SVM : assignment 1

Florentijn Degroote

April 2019

Contents

1 Exercises

1.1 A simple example : two gaussians

In the binary classification problem as depicted in figure 1, the linear (2D) decision boundary is the optimal solution. This is because the two classes have the same covariance matrices and are thus a special case of the quadratic discriminant functions. In such special cases, bayesian decision theory proves that the optimal solution always is a hyperplane, independent of the overlap between the two classes. When the covariance matrices are unequal, then the decision boundary would be quadratic.

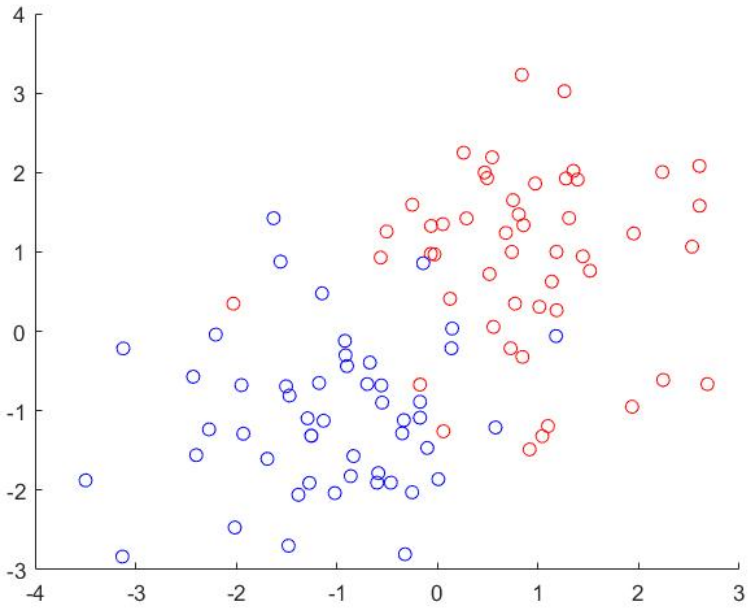


Figure 1: Graphical representation of the binary classification problem.

1.2 Support vector machine classifier

On the provided demo, created by Stanford University, one can play with SVM's using linear kernels and RBF (radial basis function) kernels, by changing the datapoints in the binary classification problem.

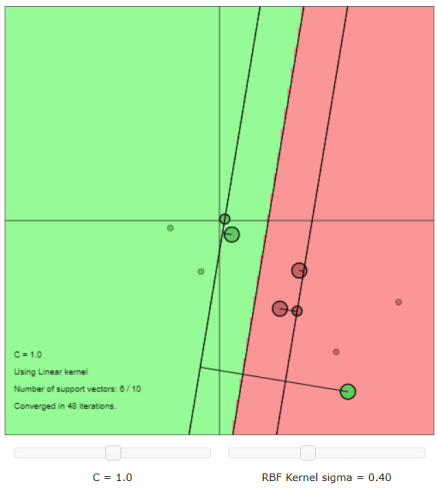
1.2.1 Linear kernel

When adding points to the starting position (figure 2a) outside the margin, only slight changes occur qua decision boundary, as shown in figure 2b. Each decision boundary corresponds with solving a new SVM optimization problem. The margin is graphically defined by the two lines that run parallel with the decision boundary, and its width is maximized as solution of the optimization problem (minimizing $2/||w||^2$), hereby giving the convex character to SVM problems. This means that there is only one true solution when the SVM has converged.

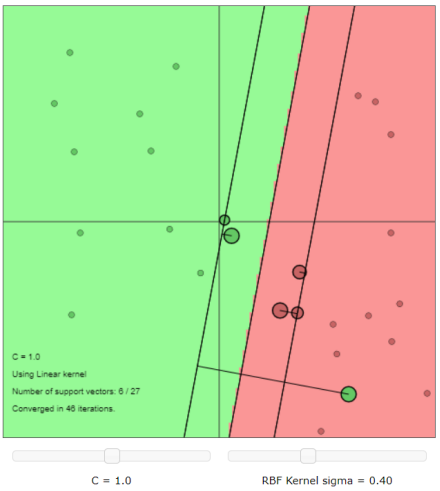
If points are added within the margin, they take the role as support vectors, depicted as big coloured circles in figure 2c. These support vectors are the non zero elements in kernel matrix, obtained when solving the convex QP problem (sparsity property). The decision boundary can be expressed by a these support vectors; that's also why they reside close to the decision boundary, as they have a big effect on how this boundary is situated. Adding points within the margin effects the decision boundary quite heavily.

On figure 2d points were added on the "wrong" side of the boundary. Graphically, we see that the boundary is influenced heavily by these changes. All these points take up the role as support

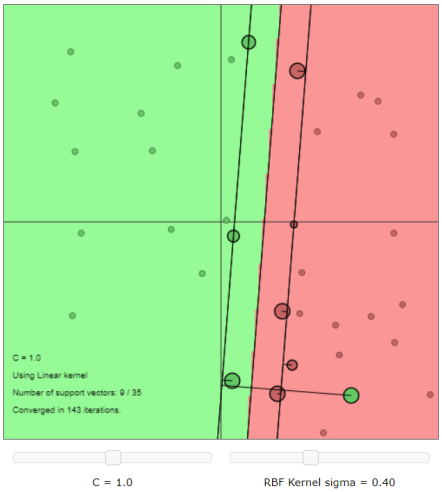
vectors, even when residing outside of the margin. They all have an influence on the width of the margin as well.



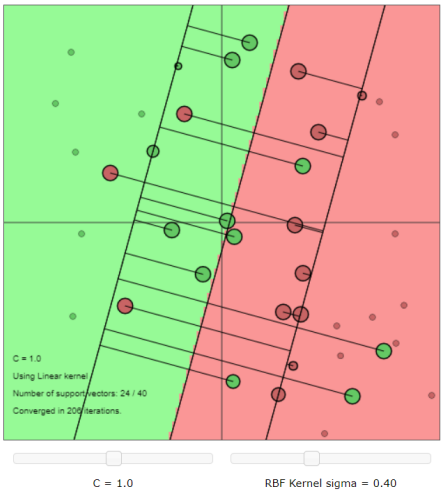
(a) Starting position



(b) Added points outside margin



(c) Added points inside margin



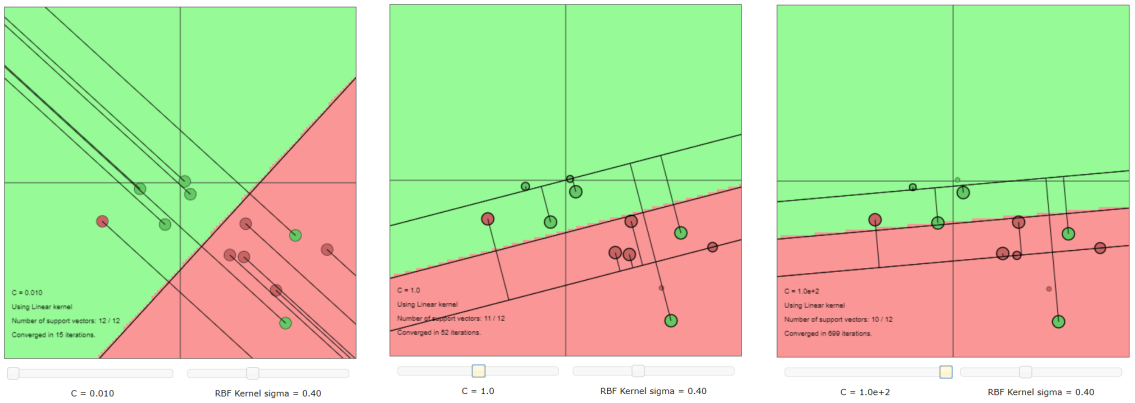
(d) Added points on the wrong side of the dicision boundary

Figure 2: Screenshots of the graphical results of the linear kernel for different data sets.

Previous results were obtained for a value of 1.0 as the c parameter. The σ parameter does not affect the linear kernel as it is specific to RBF kernels. Equation 1 contains the c parameter and shows the optimization problem in its primal representation.

$$\min_{w,b,\xi} \Im(w,\xi) = \frac{1}{2} w^T w + c \sum_{k=1}^{N} \xi_k \tag{1}$$

Choosing a high value for c prioritizes the influence of the slack variables (ξ) on the problem. Slack variables are a measure of how "wrong" a datapoint is classified. When the c value is chosen low (as in figure 3a; 3 misclassified points, margin is huge), the solution is not really affected by how many points are misclassified. For mid-range values for c, such as 1 in figure 3b, the margin is smaller and there are still three points misclassified. Though, the distance between the misclassified points and the boundary line is smaller. For big c values as depicted in figure 3c, only 2 points are misclassified. It is thus a trade-off.



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

Figure 3: Screenshots of the graphical results of the linear kernel with different c values.

1.2.2 RBF kernel

An RBF kernel permits the kernel to correctly classify non-linear data. The implemented RBF kernel in the dual representation of the SVM looks like equation 2.

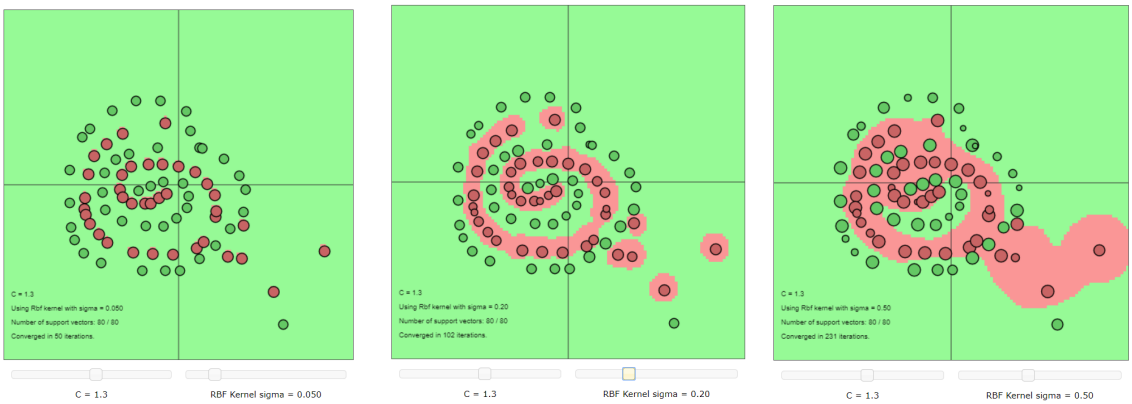
$$y(x) = sign[\sum_{k=1} \alpha_k y_k \exp\left(-\frac{||x - x_k||_2^2}{\sigma^2}\right) + b] \tag{2}$$

When wielding this kernel and adding points to the figure, the SVM will create an "island" around the minority class' points (figure 4a, to make sure that point is classified to the right class. The background colour (or default class new datapoints who are really far apart from any data point) is equal to the class with the highest amount of points (see difference figures 4b and 4c).

(a) Starting position (b) Added one more point (c) Island are formed after adding multiple points

Figure 4: Screenshots of the graphical results of the RBF kernel.

