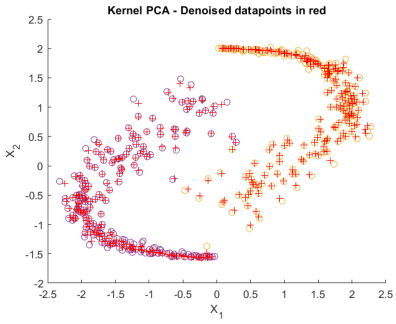
(e) 8 components



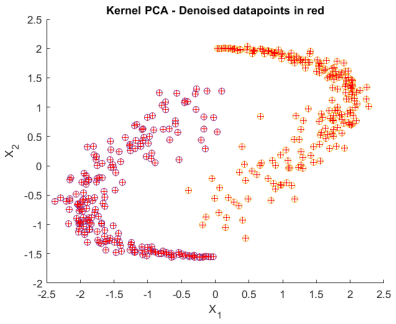
(f) 10 components



(g) 14 components



(h) 24 components



(i) 128 components

Figure 1: Kernel PCA: denoised data ('+') obtained by reconstructing the data-points ('o') using each time a different number of kernel principal components with the RBF kernel.

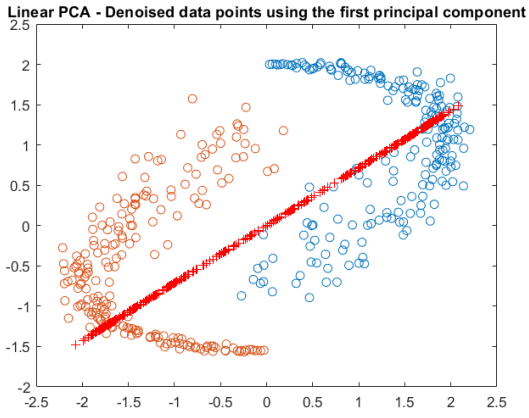


Figure 2: Linear PCA: denoised data ('+') obtained by reconstructing the data-points ('o')

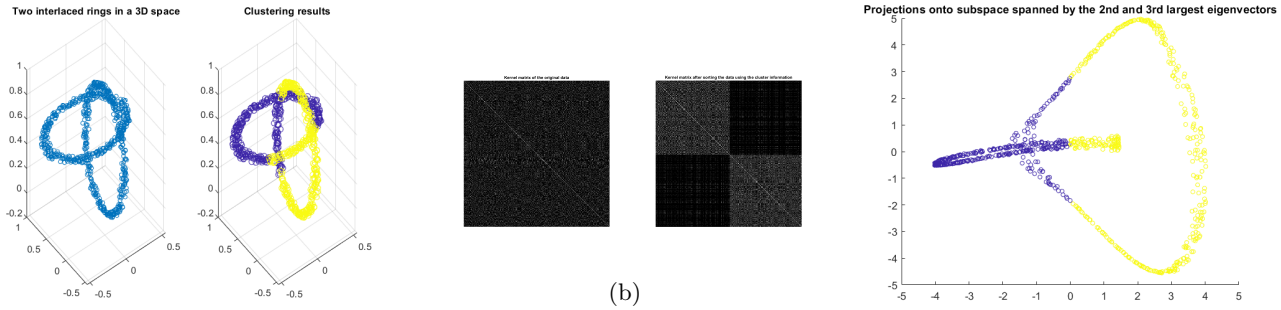
a graph-partitioning problem. A typical implementation consists of three fundamental steps:

1. Building the Similarity Graph: this step builds the Similarity Graph in the form of an adjacency matrix.
2. Projecting the data onto a lower Dimensional Space: this step is done to account for the possibility that

members of the same cluster may be far away in the given dimensional space. Thus the dimensional space is reduced so that those points are closer in the reduced dimensional space and thus can be clustered together by a traditional clustering algorithm. It is done by computing the Graph Laplacian Matrix .

