Support Vector Machines : Methods and Applications Assignments report

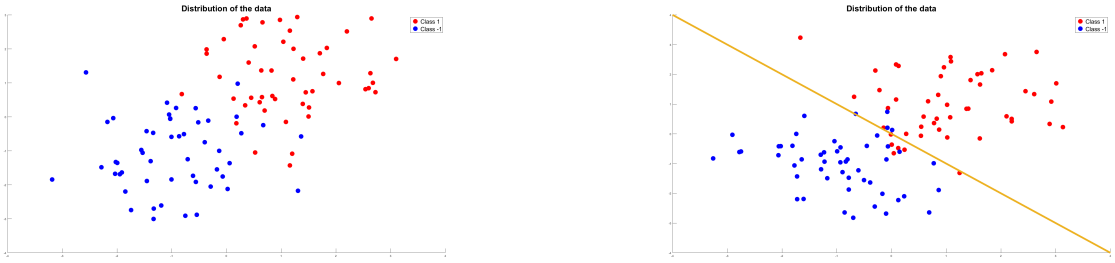
Deniz Soysal - r0875700

June 3, 2022

1 Assignment 1 : Classification

1.1 A simple example: two Gaussians

We consider two classes generated from Gaussian distributions. The dimension of the data is 2. The mean value of the first class, labelled as "1" is (1,1), while the mean value of the second class, labelled as "-1" is (-1,-1). The two classes have the same covariance matrix. The first class has 50 samples, while the second class has 51 samples. As we can see at Figure 1a, the data is not perfectly linearly separable. We will need to allow some misclassification. However, we can still try to come up with a line to classify the data, looking at the distribution. The mean of the class "1" is (1,1), while the mean of class "-1" is (-1,-1). So, a linear classifier that crosses the origin (0,0) should maximize the distance between the mean of each class and the classifier. This classifier is shown at Figure 1b.



(a) Distribution of the data

(b) Linear classification

Figure 1: Distribution of the data and linear classification

1.2 Support Vector Machine Classifier

Effect of adding more data points

In SVM, the separator hyperplane is found by maximizing the margin while requiring that all data points are correctly classified. The margin is defined as the distance between the hyperplane and the support vectors (data points closest to the hyperplane). These points are the ones used to find the hyperplane. We can also allow some misclassification with the parameter "C".

When adding more point, the classification boundary is updated if the added data point have an effect on the conditions mentionned above. If the data point added is far from the hyperplane and is already correctly classified (or is misclassified but still the maximum number of allowed misclassification is not exceded), the hyperplane will not be updated. At Figure 2, we can see the effect of adding non linearly classifiable data point when using a linear kernel with C=2.

(a) The misclassified are still allowed

(b) We exceed the limit number of allowed missclassification

Figure 2: Effect of adding more non linearly classifiable point

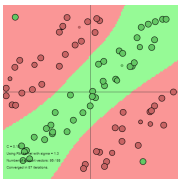
Role of C

As we have seen earlier, we can have cases where the distribution of the two classes are overlapping. Finding a classifier that perfectly separates the two classes becomes difficult. To deal with that, we can allow some misclassification. The original condition of SVM, i.e. all training data points are classified is given by [true label][output of

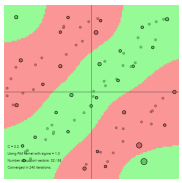
model]≥ 1, or more formally $y_k[w^T x_k + b] \geq 1$. This condition is satisfied when $w^T x_k + b > 0$ if $y_k = 1$ **and** when $w^T x_k + b < 0$ if $y_k = -1$, so when all training points are correctly classified[1]. To allow some misclassification, we introduce slack variables ξ_k, with the condition $\xi_k \geq 0$. We have as many slack variables ξ_k as training points. The condition become $y_k[w^T x_k + b] \geq 1 - \xi_k$. For data points that are not classifiable, we should tolerate misclassification by setting their corresponding ξ_k to > 1. The objective function is then defined as $min[\frac{1}{2} w^T w + c \sum \xi_k]$ under the condition $y_k[w^T x_k + b] \geq 1 - \xi_k$. So, the parameter "C" control the amount of misclassification we tolerate. Looking at this objective function, we can observe that :

- For low values of C, the model will focus on finding the hyperplane with the largest margin, allowing some misclassification.
- For high values of C, the model will focus on classifying correctly the data points, neglecting the margin.

We have try to visualize the effect of C using a RBF kernel. The results are shown at Figure 3. We can clearly see that at Figure 3a, we allow the misclassification of the two outside green points, while at Figure 3b the boundaries are more complex to classify all data points correctly.



(a) C=0.16



(b) C=3.2

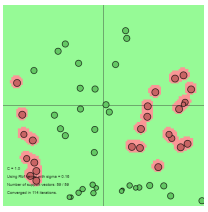
Figure 3: Effect of the parameter C for a RBF kernel with $\sigma = 1.3$

Role of sigma

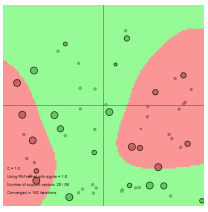
The Kernel coefficient σ define the distance of influence of a data point. For a radial basis function kernel, it is proportional to the variance of the Gaussian distribution.

- For high values of σ, further data points are considered as belonging to the same class. The decision boundaries tend to be linear. If σ is too high, we can have underfitting.
- For low values of σ, the data points need to be closer to be considered as belonging to the same class. The decision boundaries become highly non linear. If σ is too low, we can have overfitting.

This behaviour can be observed at Figure 4. We can see that $\sigma = 1.6$ is a good trade-off, while $\sigma = 0.16$ leads to overfitting and $\sigma = 7.9$ leads to a model not able to capture the data distribution. This shows the importance of **tuning the parameters** of a SVM.



(a) $\sigma = 0.16$



(b) $\sigma = 1.6$

(c) $\sigma = 7.9$

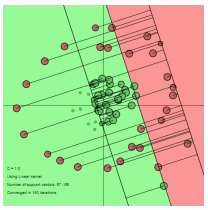
Figure 4: Effect of the parameter σ for a RBF kernel with C=1

[1] note that all these formulations are for the linear case

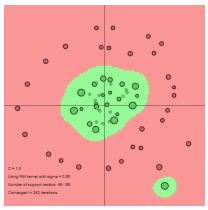
Linear kernel vs RBF kernel

We have already seen some result when using a RBF kernel (at Figure 3 and 4) and when using a linear kernel (at Figure 2). In fact, these kernels come from the *Kernel Trick*. Consider the linear SVM, given by $y(x) = sign(W^T + b)$. To define the non-linear SVM, we will map the data x onto a higher dimensional *feature space* Φ where the data will be linearly separable. So, the non-linear SVM is given by $y(x) = sign[W^T \Phi(x) + b]$. This has the advantage to allow to construct non-linear boundaries, but the disadvantage that to find the classifier we have to compute dot products in the space obtained after applying the feature map Φ, i.e. in the feature space which can be of a high dimension (sometimes infinite). To avoid that, we can use the *Kernel Trick*, which is a way of replacing the dot products in the feature space by a kernel function : $K(x, y) = \Phi(x).\Phi(y)$.

By solving the constraint optimization problem of the SVM we have defined in the section 1.2 : Role of C, we can obtain a Dual Representation of the SVM, i.e. a representation as a function of the Lagrangian multipliers. The dual formulation of the linear SVM is $y(x) = sign[\sum_{k=1}^{N} \alpha_k y_k x_k^T x + b]$. We can apply the Kernel Trick here, to obtain $y(x) = sign[\sum_{k=1}^{N} \alpha_k y_k K(x_k, x) + b]$. Based on the kernel K we choose, we can have different behaviour. For linearly classifiable data distributions, a linear kernel is enough. However, for highly non-linear data distributions, a linear kernel will not be powerful enough to classify the data, while a RBF kernel allows more flexibility. This can be observed at Figure 5, where the RBF kernel is performing better compared to the linear one (but note that at Figure 5b we observe some overfitting, due to not-tuned RBF parameters).



(a) Classification with a linear Kernel

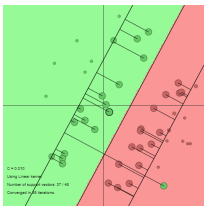


(b) Classification with a RBF kernel

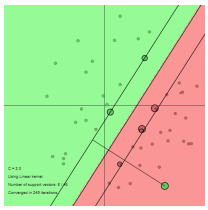
Figure 5: Comparison of linear and RBF kernels on non-linear data distributions

What is a support vector ?

In the dual formulation we have seen above, when finding the values for α_k (via a convex optimization problem), we obtain a sparse vector. So, it means that for the sum $y(x) = sign[\sum_{k=1}^{N} \alpha_k y_k K(x_k, x) + b]$, we will only take the sum over non-zero α_k values. The data points x_k related to a non-zero α_k are the support vectors. The classifier is expressed as these support vectors. At Figure 6, we can observe the support vectors as large circles with bold-lined circumference. The middle line correspond to the separator line, while the 2 parallel lines to that line correspond to the margin. We can see that all points inside the margin are support vectors.