Here, not only the c parameter can be changed, which can be interpreted just like in linear kernels, but also the σ (σ^2 =bandwidth) parameter. As one can interpret by looking at equation 2, choosing a low value for σ will result in a very shallow (multi/2)dimensional gaussian. For high values, the (multi/2)dimensional gaussian will be wide and less peaked. For the highly non-linear data set was (and holding the c variable at 1.3), taking a low σ value (figure 5a) shows that the red points have a very small island (decision boundary) around them, which is a form of overfitting. For a mid-range value, the SVM seems to generalize better and depicts the non-linear behaviour nicely (figure 5b). For high σ values, the RBF kernel is smoother and tends to behave more and more linearly (figure 5c) because the rbf itself is getting more and more flat. In this last figure, changing c to lower values will result in more misclassifications whereas higher c values will result in no misclassifications but the "islands" around the red dots will be a lot smaller, a bit like in the



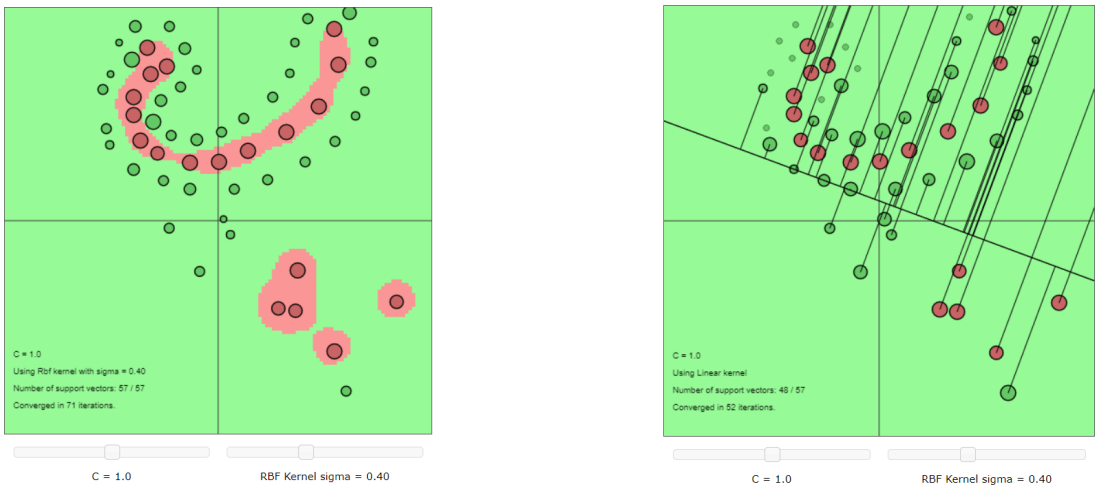
(a) σ=0.05 (b) σ=0.2 (c) σ=0.5

Figure 5: Screenshots of the graphical results of the RBF kernel.

5a. The support vectors for low sigmas include all the points, whereas the number drops and is interpretable as with the linear kernel case for higher values of σ. Both the c and σ values do have to application-specific and are subject to the focus of the results one wants of the SVM.

1.2.3 Comparison

As said, the linear kernel is fit for linear seperable data sets when it comes to classification. Also, for applications that have high dimensional datasets, the linear kernel is mostly used. Although the RBF kernel is also able to handle linear data sets, the added complexity (having to tune one more parameter in comparison with the linear kernel) makes us commit to linear kernels in these cases. For low dimensional non-linear data sets, the RBF kernel is preferred (figure 6a and 6b).



(a) High non-linearity with RBF kernel (b) High non-linearity with linear kernel

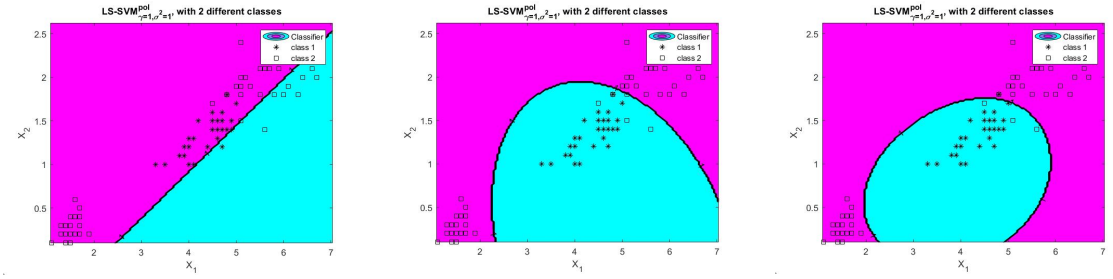
Figure 6: Highly non-linear dataset classified using different kernels.

1.3 Least-squares support vector machine classifier

1.3.1 Influence of hyperparameters and kernel parameters

Using the Iris data set, the effect of the highest order polynomial in the polynomial kernel is looked into. The regularization parameter to 1 and the intercept of the polynomial to 1, and the polynomial takes the values 1, 2 and 3. For the linear case, an error rate of 55% was achieved. The 2D representation of the classifier is shown in figure 7a. For the second order polynomial the error rate is 5%. For the third order (and higher) the error rate is 0%. Setting a higher polynomial

allows the SVM to be more flexible in function of X1 and X2. The results might look a bit like the results one would obtain from using RBF kernels, though outliers in this case would not be classified the way RBF would handle them (of course this depends on the parameters of the RBF kernel).



(a) 1^{st} order (b) 2^{nd} order (c) 3^{rd} order

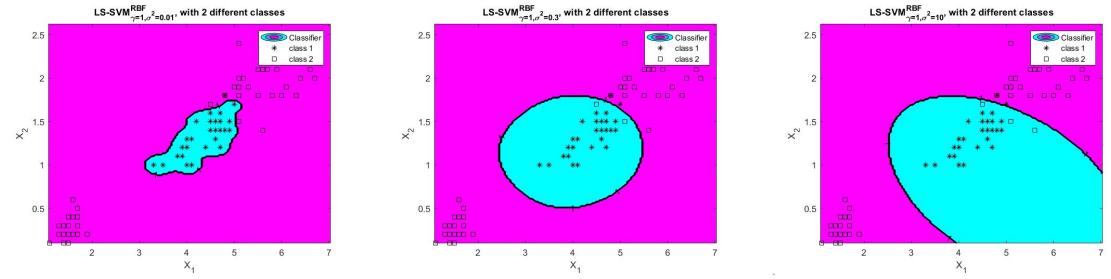
Figure 7: Graphical results for the polynomial kernel for different degrees.

RBF kernels also involve a tuning parameter (as mentioned before): the bandwidth or σ^2. The effect of different values of this parameter is shown in table 1. Again, the regularization parameter was fixed to 1. Choosing σ^2 between 0.1 and 10 results in no wrongly classified points. From value

σ^2	0.01	0.03	0.1	0.3	1	3	10	30	100
error rate (%)	0.1	0.05	0	0	0	0	0	0.5	0.5

Table 1: Error rates for different bandwidth values of the RBF kernel

30 on, the function outputs a warning that says "Simulation over the input space results in only one class". This also explains the 50%, as half of the test data belongs to the first class and the other half to the second. The results of the values 0.01, 0.3 and 10 are shown in figures 8a, 8b and 8c, respectively. Let's now fix this parameter to 0.3 and have a look at different values for the regularization parameter γ, shown in table 2



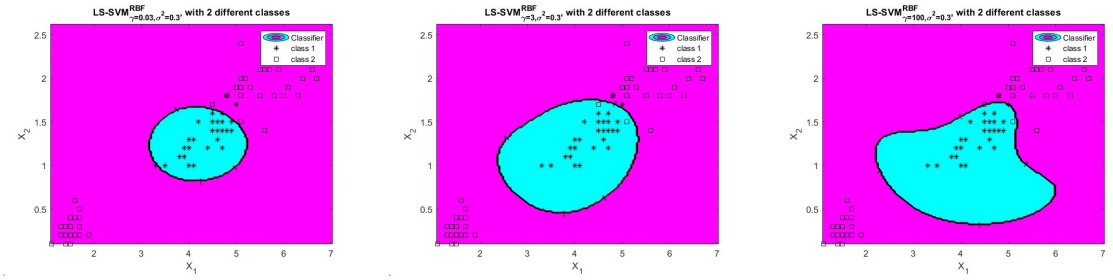
(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 0.3$ (c) $\sigma^2 = 10$

Figure 8: Graphical results for the polynomial kernel for different degrees.

γ	0.01	0.03	0.1	0.3	1	3	10	30	100
error rate (%)	0.5	0.15	0.05	0	0	0	0	0	0

Table 2: Error rates for different regularization values for the RBF kernel

For a σ^2 value of 0.3, no regularization parameter's value higher than 1 will result in errors. Again, the same warning as before was shown and now for the case of gam being 0.01. Figures of the cases where gam is chosen to be 0.01, 0.3, and 100 are shown in figures 9a, 9b, and 9c, respectively. It seems that with increasing values of the gam, the SVM is generalizing better. The results that are obtained by the sample script also work well. A very thorough comparison between my obtained results and the ones from the sample script is hard, because of the simplicity of the iris dataset.

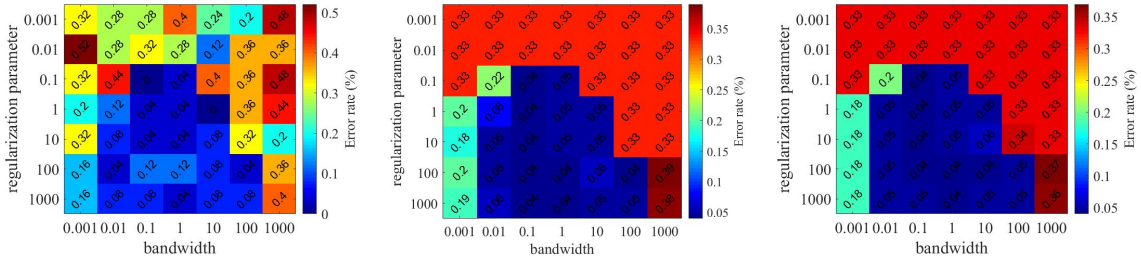


(a) $gam = 0.03$ (b) $gam = 3$ (c) $gam = 100$

Figure 9: Graphical results for the polynomial kernel for different degrees.

1.3.2 Tuning parameters using validation

For the same data set, three different validation methods are looked into, namely simple random train and validation split (75% train, 25% validation), 10 fold cross validation and the leave one out method. Results for a grid search of the hyperparameters γ and σ^2 are shown in figures 10a, 10b, and 10c.



(a) Simple train-validation split. (b) 10 fold cross validation. (c) Leave one out method.

Figure 10: Resulting error rates for different hyperparameters and validation methods.

Since the training set is quite small and simple, the random split generates very different results that do not seem consistent at first. Because the data set is small, the distribution of the actual classification is at times not depicted well by the training subset. We could have guessed that there using medium valued bandwith and medium valued regularization parameters could do well. Cross-validation and the leave one out method however, do show some consistent results and clearly show that the good parameters are found for high values for the regularization parameter and medium values for bandwidth. Results now depend way less on what sets are taken, for cross validation as a lot of bias and randomness is accounted for. For the leave one out method, the results are perfectly deterministic (because LSSVM's themselves are). It must be noted that of cross validation and even more so, the leave one out method, take longer to compute as they impose multiple models to be constructed and validated. Overall, the error rates are smaller for leave one out method than 10 fold cross validation and even smaller for simple train-validation split. This is because all but one of the points are included each time for the leave one out method, 90% of the points are included in the cross validation training set each time and only 75 % are included in the split training set.

When data sets are small, the leave one out method gives very good results but when the data sizes increase, one could opt for k-fold (with k lower than the number of training data, otherwise it's the leave one out method) cross validation because of limited resources (CPU). Choosing the value of k depends on two factors;

1. Larger k means less bias towards overestimating the true expected error (as training folds will be closer to the total dataset) but higher variance and higher running time (as you are getting closer to the limit case: Leave-One-Out CV).

2. Too large k means that only a low number of sample combinations is possible, thus limiting the number of iterations that are different.

Taking both of these into account, a trade-off should be made. 10 or 100 are arbitrary numbers that are often used. In big data cases one could be satisfied with the simple train-validation split. Now

that we determined good parameters based on the training set, we cannot immediately conclude that we have a good model. That should be determined with the test set.