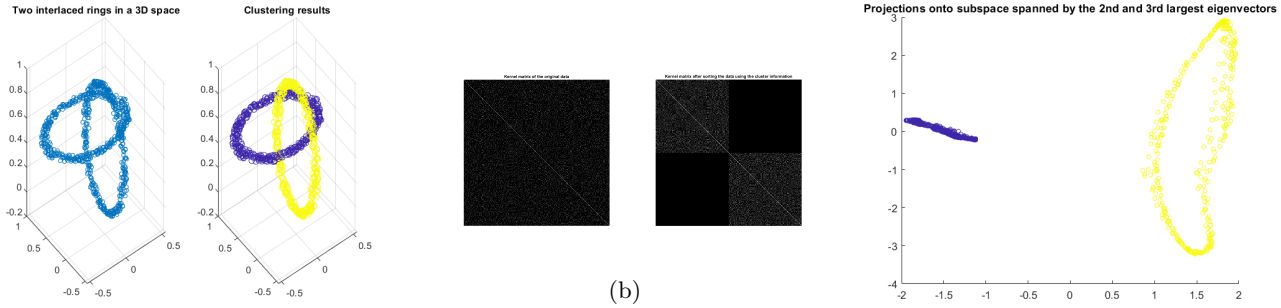
3. Clustering the Data: this process mainly involves clustering the reduced data by using any traditional clustering technique

The difference between classification and spectral clustering is that: classification objective is to find which class a new object belongs to form the set of predefined classes, while spectral clustering objective is to group a set of objects to find whether there is any relationship between them. In our experiment there are 2 clusters which corresponds to 2 rings in 3-dimensional space. The affinity matrix is computed using the RBF kernel. For this reason, the hyperparameter σ^2 can affect the performance of the clustering. We explored decreasing σ^2 values in Figures 3, 4, 5, 6, 7 with 3 plots per case, which represent: the 3D clustering result; the heatmap of the kernel matrix of the original data and kernel matrix after sorting the data using the clustering information; projections onto the subspace spanned by the 2nd and 3rd eigenvectors (the 1st eigenvector does not contain clustering information). As one can observe, when $\sigma^2 = 0.05$ the task is not well performed, being the two rings not well identified. This is due to the fact that the rings are projected into the 2D subspace in a way that is not optimal for their clustering. In all the other cases taken in exam, the projection clearly separates the two rings and for this reason also the clustering result is optimal.



(a) (b) (c)

Figure 3: $\sigma^2 = 0.05$



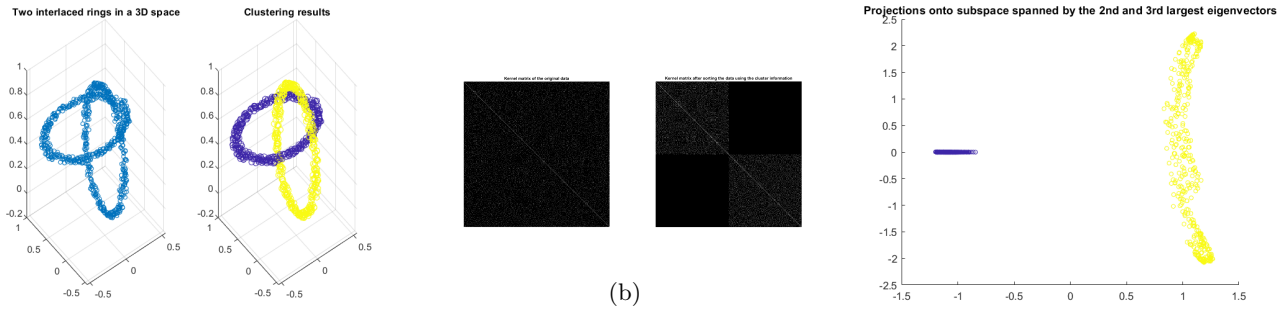
(a) (b) (c)

