Support Vector Machines: Methods and Applications

[H00H3a] – KU Leuven

Alberto Stefanelli*

Assignment 1: Classification

Separating two Gaussian distributions

Given that the artificial dataset is constituted by two Gaussian distributions such that $X_1 \sim N(\mu_1, \sigma_1)$ with mean $\mu_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and variance $\sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, and $X_2 \sim N(\mu_2, \sigma_2)$ with mean $\mu_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$ and variance $\mu_2 = \mu_1$, a straight line would be the best classifier.

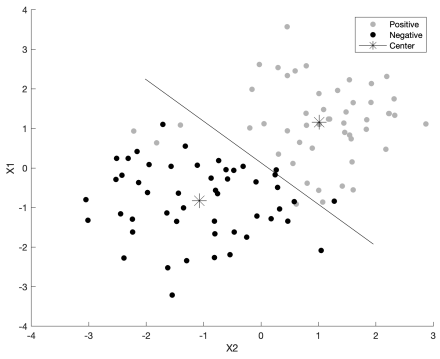

Figure 1: The artificial dataset constituted by two Gaussian distributions. The black line separating the two distributions represents the decision rule

This is because we know the distribution of the two classes a priori. As such, if $X - \mu_1 < X - \mu_2$, then X belongs to the positive class while if $|X - \mu_1| < |X - \mu_2|$, X would belong to the negative

*Alberto Stefanelli is an FWO Ph.D Fellow at the Institute for Social and Political Opinion Research, KU Leuven.

Email: alberto.stefanelli@kuleuven.be

class. Consequently, the linear classifier can be a perpendicular line connecting the two distribution centres, as shown in Figure 1. As Suykens et al. (2002) explains, "[f]or equal covariance matrices, the decision boundary becomes linear, for the distributions having a small overlap as well as a large overlap." (2002, 17)

This decision rule is optimal in the sense that it is the best possible solution to separate the two distributions as much as possible. However, depending on the difference in the means of the distributions, the classes could display greater overlap resulting in a considerable amount of misclassification. It is worth noting that, in this case, it is not possible to avoid misclassification using a linear decision boundary.

Support Vector Machine Classifiers

Support vectors are those data points that lie the closest to the margin and, consequently, influence its position and orientation. They are usually the data that are the most difficult to classify. This is because, in order to choose the support vectors, SVM maximizes the margin by finding the largest perpendicular distance between the hyperplane and the closest data points on both sides of the margin. In general, the contribution of each support vector to the decision rule is influenced by how close the support vector is to the margin. The closest, the more important is the support vector. In other words, the higher the (non-zero) weight (i.e., Lagrangian Multiplies), the more important a support vector is.

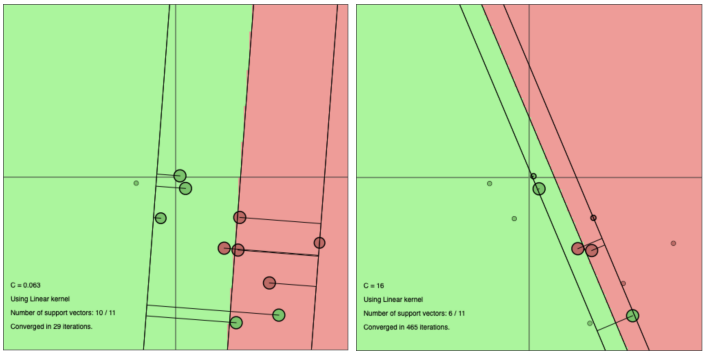

Figure 2: Support vector for lowe and high values of the parameter C

C is a regularization parameter that influences the misclassification on the objective function. It can be interpreted as a trade-off between reducing the number of weights that are non-zero (i.e., support vectors) and the amount of misclassification that the SVM allows. The underlying logic is illustrated in Figure 2. Low values of C (left-side) will result in a larger margin with a higher number of support vectors and more data points inside the planes. The cost of misclassifying data points is lower leading to a higher number of misclassified data points. In the boundary case of C=0, the SVM will ignore the errors, and just try to minimise the sum of squares of the weights (i.e., maximise the margin). On the contrary, higher values of C (right-side) mean that the SVM will try to reduce the margin as much as possible putting a higher cost on misclassified observations. The number of support vectors will be lower (i.e., 10 vs 6) and there will be a lower number of misclassified points (2 vs 3).

In the case of radial basis function (RBF) kernels, there is another parameter σ that needs to be optimized. The parameter σ can be interpreted as a measure that indicates how important is a given data point in defining the decision boundary. When σ is smaller, the decision bounder is influenced by the closest data points. In this case, points that are further away from the margin have a lower impact on the decision boundary. On the contrary, when σ is larger, the decision boundary will consider points that are further away. This is why the parameter σ can be intuitively interpreted as the amount of non-linearity (i.e., smoothness) that is allowed in drawing the decision boundary. Figure 3 illustrates this property showing how the decision becomes more linear for higher values of σ (and vice-versa).

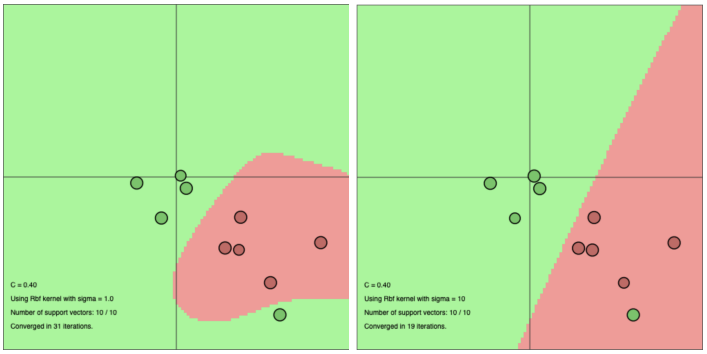

Figure 3: Support vector for lowe and high values of the parameter C

It is also worth noting that σ can also be used to control (and avoid) overfitting the training data. Figure 4 shows an RBF kernel fitted to the same synthetic data with a low (left-hand side) and a high (right-hand side) value of σ, keeping the parameter C constant. It is clear that a lower value of σ leads the SVM to attribute importance to the closest data points, to the point of the decision boundary lies in a very small circle around each data point. In this case, the SVM considers all the data points as support vectors. This results in a model that is usually less capable of generalizing to new data. On the contrary, a higher value of σ will result in an SVM that considers points that are more distant from the decision boundary leading to a substantial reduction of the number of support vectors (in this example, 45 vs 14).

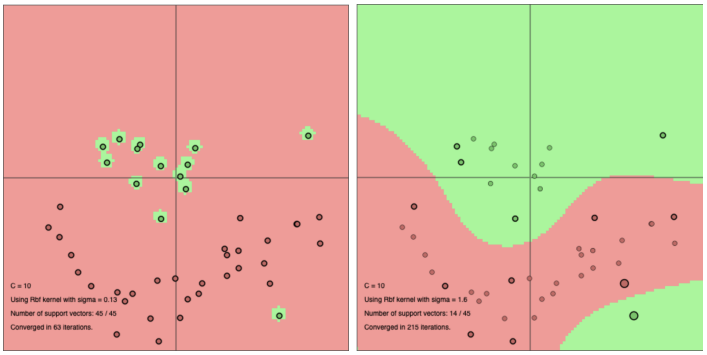

Figure 4: Support vector for lowe and high values of the parameter C

All in all, two general insights can be derived from the presented discussion. First, fine-tuning the selection of the C and the σ parameters is essential to obtain a model that strikes a balance between

good classification performance and flexibility (i.e., generalization of the model). Second, one of the main differences between linear and RBF kernels lies in the amount of non-linearity introduced by the σ parameter. This allows to introduce more complex and non-linear decision boundaries that are able to deal with non-linear data at the cost of introducing more complexity.

Least-Squares Support Vector Machine Classifier

I now apply a Least-Squares Support Vector Machine Classifier (LS-SVM) to classify the well-known iris dataset. First, I assess model performances by evaluating the impact of varying the degree of a polynomial kernel (1, 2, and 3) for an LS-SVM classifier, keeping γ fixed to 1. Figure 5 reveals that the decision boundary change significantly depending on the degree of the polynomial. Increasing the degree of the polynomial correspond to a more complex and non-linear decision boundary. This corresponds to a better performance of the model, here evaluated as a decrease of the error rate on the testing set, as shown in Table 1.

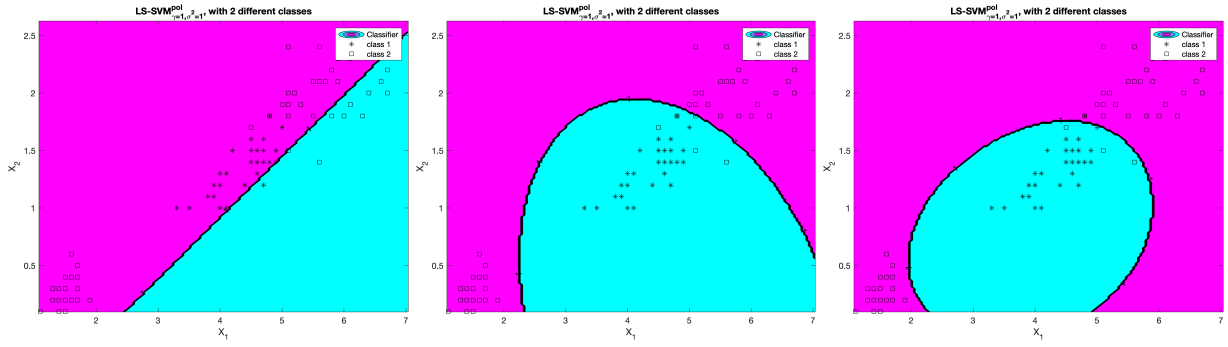

Figure 5: Decision boundary for LS-SVM with a polynomial kernels of varying degree (1, 2, and 3)

Table 1: Performance of LS-SVMs with polynomial kernels of varying degrees in Iris test dataset.

Degree	# misclassified obs.	Error rate
1	-0.21	11
2	-0.13	1
3	-0.21	0

Next, I evaluate the impact of changing σ^2 for an LS-SVM with an RBF kernel. To test how the model performs, I fit a series of LS-SVM classifies with σ^2 ranging from 0.01 to 20 and measure the error rate on the test set by calculating the percentage of misclassified data points. This measure ranges from 0 to .5, where 0 indicates the absence of misclassification and .5 indicates that the SVM classifies the data as it is at random. The value of γ has been fixed at 1. The results reported in 6 reveal that there is a window of σ^2 values that minimizes the error. For $\sigma^2 < .04$ the LS-SVM classifier tends to show an error rate ranging from .05 to .3. With a sigma σ^2 between .04 and 12,

the model shows an error rate of 0, indicating that none of the training data are misclassified. For values of $\sigma^2 > 12$, the error rate rapidly increases and reaches a plateau with $\sigma^2 > 17$ where the model performance is as bad as classifying the data points at random. I can then conclude that a σ^2 ranging from 0.04 and 12 is ideal for the classification task at hand.