1.3.3 Automatic parameter tuning

Instead of tuning the parameters ourselves, we can use the LSSVM toolbox. Two different methods are implemented there, the Nelder-Mead method that makes use of Simulated Annealing search and on the other hand, the brute force gridsearch (similar as we did in the previous subsection). The results of three runs of either algorithm is shown in table 3.

	First run		Second run		Third run	
	γ	σ^2	γ	σ^2	γ	σ^2
Brute force method	268374.7825	2.7444	536.1305	0.0518	44.3640	0.0154
Nelder-Mead method	941515.2377	7.5429	4.1244	0.04404	162242.4192	0.8891

Table 3: Error rates for different bandwidth values of the RBF kernel

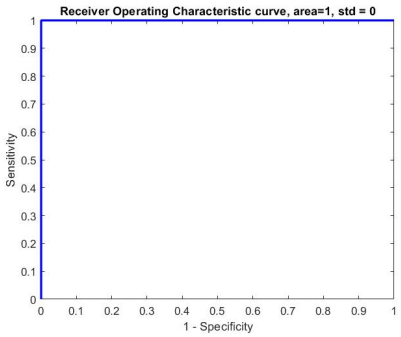
Both algorithms return different results for the hyperparameters each time, of which each of them corresponds to the results shown in figures 10a, 10b, and 10c. For both algorithms, the cost is always equal to 0.02, 0.03 or 0.04. Simulated annealing is a local search method and returns the parameters when no parameters are found in its neighbourhood (also based on the temperature, which is a parameter of the algorithm) [1]. Made easy, it returns a solution when the algorithm has found a local minimum. Brute force search returns the best (based on cost minima) solution of the options it has checked, of which the borders of the values were first based on the results of simulated annealing. The Nelder-Mead method starts in a similar matter (with the simulated annealing search) and then stops when no better result can be achieved within a defined simplex (special polytope of n + 1 vertices in n dimensions) [2]. Their results are very similar for this small data set. The mean loss outputted by the brute force algorithm is 0.0351, and 0.0359 for the Nelder-Mead method, with almost no difference in variance of those 100 runs (3.7364e-05 vs 3.4535e-05), respectively). The computation times however were different. The brute force needs 0.8 seconds on average, while the Nelder-Mead method can find good parameters in 0.36 seconds. The latter is thus a much more performant algorithm and should be preferred.

1.3.4 Using ROC curves

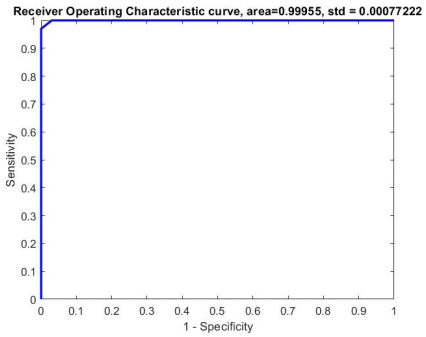
ROC curves are a measure of performance of your model. The goal is to maximize the area under the curve (AUC). This AUC is determined by the True Positive Ratio (TPR) and the False Positive Ratio (FPR). For a particular class i, TPR is the number of outputs whose actual and predicted class is class i, divided by the number of outputs whose predicted class is class i. FPR is the number of outputs whose actual class is not class i, but predicted class is class i, divided by the number of outputs whose predicted class is not class iBecause the model has been trained on the training set and thus might be overfitted to that particular data. When computing the ROC curve on that same data, results will be good, even though the ROC curve on the test set can be low. So using the ROC curve on the test set is a more gullible way of measuring performance of your model. In figure 12a, the ROC curve on the test set is generated for a bandwidth of 0.3 and gam chosen to be 100 (the same case as figure 9c). On figure 12b the ROC on the training set is shown. Because in the training set, one particular point of class one has the same coordinates of a particular of class 2, the AUC is not 100%. This is not the case for the test set, and the classifier is able to classify the points perfectly. For a worse combination of γ and σ^2, such as 10 and 0.1, respectively, the ROC curves of test and training set look worse (figures 11a and 11b). In addition, it should be mentioned that these ROC curves look pretty stair-like. This is because at a certain cut off point, the TPR and FPR rates change because more latent variables are correctly/wrongly classified, and because there are not a lot of latent variables. For huge amounts of latent variables, the ROC curves look more smooth (can already be perceived by the difference of the training set and test set).

1.3.5 Bayesian framework

The probability plot, generated within the bayesian framework of the tuned parameters (bandwidth = 0.3 & regularization = 100) is shown in figures 13a. The cases similar to 8b and 8a are shown

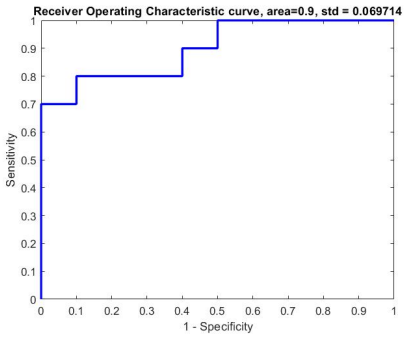


(a) ROC on test set

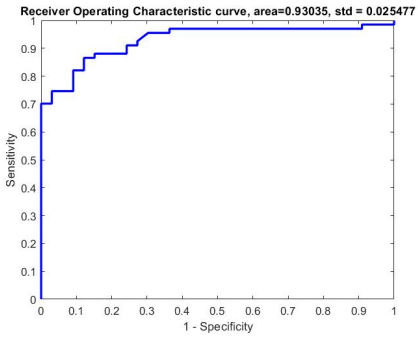


(b) ROC on training set

Figure 11: ROC's with tuned parameters.



(a) ROC on test set

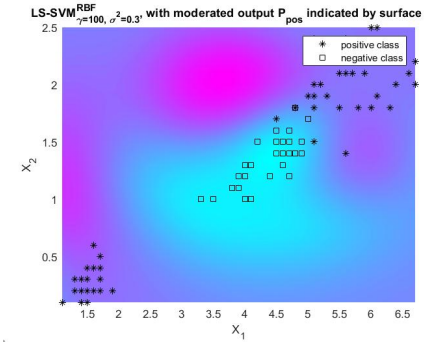


(b) ROC on training set

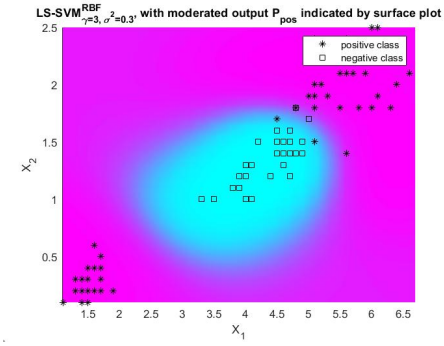
Figure 12: ROC's with a bad parameter combination.

in figures 13b and 13c. In figure 13d an extra plot is generated to interpret the effect of the bandwidth.

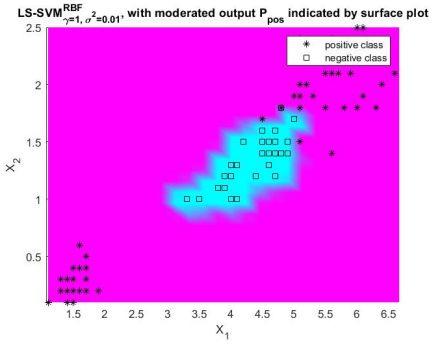
The color purple denotes high probabilities of points in that area belonging to the crosses' class. Light blue denotes a probability of zero percent of the point belonging to the cross class (or 100 % that the point belongs to the square class). For high valued regularization parameters and higher bandwidths, the probability differences do not change rapidly when hovering over the 2D figure (all probabilities between 45 and 70 percent on figure 13a). For low values of both (especially in figure 13c), the probabilities change drastically on the border.



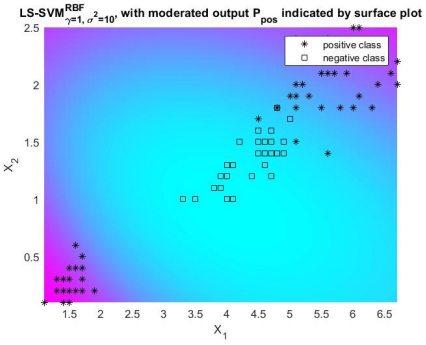
(a) Tuned parameters.



(b) $\gamma = 3$ and $\sigma^2 = 0.03$



(c) $\gamma = 1$ and $\sigma^2 = 0.01$



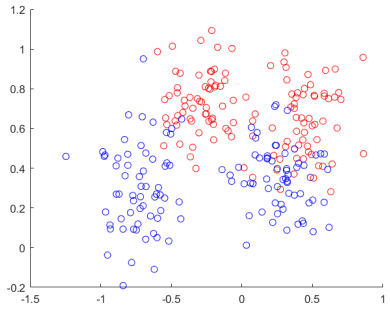
(d) $\gamma = 1$ and $\sigma^2 = 10$

Figure 13: Probability plots of different settings.

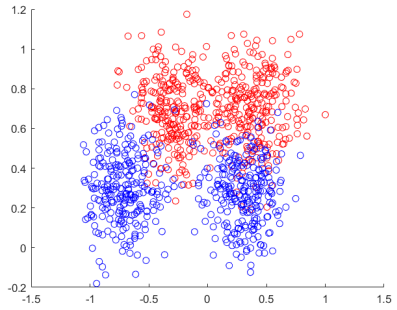
2 Homework problems

2.1 Ripley data set

Preliminary analysis consists of looking at the raw data, amongst other things. From the raw data we can conclude that the Ripley data is about a binary classficiation task, where there are two features. Now, a 2D representation of the training set (figure 14a) and test set (figure 14b) is looked at. Here we see that there is quite some overlap between the two classes in the training set, implying that some error margin can be expected. On first sight, it is questionable whether the linear kernel would do a good job, compared to the polynomial or RBF kernel. From the test set we can conclude that there are more points than in the training set and that the distributions from both sets are more or less equal. In a next step, the results of manually tuned linear and



(a) training set

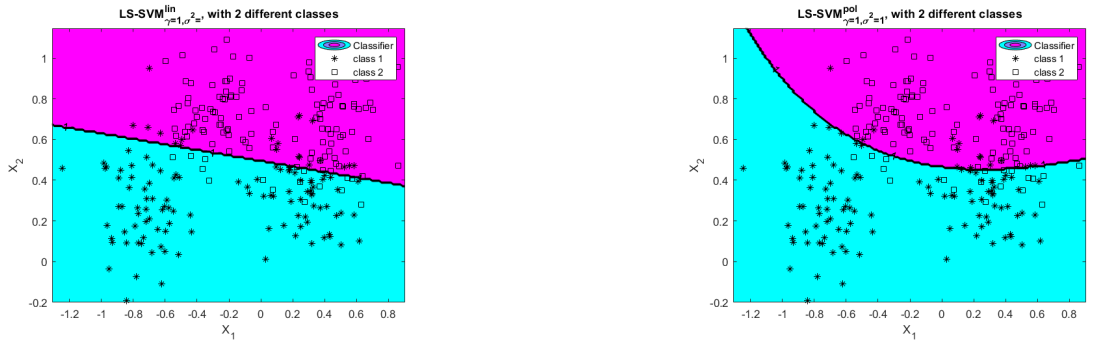


(b) data set

Figure 14: Graphical representation Ripley dataset.

polynomial kernel are looked into. The linear kernel where $\gamma = 1$ is shown in figure 15a and the best found polynomial kernel (2^{nd} degree) while keeping $t = 1$ and $\gamma = 1$ is depicted in figure 15b. The loss of the former is 0.108 and the latter has a loss of 0.094. That means that the 2^{nd} degree

polynomial outperforms the linear kernel and classifies 9.4 points wrong on 100, on average.