(a) Visualization of the number of support vector when C=0.01



(b) Visualization of the number of support vector when C=2.0

Figure 6: Number of support vectors used for defining the hyperplane, for different values for C

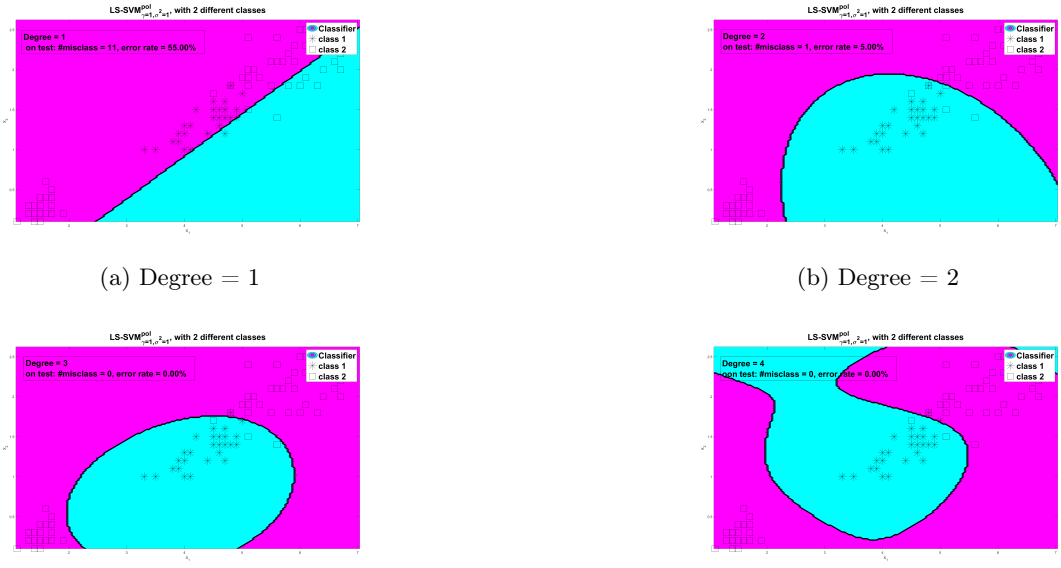
1.3 Least-squares support vector machine classifer

The formulation of SVM can become complex. Especially in function estimation, we end up with a high number of inequalities constraints. This is a drawback of SVM and explain why they are not so much used for function

estimation. LS-SVM address this problem by simplifying the constraints of the SVM. For the LS-SVM classifier, we use equalities constraints instead of inequalities constraints ($y_k[w^T x_k+b] = 1$ instead of $y_k[w^T x_k+b] \geq 1$) and square the slack variables so that they will be lower bounded by 0 without adding any constraint ($min[\frac{1}{2}w^T w + c\sum(\xi_k)^2]$ instead of $min[\frac{1}{2}w^T w+c\sum \xi_k]$). Note that after solving the constraint optimization problem in LS-SVM, we obtain $\alpha_k = \gamma e^k$. So, we do not have the sparsity property, but all data points are support vectors, with some data points having a high α while other having a low α, which can be seen as a "weight" in the final classifier[2]. Now, we will train a LS-SVM classifier on the Iris dataset. It is a two dimensional, two class dataset with 100 training samples, 67 of the "+1" class and 33 of the "-1" class and 20 test samples, 10 of the "+1" class and 33 of the "-1" class. We can already observe that the dataset is unbalanced : we have more "+1" samples in the training set.

Degree of polynomial

At Figure 7, we can observe the classification using a Polynomial Kernel. With Degree = 1, the behaviour is exactly the same as having a Linear Kernel. We can see that the Polynomial with a Degree=3 capture well the data distribution, while the one with Degree=4 start to become too complex.



(a) Degree = 1

(b) Degree = 2

(c) Degree = 3

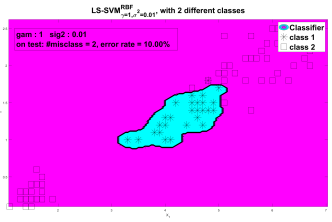
(d) Degree = 4

Figure 7: Classification of the two classes using a LS-SVM with a polynomial kernel for different degree

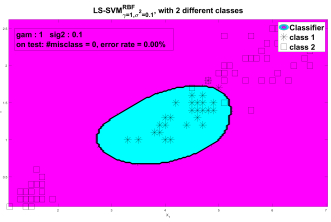
Tuning of kernel parameter σ^2

At Figure 8 we can observe the classification using a RBF kernel. We can see that for $\sigma^2 = 0.01$ the model is too complex and overfits the data, while $\sigma^2 = 10$ and $\sigma^2 = 1$ underfit the data. Using $\sigma^2 = 0.1$ seems to be a good trade-off.

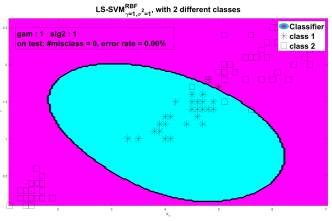
[2]Note that with pruning the lowest α we can introduce sparsity



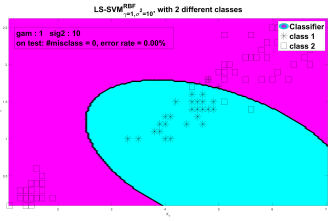
(a) $\sigma^2 = 0.01$



(b) $\sigma^2 = 0.1$



(c) $\sigma^2 = 1$

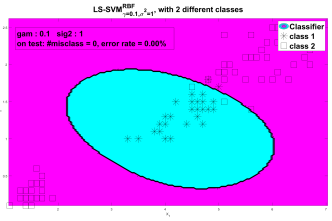


(d) $\sigma^2 = 10$

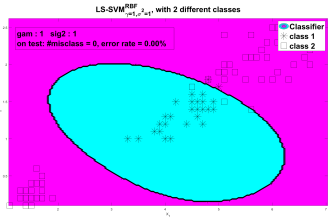
Figure 8: Classification boundaries for a LS-SVM using a RBF kernel with $\gamma = 1$ and different values for σ^2

Tuning of regularization constant γ

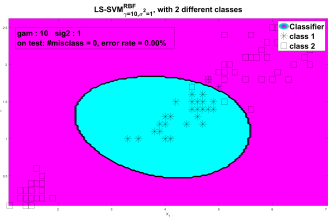
At Figure 9 we can observe the classification using a RBF kernel with $\sigma^2 = 1$ and different values for γ. We can see that for $\gamma = 10$ or 250 is well-suited to our distribution (note that this value depends on our choice of $\sigma^2 = 1$).



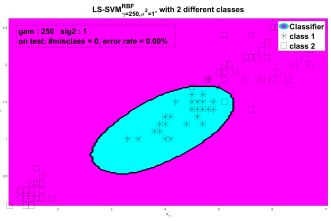
(a) $\gamma = 0.1$



(b) $\gamma = 1$



(c) $\gamma = 10$



(d) $\gamma = 250$

Figure 9: Classification boundaries for a LS-SVM using a RBF kernel with $\sigma^2 = 1$ and different values for γ

1.3.1 Tuning parameters using validation

Random split validation

The idea of *Random split validation* is simply to split the dataset onto a training set and a validation set randomly, train the model on the training set and validate it on the validation set. Here, we use 80% of the dataset for the training and 20% for the validation. At Table 1, the cost values for a range of values for γ and σ^2 are presented. We can see that for $\gamma = [10, 100]$ and $\sigma^2 = 0.1$, the cost is 0. The disadvantage of this technique is that sometimes

we can ends up with a validation set containing "easier" samples to classify, which will leads to a high validation accuracy which will not really be representative of the actual generalization performance of the classifier. We have noticed that the performance obtained was highly different when running the script multiple times. One way to deal with that is using **cross-validation**.

γ \ σ^2	0.001	0.01	0.1	1	10
0.001	0.30	0.25	0.35	0.30	0.50
0.01	0.30	0.30	0.35	0.25	0.20
0.1	0.60	0.30	0.05	0.00	0.15
1	0.20	0.15	0.00	0.05	0.05
10	0.20	0.10	0.00	0.10	0.10
100	0.50	0.10	0.00	0.00	0.00
1000	0.40	0.00	0.05	0.00	0.15

Table 1: validation cost using the random split technique (split=0.8) for different values of γ and σ^2

10-fold cross validation

The idea of k-fold cross-validation is to divide the dataset into k folds, leave apart one of the folds and train the classifier on the *k-1* folds, then evaluate the classifier on the left apart fold. We repeat this operation "k" times (each fold is left apart one time). The advantage of k-fold cross validation is that at the end we will compute a mean accuracy over all the iterations, so we don't have the same problem as in *Random split validation* where we can end up with a "easy" validation set. However this method has also the disadvantage that it is computationally expensive (we train the classifer "k" times). At Table 2, we can observe the cost values for a range of values for γ and σ^2 using 10-fold cross validation. The results are more realistic (and pessimistic) than Table 1, but again $\gamma = [10, 100, 1000]$ and $\sigma^2 = 0.1$ leads to a low cost. We have also observed that the results are most consistent when running the script multiple times. It is important to remember that the training set is unbalanced : we have 67 "+1" samples and 33 "-1" samples. In unbalanced cases, one can prefer to use *stratified cross validation*, where we force that each fold will have the same partition of "+1" samples, i.e. 67%. This is not done here.