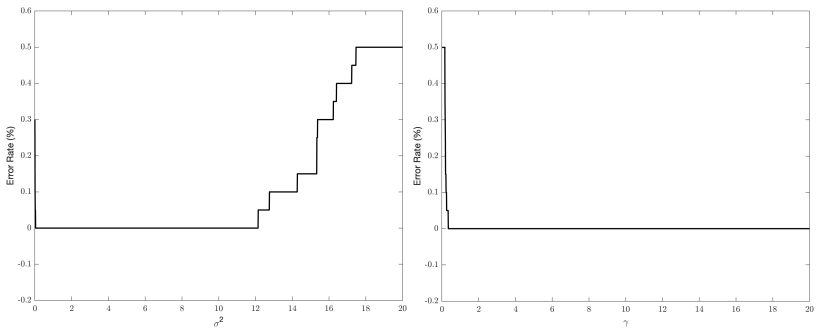

Figure 6: Performance of LS-SVM with RBF kerner in iris test dataset with different γ and σ^2

Based on this, I fixed $\sigma^2 = 6$ and fit a series of LS-SVM classifies varying the parameter γ using a range of values ranging from 0.01 to 20 with 0.1 increments. Figure 6 reveals that the error rate plateau at 0%, meaning non of the testing data are misclassified, for values of $\gamma > .34$. Although I have no information on the upper limit of γ, for the data at hand, a sensible range of γ would range from .34 to 30.

I can conclude, that both the σ^2 and γ of the LS-SVM with RBF kernel are important to achieve good performances in a classification task.

Tuning parameters using validation

In this section, I use automatic tuning algorithms to find the best hyper parameters σ^2 and γ for an LS-SVM model with an RBF kernel. These algorithms further split up the training data into a training and a testing part, leading to more robust insights on which are the best parameters of an LS-SVM for a given dataset. Three different algorithms are used namely, random split of the data, 10-fold cross validation, and leave-one-out cross validation. Based on the insights The range of values for the parameter tuning is selected to range from 0.01 to 12 for both σ^2 and γ.

The results are presented using a heatmap plot that shows the classification error for each combination of σ^2 and γ. Three important insights can be drawn from Figure 7. First, as also shown before, we can notice that there is not a single pair of σ^2 and γ that leads to a good performance. Rather more than a combination of σ^2 and γ leads to good results. Second, the most variation in error rates can be found in the models fitted with random split of the data. Third, the difference between 10-fold cross validation, and leave-one-out cross validation is minimal suggesting that both approaches perform equally. To select the best hyper parameters σ^2 and γ, I compare the results from the three different methods and pick the combination with the lowest error across the 3 different methods. In this case, the lowest error rate is for a $\sigma^2 = 5$ and a $\gamma = 3$.

Regardless of the results above presented, cross validation should be preferred over simple validation, especially for small datasets like the one currently used. The reason is that cross validation

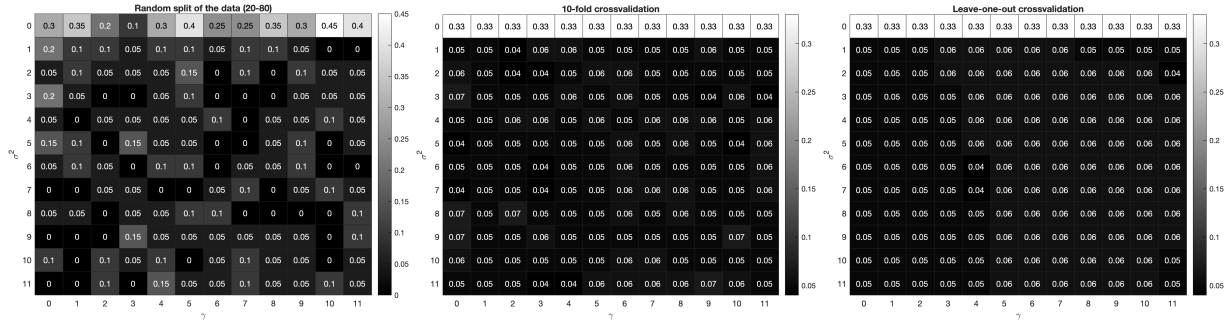

Figure 7: Classification error for each combination of σ^2 and γ

uses all the available training data to find the best hyper parameter σ^2 and γ while the random split of the data method usually employs 70 or 80 percent of the training data. This explains the results displayed in Figure 7. Although the random split method has the lowest error rate, it may not the best choice to assess the performance of an LS-SVM for a small dataset such as the one used.

Automatic parameter tuning

In this section, I compare two different methods for automatically tuning the σ^2 and γ parameters, namely the "Nelder-Mead" method (i.e. "Simplex") and the "brute force grid search". Both methods use a two-step approach. In the first step, a Coupled Simulated Annealing determines a range of good starting values and then, in a second step, these are fine-tuned using the Simplex or Gridsearch method. Both the Simplex and grid search algorithms are used to find a local minimum and, thus, different runs should be used employing different initial values. Regarding the cost (ϵ), both algorithms are set to use 10-fold cross validation to calculate the error rate on the training set.

To assess whether any difference exists between the "Nelder-Mead" method (i.e. "Simplex") and the "brute force grid search", we extract the fine-tuned σ^2 and γ parameters from 100 different runs. We also extract the cost (ϵ) to assess whether there is a difference in model performance. We plot the parameters on a log scale for an easier interpretation and calculate their median on the untransformed scale. The results are reported in Figure 8 and Figure 9.

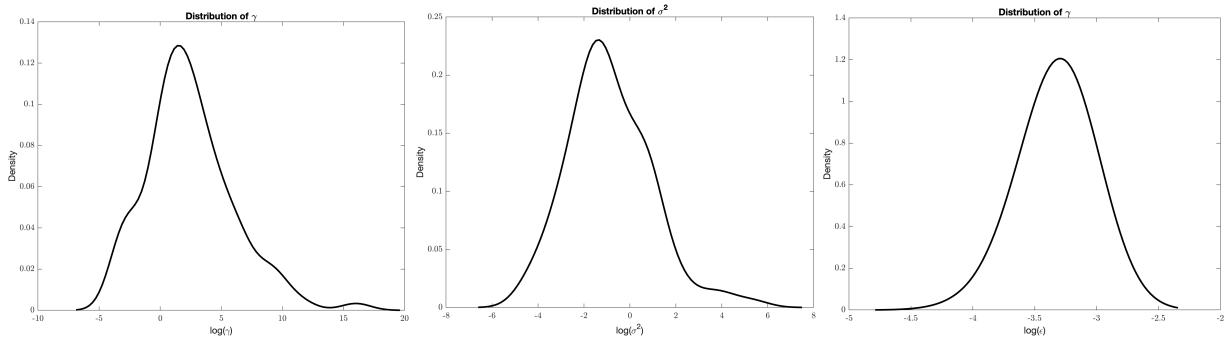

Figure 8: Simplex method – Distribution of σ^2 and γ and ϵ (logged)

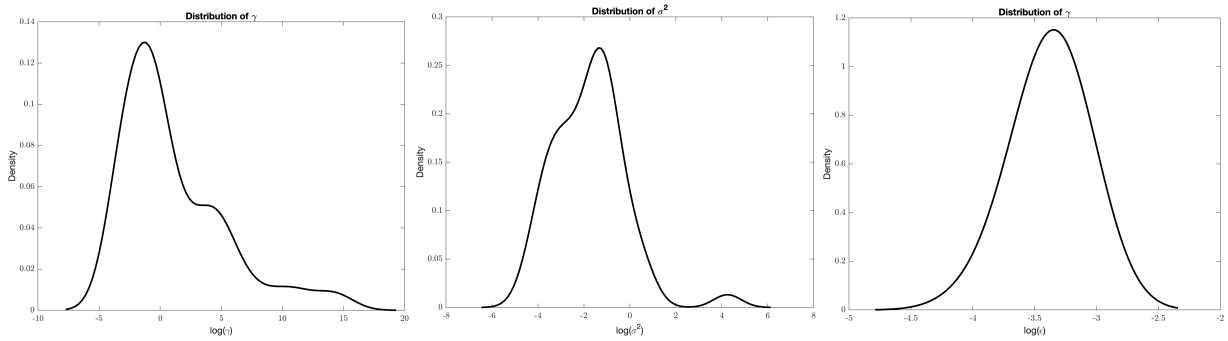

Figure 9: Force gridsearch method – Distribution of σ^2 and γ and ϵ (logged)

The figures reveal strong differences between the two methods, yet similar model performance ($\epsilon = 0.04$), confirming what was observed before: there are multiple pairs of σ^2 and γ that lead to good model performances. The distribution of γ is rather different, with a median of 3.54 for the Simplex algorithm and a median of 0.44 for the brute force grid search. The distribution of σ^2 is more similar across the Simplex and the brute force grid search, with a median of 0.34 and 0.21, respectively. Concerning the execution time for the 100 runs, we notice a significant discrepancy, with the Simplex method being faster compared to the brute force grid search. Specifically, 100 runs took approximately 4 minutes to finish using the Simplex method while it took approximately 7 minutes for the brute force grid search.

ROC Curve to evaluate model performance

Model performance can be also assessed using a Receiver Operating Characteristic (ROC) curve and looking at the area under the curve. The ROC curve represents the trade-off between the true positive rate and the false positive rate and, thus, the larger the area under the curve, the better the performance of the model. Usually, the ROC curve is calculated using the test set rather than the training set. This is because the ROC curve is typically employed to assess whether the trained model is able to generalize to new data. I will use the σ^2 and γ parameters obtained from the brutal force grid search method to fit an LS-SVM with an RBF kernel and assess its performance on the test set using a ROC curve.

The results indicate that the selected σ^2 and γ parameters are appropriate to classify the test data. The area under the curve reported in Figure 10 is equal to 1 on test data meaning that a perfectly separating classifier is found on the iris dataset.