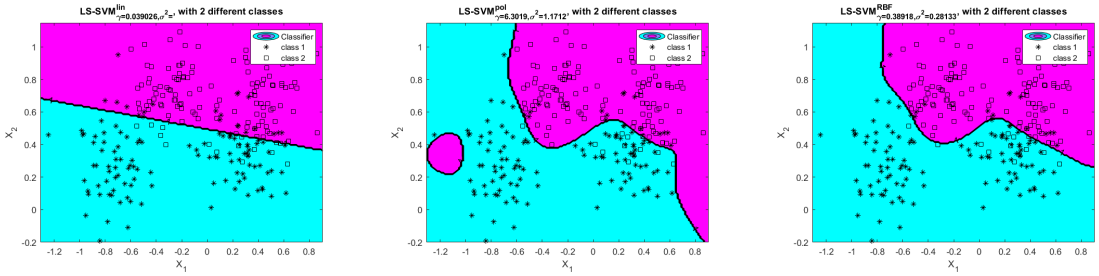


(a) Linear kernel

(b) Polynomial kernel of second degree

Figure 15: Manually found kernels.

To find a good setting of parameters for the RBF kernel, as well as the polynomial kernel, it is good practice to use the automatic parameter tuning algorithms provided in the LSSVM toolbox as both kernels have multiple parameters to tune. In addition, the Nelder-Mead method was also used to determine a good γ for the linear kernel. Figures 16a, 16b, and 16c show the graphical results when wielding the best parameters thrown by the Nelder-Mead method (3 runs). For the linear kernel, the loss on the test set is 0.106. The polynomial kernel has a loss of 0.094 and the loss of the RBF kernel is 0.0910. The self found polynomial kernel performs better than the one



(a) Linear kernel.

(b) Polynomial kernel of 7 degrees.

(c) RBF kernel.

Figure 16: Applied results of the Nelder-Mead method for different kernels on the Ripley data set.

outputted by the Nelder-Mead method. the AUC of the ROC of this second order polynomial kernel is 0.96538, which is better than the linear kernel outputted by the Nelder-Mead method (AUC=0.95872) and slightly worse than the RBF kernel outputted by the Nelder-Mead method (shown in figure 17).

Conclusion

With an error rate of about 10% on the test set, the model is still not very accurate. It might be a good option to look for more features in order to distinguish the classes better or to find a better method (such as other kernels). If a choice between a linear kernel, polynomial kernel and RBF kernel would have to be made, linear could be a better choice since it is computationally cheaper and the error rates are quite similar. If every percent of reduction on the error rate is imperative, the RBF kernel would be my suggestion.

2.2 Breast Cancer Wisconsin (Diagnostic)

This data set originally contained 32 columns, of which the first was the ID and was omitted. The second was an indication of having cancer or not (here -1 means benign, 1 means malignant). Finding information about the 30 variables at hand is not straightforward and are all assumed to be important. The training data consists of 250 benign tuples, and 150 malignant. The test set is smaller (107 benign and 62 malignant). As this is a medical application, not detecting cancer (FN) is worse than wrongly detecting cancer (FP). So the amount of FN's of the test set will be the one and only measure of performance.

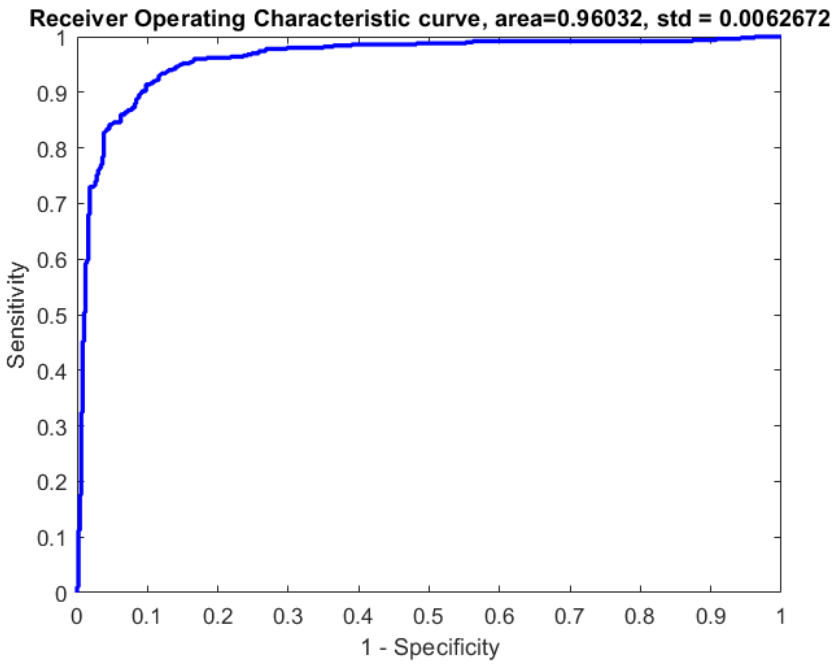
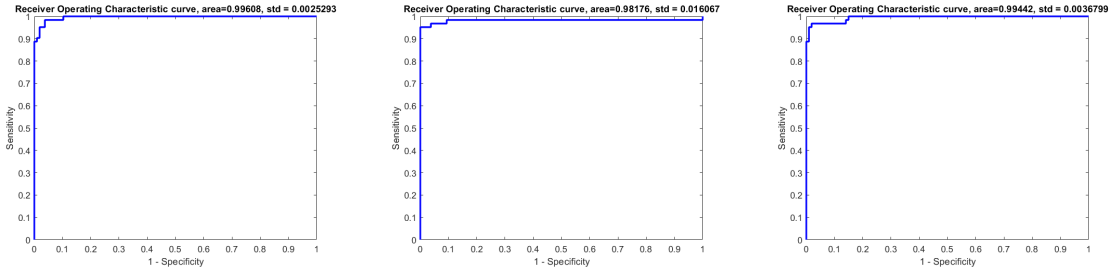


Figure 17: ROC of the best RBF kernel.

Using the Nelder-Mead algorithm a linear kernel was found that only had 1 FP, and 6 FN, resulting in an error rate of 4.14%. The polynomial kernel of third degree, outputted by the algorithm gave 0 FP and 6 FN (3.55% error rate). Similarly, results of the RBF kernel showed 1 FP and 3 FN (2.37% error rate).

The ROC curves can give us good insights on the model. Because our go-to is minimizing false negatives, we must keep the TPR as high as possible. This corresponds with the Sensitivity axis in the ROC curve. On figure 18a the linear kernel is depicted, 18b the polynomial, and 18c the RBF kernel.



(a) Linear kernel. (b) Polynomial kernel of 3 degrees. (c) RBF kernel.

Figure 18: Applied results of the Nelder-Mead method for different kernels, on the Breast Cancer data set.

Conclusion

On the ROC curves of the three best kernels within their category, the AUC is not the most important measure. We graphically see that, for the polynomial kernel, the sensitivity never reaches 100%. Comparing this kernel to the linear and RBF kernel where 100% *is* reached, we can discard this option. Looking closer to the RBF kernel, we see that the split point (threshold) of having less than 100% sensitivity is at a 14.95% FPR (=1-specificity). For the linear kernel it is only at 9.839% FPR. Therefor the linear kernel is for sure preferred in this application.

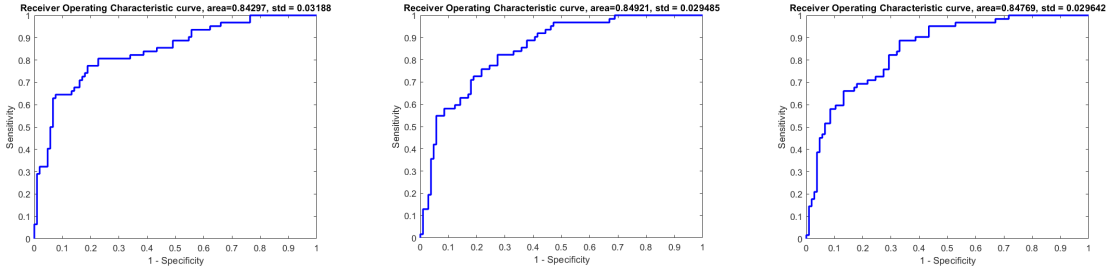
It could be interesting to have extra features in the LSSVM toolbox with regard to misclassification, so that not only the error rate of all classes can be used as a metric to run the automatic parameter tuning alogirthms upon, but also misclassifications of specific classes (in this topic the malignant class).

2.3 UCI Diabetes

The diabetes data set is again a multivariate (8 features), two-classed data set. The training set consists of 300 tuples whereof 95 belong to the positive class and 205 the negative. For the test set the numbers are 62 and 106, respectively.

Information of this data set is not available on the link that one can find on the exercise sheet. Instead, I found another site[3], where the information did not accord with the data sets. I again assumed that all features are important and, just like with all other data sets, I checked the preprocess option.

Because this data set contains medical data, FN's are more important than FP's. Similarly as with the breast cancer data set, three ROC curves for the best linear, polynomial, and RBF kernel are shown in figures 19a, 19b, and 19c, respectively.



(a) Linear kernel. (b) Polynomial kernel of 3^{rd} order. (c) RBF kernel.

Figure 19: Applied results of the Nelder-Mead method for different kernels, on the UCI obesitas data set.

Conclusion

Because the ROC curves are not as nicely looking as with the breast cancer case, suggesting one of the three kernels is not so easy. When going for no false negatives, the polynomial kernel is the best option as its vertex is at 68.87% for the 1-specificity, which is the lowest of the three kernels. In any other case, the end-user of the model needs to make a trade-off of between TPR and FPR, and overlay the three figures in order to decide which kernel she/he would use. Better models could (but not necessarely *can*) be obtained by having more data (tuples/features) or using a different approach to the problem.

References

[1] Emile Aarts and Jan Korst. Simulated annealing and boltzmann machines. 1988.

[2] JA Nelder and RA Mead. Simplex method for function minimization, the computer journal, 7. 1965.

[3] diabetes data set. http://archive.ics.uci.edu/ml/datasets/diabetes. Accessed: 2019-04-19.

SVM : assignment 2

Florentijn Degroote

May 2019

Contents

1 Exercises

1.1 Support vector machine for function estimation

1.1.1 Focus on linear kernel

To understand what effect the parameters *bound* and e have, we first take a look at the math behind the general optimisation problem for regression in SVM's:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}w^T w + bound \sum_{k=1}^{N}(\xi_k + \xi_k^*) \\ \text{s.t.} \quad & y_k - w^T\phi(x_k) - b \leq e + \xi_k \\ & w^T\phi(x_k) + b - y_k \leq e + \xi_k^* \\ & \xi_k, \xi_k^* \geq 0 \end{aligned} \tag{1}$$

Which can be rewritten in dual form, where $K(x_k, x_l) = \phi(x_k)^T\phi(x_l)$:

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

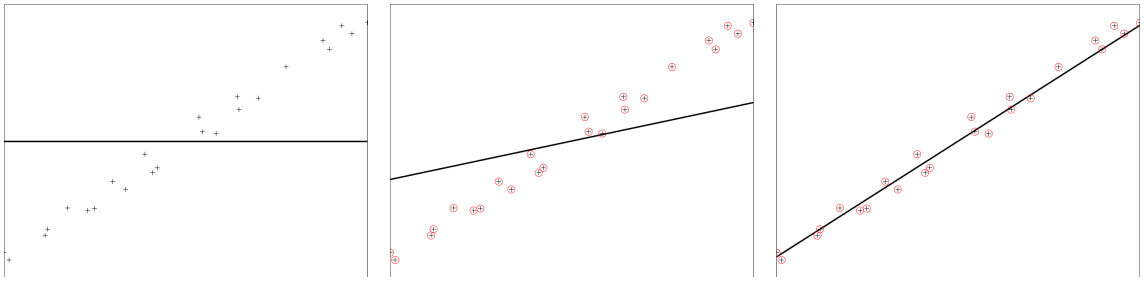
The *bound* parameter acts as a regularizing parameter; when chosen zero, only the norm $||w||$ will be minimized. When *bound* is chosen higher and higher, more importance will go out to points that are not within the *bounds* of the function approximation. When the bound parameter is infinity, only the the error will be taken into account, meaning the problem is a pure **classical least squares fit**; regression SVM's are thus an extension of the classical least squares fit. This parameter's value is thus a trade-off. The actual bounds wherein a point is considered erroneous are determined by the parameter e. When e is chosen zero, every data point will have effect on the result of the optimization problem, as no point exists between the (non-existing) bounds of the function approximation. As e grows larger, points close to the function approximation will not have any effect on the optimization function. The absence of effect of some points on the outcome of the regression SVM can be described to the **sparsity property**.