γ \ σ^2	0.001	0.01	0.1	1	10
0.001	0.33	0.33	0.33	0.33	0.33
0.01	0.33	0.33	0.33	0.33	0.33
0.1	0.33	0.21	0.04	0.05	0.33
1	0.20	0.06	0.04	0.05	0.05
10	0.18	0.05	0.04	0.05	0.05
100	0.17	0.05	0.04	0.05	0.05
1000	0.18	0.05	0.04	0.04	0.04

Table 2: validation cost using the 10-fold cross validation technique for different values of γ and σ^2

γ \ σ^2	0.001	0.01	0.1	1	10
0.001	0.33	0.33	0.33	0.33	0.33
0.01	0.33	0.33	0.33	0.33	0.33
0.1	0.33	0.20	0.04	0.05	0.33
1	0.18	0.05	0.04	0.05	0.05
10	0.18	0.05	0.04	0.05	0.06
100	0.18	0.05	0.04	0.04	0.04
1000	0.18	0.05	0.05	0.04	0.05

Table 3: validation cost using the leave one out technique for different values of γ and σ^2

Leave one out cross validation

In the case where the training set is small, one can prefer using leave one out cross validation, which a special case of cross validation where k = number of training samples. This way, we only leave one sample left apart at each iteration and train the classifier on the rest of the data. The advantage is that the estimation of the performance of the classifier will be less pessimistic : indeed, in cross-validation, we train at each iteration the classifier on a subset of the training set, which when the training set is small can lead to a lower estimate of the actual performance because generally using more data for the training leads to better performance. In these cases, Leave-one-out can give a more realistic estimate of the performance . However, the disadvantage is that we train the classifier N times, N being the size of the training set. The result are shown at Table 3, where we can observe that the result are very similar to Table 2.

1.3.2 Automatic parameter tuning

Even if solving SVM and LS-SVM is a convex optimization problem, the tuning of the parameters is not a convex problem[3] and we need algorithms to find a selection of parameters that results in a local minima for the cost function which is hopefully close enough to the global minima. We will compare two algorithms for tuning the parameters of a classifier : Nelder-Mead simplex algorithm, a derivative-free method to approximate a local optimum and Grid Search, which is simply the idea of training the classifier for a range of parameters and choosing the one with the lowest error. The results are presented at Table 4 and 5. We can observe that the minimal cost is similar for the two methods and also at each run. However, the parameter are highly different with regards to the method and changes also at each run for the same method ! This shows that we have many local minimas. We have also observed that the gridsearch method takes more time.

Method	Run ID	γ	σ^2	cost value
Simplex	1	8.2675	0.3682	0.04
Simplex	2	30.0056	0.524885	0.04
Simplex	3	9.9938	0.064402	0.03

Table 4: Tuning of the LSSVM using the simplex method for 3 different run (the cost function is crossvalidatelssvm)

Method	Run ID	γ	σ^2	cost value
Gridsearch	1	0.087483	0.12013	0.04
Gridsearch	2	1.4228	0.034115	0.04
Gridsearch	3	129.3096	0.07070565	0.04

Table 5: Tuning of the LSSVM using the gridsearch method for 3 different run (the cost function is crossvalidatelssvm)

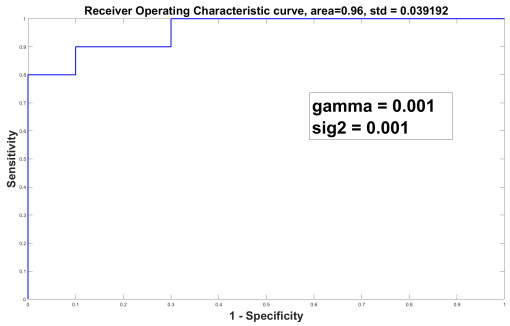
1.3.3 Using ROC curves

Using accuracy to evaluate a system is not always appropriate : with accuracy we assume a uniform cost between the False Positive and the False Negative. Moreover, accuracy is unstable, depending a lot on the balance of the classes in our training-validation split. One can prefer to do a ROC analysis. The ROC curve plots the true positive (positive predictions which are actually positive) rate against the false positive (positive predictions which are actually negative) rate. The perfect case is TP = 1 and FP = 0. So classifiers with a ROC curve that goes up quickly (in other terms that maximize the area under the ROC curve) will be better. At Figure 10 we compare the ROC curves for a LS-SVM with a RBF kernel. The parameters are $\gamma = 0.001$ and $\sigma^2 = 0.001$ at Figure 10a and $\gamma = 100$ and $\sigma^2 = 0.1$ at Figure 10b. We can observe that the ROC curve at Figure 10b is steeper, so the corresponding classifier is preferred. We compute the ROC curve on the test set rather than on the training set to estimate the generalization performance of the model, to see how it will perfom on unseen data.

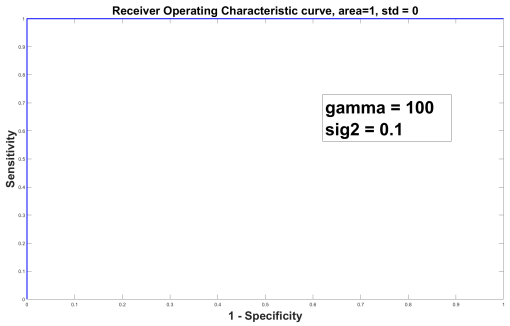
1.3.4 Bayesian Framework

We can use Bayesian inference to estimate the posterior class probabilities. In the Figure 11, the color scale correspond to the probability of belonging to the positive class. We can observe the influence of the parameters : The classifier at Figure 11a captures more the distribution of the data compared to the one at Figure 11b.

[3]But note that in comparison with MLP, we have much less parameters to tune (in MLP, we have to tune all the weights)

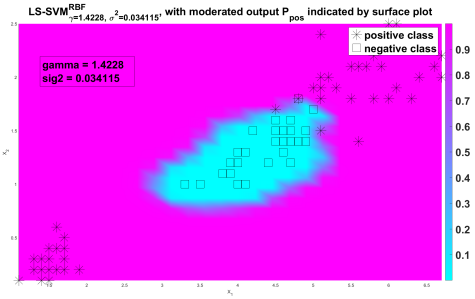


(a) ROC curve for $\gamma = 0.001$ and $\sigma^2 = 0.001$

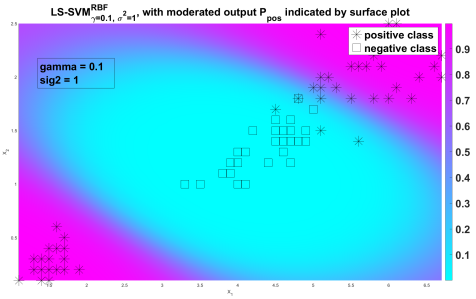


(b) ROC curve for $\gamma = 100$ and $\sigma^2 = 0.1$

Figure 10: ROC curves



(a) Bayesian Inference with LS-SVM with parameters γ =1.4228 and $\sigma^2 = 0.034115$



(b) Bayesian Inference with LS-SVM with parameters γ =0.1 and $\sigma^2 = 1$

Figure 11: Bayesian Inference with LS-SVM

1.4 Homework problems

1.4.1 Ripley dataset

The dataset is two dimensional with two classes, +1 and -1. We have 250 training data points with 125 data points for each class and 1000 test data points, with 500 data points for each class. We will use the LS-SVM to classify the two classes. To tune the parameters, we will use a grid search 10-fold cross validation. At Figure 12, we can observe the training and test distribution of the dataset. We observe that the classes are overlapping, and that we have a mixture of two decision boundaries. Therefore we will need a non-linear classifier, and we except that the RBF kernel will perform better than the other ones. At Figure 13, we can observe the decision boundaries of a LS-SVM for different kernels. The LS-SVM with RBF kernel is the one that capture the best the data distribution. This can also be observed at the error rate at Table 6 and Figure 14, where we see that the LS-SVM with a RBF kernel has a higher AUC-ROC[4] and a lower error rate.

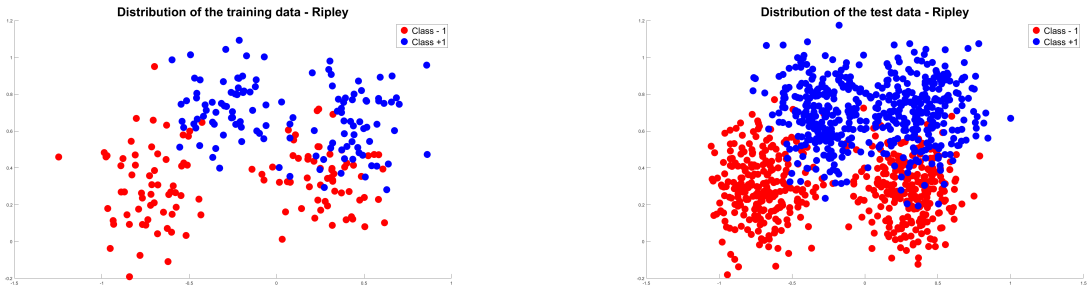
[4]area under the ROC curve

Kernel	# missclassied	Error rate [%]	AUC-ROC
Linear	108	10.80	0.959
Poly	99	9.90	0.96026
RBF	95	9.50	0.96842