Bayesian framework

I now classify the iris dataset using an SVM with a Bayesian framework. Contrary to a traditional LS-SVM, we now have a distribution of the posterior probabilities of each data point belonging to a given class instead of having a single class label. Figure 11 shows the SVM classification of 3 different models that use different values of σ^2 and γ. I fit a model using the fine-tuned parameters form the brutal force grid search ($\gamma = 0.44$ $\sigma^2 = 0.21$, a model with values of $\gamma = 0.1$ $\sigma^2 = 0.1$,

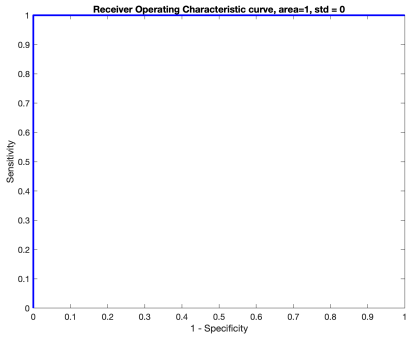

Figure 10: ROC curve for the test data using the σ^2 and γ obtained form the force gridsearch method

and a model with $\gamma = 60$ $\sigma^2 = 20$. This allows me to visualize the classification boundary of an SVM with a Bayesian framework with varying σ^2 and γ.

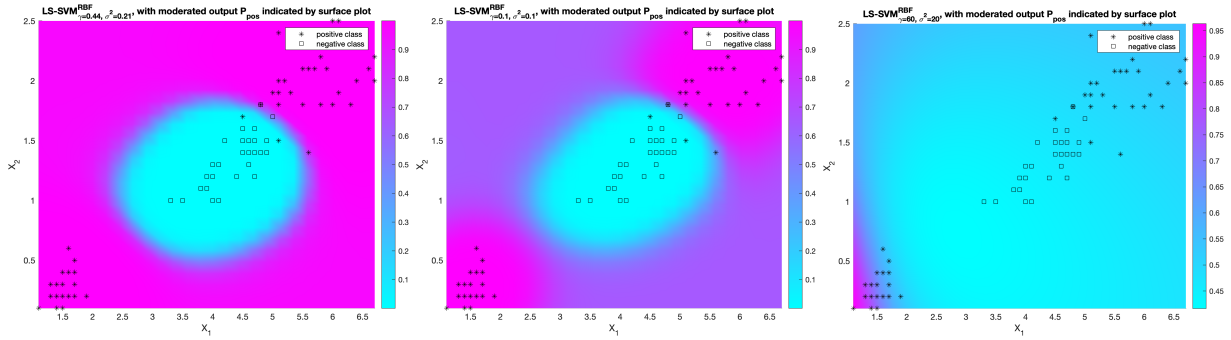

Figure 11: Posterior probabilities obtained from Bayesan SVM using varying values of σ^2 and γ

The intensity and the gradient of the colour of the plot indicate the probability of a given data point in the plot to belong to either class 1 or class 2. In this case, the blue colour represents the probability of a given data point to belong to the negative class while the purple colour represents the probability of belonging to the positive class. The colour bar on the left side of each plot displays a colour gradient that goes from blue to purple and represents the probability of belonging to the positive class. Low values (blue) indicate the areas of the plot with a low probability of belonging to the positive class while high values (purple) indicate areas with a high probability of belonging to the positive class.

The results at varying levels of γ and σ^2 follow what was observed earlier. A model that is well fine-tuned (left-side plot) is useful for classifying the data. It displays a smooth decision boundary with a low number of misclassified data points. It is worth noting that in the Bayesian framework, the data points that lie close to the decision boundary are in an area of the plot where the gradient between blue and purple is the highest. This means that the posterior probability of belonging to the positive class (versus the negative class) is close to 0.5. This corresponds to the area with the data points that are the most difficult to classify. Moving to the model with $\gamma = 0.1$ and $\sigma^2 = 0.1$ (central plot) we can notice a higher level of uncertainty in those areas where we do not have any data point, namely the bottom right and the top left quadrants of the plot. The model is still able to properly classify the data point, yet the decision boundary is not as well defined with

a large area where the posterior probabilities are close to 0. Finally, for a model with $\gamma = 100$ and $\sigma^2 = 20$ (right-side plot), we can notice that the model is not able to draw a clear decision boundary. It is worth noting that, in this case, the model adjusts the posterior probabilities to reflect the higher uncertainty of the model. The posterior probabilities range from 0.40 to 0.95 suggesting high uncertainty for almost all the data points

All in all, I can conclude that the posterior probabilities are a useful instrument to assess the areas with high levels of uncertainty and the overall ability of a model to correctly classify the data.

Homework problem – Ripley dataset

The Ripley dataset is a synthetic dataset composed of a mixture of two bidimensional normal distributions with different means and the same variance. It is composed of 250 observations for the training and 1000 for the testing. The classes are balanced (.5) for both the training and the testing sets.

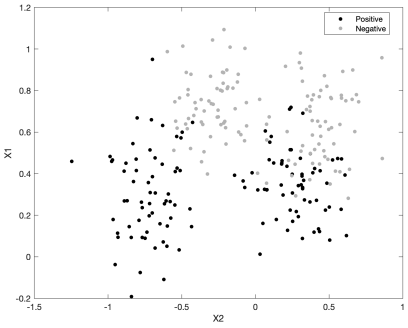

Figure 12: Scatter plot for the Ripley dataset

Figure **??** reveals there is not much separation between the two classes. Consequently, a linear kernel may not be the best to separate the two classes. For this reason, I use different kernels and use a parameter tuning method to select the best hyper parameters. To reduce the computation time, I employ a simplex method with parameter tuning via 10-fold cross validation. To assess which kernel is the best at classifying the data, I extract the median error (cost) of the 100 runs for each kernel.

Table 2: Performance of different kernels and the median of the hyper parameters from 100 runs of the simplex algorithm in the Ripley dataset

Kernel	$\tilde{\gamma}$	$\tilde{\sigma^2}$	$\tilde{t}, \tilde{Degree}$	$\tilde{Cost}$
Linear	0.02			0.14
Radial	4.80	0.67		0.12
Polyno-mial	6.30		1.64, 5	0.12

Results reported in Table 2 show that the three kernels perform rather similarly. The lowest error is achieved using either a polynomial or an RBF kernel. Given that for the Polynomial kernel we need to fine-tune both the degree and the parameter t, an RBF is selected for the classification task. Given that the 3 models perform quite similarly, there is not much difference in their ROC curves. However, I report the ROC curve of the RBF model since it shows the largest area under the curve being, which is 0.968 (Figure 13).

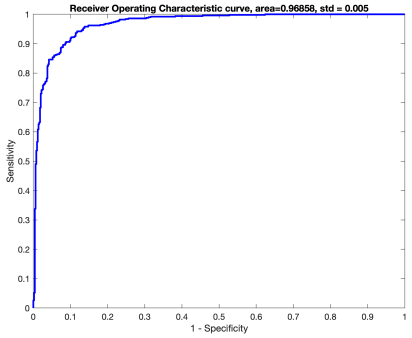

Figure 13: ROC curve for the for RBF kernel the test data using the σ^2 and γ obtained form simplex methods with parameter tuning via 10-fold crossvalidation

I consider the selected method satisfactory for the task at hand given that the model shows good performance both in terms of classification and generalization. However, simpler models such as an SVM with a linear kernel (or a logistic regression) are also appropriate to classify the Ripley dataset and, depending on the context of the application, may be preferred for their simplicity.

Homework problem – Wisconsin Breast Cancer dataset

The Wisconsin Breast Cancer dataset is composed of 400 observations for the training and 168 for the testing. The 30 explanatory variables are extracted from digitized images of a set of characteristics of the cancer cell nucleus. It is important to notice that the classes are unbalanced showing an average of 37% of malign and 63% benign breast cancer. I proceeded using a similar approach to the one used for the Ripley dataset and use an automatic parameter tuning method.

To visualize the data, I use a t-Distributed Stochastic Neighbor Embedding (TSNE) that allows me to reduce the dataset to a 2 dimensional array that can be plotted. In short, TSNE is a probabilistic approach that can be used to map high-dimensional data distribution to a low dimensional space. The main idea is to find the distance (in this case, euclidean) between two data points in a 2-D space that reflects the distance between two data points in the high-dimensional space. Figure 14 reveals that the two classes are well delimited with only a few points that may be difficult to classify. Consequently, I expect good classification accuracy for the Breast Cancer dataset.

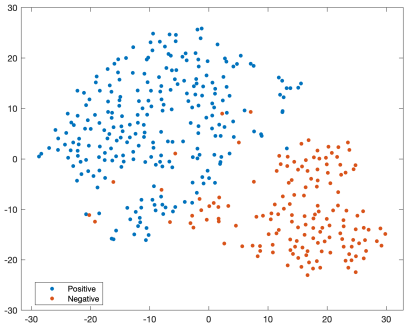

Figure 14: TSNE plot for the UCI Diabetes dataset

Table 3: Performance of different kernels and the median of the hyper parameters from 100 runs of the simplex algorithm in the Wisconsin Breast Cancer dataset

Kernel	$\tilde{\gamma}$	$\tilde{\sigma^2}$	$\tilde{t}, \tilde{Degree}$	$\tilde{Cost}$
Linear	0.14			0.03
Radial	40.7	37.5		0.01
Polyno-mial	69		44.8, 5	0.11

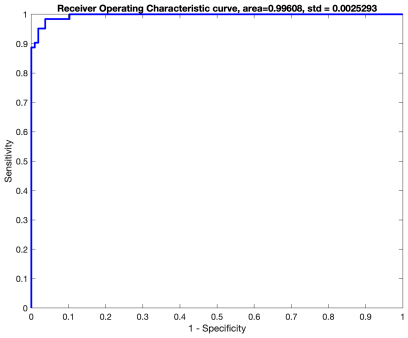

Figure 15: ROC curve for the for linerar kernel on test data using the σ^2 and γ obtained form simplex methods with parameter tuning via 10-fold crossvalidation

Results reported in Table 3 show that the linear and the RBF kernel perform very similarly with an area under the curve being that is greater than 0.996 for both models. Given the excellent performance of the linear kernel with $\gamma = 0.14$, I believe that it is the best model for the classification of the Wisconsin Breast Cancer dataset. Its ROC curve is reported in Figure 15. A non-linear PCA for dimensionality reduction could be useful to reduce computational time in similar applications.