Figure 4: $\sigma^2 = 0.02$

3 Fixed-size LS-SVM

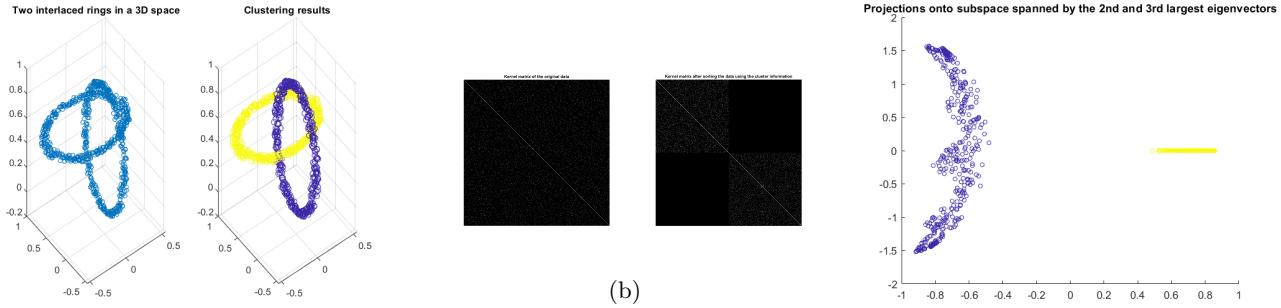
In large scale problems, to reduce the computational time, based on the Nystrom approximation, an approximation to the feature map can be obtained. This map can consequently be used to construct parametric models in the primal representation of the LS-SVM model. The approximation of the feature space is based on a fixed subset of data-points. One way to select this fixed-size set is to optimize the entropy criterion (kentropy) of the subset.

Therefore, a fixed size LS-SVM takes a working set of M support vectors and select vectors according to the entropy criterion (instead of a random subsample as in the Nystrom method). Typically, $M << N$. In practice, first of all a working set of M support vectors is chosen. Then, randomly a point from the training data pool is selected and replace one of the M support vectors chosen randomly. If the entropy increases that point is accepted as a support vector, otherwise it is rejected and returns in the training data pool.



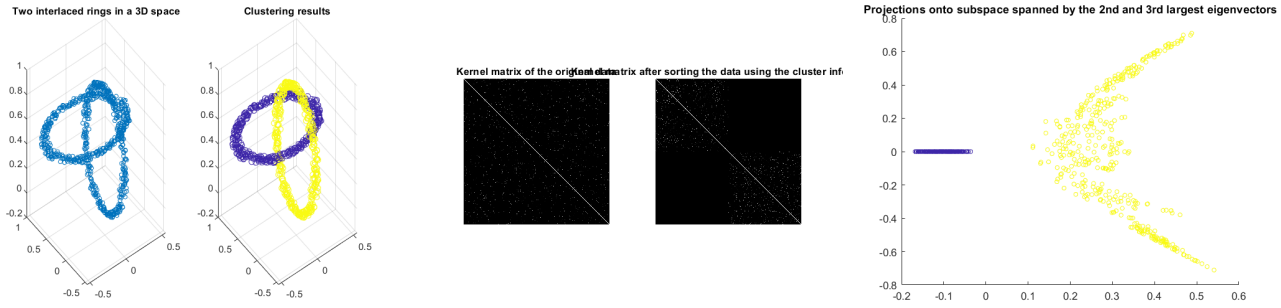
(a) (b) (c)

Figure 5: $\sigma^2 = 0.01$



(a) (b) (c)

Figure 6: $\sigma^2 = 0.005$



(a) (b) (c)