Table 6: Results on a test set of size 1000 - Ripley dataset

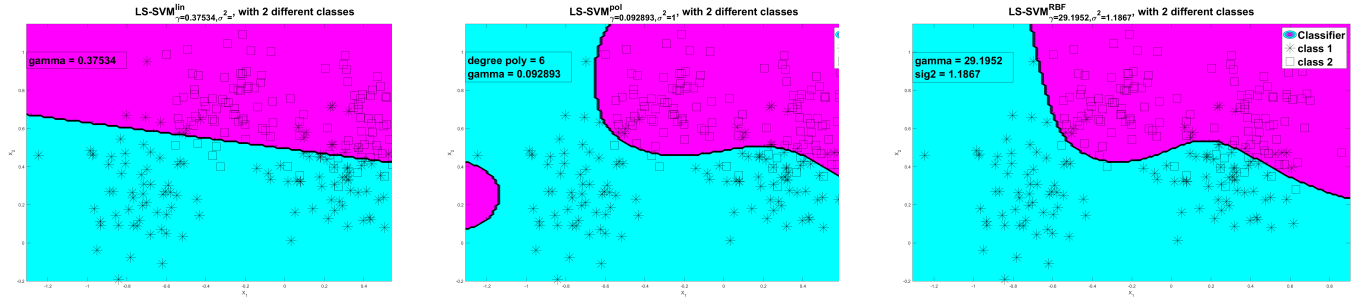
Kernel	# missclassied	Error rate [%]	AUC-ROC
Linear	7	4.14	0.99563
Poly	11	6.51	0.97483
RBF	4	2.37	0.99563

Table 7: Results on a test set of size 169 - Wisconsin Breast Cancer dataset



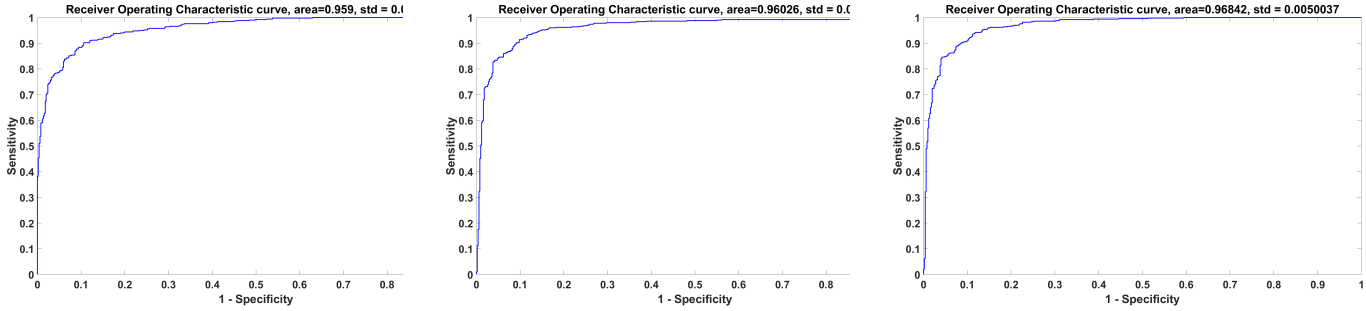
(a) Training set (b) Test set

Figure 12: Distribution of the training and test set for the Ripley dataset



(a) Classification with Linear Kernel (b) Classification with Polynomial Kernel (c) Classification with RBF Kernel

Figure 13: Classification of the Ripley dataset with a LS SVM for different kernels

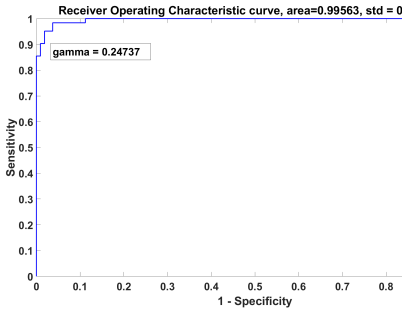


(a) ROC curve with Linear Kernel (b) ROC curve with Polynomial Kernel (c) ROC curve with RBF Kernel

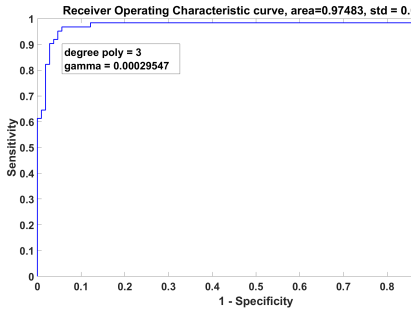
Figure 14: ROC curves for different kernel - Ripley dataset

1.4.2 Wisconsin Breast Cancer dataset

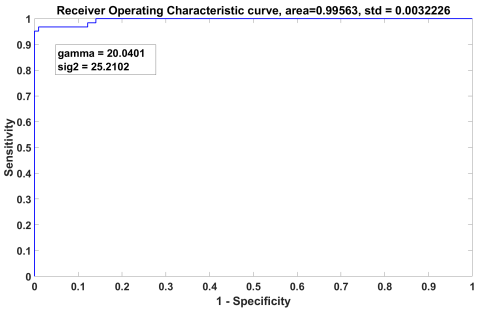
This time, the data is of higher dimension : 30 features. We have 400 training data points, with 250 of the "-1" class and 150 of the "+1" class and 169 test data points, with 107 of the "-1" class and 62 of the "+1" class. So the dasaset is unbalanced. The data is high dimensional so it will be difficult to visualise it. However, one way of doing it would be to use PCA and only keep the first 2 component (or 3), which is not done here. At Figure 15 and Table 7, we can observe the performance of the LS-SVM for different kernels. We observe that the RBF kernel and the Linear Kernel performs very well. This can be interpreted via the high dimensionality of the data : 30 features for only 400 training samples, which makes it quite likely that a linear hyperplane will be enough to separate the two classes.



(a) ROC curve with Linear Kernel



(b) ROC curve with Polynomial Kernel

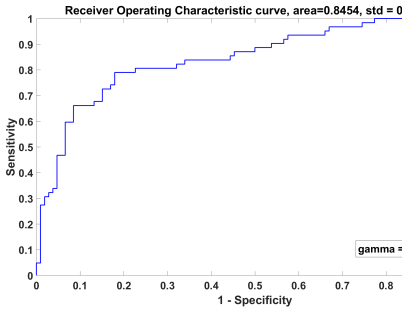


(c) ROC curve with RBF Kernel

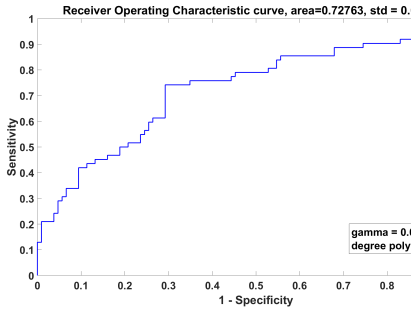
Figure 15: ROC curves for different kernel - Wisconsin dataset

1.4.3 Diabetes dataset

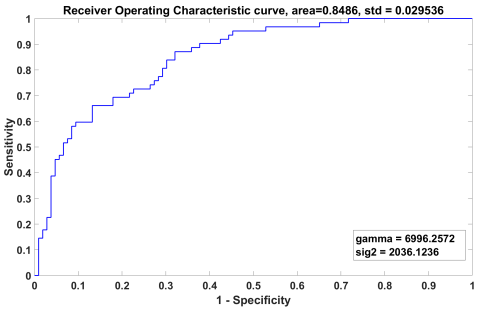
The diabetes dataset is of dimension 8. We have 300 training samples, with 205 "-1" samples and 95 "+1" samples and 168 test samples, with 106 "-1" samples and 62 "+1" samples. Thus again the dataset is unbalanced. As for the breast dataset, the data is high dimensional so it will be difficult to visualise it. At Figure 16 and Table 8, we can observe that the RBF kernel and the Linear kernel performs well.



(a) ROC curve with Linear Kernel



(b) ROC curve with Polynomial Kernel



(c) ROC curve with RBF Kernel

Figure 16: ROC curves for different kernel - Diabetes dataset

Kernel	# missclassied	Error rate [%]	AUC-ROC
Linear	38	22.62	0.8454
Poly	50	29.76	0.7276
RBF	35	20.83	0.8486

Table 8: Results on a test set of size 168 - Diabetes dataset

1.4.4 Improve performance in case of unbalance

We have seen that the Wisconsin Breast Cancer datase and the Diabetes dataset are unbalanced. For unbalanced data, we can introduce a weight per class in the objective function, by increasing the weight of the underrepresented class and/or decreasing the weight of the overrepresented class. Another way to deal with unbalance is to use the Bayesian framework and change the prior class distribution to match the training data distribution. Stratified cross validation can also be used to ensure that the proportion of the class are the same for the different folds.

2 Assignment 2 : Function Estimation and Time Series Prediction

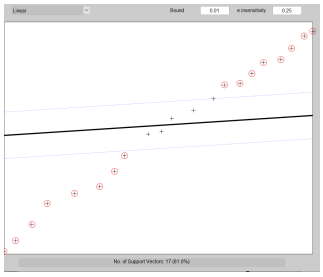
For function estimation with SVM, we use the ϵ-insensitive loss function. In the linear case, this leads to a model in the form $\hat{y} = w^T x + b$, where we have to find w and b under the constraint that all point are within the ϵ tube, i.e. $|y_i - \hat{y_i}| \leq \epsilon$. Similarly to classification, we can introduce slack variables ξ to allow that some datapoints are outside of the ϵ tube. To have non-linear regression, we have to work in the feature space $\Phi(x)$ or we can apply the Kernel Trick.