Homework problem – UCI Diabetes dataset

The Diabetes dataset is composed of 300 observations for the training and 168 for the testing. Each observation has 8 features that refer to demographic characteristics (e.g., age) and some medical information (e.g., level of glucose) of the patients in the dataset. It is important to notice that the classes are unbalanced with 31% of the respondents having been diagnosed with diabetes in the train set and 37% in the test set. I proceeded by applying a similar approach to the one used for Ripley and Wisconsin Breast Cancer dataset and use an automatic parameter tuning method.

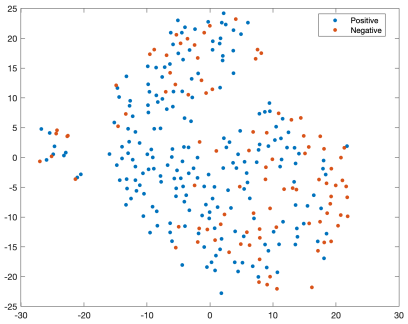

Figure 16: TSNE plot for the UCI Diabetes dataset

Table 4: Performance of different kernels and the median of the hyper parameters from 100 runs of the simplex algorithm in the UCI Diabetes dataset

Kernel	$\tilde{\gamma}$	$\tilde{\sigma^2}$	$\tilde{t}, \tilde{Degree}$	$\tilde{Cost}$
Linear	0.03			0.24
Radial	51.5	416		0.24
Polyno-mial	0		24, 3	0.28

Results reported in Table 4 confirm my intuition and show that none of the fine-tuned SVMs is able to accurately classify the data as it was for the Ripley and the Wisconsin Breast Cancer datasets. Given the similar performance of the fine-tuned linear and RBF kernel, we further inspect their ROC curves using the respective fine-tuned hyper parameters. Unsurprisingly, the ROC curves are very similar with an area under the curve of 0.85 for the RBF kernel and of 0.84 for the linear kernel. I decided to select the RBF kernel given its slightly better performance and more flexibility compared to the linear kernel. Its ROC curve is reported in Figure 17. The reason for the relatively poor performance of the SVM models can be intuitively assessed by visualizing the classification of the data points (not shown). Most of the data lies close to the decision boundary with severe overlap between the patients diagnosed with diabetes and the ones with a negative diagnosis.

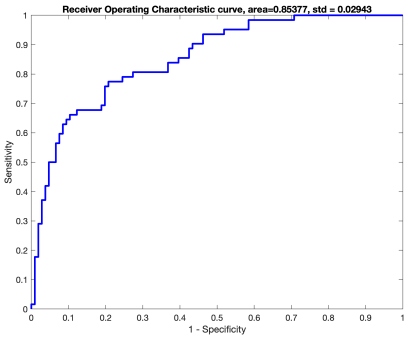

Figure 17: ROC curve for RBF kernel on the test data using the σ^2 and γ obtained form simplex methods with parameter tuning via 10-fold crossvalidation

In the case of the UCI Diabetes dataset, SVMs are not able to classify the patients with enough accuracy and, thus, other methods such as neural networks should be used to assess whether better classification can be achieved.

Assignment 2

Support vector machine for function estimation

In this section, I will use SVM for function estimation. The following examples are based on a synthetic dataset constructed to assess the behaviour of SVM regression in different contexts. The purpose is to minimize the C value used to determine the trade-off between minimizing the error and minimizing the weights. Two parameters of interest will be tuned. The first one is the "bound", which controls the magnitude of the weights used for the minimization of the sum of squares. The second parameter is the ϵ which controls the size of the margin (i.e. the $\epsilon - tube$) and is useful to find the support vectors.

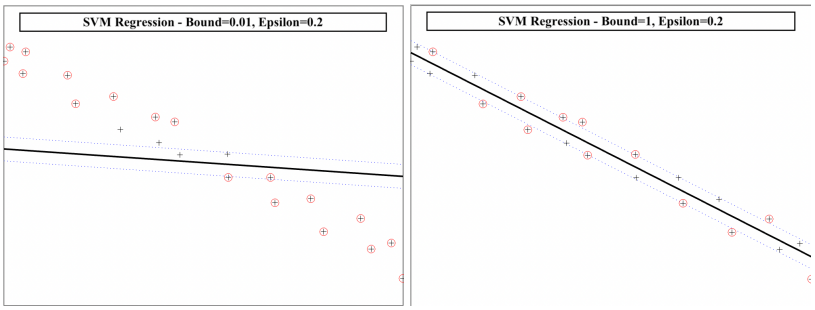

Figure 18: SVM regression with varying bound (0.01, 1)

Figure 18 reports the impact of modifying the bound, keeping ϵ fixed at 0.2. We can notice that decreasing the value of the bound leads to a decrease in the weights. This has the consequence that the sum of squares is not properly minimized.

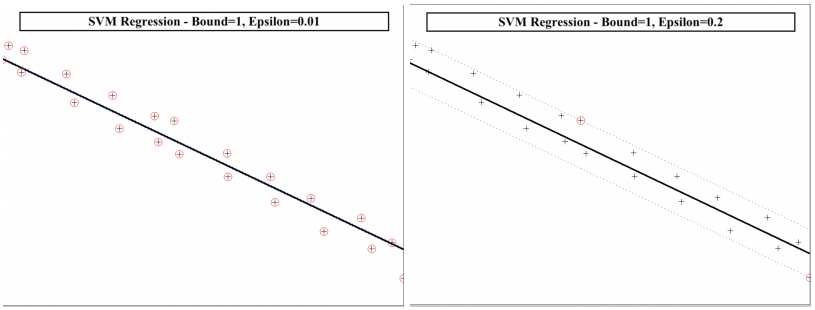

Figure 19: SVM regression with a liner kernel with varying ϵ (0.01, 2)

Figure 18 reports the impact of modifying the ϵ, keeping the bound fixed at 1. In this case, we can notice that lowering ϵ reduces the size of the $\epsilon - tube$ with the consequence of having more support vectors (circled in red in the figure). On the contrary, setting $\epsilon = 0.2$ leads to a substantial reduction of the support vectors (22 VS 2). Having fewer support vectors is preferred since the model complexity and the probability of overfitting decrease leading to a more generalizable model. Furthermore, having a low number of support vectors is the same as having an SVM with a sparsity property, which is favourable for generalization purposes or when dealing with large datasets.

Next, I create a synthetic dataset with a more complex data structure where I expect a linear regression not to perform as well as in the previous case. I then compare the performance of a linear, RBF, and polynomial kernel on this dataset. I fix the bound=10 and $\epsilon = 0.5$ and fine-tuned the hyper parameters of the polynomial (degree) and of the RBF (σ^2) kernels with the aim of reducing the number of support vectors. Although this procedure is sub-optimal, it severs as a way to compare how the different kernel performs.

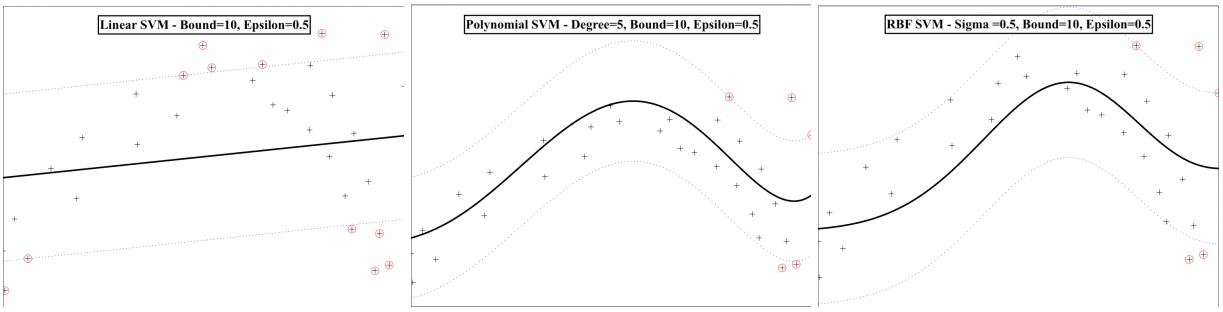

Figure 20: SVM regression with different kernels (linear, polynomial, RBF) with varying hyper parameters

Figure 20 shows that an SVM regression with a polynomial and an RBF kernel performs equally well. The number of support vectors is the same (5) and the regression line correctly follows the pattern of the data. Unsurprising, the linear kernel performs quite poorly with the data at hand. Even with 16 support vectors, it is unable to draw a line that follows the curvilinear nature of the data.

Three important differences can be noted between an SVM regression and a classical least squares fit. First, as we have noticed earlier, the SVM can use a subset of data points to draw the regression line (i.e., the support vectors) while least squares uses all observations for estimating the regression slope. Second, in classical least squares regression, the data point that has a predicted value that is different from the observed values will have a positive error term (residual). On the contrary, in SVM all the observations within the $\epsilon - tube$ have an error equal to zero. So, instead of having an error for all observations, SVM allows only the data to be outside the $\epsilon - tube$ to have a positive error. Third, in SVM we can use different kernels, allowing us to easily model non-linear relationships between variables. In a traditional regression setting, this is not possible.

Regression of the sinc function

For this exercise, I used the sinc function (with white noise) to create a training and a testing artificial datasets. Given the distribution of the data is known a priori, I decided to employ an RBF kernel for the function estimation. First, I select a predetermined range for the hyper parameters γ and σ^2 (ranging from 1 to 10^6) and calculate the mean squared error (MSR) for every combination.

The heatplot reported in Figure 21 (left-side) reveals that the model performs the best with hyper parameters $\gamma = 1000000$ and $\sigma^2 = 1$. It is worth noting that the MSR plateaus with $\gamma > 1000$, keeping $\sigma^2 = 1$. Given the relatively low MSR, I can conclude that the model fits the data very well even if the distribution is non-normal and noisy (Figure 21 right-side).

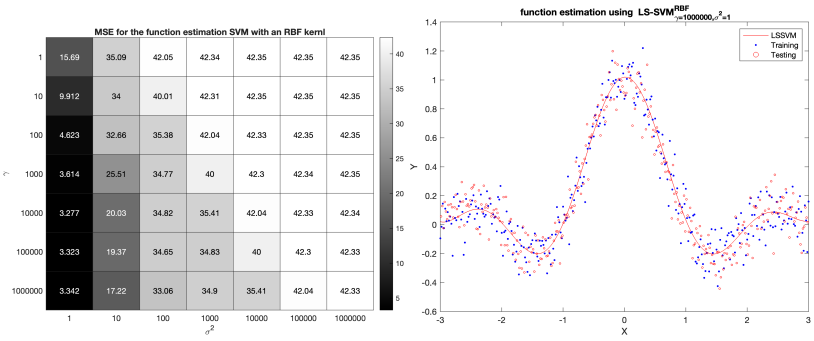

Figure 21: Heatmap for MSR with varying hyper parameters $\gamma$and σ^2 (left-side) and best fitting model (right-side)

Next, I use an automatic hyper parameter tuning using the simplex and grid brute force algorithm as previously done. As done before, I extract the median of $\tilde{\gamma}$ and $\tilde{\sigma}^2$ from 100 runs of both the simplex (Figure 22) and brute force (Figure 23) algorithm. I also plot the distribution of γ and σ^2 on the log scale.