For regression using a linear kernel, the data set that is shown in figure 1a is used. On this figure, the black line represents the function estimation for a value of 0 for both e and *bound*. It is completely flat because the norm of w was the only parameters that was affecting the optimization problem. In figure 1b, the bound parameter was altered to 0.001, resulting in a slighlty better, yet underfitted function estimation as the SVM is not able to incorporate the data's structure well due to the model being restrained too much. in figure 1c, this seems to be better. Now that a preliminary value of *bound* is determined, the value of e can be altered! In figure 2a, the bounds are visible sparsity effect is already noticeable (red circles around data points denote support vectors). After some tweaking, a good result was found for 0.1 as value for *bound* and 0.004 for e. Only 4 out of the original 24 points are now support vectors, shown in figure 2b. Making the *bound* parameter bigger does effect on the regression line, whereas increasing e leads to the undesirable function estimation shown in figure 2c, where no points lie in between the bounds.

1.1.2 Kernel analysis on a more complex data set

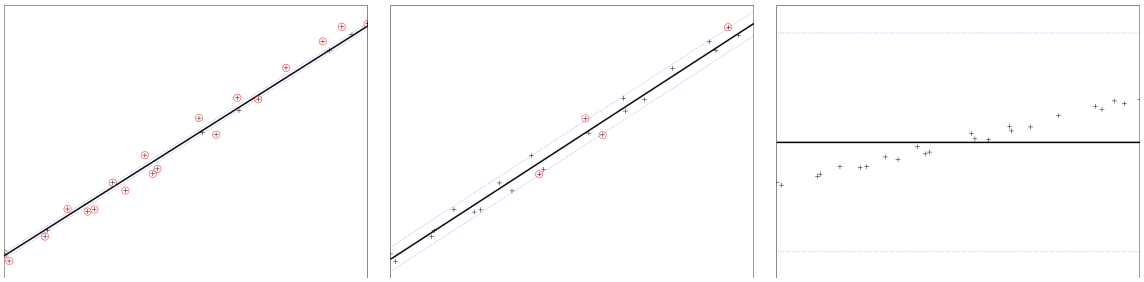
In a next analysis, a more complex data set was erected with the goal of finding the best kernel to represent the best function approximation. The best possible fit of a linear kernel, quadratic kernel, cubic kernel, quartic kernel, exponential RBF kernel and gaussian RBF kernel are shown in figures 3a, 3b, 3c, 3d, and 3e, 3f respectively. For all kernels, *bound* was set to 10, and e was 0.01.

The dataset seems to be best depicted by a regression SVM with a cubic, quartic or RBF kernel. Also quintic kernels and higher level polynomials do well, but are more complex. Occam's razor states that one should always reduce the complexity of the model as much as possible while



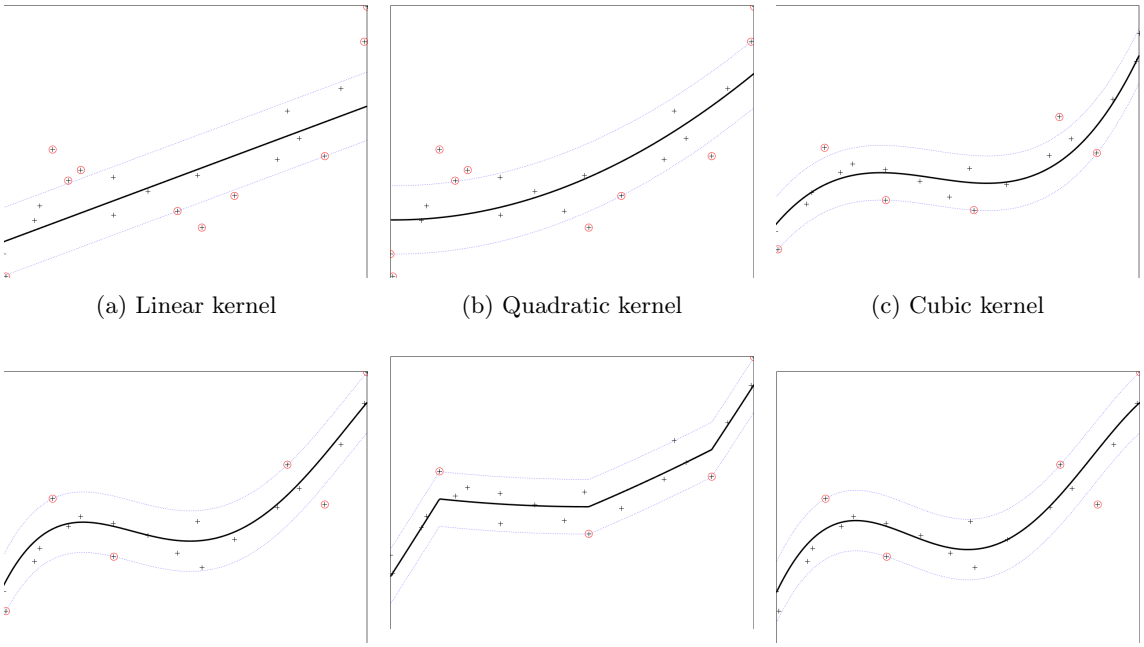
(a) *bound*=0,*e*=0 (b) *bound*=0.001,*e*=0 (c) *bound*=0.01,*e*=0

Figure 1: Effect of changing *bound* parameter.



(a) *bound*=0.01,*e*=0.001 (b) *bound*=0.1,*e*=0.004 (c) *bound*=0.1,*e*=0.1

Figure 2: Taking into account *e*.



(a) Linear kernel (b) Quadratic kernel (c) Cubic kernel

(d) Quartic kernel (e) Exponential RBF kernel $\sigma = 0.25$ (f) Gaussian RBF kernel $\sigma = 0.14$

Figure 3: Regression of non-linear data set with different kernels.

being able reproduce the general scheme of the data. Applied to this data set, the cubic kernel is preferred over the quartic. Both the gaussian and exponential kernel have 4 support vectors, whereas the cubic and quartic have 6. Because the function estimation of the Gaussian kernel seems to be smoother and because it has less support vectors than the cubic kernel, I would think that this kernel is best suited for this data set.

1.2 A simple example: the sinc function

1.2.1 Regression of the sinc function

Using LS-SVM's with RBF kernels, two parameters need to be decided, namely γ and σ^2. The former acts as a regularizing parameter, just like *bound* in the previous subsection. Having this value very low results in the SVM not being able to generalise the underlying data distribution well. On the other hand, picking this parameter's value too high, the kernel would try to minimise the fitness error as much as possible. Knowing that in the training data is noisy, the approximation could very well result in overfitting. The latter represents the width of the zone of influence of each data point in the training set. This parameter could interpreted as how big the area is to which a certain datapoint has influence on the result of the function estimation. Having this parameter too low will result in data points having a big say in how the function estimation looks like within their close proximity, resulting in overfitting. Having it too low will lead to the data points' influence being very wide so that they interfere with eachother. This could result in bad function approximations.

Let's now conduct a graphical analysis of function estimations with a RBF kernel on a noisy sinc function. Figure 4a and 4d show the resulting function approximation and training set for an arbitrarily chosen $\sigma^2 = 1$ and $\gamma = 0.01$. Increasing the value of γ allows the kernel to better estimate the function, shown in figures 4c and 4e. For $\gamma = 10000$, the function estimation seems satisfactory (see figures 4b and 4f). If we then decrease the σ^2 parameter, the function estimation gets a lot worse as we also conferred in the previous paragraph (figures 5a and 5d). Increasing σ^2 leads to an overfitted solution (figures 5b and 5e). It must be noted that the situation in figures 4b and 4f not the only configuration that works out. In figures 5c and 5f, for $\gamma = 100$ and $\sigma^2 = 0.4$, the function estimation is also solid, and is actually preferred over the first, as the gamma parameter is lower (possibly averting overfitting and making the kernel more robust). Keeping in mind the parameter space and the trade-off that one makes when choosing the two parameters makes me believe that there is no optimal pair of hyperparameters. Instead, lots of pairs do work out well, while others are for sure not suitable.

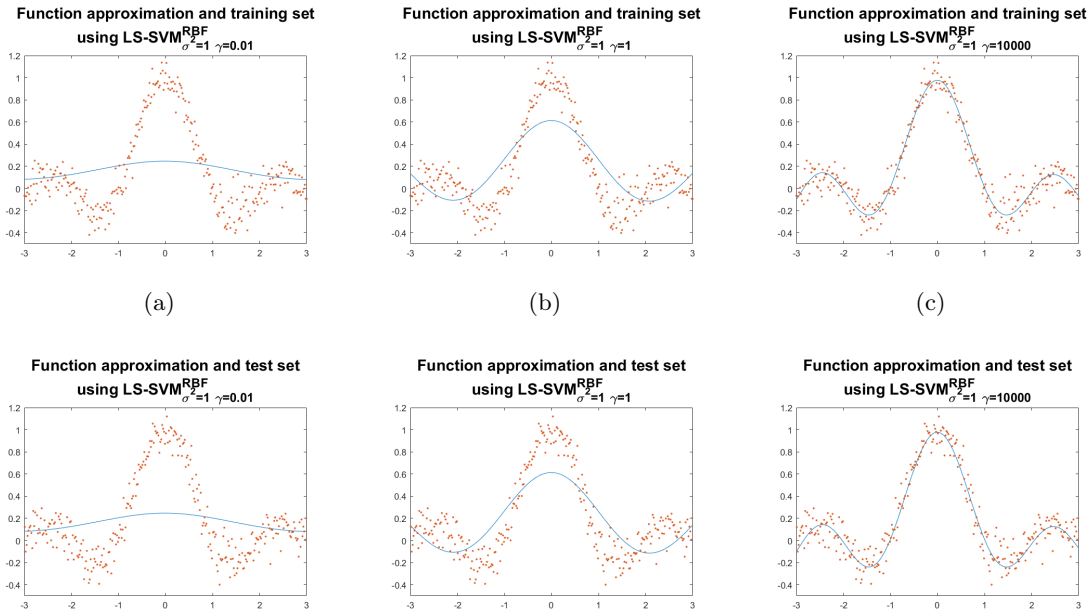


Figure 4: Effect of parameter γ.

The RMSE of the conducted experiments are shown in table 1 and 2.