2.1 Support vector machine for function estimation

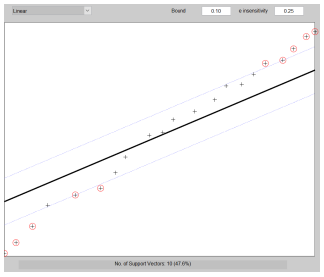
2.1.1 Linear Data

Effect of the bound

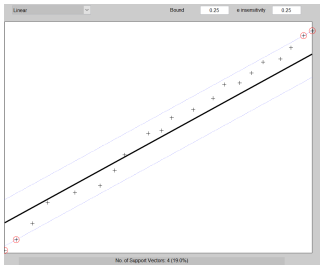
At Figure 17, we can observe the result of the regression with SVM for several values of the bound. We can observe that the bound is a regularization parameter : it limits the value of the weights. Indeed, at Table 9, we can see that lower the bound is, lower will be the value of the weight. Remember the formulation of the linear SVM for regression : $\hat{y} = w^T x + b$. Therefore, the weight is the slope of the regression line ! If we impose hard constraint on the weight value, we will not able to fit a linear distribution with a high slope.



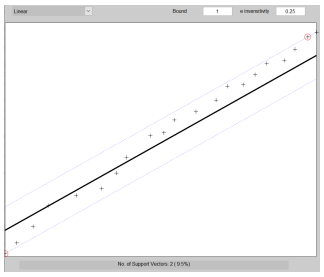
(a) bound=0.01



(b) bound=0.1



(c) bound=0.25



(d) bound=1

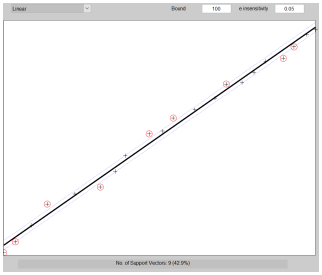
Figure 17: Regression with SVM, for ϵ=0.25 and different values for the bound

Bound	$\|w_0\|^2$	Number of support vectors
0.01	0.011515	17
0.10	0.493776	10
0.25	0.828792	4
1.00	0.872981	2

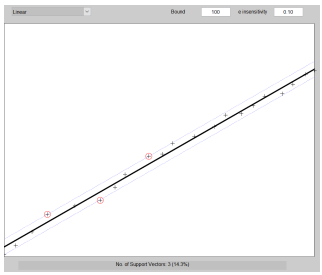
Table 9: Value of the weights and number of support vectors for different bound values (ϵ=0.25)

Effect of ϵ

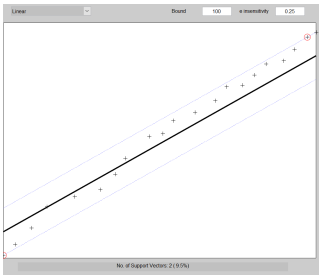
When we learn w and b, we impose that all point are within the ϵ tube, i.e. $|y_i - \hat{y}_i| \leq \epsilon$ (we can also allow some point outside of the tube with slack variables). By solving the Karush-Kuhn-Tucker (KKT) conditions, we find that all data points inside the epsilon-tube have a lagrangian multiplier $\alpha = 0$. So, the support vectors are all the points outside of the ϵ tube ! Therefore, the parameter ϵ will have a strong effect on the regression line. The larger ϵ will be, larger errors we will tolerate : the model will be simpler, and we will use only a few support vectors. For smaller ϵ values, the number of support vectors will be higher. At Figure 18, we can observe the result of the regression with SVM for several values of ϵ. At Table 10, we can observe that for lower value of ϵ, the number of support vectors increases.



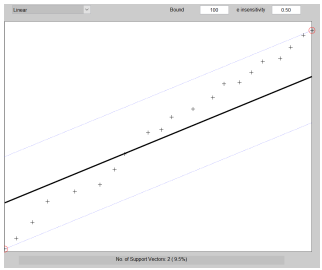
(a) $\epsilon = 0.05$



(b) $\epsilon = 0.1$



(c) $\epsilon = 0.25$



(d) $\epsilon = 0.50$

Figure 18: Regression with SVM, for bound = 100 and different values for ϵ=0.25

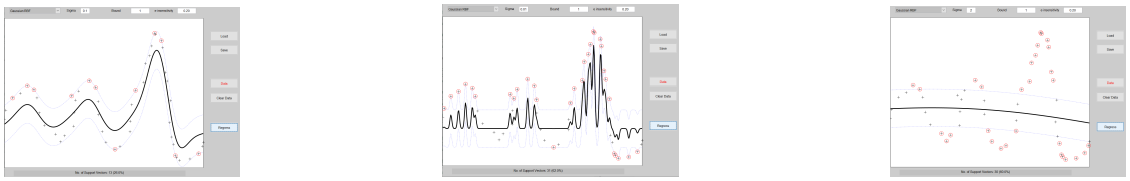
ϵ	$\lvert w_0 \rvert^2$	Number of support vectors
0.05	1.341234	9
0.10	1.302369	3
0.25	0.872981	2
0.50	0.468242	2

Table 10: Value of the weights and number of support vectors for different ϵ values (bound=100)

2.1.2 Non Linear Data

Now, we define the data as being non-linear. Using a Linear Kernel will not be enough to find a regression function that fits the data. In these cases, a RBF kernel is often preferred. It is important to tune the parameters of the RBF kernel. At Figure 19, we can observe the regression results with a SVM using a RBF kernel with different parameters. We can observe that Figure 19a fits well the data, while Figure 19b overfits and Figure 19c underfits.

SVM regression vs Least Squares Fit



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

Figure 19: SVM regression with a RBF kernel for different parameters

The main advantage of SVM regression over Least Squares Fit is that we only use a subset of the data to define our regression line. The lost function is also different : we use a ϵ-insensitive loss, against a L2 loss for Least Squares Fit.

2.2 A simple example: the sinc function

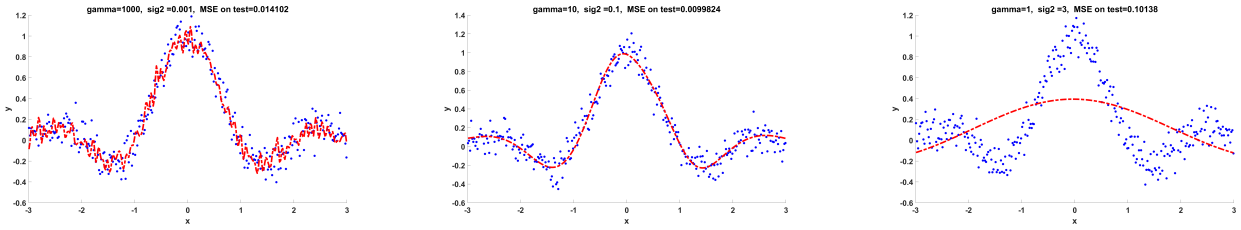
Now, we will use the LS-SVM to perform regression.

2.2.1 Regression of the sinc function

In this exercise, we construct a LS-SVM regression model with the RBF kernel. We will first tune the parameters manually, then use algorithms to tune them automatically.

Manual tuning of the parameters

We try several values for γ and σ, plot the prediction on the test set and report the MSE. At Figure 20, the results are depicted. We can clearly observe that Figure 20b fits the best the data, while Figure 20a overfits and Figure 20c underfits. This can also be observed at Table 11, where σ=0.10 and γ=10 gives the lowest MSE.



(a) $\sigma = 0.001, \gamma = 1000$ (b) $\sigma = 0.1, \gamma = 10$ (c) $\sigma = 3, \gamma = 1$

Figure 20: LS-SVM regression of the sinc with a RBF kernel for different parameters

σ	γ	MSE
0.001	1000	0.014102
0.10	10	0.0099824
3	1	0.10138

Table 11: MSE for the different σ and γ combination

Automatic tuning of the parameters

The *tunelssvm* method allows to automatically tune the LS-SVM. We use leave one out cross validation and compare the Grid Search method and the Simplex method. As we have seen in Assignment 1, the Nelder-Mead simplex

algorithm is a derivative-free method to approximate a local optimum and Grid Search is simply based on the idea of training the model for a range of parameters and choosing the one with the lowest error. The result are shown at Figure 21. We can observe that the results are similar, and the MSE is around 0.010 for both methods. However, it is important to note that the results are not consistent when running the script multiple times. Indeed, even if solving SVM and LS-SVM is a convex optimization problem, the tuning of the parameters is not a convex problem : we need to select parameters that results in a local minima for the cost function which is hopefully close enough to the global minima. This explains why the results varies a lot between different runs. We have also observed that the gridsearch method takes more time.

(a) Using Grid Search (b) Using Simplex

Figure 21: LS-SVM regression, tuning the parameters using Grid Search and Simplex

2.2.2 Application of the Bayesian framework

Here, we will use the Bayesian Framework to tune and analyse the LS-SVM regressor. The idea is to find the parameters that will maximize the posterior probability of the parameters given the data. We have 3 level of inference in the Bayesian Framework. Applied to LS-SVM, these becomes the Level 1, where we tune the parameters α and b ; The Level 2 where we tune the regularization constants ; The Level 3 where we tune the kernel parameters (i.e. σ of the RBF kernel). In LSSVMlab, we can use the command bay_lssvm to compute the posterior at each Level, and bay_optimize to optimize the parameters for each Level. After tuning, we can plot the regression function. The result is depicted at Figure 22.