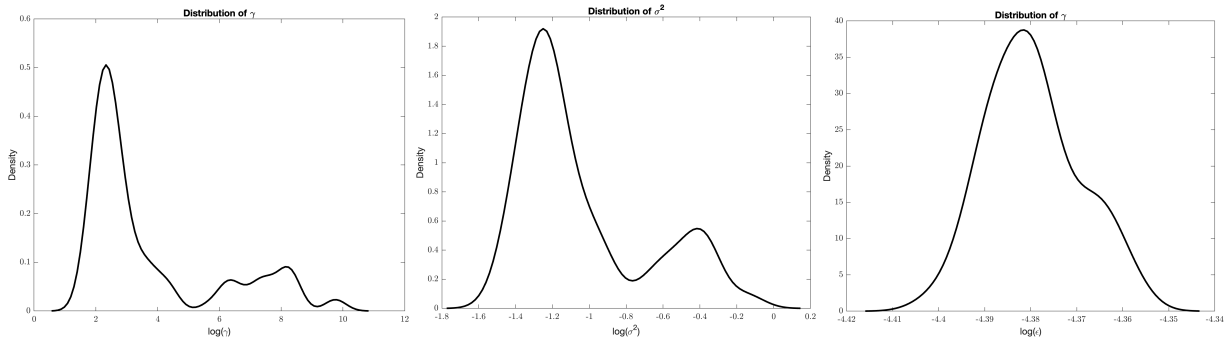

Figure 22: Simplex – Distribution of the γ and σ^2 parameters from 100 runs

Comparing Figure 22 and Figure 23 we can notice that the two algorithms perform in a similar fashion in regards to the error term (cost) both having a median MSE equal to 0.01. This is not surprising given the results obtained in the previous exercise. It is worth noting that the automatic parameter selection outperforms what was previously found with the manual tuning of the parameters. Comparing the distributions of the MSE between the simplex and the grid search, we can notice that the one obtained from the grid search is slightly less wide. Consequently, I use $\tilde{\gamma} = 13.83$ and $\tilde{\sigma}^2 = 0.31$ obtained from the grid serach to fit a new model. Nonetheless, as previously noted, it is unlikely that there is a single pair of γ and σ^2. Rather, multiple optimal pairs (i.e. local minima) are likely to be present.

Bayesian framework for LS-SVM regression

The Bayesian framework can be used to fine-tune the parameter of an LS-SVM regression. The general idea behind this approach is that by setting prior distributions on the parameters, it is possible to calculate their posterior probabilities given the data at hand. The general equation for

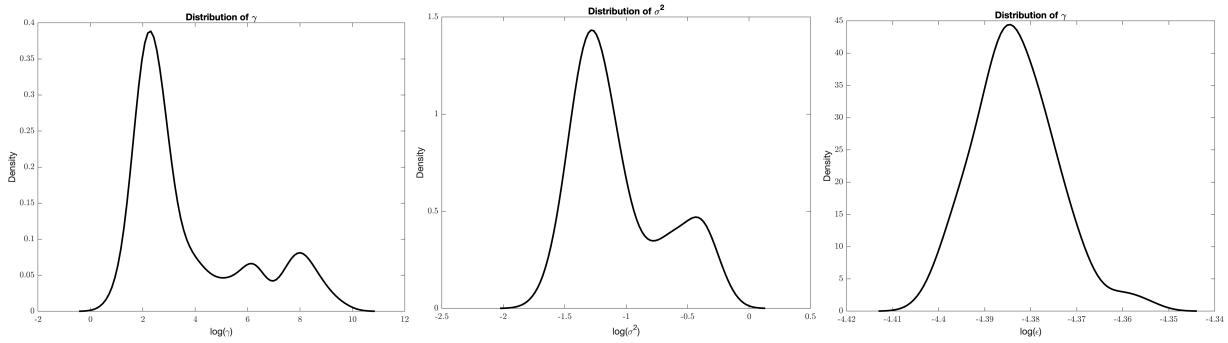

Figure 23: Grid brute force – Distribution of the γ and σ^2 parameters from 100 runs

the Bayes' Theorem for probability distributions is often formalized as follows $Posterior \propto Likelihood \cdot Prior$. In case we want to obtain the probability that the data points are generated by a model with unknown hyper parameters, we can express this as follow

$$P(parameters|data) = \frac{P(data|parameters) \cdot P(parameters)}{P(data)}$$

The model can be optimized with respect to 3 criteria, usually referred as levels. First, we can optimize for the α (i.e., the weights) and b (i.e., the bias) to obtain their posterior probabilities. This is the formulation with respect to the dual problem. This is obtained

```
[~ , alpha , b ] = bay_optimize ({ Xtrain , Ytrain , 'f', gam , sig2 } , 1)
```

These parameters are used in two subsequent levels where we can optimize for the γ and, for an RBF kernel, σ^2 using the following code:

```
[~,gam] = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 2);
[~,sig2] = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

Each level has a cost (negative logarithm of the posteriors). These error bars represent the model uncertainty of the parameters distributions. By By comparing the distribution and the error bars of the prior of the hyper parameters and their posterior, we can assess whether a model performs better. These can be obtained using the following code