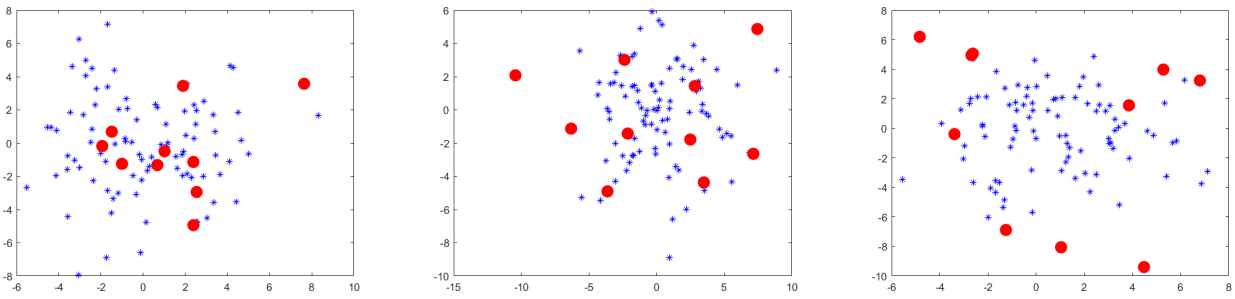
Figure 7: $\sigma^2 = 0.001$

One would be interested in solving a model in the primal representation when the number of samples in the dataset is huge. A solution in the dual space is more advantageous in case of large dimensional inputs, that translates in a dataset with a huge number of attributes. Here, as previously mentioned, we are working in primal space.

As we can observe in Figure 8, for low values of σ^2, say $\sigma^2 = 0.01$ the support vectors tend to be more close than for higher values. The extreme case is for $\sigma^2 = 100$ for which the support vectors tend to be very distant from each other.

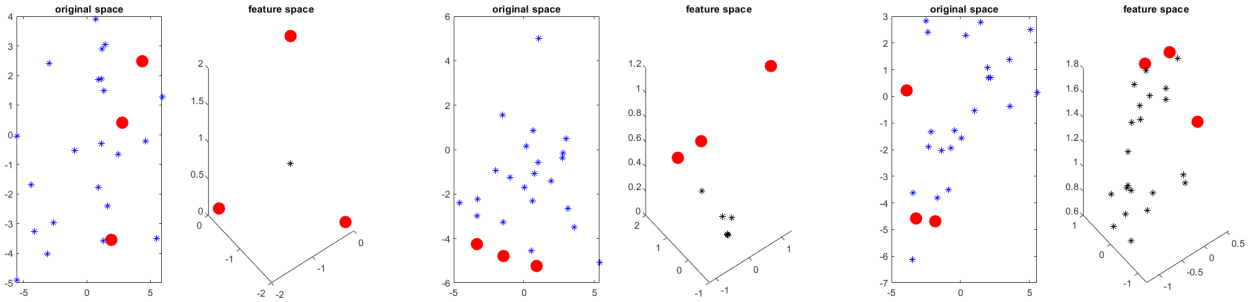
Using the Nystrom approximation method, the mapping of data to the feature space can be evaluated explicitly. This gives the features that one can use for a linear regression or classification. The decomposition of the mapping to the feature space relies on the eigenvalue decomposition of the kernel matrix. In Figure 9 we can observe the data-points in the found feature space and the differences in different values of σ^2. As one can observe, for lower values of σ^2, say $\sigma^2 = 0.01$ all the points stand in specific regions of the feature space, overlapping. On the other hand, for higher values of σ^2, for example $\sigma^2 = 100$, we can observe how all the points are distant in the feature space. Hence, one would like to find a trade-off in σ^2 so as to approximate with the few support vectors, the best possible solution.

Now, we compare the results of fixed-size LS-SVM to ℓ_0-approximation in terms of test errors, number of