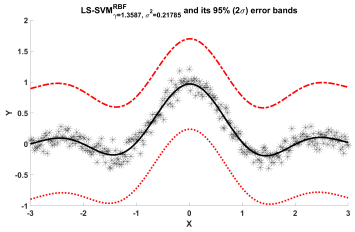


Figure 22: Regression using the Bayesian Framework to tune the parameters

2.3 Automatic Relevance Determination

The Bayesian Framework can also be used to select the most relevant inputs. With the command bay_lssvmARD in LSSVMlab, we assign a weight to each dimension of the kernel. Then, we use the 3rd level of the Bayesian Framework to find the most relevant inputs. To verify the results, we have divided the dataset into 80% of training samples and 20% of test samples. Then, we tune and train a LS-SVM by using only one input, for each of the input. The results are presented at Table 12. We can see that the results are coherent : we find that X_1 is the most relevant input, and it is the one that leads to the lowest MSE. We can observe the regression using X_1 at Figure 23.

Input selection with cross validation

Another way of doing feature selection is to do the following : drop one feature, apply cross-validation on the rest of the data and report MSE. Drop another feature, apply cross-validation on the rest of the data and report MSE. Do that for all the features, and the highest MSE will correspond to when we drop the most relevant feature.

Input	Relevance rank via bay_lssvmARD	MSE on test set
X_1	1	0.02124
X_2	3	0.15758
X_3	2	0.1653

Table 12: Relevance rank and MSE for each of the input

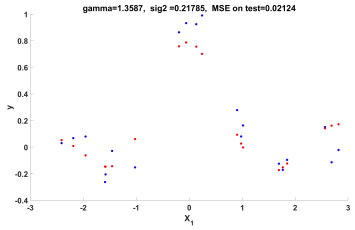


Figure 23: Regression using X_1 as input

2.4 Robust regression

One of the drawback of LS-SVM is that outliers have a lot of importance in the loss function because we square the errors (we use a L2 estimator). The derivative of such estimator is not bounded. A possible solution is to use a Hubber loss function, which is a quadratic function that becomes linear for high input values. Another solution is to use a robust version of the LS-SVM, by assigning a weight v_k to every datapoint. If we don't have outliers in the data, every data points will have a weight $v_k = 1$. On the other hand, when outliers are present (points with large residual), we will assign a low v_k to these outliers in order to limit their impact on the loss function. To compute the weights to assign to the data points, we will compare 4 methods : 'whuber', 'whampel', 'wlogistic', 'wmyriad'.

At Figure 24, we can observe the regression using the not robust version of LS-SVM. It is clear that the outliers have a strong effect : around [-4...-2] and [2...4] on the x-axis, we can observe that we have groups of 3 outliers. These have as effect that the estimated function is bring upwards around [-4...-2] and [2...4]. This is an unwanted effect and we want to be robust against such outliers. At Figure 25, we can observe the regression using **robust** LS-SVM, for 4 different weighting function. We can see that for all weighthing function (Huber, Hampel, Logistic and Myriad), the impact of the outliers is reduced compared to Figure 24. We can observed that the estimated function is similar when using Huber, Hampel and Logistic. On the other hand, when using Myriad, we observe that the estimated function overfits slightly the data.

MAE is preferred to MSE in presence of outliers, because in MSE we square the errors which increases the effect of outliers.

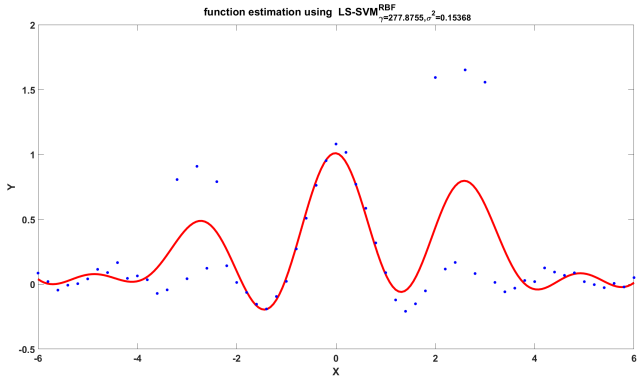
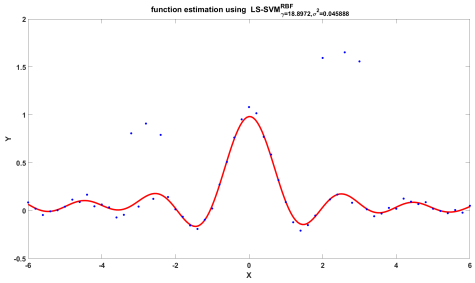
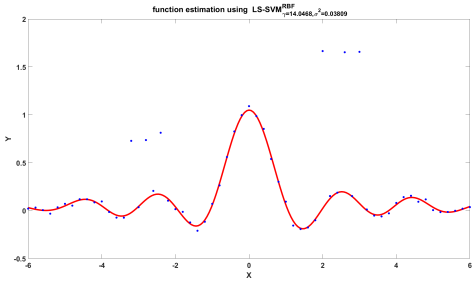


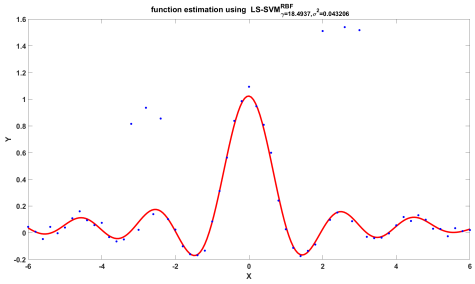
Figure 24: Regression using Not Robust LS-SVM



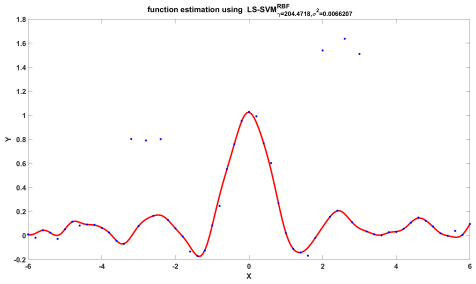
(a) Using Huber function



(b) Using Hampel function



(c) Using Logistic function



(d) Using Myriad function

Figure 25: Regression with weighted robust LS-SVM, for different weighting function

2.5 Homework problems

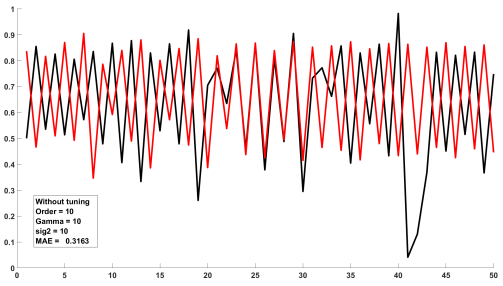
The goal here is to predict the next M points of a dataset based on the first N data points. What we do is therefore time-series prediction. To do so, we train the model on the first M data points, and use this trained model to predict the next N data points. The data is one dimensional. The model is trained as follows : we divide the dataset into pairs of input and target, where the input is a window of data of size p, and the target is the next point, i.e. point $p+1$. We shift the window accross the whole x-axis to cover the whole dataset. Consider a time series in the form [1;2;3;4;5]. If we select a window size of 2, the pairs of (input;target) will be [(1,2;3),(2,3;4),(3,4;5)]. After training the model, to predict what is the next unknown value, we place the window at the end of the dataset and predict the next point, i.e. in our example : (4,5;next point). After, we can use the predicted next point to predict even further, i.e. in our example (5,next point;even further).

2.5.1 Logmap dataset

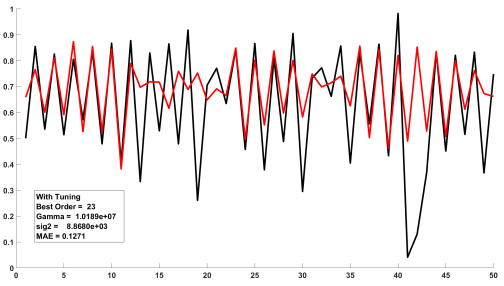
In the Logmap dataset, we have 150 data points, and we need to predict the next 50 datapoints. We will tune the LS-SVM to find the best following parameters : order (i.e. window size), γ and σ^2. The methodology will be as follow : for a range of order values from 1 to 40, tune a LS-SVM with the tunelssvm method using the 'simplex' method and 10-fold cross validation, report the MAE for each iteration of the for loop. Then, the lowest MAE will correspond to the *Best Order*, and finding the *Best* γ and the *Best* σ^2 will be trivial because they will be the values corresponding to the iteration of *Best Order*. At Figure 26, we can observe the Time series Predictions without and with tuning. We can see that with tuning, the predicted function is more accurate. Moreover, with tuning, we have a MAE of 0.1271, while without tuning ,we have a MAE of 0.3163.

2.5.2 Santa Fe dataset

In the Santa Fe dataset, based on the first 1000 data points, we have to predict the next 200 data points. The exercice is similar to the previous one, so we will use the same methodology to tune the order, γ and σ^2. However this time we will try for order values from 1 to 100. The results are presented at Figure 27. We can see that with tuning, the predicted function is more accurate. Moreover, with tuning, we have a MAE of 11.4159, while without tuning ,we have a MAE of 14.2953.