```
bay_errorbar({Xtrain, Ytrain, 'f', gam, sig2}, 'figure');
```

Automatic Relevance Determination (ARD)

Using a Bayesian framework it is also possible to perform Automatic Relevance Determination (ARD). In short, ARD is a technique that can be used to 'prune' away those predictors that are marginally irrelevant. This can be used to prevent over fitting and increase the generalization of

the model in cases where the data at hand have many features (i.e. high dimensional data) and there is the necessity to only keep the features that matter the most.

To assess how ARD works, I use an artificial dataset constructed with 1 dependent variable and 3 predictors of which 1 is constructed to be related to the dependent variable. This allows me to test whether ARD is able to identify the most relevant predictor, namely the one that is linear dependent with the outcome variable. I first need to fine-tune the hyper parameters for the synthetic data. To this end, I use the procedure detailed in the previous section using as priors $\gamma = 20$ and $\sigma^2 = 0.5$.

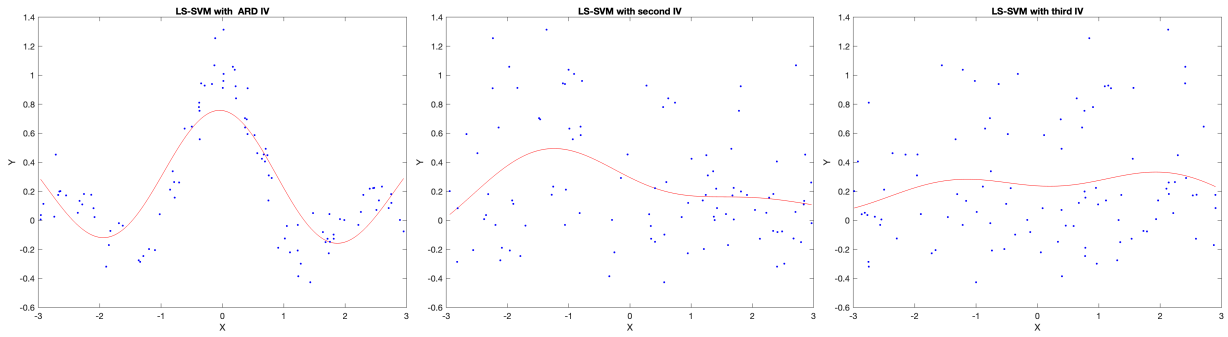

Figure 24: SVM Regressions with (1) only the most relevant predictor obtained from the ARD, (2) the unrelated predictor 1, (3) the unrelated predictor 2

Using the fine-tuned hyper parameters $\gamma = 8.1$ and $\sigma^2 = -0.22$, the ARD correctly points out that the most relevant variable is the one constructed to be a function of the dependent variable. To visualize the impact of using all the predictors, I fit 3 LS-SVM regressions using, (1) only the most relevant predictor obtained from the ARD, (2) the unrelated predictor 1, (3) the unrelated predictor 2. Figure 24 reveals that the two unrelated predictors are irrelevant while the one using only the most relevant predictor is better at predicting the synthetic data. It is worth noting that the model with the most relevant predictor should be re-fine-tuned to obtain better results.

We can also use the cross validate function to 'prune' away those predictors that are marginally irrelevant. The process would require defining 'a minimal cost criterion' that indicates the trade-off between removing a predictor and an increase in MSE. First, the MSE would be calculated for a fine-tuned model with all the variables. Second, a model is fitted removing a covariate and fine-tuned again. If the increase in MSE is less than the cost criterion, the algorithm continues and repeats the process removing another covariate, otherwise, it stops.

Robust regression

LS-SVM regression can be used also in the presence of outliers. This can be done by incorporating robustness into the estimation. In a nutshell, the robust estimation of an LS-SVM is done by weighting the errors in the loss function: observations with small errors will have larger weights while observations with large errors will have smaller weights. Since outliers will generally have large errors, they will be down-weighted and, thus will have less impact on the calculation of the loss function. To showcase the properties of robust LS-SVM regression estimation, I construct a

synthetic dataset with several outlines and compare the results obtained from a robust with the one obtained from a non-robust LS-SVM.

First, I fit an LS-SVM with an RBF kernel without taking into account the outliers. I fine-tune the hyper parameters using a grid search with 10-fold cross validation and visualize the regression results. Second, I fit another model that uses robust crossvalidation and huber weights. The results reported in Figure 25 show substantial differences between the two models. The non-robust model is quite sensitive to the outliers while the robust version is not, proving its utility when dealing with outliers. It is worth noting that when comparing the performance of the model the preferred loss function is the mean absolute error (i.e., L1) and not the classical mean squared error. The reason is that, as the name suggests, the latter uses a squared term and, consequently, large errors such as the one introduced by the outliers have more impact on the metric leading to inaccurate results.

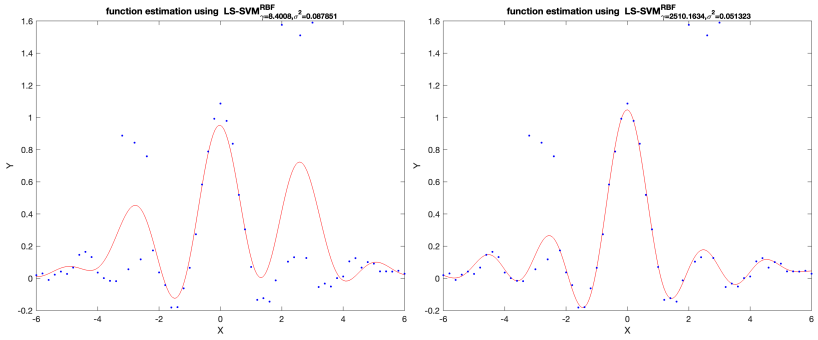

Figure 25: Non-ronust (left-side) and robust (right-side) SVM regressions in the prsence of outliers.

Second, we compare the performance of an LS-SVM with an RBF kernel fitted to the data under different weighting schemes. As reported in the user's guide of LS-SVMlab, the software automatically tunes the parameters of the huber and myriad weight functions. For the Hampel weight, the parameters are set to 2.5 and 3, following the software specifications. For the model using the logistic weight function, no parameters need to be tuned. Table 5 shows that the different weighting schemes perform equally with no substantial differences in the data at hand.

Table 5: Mean Absolute Error for LS-SVM with an RBF kernel under different weighting schemes

Weighting scheme	**Mean Absolute Error**
Non-Robust SVM	10.52
Whuber	7.75
Hampel	7.75
Logistic	7.77
Myriad	7.77

Homework problems: time series prediction on the Logmap dataset.

In this section, I will use LS-SVM to perform time series prediction on the Logmap dataset. To optimize the γ, σ^2, and the order, I decided to use a holdout sample approach using the last 50 points of the training data. The idea is to find the γ, the σ^2, and the order that minimize the MSE on the last part of the training dataset. I decided to use an LS-SVM with a RBF kernel given its good performance in the previous regression tasks. First, I inspected the data and selected a range of values for the lag (i.e., order) going from 1 to 60. Second, for each lag value, I perform an automatic parameter selection to find the γ and σ^2 hyper parameters. As done before, I used multiple runs (i.e., 50) of the simplex algorithm and excreted the median of γ and σ^2. These are used to predict the last 50 points of the training set and extract the MSE. I then selected the combination of γ, σ^2, and the order with the lowest MSE on the holdout sample.

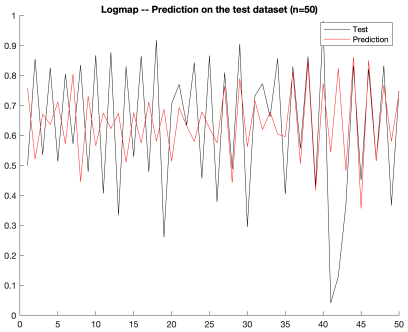

Figure 26: SVM Regressions on the Logmap dataset using fine tuned γ and σ^2 and order

This strategy yielded a model with a MSE=0.056 with $\gamma = 7752.07$, $\sigma^2 = 27.71$, and order=55. Figure 26 shows that the model prediction is far from being perfect and that in some portions of the time series the model is unable to accurately predict the next data points (e.g. x=42). The MSE on the test set is relatively low (0.074), yet it is higher compared to the one on the holdout sample. It is worth noting that this procedure requires substantial computational power with a running time of approximately 40 minutes on a modern laptop. Consequently, the proposed optimization strategy may not be the best choice in scenarios where the data at hand is very large or when the time for fine-tuning the model and forecasting is limited. Additionally, a better approach would require the usage of multiple holdout samples to avoid fine-tuning a model on a single, potentially biased or unrepresentative, portion of the time series.

Homework problems: time series prediction on the Santafe dataset

In this section, I will use LS-SVM to perform time series prediction on the Santafe dataset. The choice of order = 50 should be verified empirically unless we have accurate information on how the data has been generated. For this purpose, I fit a model with order=50 and fine-tune the γ and σ^2 parameters using an automatic parameter selection procedure with 100 runs (Simplex algorithm). We use the median of the parameters γ and σ^2 to predict the test set and calculate the MSE.

This fine-tuning procedure yields a model with an MSE=2210 using a $\gamma = 39$ and a $\sigma^2 = 71$. I can then conclude that the model does not show good performance on the test set and, thus,

another approach shall be used. It is worth noting that the median MSE resulting from the automatic parameter selection procedure is sensibly lower (i.e., MSE=220) suggesting that the model is able to reproduce the training set but fails to accurately predict future data points that are not used for model training. Consequently, I optimized the order along with the γ and σ^2. In this case, the choice of using a validation set is the most appropriate one to fine-tune parameters and, in practice, corresponds to the holdout sample approach used in the previous section. In this case, the validation set (i.e. hold out sample) is equal to 200 data points to match the testing dataset.

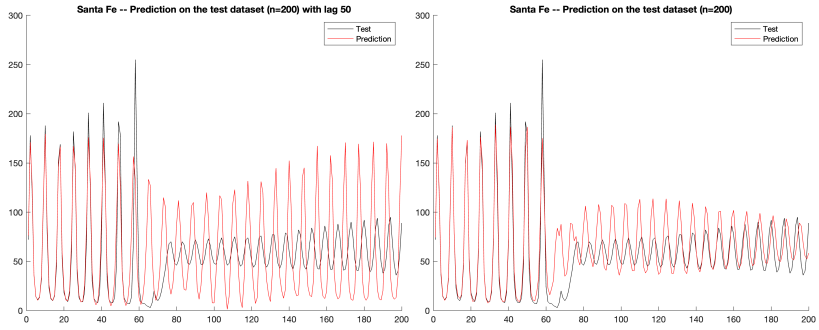