(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 1$ (c) $\sigma^2 = 100$

Figure 8: $fixedsize\_script1$, fixed-size LS-SVM with different σ^2 values

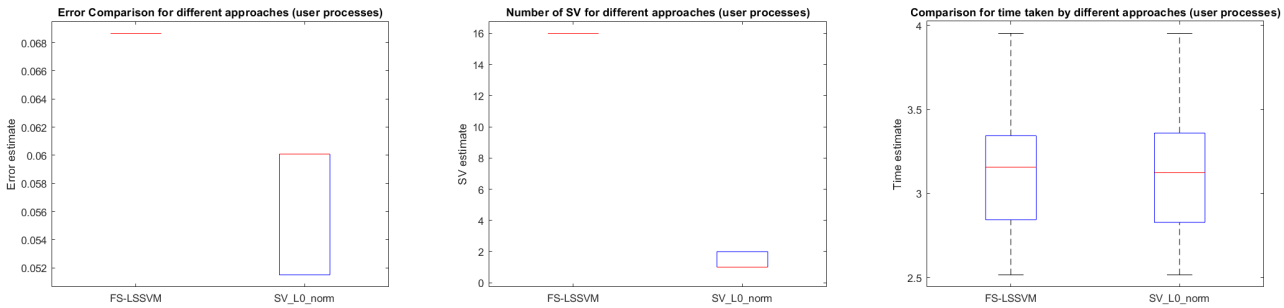


(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 1$ (c) $\sigma^2 = 100$

Figure 9: $fixedsize\_script2$: fixed-size LS-SVM with different σ^2 values

support vectors and computational time. The Figure 10 shows what we are interested about. We can see how the median error rate is very low for both methods, being in favor of the ℓ_0-approximation. As we can also expect, the number of suport vectors does not shows variability since it is fixed, while in ℓ_0-approximation there is variability and the number of used support vectors is also lower. In terms of computational time they show comparable performance.

We should highlight that results over runs of the $fslssvm\_script$ are different. Indeed, we can also have that FS-LSSVM performs better in terms of test error, and that the number of support vectors found is the same. The only thing that remains the same over different runs is the computational time that is always comparable.



(a) Test error (b) Number of support vectors (c) Computational time

Figure 10: $fslssvm\_script$: comparison of the results of fixed-size LS-SVM to ℓ_0-approximation

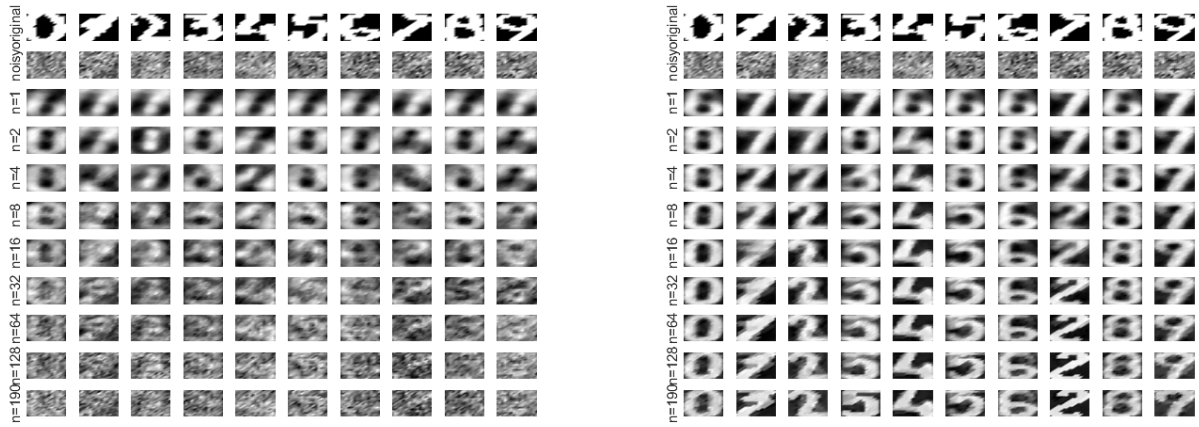
4 Homework problems

4.1 Kernel principal component analysis

In this homework problem, we do image denoising using kernel PCA. The dataset used in this exercise consists of images of handwritten numerals (0 to 9), extracted from a collection of Dutch utility maps. Approximately 20 patterns per class (digit 9 has 18 images, total of 198 patterns) have been digitized in binary images.

As a rule of thumb, σ^2 is calculated as the mean of the variances of each dimension times the dimension

(number of features) of the training data. In formula: $\sigma^2 = p * \mathbb{E}[\text{var}(x_i)]$, where p is the number of features and i the feature of interest varying from 1 to p. In our case $\sigma^2 = 35.9078$. In Figure 11 we illustrate the difference between linear and kernel PCA by giving an example of digit denoising for $noisefactor = 1.0$, in each row using a different number of principal components. We can observe how the performance of the linear PCA is very poor, with digits that are not correct most of the times. Moreover the performance degrades as the number of principal components increases. At the opposite, the Kernel PCA works in a better way, reaching a very surprising performance for $n = 64$ where n refers to the number of selected principal components. Indeed in this case we can recognize 9 out of 10 digits: only the digit '7' is not correctly identifiable. However, it is identifiable for lower values of n, but a worse denoising is computed for the other digits.