Instead of tuning the parameters manually, we can also make use of the built-in automatic hyper-parameter tuning algorithms. Two mayor algorithms will be looked into, namely the grid search algorithm (which scans a grid of parameters and picks the best configuration, based on a measure) and the Nelder-Mead method (gradient-based optimization, which is faster than brute force grid search). In both cases, 10 fold cross-validation is applied.

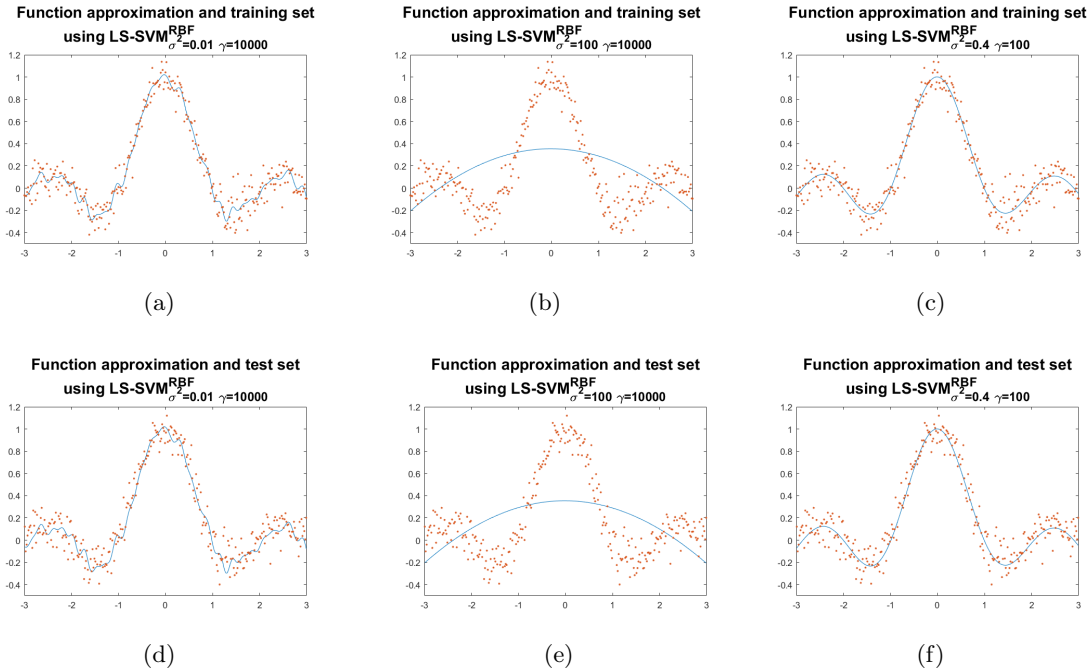


Figure 5: Further tweaking.

γ	0.01	0.1	1	10	100	1000	10000
RMSE	0.3438	0.291	0.2258	0.1779	0.1218	0.1124	0.1061

Table 1: RMSE for different values of γ when $\sigma^2 = 1$.

σ^2	0.01	0.1	1	10	100
RMSE	0.1111	0.1071	0.1061	0.2557	0.3374

Table 2: RMSE for different values of σ^2 when $\gamma = 10000$.

Results (in terms of the mean μ and variance σ^2) of 30 runs for computation times and RMSE's of the solution of both techniques are shown in table 3. The Nelder-Mead method is significantly

	μ_t	σ_t^2	μ_{RMSE}	σ_{RMSE}^2	μ_γ	σ_γ^2	μ_{σ^2}	$\sigma_{\sigma^2}^2$
Nelder-Mead method	0.7318	0.0026	0.0990	7.96e-05	131.1567	2.18e+07	0.4407	0.0110
Gridsearch	1.1678	0.0011	0.0992	1.10 e-4	352.4359	2.62e+05	0.4530	0.0053

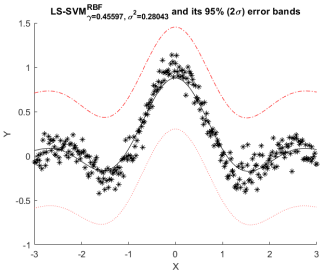
Table 3: Statistics of 30 runs for the two optimization procedures.

faster and RMSE values are also slightly better. For both methods, there is quite some variation in the solution of γ ($> 10^5$) and less so for σ^2.

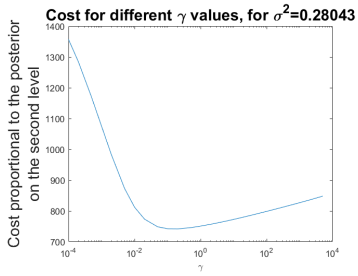
1.2.2 Application of the Bayesian framework

In the bayesian framework, parameters of the model can be evaluated on a measure called the likelihood, representing how likely it is that a given set of parameters make up for an SVM that is able to classify the data well (in case of classification) or represent the data well (in form of a function estimation, in regression). In total, three different interwoven levels make it possible to infer information with respect to specific parameter(s). The first level handles the parameter w and b, that are results from the SVM optimisation itself. The second level handles the regularization parameter γ and the third (optional) level infers information about any kernel parameter, in our case σ^2, as we are working with RBF kernels. Next to calculating the cost (negative logarithm of the posterior), the LS-SVM toolbox also permits the user to let a function decide the best model parameters (γ and σ^2) with the function *bay_ optimize()*. The framework needs to be initialized

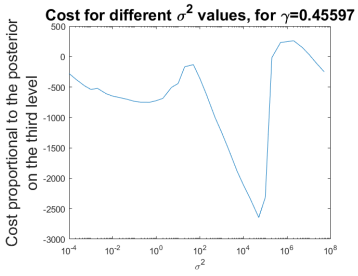
with an initial "guess" of the parameters, aka prior, and in our case $\gamma = 10$ and $\sigma^2 = 0.4$ were wielded. The resulting regression is and 95 % error bars are shown in figure 6a. Figures 6b and 6c show the cost function for different parameter values according with level 2 and 3, respectively, each time calculated with the result of the *bay_ optimize()* function. As one can see, the calculated



(a)



(b)



(c)

value for γ is the actual minimum in the log-likelihood related cost function. For σ^2, 0.28043 is a local minimum rather than a global optimum.

1.3 Automatic Relevance Determination

In ARD, the bayesian framwork and its three levels of inference to calculate the posterior can be used to determine the most fitting features of a given data set that accord most with the (in this case regression) SVM, defined by the parameters γ and σ^2. The ARD first starts by considering all input dimensions important, and prunes them one by one. An input dimension is pruned if there is a σ^2 of the subset without the considered input dimension which involves a higher likelihood than the best likelihood for the total set. For the provide code, ARD detects that only the first input is relevant. Figure 7 shows that indeed, this is the case as input dimension two and three are random noise.

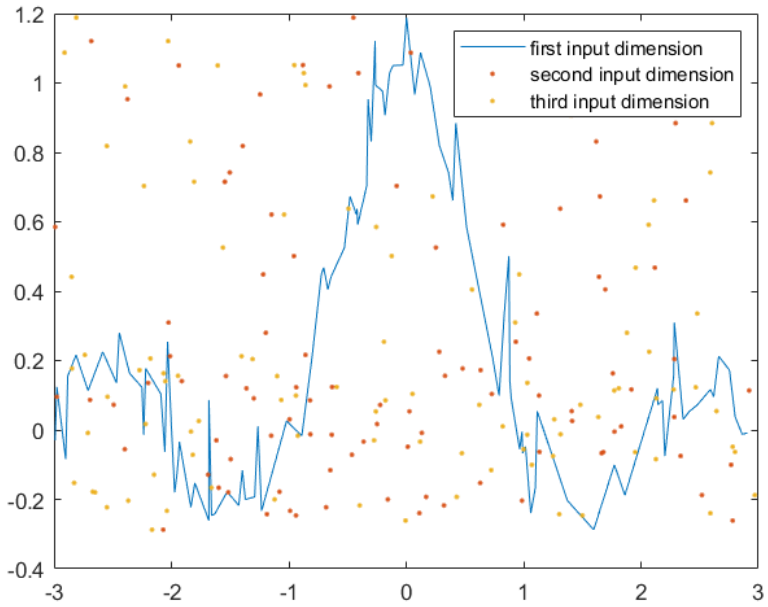


Figure 7

Instead of the cost function wielded by the ARD, which is based on the log likelihood of the third level of the bayesian framework, other measures can be taken into consideration to determine whether an input dimension is relevant or not. One of those measures is the mean-squared-error (MSE) of the data set on the function estimation with the given parameters γ and σ^2. In order to get a truthful estimation of the SME, cross-validation can be used. The reported statistic is then

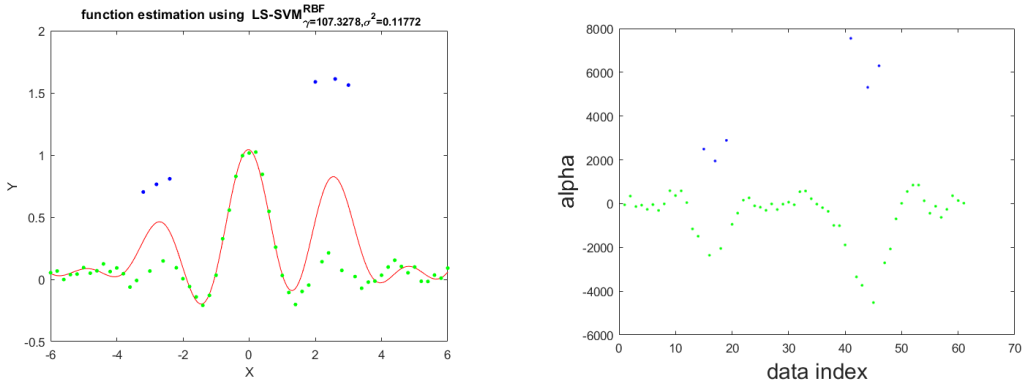
the mean MSE. These are listed in table 4 for the 7 different configurations. These results lead to the same conclusion as the ARD.

Configuration	1	2	3	{1,2}	{1,3}	{2,3}	{1,2,3}
Mean MSE	0.0270	0.1744	0.1676	0.0541	0.0650	0.1836	0.0962

Table 4: Results of cross validation for the different configurations of the input dimensions.

1.4 Robust regression

For regression with outliers, extra measures were taken and added to the LS-SVM toolbox to detect outliers and not let the function estimation be influenced by these data points. In figure 8a, the function estimation is depicted for parameters obtained by the *tunelssvm()* function for the data points shown in green and blue (of which the blue points denote the outliers). The function estimation in the region of these outliers seems to not follow the sinc function, due to the function trying to model the outliers as well. Figure 8b shows the alpha values (weights) of each data point. As we can see, the weight of the outliers and the points close to the outliers in x-direction have a big (opposite) influence. SVM in its classical form is thus very sensible to outliers. In order to



(a) (b)

deal with outliers in the data, statistical methods have been developed such as the Huber loss or Hampel loss, that involve using a weighted cost function. These cost function effectively penalizes the outliers' weight on the function estimation. Also logistic losses or myriad weighted myriad filters can be used. For the formulas, I'd like to refer to figure 9 [1].