Figure 27: SVM Regressions on the Santafe dataset using fine tuned γ and σ^2 and order 50 (left-wing) and 48 (right-wing)

This procedure yield a model with an MSE of 508.97, sensibly lower than the one obtained using order = 50. The two models are shown in Figure 27. Although the model yielded a better performance, the computational time was rather long (1.5 hours) and, consequently, a different approach shall be used in future applications. Instead of optimizing the γ, σ^2 and order parameter in a single stage, a multi-stage approach could be applied. In the first stage, random "starts" for the order parameter are taken from a uniform distribution and used to estimate a set of models with optimized γ and σ^2 parameters using 5 runs of an automatic parameter selection with 10-fold cross validations. The best performing model is selected and the order parameter is used to construct a -10 to + 10 around it to use in the second-stage. Results from the second-stage are used in a third stage with an interval of -3 to + 3. In the last stage, the model is fine-tuned using an automatic parameter selection of γ and σ^2 with 100 runs with 1 unit increment in the order. Rules for similarly performing models can be introduced at each stage to avoid local minima.

Assignment 3

Kernel principal component analysis

Kernel principal component analysis (KPCA) can be used in situations where the data are non-linearly distributed. Similarly to what we have seen so far, the introduction of non-linearity can be achieved through mapping the data from the original space into a higher dimensional feature space using the so-called "kernel trick". This allows to keep the problem in the form of an eigenvalue problem as done in traditional (linear) PCA.

KPCA can also be used to denoise data. The general idea is to use KPCA to find the direction with maximal variance of a given target space. This means being able to extract the components that matter the most, leaving the noise out. In this case, the principal components should capture and represent the main structure of the data while, at the same time, excluding the unnecessary (noisy) information. Along with the hyper parameters σ^2, the other relevant parameter to tune is the number of principal components to extract. If the number of components is lower than the optimal number, the model will disregard important information present in the data and, thus, will not be able to properly represent its structure. On the contrary, if the number of components is higher than the optimal number, the model will also extract the noise present in the data, making the denoising procedure less effective.

I showcase the ability of KPCA using synthetic non-linear data. As mentioned before, linear PCA will not be able to extract the main structure of the data and, thus, to property denoise it. This is exemplified in Figure 28. The linear PCA model is completely unable to capture the structure of the data. On the contrary, a KPCA model with 6 principal components and $\sigma^2 = 0.4$ is able to recognize that the data is composed of two curvilinear distributions. While in linear PCA the maximum number of components is equal to the number of variables, in KPCA it is equal to the number of observations. It is worth noting that in KPCA the boundary case when the number of extracted components equals the number of data points, a representation of the original noisy data is returned since we are not disregarding any variance and, consequently, noise.

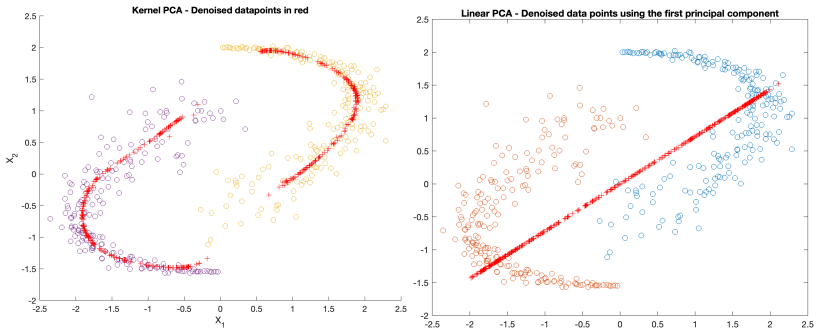

Figure 28: Denoising non-linear daata using linear (left-side) and kernel (right-side) PCA

A way to tune the number of components and the hyper parameter in the presence of noisy data would entail a multi-stage optimization. First, a KPCA is fitted to the data and to the eigenvectors (i.e., the principal components) are extracted. Each eigenvector is associated with an eigenvalue which represents the explained variance in the data. A high eigenvalue means high explained variance. Second, by visually exploring the eigenvalue distribution or using an arbitrary threshold, a smaller subset of the eigenvalues is selected. Third, these selected eigenvectors are used to denoise the data. Fourth, the kernel parameters are fine-tuned using an automatic selection algorithm that minimizes the reconstruction error between the original and the de-noised data. Intuitively, the reconstruction error is similar to the MSE and corresponds to the error between the original data point and the projection of the new reconstructed image.

Fixed-sized LS-SVM

SVMs generally rely on convex optimization theory. Usually, the SVM is formulated in the primal space as a constrained optimization problem and then the problem is solved in the dual space using the Lagrange multipliers (i.e. the support vectors). Solving in the primal space can be interpreted as a parametric approach since the model depends on the dimension of the input space. On the contrary, the dual problem corresponds to a non-parametric approach since the size of the $|W|$ will be fixed independently of the number of data points. Consequently, for high dimensional data, it may be useful to solve in the dual space while for large datasets is better to solve the problem in the primal space.

The primal-dual representation is useful in a fixed size LS-SVM algorithm where SVMs are applied to larger data sets. This is because, as mentioned before, the problem can be solved in the primal instead of the dual. The algorithm for fixed size LS-SVM consists of multiple steps. First, a random subset of points is selected from the data (usually standardized and normalized). Second, a random point in the subset is replaced by taking, at random, a point from the original training data. If the quadratic Renyi entropy increases, the point is kept, otherwise is returned to the training set. The algorithm stops when the change in the entropy is smaller than a given threshold or after a fixed number of iterations.

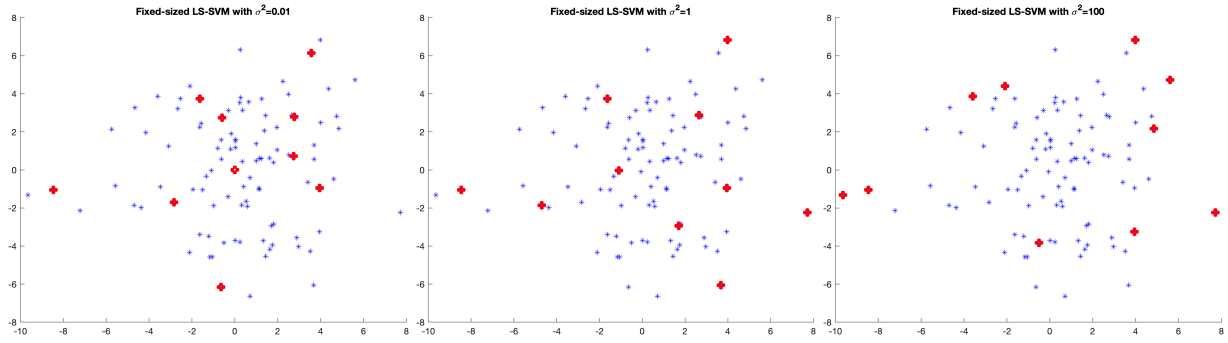

Figure 29: Fixed-sized SVM with varying σ^2 (i.e., 0.01, 1, 100)

We can notice from Figure 29 that the selection of the support vector is similar across different values of σ^2, yet larger values of σ^2 result in the selection of a subset of the points that is at the boundary of the original input space. This is because, the algorithm always tends to maximise the entropy and, thus, select points that are further away from each other, and thus more spread.

Another technique can be used to achieve sparsity and, thus, reduce the number of support vectors in large datasets. This method is called l_0 approximation and is based on an iterative approximation to the l_0-norm. In this section, I will compare the l_0 approximation with a fixed-size LS-SVM using an RBF kernel. The two techniques are contrasted on the number of support vectors, error term, and computation time. I use the Wisconsin Breast Cancer dataset to perform this comparison. This version of the dataset is composed of 400 observations for the training and 168 for the testing. The dataset has 9 features that refer, *inter alia*, to a set of characteristics of the cancer cell nucleus.

Figure 30 reveals that the support vectors used by the fixed-sized LS-SVM are approximately 105 which is a substantial reduction compared to the 682 original data points in the dataset (85% reduction). The l_0 approximation results in an even lower number of support vectors that are now

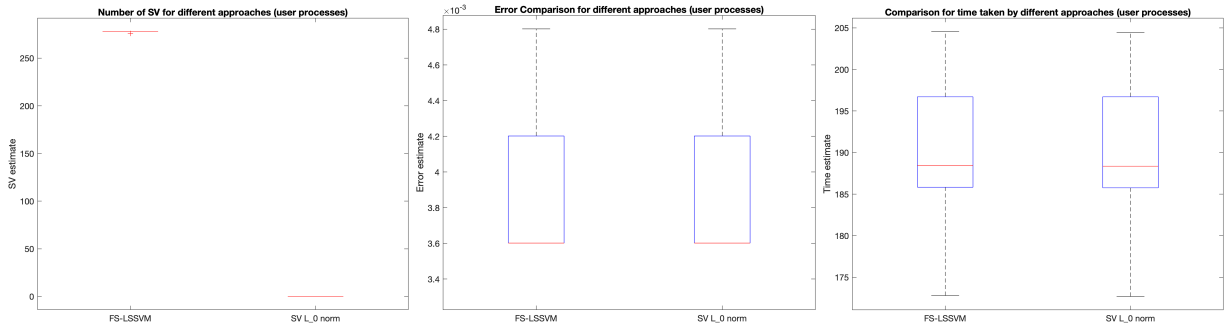

Figure 30: Comparing l_0 approximation with a fixed-size LS-SVM using an RBF kerel

approximately 10. This reduction comes with an increased error on the test set. However, the increase in the error is rather limited as shown in the central plot of the figure. Finally, the computational time between the two approaches is very similar. I can then conclude that both approaches are suitable to reduce the number of support vectors. Yet, l_0 approximation may be preferred in for very large scale data problems where the number of observations needs to be substantially reduced and sparsity achieved.

Homework problems: Kernel principal component analysis