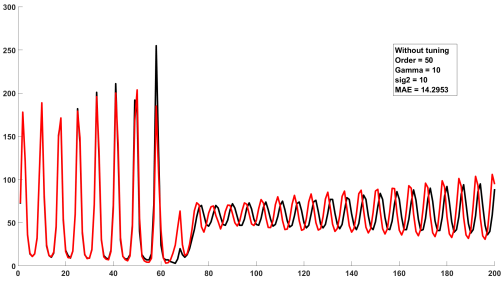


(a) Without tuning : order=10,γ=10,σ^2=10. MAE = 0.3163

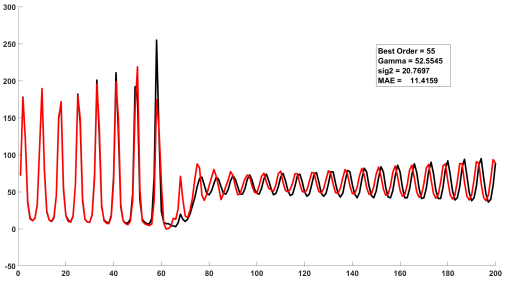


(b) With tuning : order=23,γ=1.01e7,σ^2=8.86e3. MAE = 0.1271

Figure 26: Time series prediction for the Logmap dataset, without and with tuning



(a) Without tuning : order=50,γ=10,σ^2=10. MAE = 14.2953



(b) With tuning : order=55,γ=52.55,σ^2=20.77. MAE = 11.4159

Figure 27: Time series prediction for the Santa Fe dataset, without and with tuning

Is order=50 a good choice ?

As we can see at Figure 27, we have obtained order=55 after tuning. Therefore, order=50 is a good choice if we also tune γ and σ^2 afterwards. Higher the order value is, more information we will take into account when doing the prediction. However, this can also reduce the sensitivity of the prediction. In the Santa Fe dataset, using order=50 seems to be a good trade-off.

Is using the validation set to optimize hyperparameters a good idea ?

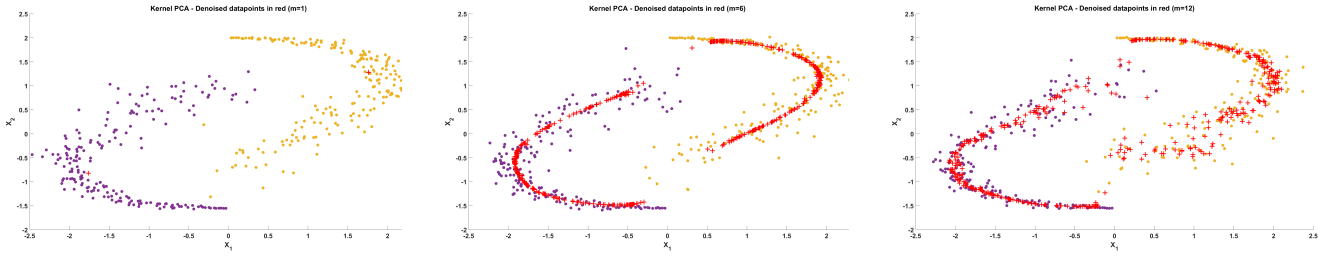
Usually, one uses a validation set to optimize hyperparameters. Here, we are in a Time Series Prediction framework, which is a bit special. It will be important to **not mix** the training, the validation and the test set if we want to avoid overfitting. The idea that we can implement will be to split based on the time axis, e.g. for a dataset correspond to 100 seconds data, use the first 60 seconds for training, the following 20 for validation and the last 20 for test.

3 Assignment 3 : Unsupervised Learning and Large Scale Problems

3.1 Kernel principal component analysis

How can we do denoising using PCA.

The idea of PCA is to find the direction of maximal variance, by computing the largest eigenvectors of the covariance matrix of the data distribution. Large eigen values correspond to components that are the most informative for the data distribution, while very small eigen values are usually linked to noise. Therefore, PCA can be used to do dimensionality reduction and denoising : by only keeping the large eigenvalues, we reduce the dimensionality of the data and perform denoising. The number "m" of eigen values we keep has to be chosen carefully : if "m" is too small, we will remove too much information, and will not be able to reconstruct the original data. On the other hand, if "m" is too large, we will not remove enough noise from the data distribution. We can observe this at Figure 28 : when using only one component, the denoised data poitns are not enough to represent the structure of the data. When using 12 components, the denoised datapoints are still noisy. Using 6 components is a good trade-off : the data structure is well represented and the noise is removed.



(a) Denoising using KPCA with 1 principal component

(b) Denoising using KPCA with 6 principal components

(c) Denoising using KPCA with 12 principal components

Figure 28: Denoising of the yin-yang data using KPCA with 1, 6 and 12 components (σ^2=0.4)

Linear PCA VS Kernel PCA.

For non-linear data distribution, a linear PCA is limited. If we consider the ying-yang dataset of Figure 28, linear PCA will only finds linear directions, which does not really gives the actual direction of maximal variance. This is depicted at Figure 29.

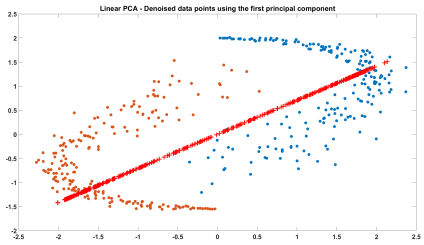


Figure 29: Denoising of the yin-yang data using linear PCA

The maximal number of components in Linear PCA is the dimensionality D of the data. On the other hand, in Kernel PCA, we start with a Kernel Matrix, sized NxN (where N is the number of data) and then we perform eigen values decomposition. Therefore, in Kernel PCA, there will be N eigenvalues and N eigenvectors.

Tuning of the parameters

For the tuning of the parameters, one can follow a grid search approach to test different combination of the parameters, and select the one that gives the lowest error between the denoised pattern and the desired pattern.

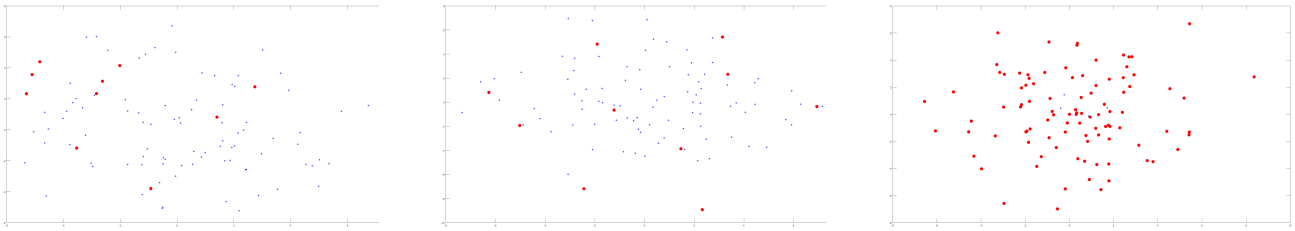
3.2 Fixed-size LS-SVM

In the primal formulation of SVM and LS-SVM, the unknown are w, of size the dimensionality D of the data, and b, the bias term. On the other hand, in the dual formulation, the unknwon are the lagrangian multipliers α, of size the number of samples N, and b, the bias term. Therefore, the choice between the primal formulation and the dual formulation depends on the problem : If we have a small number of samples in a high dimension (D»N), the dual is preferred. When we have a high number of samples in low dimension (N»D), the primal is preferred.

If we use a RBF kernel, we need to work in the dual because the feature map in the primal will be of infinite dimension. However, one may want to work in the primal if we are in a case where the number of training samples is much higher than the dimensionality of the data (N»D). To work in the primal, we will need to *approximate the feature map*. The Nystrom method is a way to do so. It allows to obtain a finite dimensional approximation of the feature map $\Phi(x)$. The idea is to use a subset of the dataset to build a kernel matrix of size MxM from which we will compute the eigenvectors to approximate the feature map. The choice of the subset M of the dataset is important. Selecting a random subset often works pretty well, but a more robust choice is by using the Quadratic Renyi Entropy, which leads to a subset M that represent well the complete dataset N. The algorithm to choose the subset M is defined as follow : we start with a random subset M. We pick one support vector x^* from the subset M and one sample x^{t*} from the training set. If using x^{t*} instead of x^* improve an entropy criterion we define, we replace x^* by x^{t*}. We repeat this processus until obtaining a *good* group of support vectors.

Effect of sig2 and to what subset the algorithm converges?

The Renyi Quadratic Entropy is given by $-log \int p(x)^2 dx$. Based on Information Theory, the amount of information of a variable is given by the uncertainty of the possible outcomes of the variables. Outcomes that does not occur often carry more information. The Renyi Quadratic Entropy is based on that. We will use this Entropy criterion to select the support vectors, by selecting the ones that maximizes this criterion. This way, we will select the samples that carries the most information about the data distribution. Instead of computing the Renyi Quadratic Entropy by integrating the probability distribution, we can approximate it using an estimator based on an RBF kernel. We have to choose the bandwidth of this RBF kernel carefully. Indeed, at Figure 30, we can observe the resulting support vectors for different values of σ^2. We can see that when σ^2=1000, the number of support vectors is too high and close to N, the size of the dataset, which removes the advantage of working with a smaller subset of the data. On the other hand, when σ^2=0.0001, we don't have anymore the characteristics of the data distribution. σ^2=10 seems to be a good trade-off. Moreover, in this case, the dataset is randomly distributed. Therefore, having a random subset will perform similarly to using Renyi Quadratic Entropy.