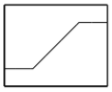
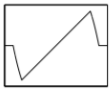
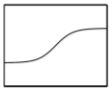
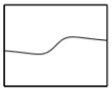
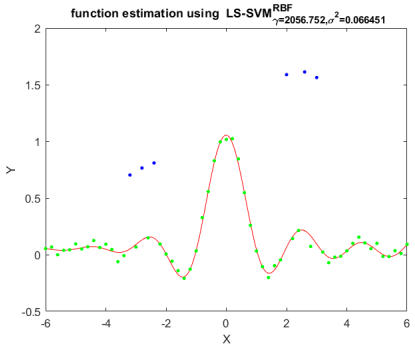
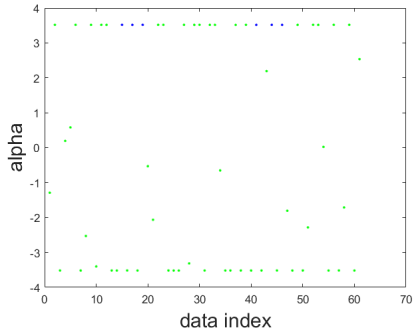
	Huber	Hampel	Logistic	Myriad
$V(r)$	$\begin{cases} 1, & \text{if } \lvert r\rvert < \beta; \\ \frac{\beta}{\lvert r\rvert}, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} 1, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2-\lvert r\rvert}{b_2-b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$\frac{\tanh(r)}{r}$	$\frac{\delta^2}{\delta^2+r^2}$
$\psi(r)$				
$L(r)$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < \beta; \\ \beta\lvert r\rvert - \frac{1}{2}\beta^2, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2 r^2 - \lvert r^3\rvert}{b_2-b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$r\tanh(r)$	$\log(\delta^2+r^2)$

Figure 9: Weight functions $V(r)$, score functions $\psi(r)$ and loss functions $L(r)$ for the Huber, Hampel, Logistic and Myriad approaches [1].

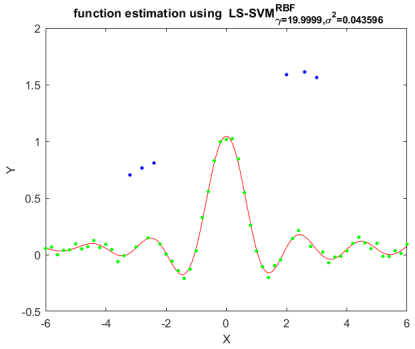
Figure 10 encapsulates the results for the four different techniques. It seems that for logistic and Huber, the weights have been bounded, so that the influence of the data points is minimized. Though, the best results are obtained for Hampel and Myriad, where the function estimation seems to be minimally affected by the outliers, and where the weights of the points are not among the biggest weights anymore.



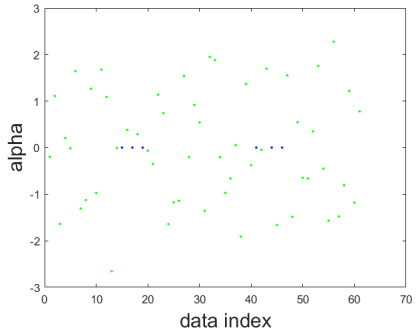
(a) Function estimation for Huber



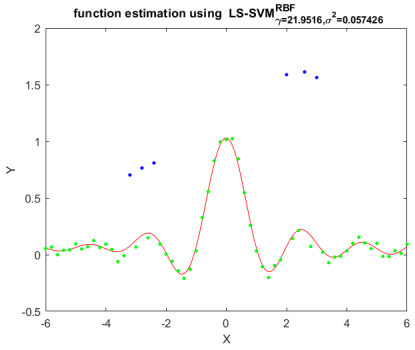
(b) alpha values for Huber



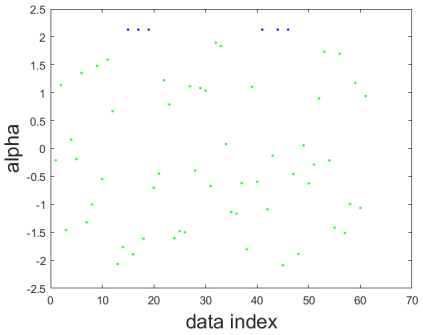
(c) Function estimation for Hampel



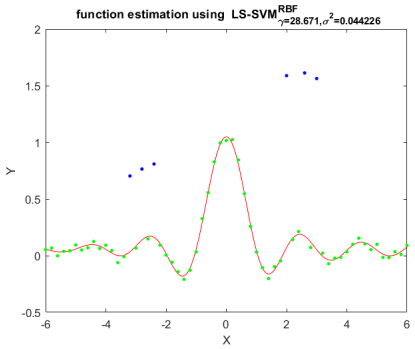
(d) alpha values for Hampel



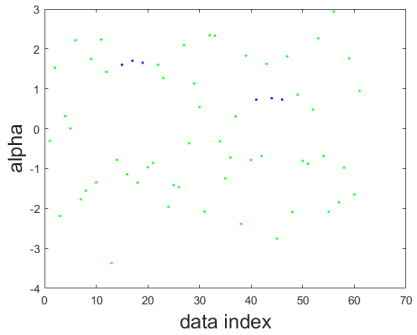
(e) Function estimation for logistic



(f) alpha values for logistic



(g) Function estimation for Myriad



(h) alpha values for Myriad

Figure 10: Robust regression techniques

Because of the square in mean squared error, large errors have relatively greater influence on MSE than smaller error do. Since MAE does not involve squaring the error, the measure is more

robust to outliers and is preferred in this setting.

2 Homework problems

2.1 Logmap dataset

For the logmap dataset, SVM regression is used in combination with the linear autoregressive model of the training and test data. The training set consists of 150 points, and the test set of 50 points. The example code of the assignment sheet generates a inferior model, not predicting the right tendencies of the data, as shown in figure 11.

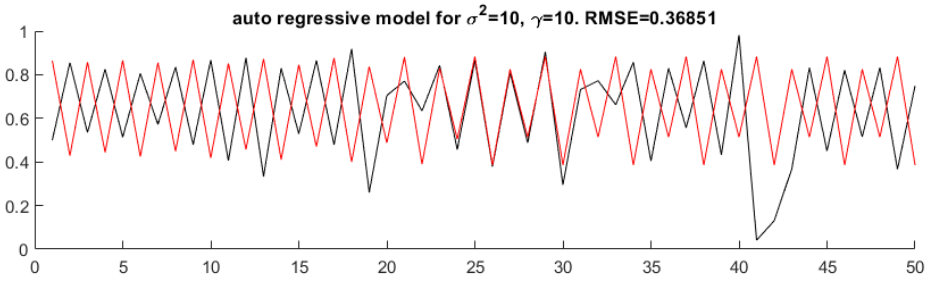
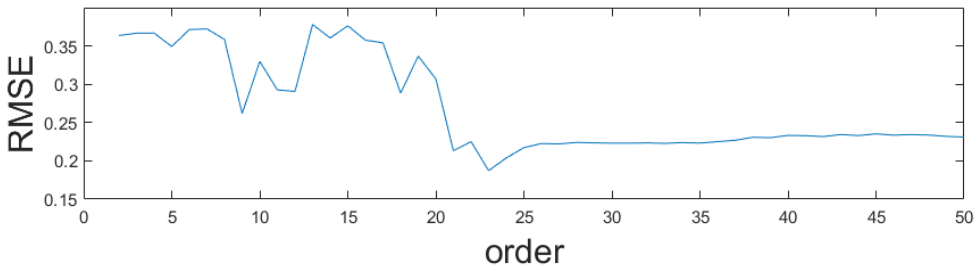
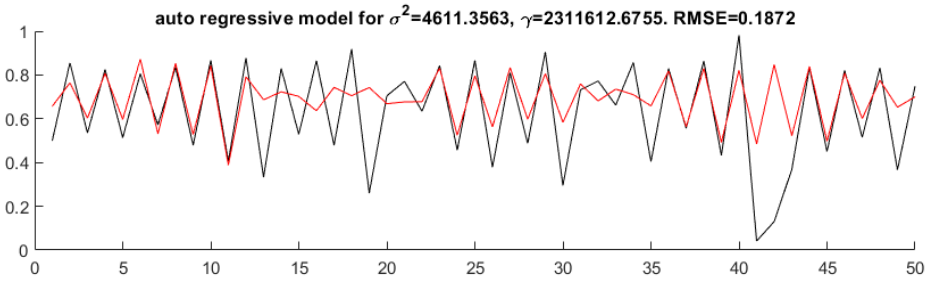


Figure 11

In order to correctly validate the model on overfitting, a validation set should be erected on which extra analysis should be conducted (e.g. leave-one-out, k-fold cross validation). In this exercise, 10-fold cross validation was used to determine the parameters σ^2 and γ. This cross-validation splits the training set into a training and test set. Model orders of size 2 up until 50 are looked into. For each of these model orders, the *tunelssvm()* was wielded 10 times, and the best configuration of the parameters was being kept, on basis of the RMSE measure (note that RMSE was measured on the test set). In figure 12a, the best RMSE values for the different model orders are registered. The best one is registered for a model order of 23. Results of this particular configuration are shown in figure 12b. The results are far from perfect, still, but the prediction



(a)



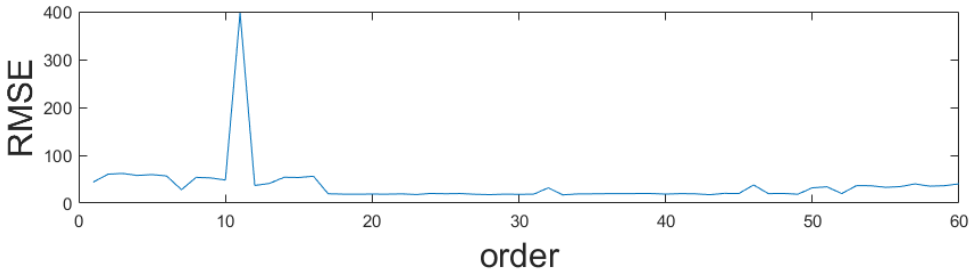
(b)

seems to be able to detect the correct frequency and in certain intervals, the right amplitude as well. Other measures could be used to choose the best model upon, in order to get different predictions (for example a weighted RMSE that puts a lot of weight on the first data points in comparison

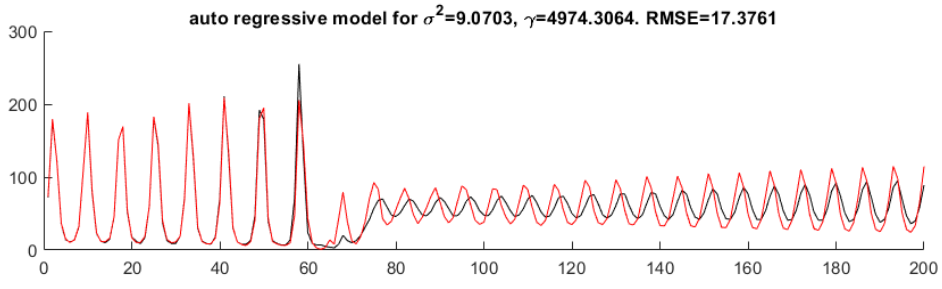
with the last could lead to a model that is able to fit the first 20 data points better than in this case). One could note that the values for σ^2 and γ are quite large, especially compared to the results of a two dimensional problem, like in section 1. An important aspect of the explanation lies in the fact that the input is now 23-dimensional. The 150 data points are scattered in this 23-dimensional space and are very distant from each other. This permits the width of the RBF kernel (σ^2) to be large. The big value of the regularization parameter indicates that the error of the data points is imperative for a good function estimation.

2.2 Santa Fe dataset

For the santa Fe dataset, the first 1000 laser measurements act as training set and the 200 consecutive measurements as test set. When visualising the data, I would think that 50 is not a bad choice for the model order, because the general tendency of the laser (frequency as well as collapses) can be depicted within a window of 50 data points. The question is however, if there are lower orders that perform as good as, or even better, than 50. It is sensible to use the performance of the recurrent prediction on a validation set to optimize hyperparameters and the model order, when the validation set is not the test set. It is possible that by tweaking the parameters more than necessary, so that the results on the validation set are outstanding, the results or the test set are disappointing. In that case, we have overfitted our solution to the validation set and are able to detect this with the test set. The validation is then, of course, not part of the training set (leave-one-out/k-fold cross-validation). The test set is thus used as a final judgment on the model. For this homework problem, the test set will, just like in the previous problem, be used to determine the best model order. As the test set is thus also used to tune a parameter with, the borderline between the test set not being a kind of validation set is vague. Analogous to the previous exercise, *tunelssvm()* is ran 10 times for each model order ranging from 2 to 60. The best RMSE values for each order are shown in figure 13a. The best prediction is shown in figure 13b.