In this section, I will use kernel principal component analysis (KPCA) to perform image de-noising. As mentioned before, image de-noising is a technique that can be used to obtain cleaner images by removing noise introduced in the data generation or collection process. The dataset used is the Dutch utility maps and consist of images of handwritten numerals (0 to 9). Approximately 20 patterns per class are present in the dataset. For the purpose of this exercise, I first compare the performance of linear against a kernel PCA in de-noising the digits when a high amount of Gaussian noise is synthetically introduced in the digits. As noted before, Figure \@fig(fig:fig29) shows the linear kernel is completely unable to de-noise the images, regardless of the number of principal components. On the contrary, kernel PCA with 16 or more principal components is able to detect most of the digits. Additional improvements can be made by fine-tuning the kernel parameters (*vedi infra*).

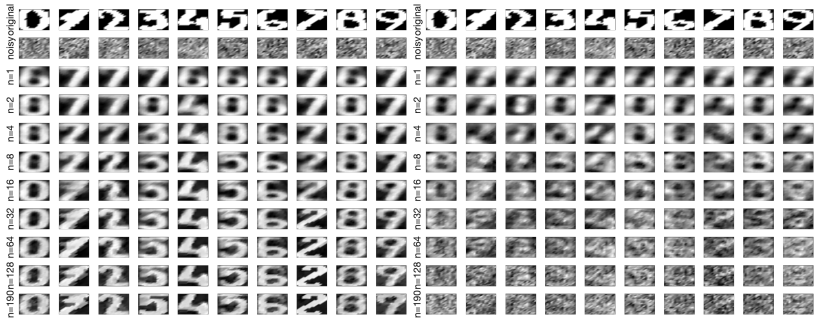

Figure 31: Comparison of the kernel (left-side) and linear (right-side) in denoising the Dutch utility maps digits.

Next, I assess the impact of varying values of σ^2 in de-noising the digits. By modifying the `sigmafactor` parameter, I obtain 3 values of σ^2 that correspond to a low (0.051), median (5.1), and high (512) value for σ^2. Figure 32 indicates that for relatively low values of σ^2 the KPCA model fails to de-noise the data. On the contrary, with a median value of σ^2, the model is able to de-noise the data and I am able to recognize the digits. This can be explained by looking at how the σ^2 parameter works in a KPCA. As mentioned before, the main idea of KPCA is to find the direction of maximal variance around a target point 0. Similarly to what happens for traditional PCA, the eigenvalues are obtained from a matrix of kernels (i.e. the gram matrix $K(x_i, x_j)$) that has a similar purpose to the correlation matrix used in linear PCA. In this case, $k(x_i, x_j) = exp(-norm(x_i - x_j)^2/\sigma^2)$ and, thus, σ^2 can be interpret as a parameter that controls the "distance" between the two kernels. When σ^2 is very low, $K(x_i, x_j)$ will have the maximum distance (the upper bound of 1), regardless of the actual distance of the kernels. On the contrary, if σ^2 is high, the distance will be at its minimum (the lower bound of 0), again regardless of their actual position. This implies that in these two boundary cases the spatial representation of all the data points will be approximately the same, leading to the observed inability to de-noise the digits.

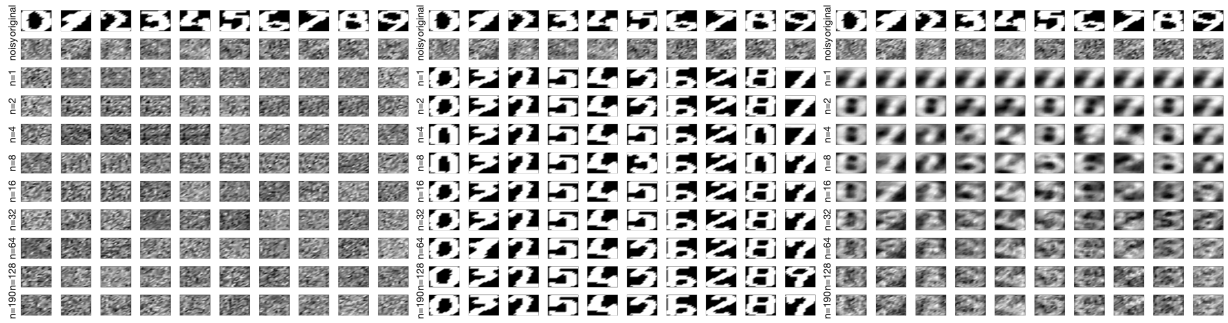

Figure 32: Comparison of the kernel with varying σ^2 (low, middle, high).

Finally, I will optimize the parameters σ^2 and the number of principal components for the KPCA model. Similarly to what has been done for the time series prediction, I use a manual grid search using different σ^2 (i.e., 0.1, 1, 10, 25, 50, 100, 200, 400) and principal components (i.e., 2, 4, 8,1 6, 32, 64, 128, 190). The optimization routine involves, first, training a model using the digits without noise (i.e., X in the original digits dataset). The eigenvectors computed from this non-noisy data are subsequently used to de-noise a validation set (i.e., Xtest1) where noise has been synthetically added (i.e., `noisefactor=0.3`). The MSE (i.e. reconstruction error) between the de-noised dataset and the original digits without noise is computed and the best σ^2 and number of principal component parameters are extracted and used to predict a test set (i.e', Xtest2) where noise has been synthetically added (i.e., `noisefactor=0.3`).

The results from the grid search are shown in Figure 33 (left-side) along with the prediction on the test set (right-side). The best model performance is achieved with $\sigma^2 = 50$ and 128 principal components. It is important to notice that these parameters can be further optimized using a more fine-grained grid search. I can conclude that a fine-tuned KPCA model is able to properly reconstruct the digits and, thus, that the proposed approach is useful to de-noise the data at hand.

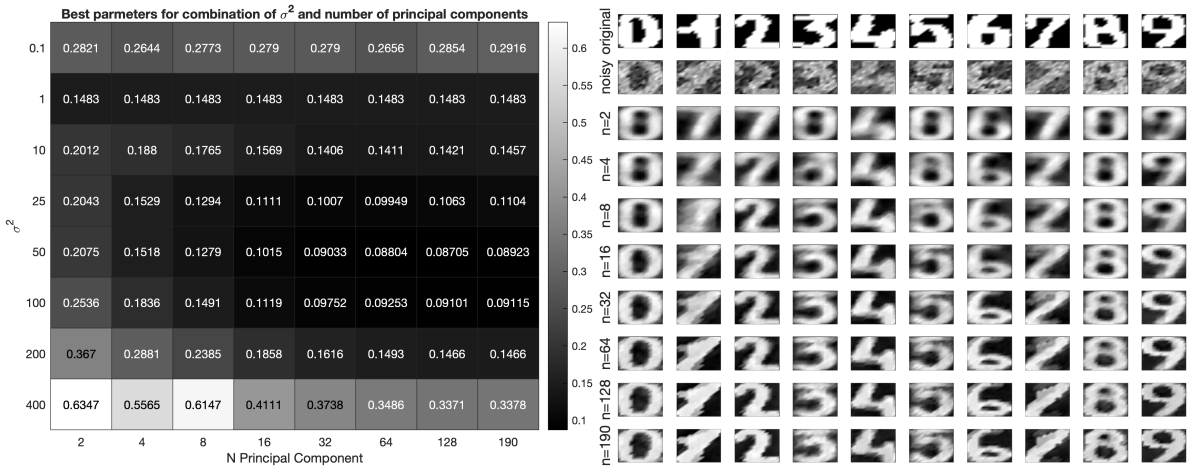

Figure 33: Gridsearch for σ^2 and PC (left-side) and validation on the test set (right-side)

Homework problems: Fixed-sized LS-SVM for the Shuttle (statlog) dataset

In this section, I use Fixed-sized LS-SVM to classify the Shuttle dataset. The Shuttle dataset is a large dataset with 58,000 observations, 7 different classes, and 9 explanatory variables that represent the conditions of the landing of an aircraft. When classifying multi-class data two approaches can be used. One consists of transforming the classification problem into a binary problem by looking at each pair of classes (e.g., Class 1 VS Class 3). The other approach entails collapsing multiple classes together to obtain only two classes. Since the first 1 class accounts for 80% of the observations, I decided to follow the second approach and group all observations that belong to classes 2 to 7 into a single class. Given the limited amount of computational power at my disposal, I selected a random subset of the data

To visualize the data, I standardized the input data (all continuous variables) and use a t-Distributed Stochastic Neighbor Embedding (TSNE) that allows me to reduce the dataset to a 2-dimensional array that can be plotted.

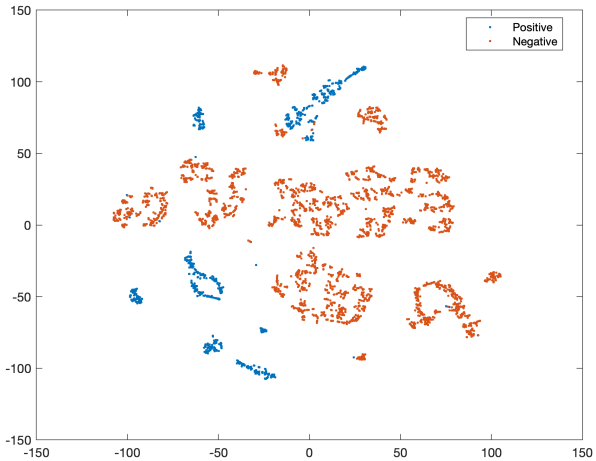

Figure 34: TSNE dimensionality reuction for the Shuttle (statlog) dataset

Figure 34 shows that, in most of the areas, the classes are well delimited and, thus, I expect a good classification accuracy for the binary problem for most of the data points. Given the large size of the dataset, a l_0 approximation may be desirable since, as seen earlier, it reduces the number of support vectors more aggressively compared to Fixed-sized LS-SVM without a substantial increase in the error. However, we compare Fixed-sized LS-SVM and l_0 to be sure that what we have observed before holds in Shuttle dataset.

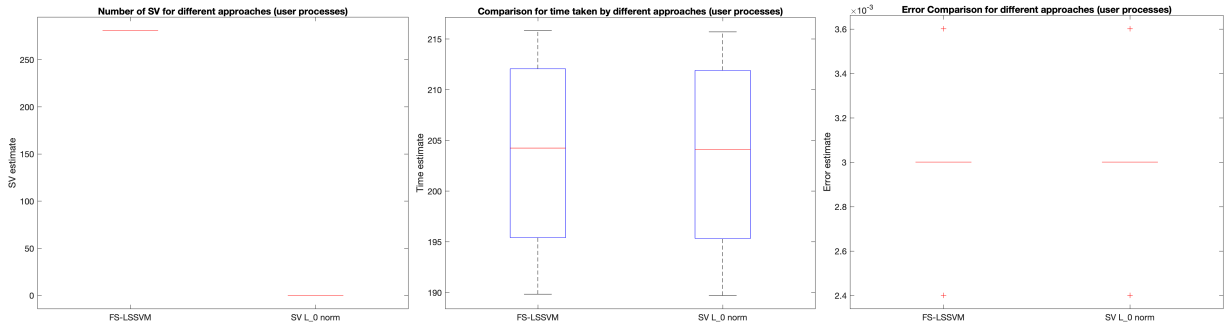

Figure 35: Comparing l_0 approximation with a fixed-size LS-SVM using an RBF kerel

Figure 35 shows the comparison between the two techniques on the number of support vectors, errors, and computation time. Both techniques are useful to reduce the number of support vectors. However, as expected, the l_0 approximation is more aggressive and uses as support vectors less than 1% of the initial 5000 observations while the Fixed-sized LS-SVM retains approximately 5% of the initial vectors. Given the computation time and the error rate between the techniques are similar, in this case, l_0 is the preferred technique.

Homework problems: Fixed-sized LS-SVM for the California dataset

In this section, I use Fixed-sized LS-SVM on the provided California dataset. The dataset consists of 20640 observations and contains information on the houses found in a given California district and some info based on the 1990 census data in relation to the block in which the house is located (e.g. total number of households in a given block). In this case, the dependent variable is the median house value within a given block.

To analyse the data, I follow a similar approach to the one used for the Shuttle (statlog) dataset. First, I randomly select 1000 observations to reduce the computational burden. Second, I fit a TSNE model to plot the dataset in two dimensions and used a colour gradient that goes from blue (least expensive) to yellow (most expensive). Figure 36 shows that the separation between low and high-income houses is not as clear, yet the most expensive houses are located on the right- and left-hand sides of the data cloud. Consequently, I expect both Fixed-sized LS-SVM and l_0 approximation to need more support vectors compared to the previous example.

Figure 37 compares the Fixed-sized LS-SVM and l_0 approximation approach. My intuition is confirmed for the Fixed-sized LS-SVM. The model retains 120 support vectors, equal to approximately 12% of the initial observations. However, the l_0 approximation retains less than 5% of the initial vectors and shows exactly the same error as the Fixed-sized LS-SVM. Consequently, even using

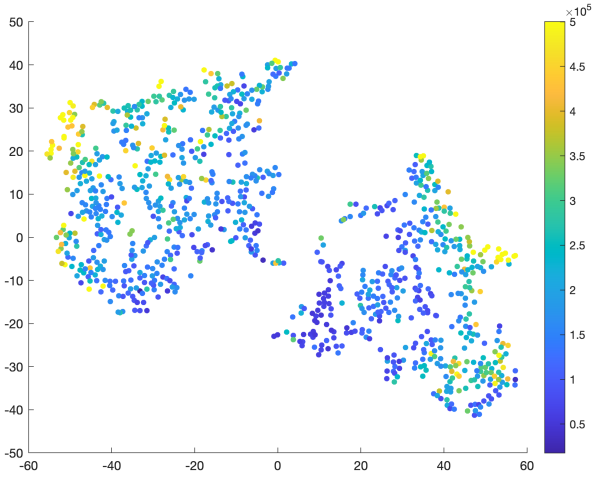

Figure 36: TSNE dimensionality reuction for the California house dataset

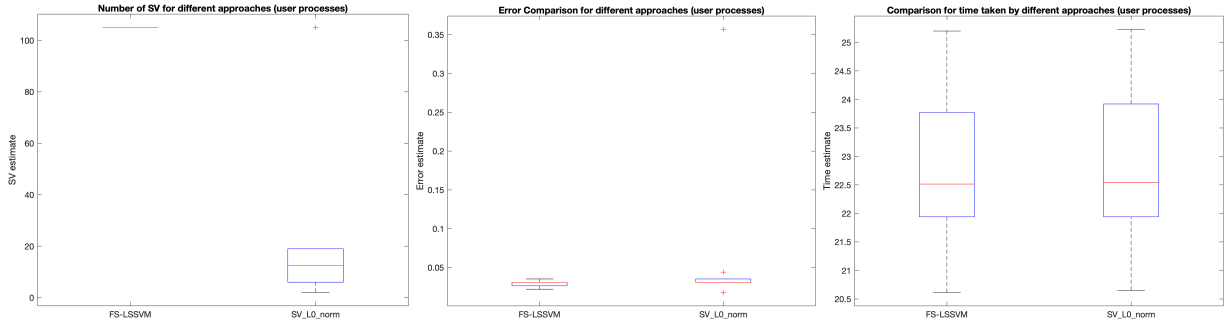

Figure 37: Comparing l_0 approximation with a fixed-size LS-SVM using an RBF kerel

continuous data where the separation between high and low income is not as clear, the l_0 approximation appears to achieve more sparseness compared to the fixed size LS-SVM, and thus, may be preferred. Nonetheless, to be sure that the l_0 approximation is better at classifying the data, the two approaches should be compared using a similar approach to what has been done many times before where the ROC curves or the MSE is calculated between a train and a test set.

References

Suykens, Johan A. K., Tony Van Gestel, Joseph De Brabanter, Bart De Moor, and Joos P. L. Vandewalle. 2002. *Least Squares Support Vector Machines*. World Scientific.