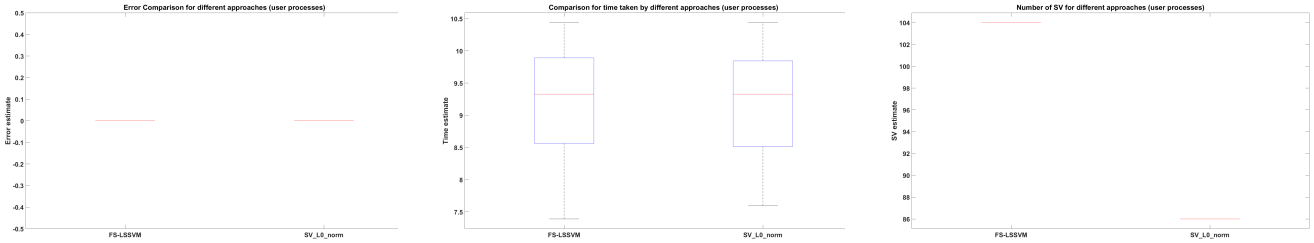


(a) σ^2=0.0001 (b) σ^2=10 (c) σ^2=1000

Figure 30: Selected subset M using the Quadratic Renyi Entropi criterio, for different values of σ

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

At Figure 31, we compare fixed-sized LS-SVM and l_0-approximation on the Shuttle dataset. We observe that the error and the time taken is similar between the two methods. On the other hand, the number of support vectors is higher for fixed-sized LS SVM.



(a) Errors (b) Time taken (c) Number of support vectors

Figure 31: Comparison, using a polynomial kernel, between fixed-sized LS-SVM and l_0-approximation on the Shuttle dataset

3.3 Homework problems

3.3.1 KPCA

Linear PCA vs Kernel PCA

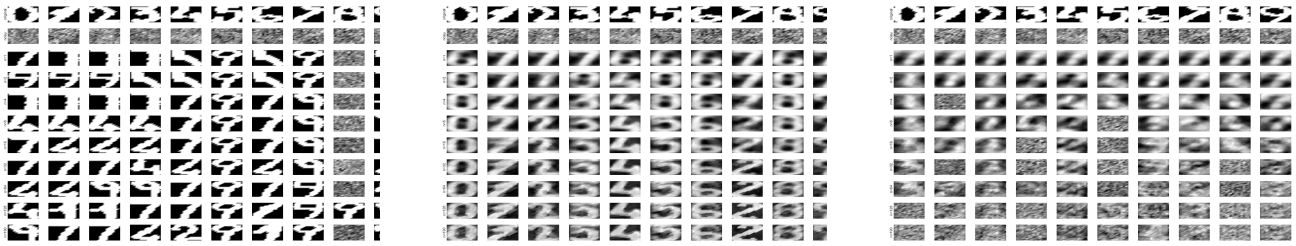
At Figure 32, we compare the performance of Linear PCA and KPCA for denoising. The parameter used for the noise is noisefactor=1.0, while the parameter for KPCA is $sig2$=35.9078. We can clearly observe that KPCA provides a better reconstruction. This can be explained by the ability of kernel PCA to extract **non-linear** maximum variance direction.

(a) Using KPCA with σ=35.9078 (b) Using Linear PCA

Figure 32: Denoising using KPCA and Linear PCA on a handwritten digits dataset

Effect of sigma

At Figure 33, we can observe the effect of the parameter Sigma Factor. It become clear that tuning this parameter is crucial. For low values, the noise is completely removed, but the digits reconstructed are false ! For example, we reconstruct the digit 4 for a noisy 0. For high values of Sigma Factor, we are not able to remove enough noise. At Figure 33b, Sigma Factor =1, with n=32 seems to be a good trade-off (8th row on Figure 33b).



(a) Sigma Factor = 0.01 (b) Sigma Factor = 1 (c) Sigma Factor = 100

Figure 33: Denoising using KPCA, for different Sigma Factor values

Tuning of parameters

The parameters to tune are sigma and the number of principal components we take into account. To tune these parameters, we will follow a grid search method : we try several values (number of principal components = [16...128], sigmafactor=[0.1...5]). Then we look at the mean-square error between the original image and the denoised image. The best parameters obtained when tuning on Xtest1 are number of principal components=36, $\sigma^2 = 25.5$, and the best parameters when tuning on Xtest2 are number of principal components=126, $\sigma^2 = 25.5$. The results for the best parameters, on Xtest1 and Xtest2 are depicted at Figure 34, where the third row correspond to denoising using the tuned sigmafactor and the tuned number of principal component, while the fourth row correspond to denoising using just the tuned number of principal component and the default sigmafactor. We can clearly see that tuning helps. This is more visible for Xtest2, where we see at Figure 34b that with the tuned parameter the reconstructed images are much more accurate. Moreover, MSE is 29 before tuning and 27 for Xtest1, while MSE is 6 before tuning and 1 after tuning for Xtest2. One hypothesis we can make is that the samples in Xtest1 are more *challenging* than those in Xtest2.

(a) Denoising on Xtest1 (b) Denoising on Xtest2

Figure 34: From top to bottom : Original images, noisy images, denoised images with tuned sig2 and number of components, denoised images with only tuned number of components

3.3.2 Fixed-size LS-SVM

Shuttle

The Shuttle dataset contains 58000 samples, with 9 numerical attributes belonging to 7 classes. The dataset was generated to extract rules for determining the conditions under which an autolanding would be preferable to manual control for a spacecraft [5]. The 7 classes are: Rad Flow, Fpv Close, Fpv Open, High, Bypass, Bpv Close, Bpv Open. 80% of the samples belong to class 1. Therefore, a classifier that always predict "class 1" will already have 80% of accuracy. We aim a higher performance than that. In our case, we only consider 700 samples in a binary classifation framework : we want to separate data belonging to class 1 to data not belonging to class 1. We end up with 572

[5] see https://sci2s.ugr.es/keel/dataset.php?cod=108

samples belonging to class 1 and 128 samples belonging to the other classes. Therefore, a classifier that always predict class 1 will have 81.714% accuracy on our subset of the data.
The dataset being too big to run on my computer, I have considered the first 700 data points. The results are shown at Figure 31 as we already have seen. We observe that the error and the time taken is similar between the two methods. On the other hand, the number of support vectors is higher for fixed-sized LS-SVM.

California

The dataset we use has 20640 samples. It takes too much time to run on my computer, therefore I consider only the first 2000 samples. Here, we are in a regression framework. The attributes are median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude. The output is the median house value. To visualize the data, let's plot the median house value as a function of the median income. We expect that the median house value will increases when the median income increases. This is indeed the case, as shown in Figure 35. However, it looks like the Median House Value saturates at 60 000.

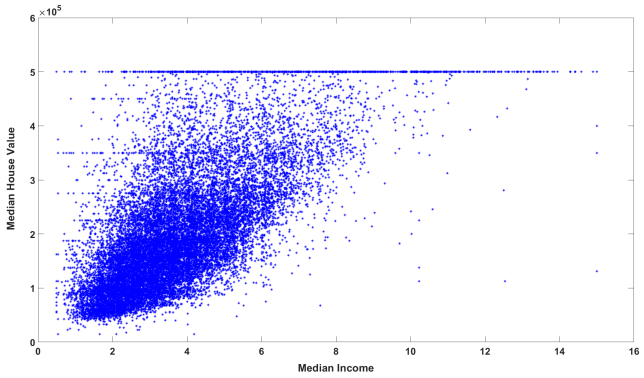
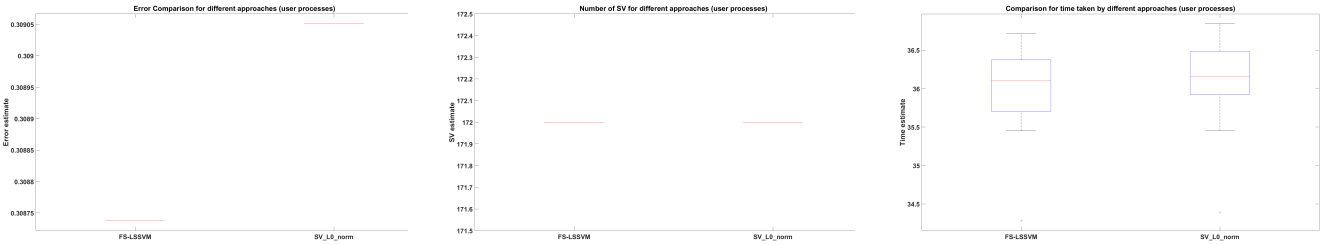


Figure 35: Regression using Not Robust LS-SVM



(a) Errors (b) Number of support vectors (c) Time taken

Figure 36: Comparison, using a polynomial kernel, between fixed-sized LS SVM and l_0-approximation on the California dataset

At Figure 36, we compare fixed-sized LS SVM and l_0-approximation on the Shuttle dataset. We observe that the error and the time taken is similar between the two methods. On the other hand, the number of support vectors is higher for fixed-sized LS-SVM.