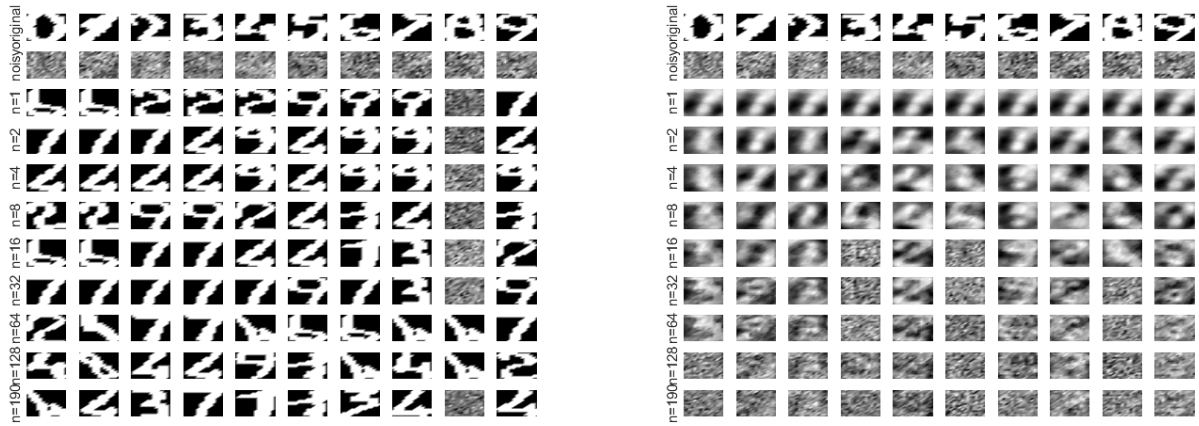


(a) Linear PCA denoising

(b) Kernel PCA denoising

Figure 11: difference between linear and kernel PCA with $noisefactor = 1$ and $sigmafactor = 1$ on the test set for handwriting recognition.

Now we try to scale σ^2 value, using the $sigmafactor$ parameter, in order to understand how much influence has this parameter on the results of kernel PCA. We try a $sigmafactor = 0.01$ and $sigmafactor = 100$. The results are respectively shown in Figure 12a and 12b. For low values of σ^2 the denoised images are very sharp, but they are most of the time denoised in a not proper way, since the digits we can recognize are not the ones we expect. On the other hand using values of σ^2 that are much higher, we have images that are not so denoised. As we can observe when number of principal components increases, the noise is still there, while for low number of components the denoising is not good since the digits that we car recognize are not correct. We can conclude that choosing a proper value for σ^2 is fundamental to have a good denoising.



(a) Kernel PCA: $sigmafactor = 0.01$

(b) Kernel PCA: $sigmafactor = 100$

Figure 12: kernel PCA with $noisefactor = 1$ on the test set for handwriting recognition.

4.2 Fixed-size LS-SVM

We now investigate the use of fixed-size LS-SVM for two additional datasets: the Shuttle dataset (classification) and the California housing dataset (regression).

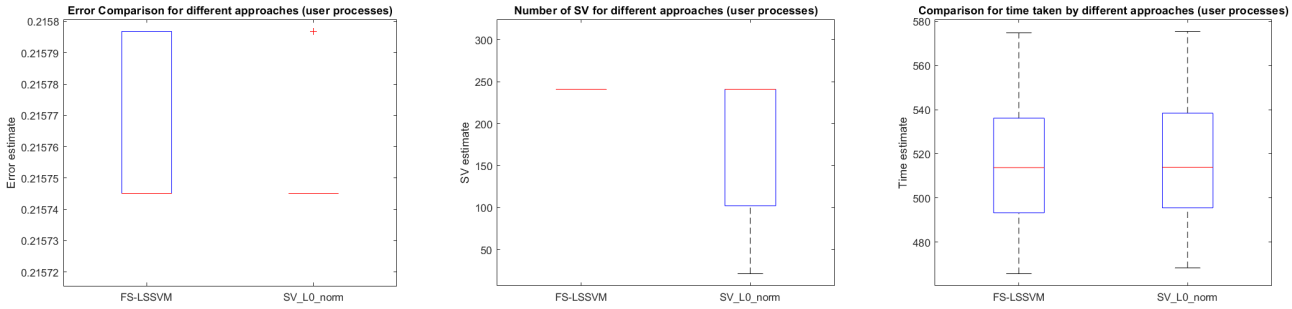
4.2.1 Shuttle dataset: classificaiton

The shutte dataset counts 58000 samples, 9 attributes and 7 classes highly unbalanced. In Table 1 we explored the classes of the dataset, with their count and percentage.

Class	Count	Percentage
1	45586	78.60%
2	50	0.09%
3	171	0.29%
4	8903	15.35%
5	3267	5.63%
6	10	0.02%
7	13	0.02%

Table 1: Shuttle dataset informations about classes

Now we use the same script we used before, $fslssvm\_script$, just changing the input (originally the subsampled shuttle dataset was used), to observe how the FS-LSSVM behaves when a large number of samples is given as input. The results are shown in Figure 13. In this case the FS-LSSVM has a higher median error rate compared to the sparser solution using the ℓ_0-approximation. This last solution has also variable number of support vectors which can be lower than the one of the FS-LSSVM which is fixed. The computational time appears instead to be quite similar for both approaches.

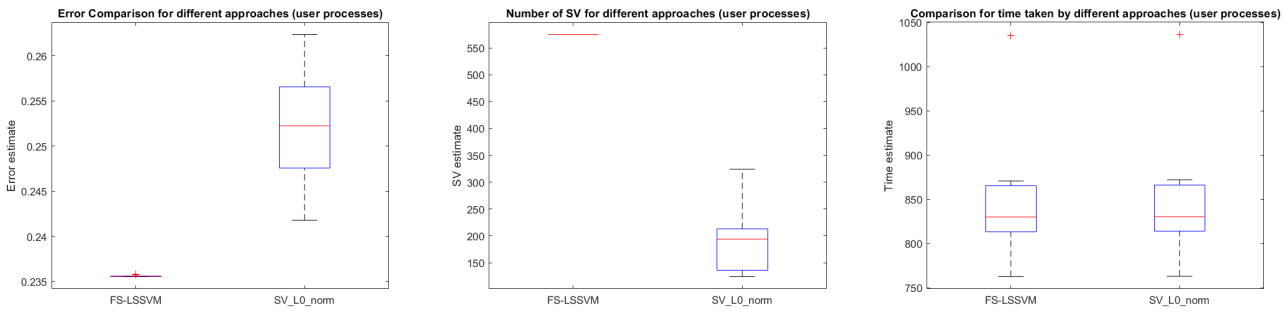


(a) Test error (b) Number of support vectors (c) Computational time

Figure 13: $fslssvm\_script$: comparison of the results of fixed-size LS-SVM to ℓ_0-approximation for the entire shuttle dataset

4.2.2 California housing dataset: regression

The california housing dataset counts 20640 samples with 9 attributes. The problem now regards regression. We use the same script that we used for the shuttle dataset in order to observe the difference in performance between the two approaches for a regression task. The experiment is represented in Figure 14. We can observe how the FS-LSSVM outperforms the ℓ_0-approximation on the test error, reaching always the same performance, while the ℓ_0-approximation performance varies for different runs. Moreover, the number of support vectors is again fixed for the the FS-LSSVM and again higher than the one of the other strategy, for which the mean is lower than a third of the FS-LSSVM. To conclude, regarding computation time, we can clearly observe again similar performance between the two strategies.



(a) Test error (b) Number of support vectors (c) Computational time

Figure 14: *fslssvm_script*: comparison of the results of fixed-size LS-SVM to ℓ_0-approximation for the entire california housing dataset