(a)



(b)

References

[1] Kris De Brabanter, Kristiaan Pelckmans, Jos De Brabanter, Michiel Debruyne, Johan AK Suykens, Mia Hubert, and Bart De Moor. Robustness of kernel based regression: a comparison of iterative weighting schemes. In *International Conference on Artificial Neural Networks*, pages 100–110. Springer, 2009.

SVM : assignment 3

Florentijn Degroote

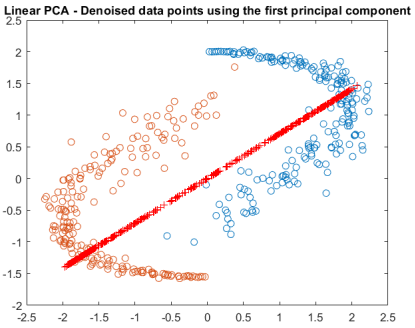
May 2019

Contents

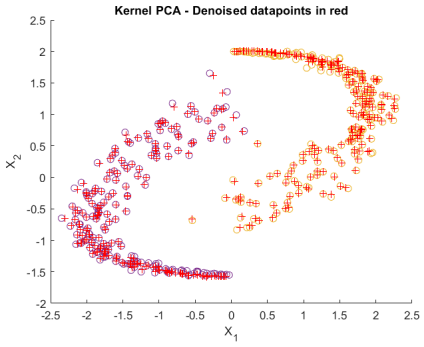
1 Exercises

1.1 Kernel principal component analysis

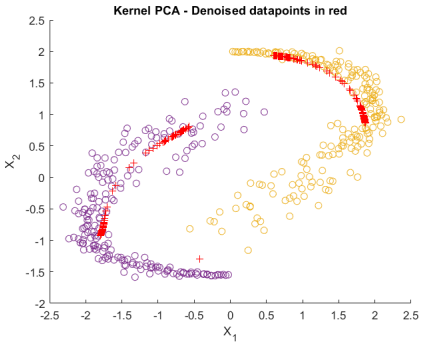
When denoising a data-set, such as the given yin and yang data set, the main goal is to generalise so that the underlying distribution is represented without the noise. This can be done by a kernel component analysis, especially, the non-linear variant. When reconstructing the data set, based on this non-linear kernel, the noise should be evaporated. Just like with auto-encoders, a dimension-changing technique is used. In the case of non-linear kernel PCA, it is possible to scale up the dimension, as opposed to traditional linear PCA which acts as a dimension reduction technique. PCA finds new directions based on the covariance matrix of original variables. It can extract a maximum of eigenvalues equal to the amount of features. KPCA finds new directions based on kernel matrix. KPCA's big advantage is that it can handle non-linear data sets, in contrary to traditional linear PCA (see figure 1a). It can extract n (number of observations) eigenvalues. Increasing the number of principal components will retain more and more information of the input space. When the number of principal components is chosen too large, the noise will also be assimilated in the model, as shown by figure 1b. On the other hand, a very low number of principal components will lead to bad specification/reconstruction of the data, depicted on figure 1c. The number of principal components is thus a trade-off. 5 seems to reasonably reconstruct the underlying data distribution, while effectively having reduced the noise (figure 1d). The measure to tune parameter upon could be based on the error (e.g. RMSE) between the original denoised data set and the data outputted by the KPCA. A part of the training set could herein be erected as validation set and methods like leave-one-out and cross-validation should be a good choice to alter parameters hyper and kernel parameters upon, for a number of principal components. As extra note: results of the KPCA's denoted in figure 1 are obtained with the default values of the script, and with the RBF kernel.



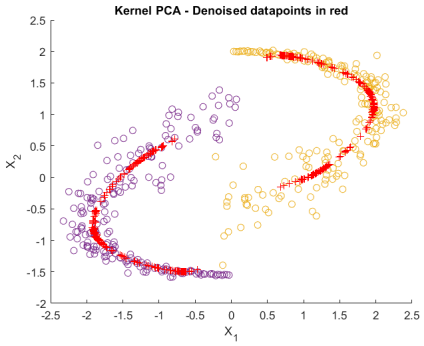
(a) Standard (linear) PCA, 1 principal component



(b) KPCA, 25 principal components



(c) KPCA, 2 principal components



(d) KPCA, 5 principal components

Figure 1: PCA and KPCA on the yin-yang data set

1.2 Spectral clustering

Spectral clustering can be interpreted as the weighted version of KPCA. The weights are equal to the inverse of the degree matrix of the graph. Solving the optimization problem is harder because finding the optimal centering is not as trivial anymore. In this part of the exercise, we will try to differentiate two clusters from one another.

To go more in depth; executing spectral clustering involves computing the RBF kernel (affinity) matrix which contains the correlations between the data-points. These correlations are in the kernel matrix in their complete dimensions (n by n, where n is the number of datapoints). The thing about spectral clustering is that it generates a (in this case) single-dimensional vector $sign[\alpha_i]$, with length equal to the size of the input data, based on the correlations, in such a way that the vector assigns a class to each data point (assign the data to their cluster). The single-dimensional vector is for the binary case. In the end, the optimization problem solved to obtain this vector minimizes the inter-cluster similarities. This is all done in an unsupervised way, so no prior information on the classes is necessary. Spectral clustering is tied with k-means clustering, as it could isolate non-linear clusters in the feature space, which is trivial for k-means clustering to cluster.

Spectral clustering for multiple clusters is also possible, and involves having more constraints on the optimization problem. In essence, the method is comparable to a one-vs-all classification for all number of clusters.

On the other hand, classification in SVM's involves a boundary in the same feature space is the data (of dimension a-1, where a is the number of dimensions of the input data). The training of such a classification SVM model is done in a supervised way. That is, training data is labeled and predefined to a certain class. Splitting the data into training,validation, and test set is crucial.

σ^2 is the RBF kernel parameters that determines the width of the kernel. σ^2 has an influence on the kernel matrix in general kernel PCA and spectral clustering because the kernel matrix/similarity matrix Ω_c can be written as;

$$\Omega_c = \begin{bmatrix} (\varphi(x_1) - \hat{\mu}_\varphi)^T(\varphi(x_1) - \hat{\mu}_\varphi) & \dots & (\varphi(x_1) - \hat{\mu}_\varphi)^T(\varphi(x_N) - \hat{\mu}_\varphi) \\ \vdots & \ddots & \\ (\varphi(x_N) - \hat{\mu}_\varphi)^T(\varphi(x_1) - \hat{\mu}_\varphi) & \dots & (\varphi(x_N) - \hat{\mu}_\varphi)^T(\varphi(x_N) - \hat{\mu}_\varphi) \end{bmatrix} \tag{1}$$

and each element of the kernel matrix can be described by;

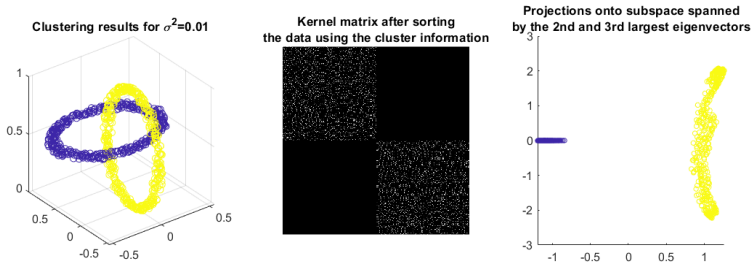
$$\Omega_{c,kl} = (\varphi(x_k) - \hat{\mu}_\varphi)^T(\varphi(x_l) - \hat{\mu}_\varphi), \quad k, l = 1, ..., N \tag{2}$$

Each score variable is then equal to;

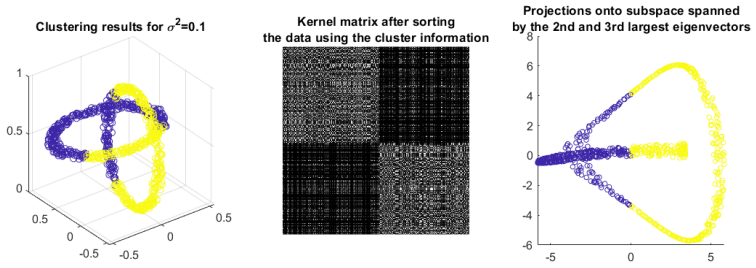
$$\begin{aligned} z(x) &= w^T(\varphi(x) - \hat{\mu}_\varphi) && (3) \\ &= \sum_{l=1}^{N} \alpha_l (\varphi(x_l) - \hat{\mu}_\varphi)^T(\varphi(x) - \hat{\mu}_\varphi) && (4) \\ &= \sum_{l=1}^{N} \alpha_l \left(K(x_l, x) - 1/N \sum_{r=1}^{N} K(x_r, x) - 1/N \sum_{r=1}^{N} K(x_r, x_l) + 1/N^2 \sum_{r=1}^{N} \sum_{s=1}^{N} K(x_r, x_s) \right) && (5) \end{aligned}$$

Knowing that $K(x, x_k) = e^{\frac{-||x - x_k||_2^2}{\sigma^2}}$ and keeping in mind equation 5 we can deduce that a very small value of σ^2 will impose that the scores and the elements of the centered matrix Ω_c will be close to zero. On the other hand, when σ^2 is very large, the whole matrix will contain 1's. In the best case, there should be no dissimilarity between data-points of a same cluster (values close to 1) and the the two non-diagonal blocks of the sorted kernel matrix should contain close to zero elements, indicating that the between cluster similarity is small/dissimilarity is big. Figure 2 shows the results for varying σ^2 values. In this case, $\sigma^2 = 0.01$ seems to be doing well and cluster the data well.

(a) $\sigma^2 = 0.000001$



(b) $\sigma^2 = 0.01$



(c) $\sigma^2 = 0.1$

(d) $\sigma^2 = 1000$

Figure 2: Spectral clustering with two clusters

1.3 Fixed-size LS-SVM

In general, problems where the number of data points is significantly higher than the number of dimensions, ought to be solved in the primal representation. The dual representation on the contrary is kernel based, and is nothing else than inner-product of data points. Each data point has then an associated an alpha value. The number of dimensions does not affect the dual problem in any way, so should be preferred to solve a problem which is very high dimensional in comparison with the number of training data (such as micro-array data). An aspect that also needs to be taken into account is the choice of kernel function. When for example the feature map is infinite dimension, you can always solve the problem in the dual. Another way of handling large data sets lies in the use of fixed size LS-SVM's.

Because the eigenvalue decomposition increases sharply with the size of the feature space,

finding a way to reduce the space without loosing to much information would benefit computation times and memory in the primal. The fixed-size LS-SVM is a method reaches these goals by taking a fixed amount (M) of points, which is a subset of the original data set. Picking M points at random could in some cases be enough, but the risk exists that the M data points do not fully represent the distribution of the original data set. In order to solve this problem, the quadratic Renyi entropy can be used as a measure to detect whether a randomly chosen extra point a holds more information than a random chosen point b of the subset M. If that is the case, b is replaced by a. Multiple iterations of this procure lead to a data set M that resembles quite well to the original one. The quadratic Renyi entropy is calculated as in equation 6.

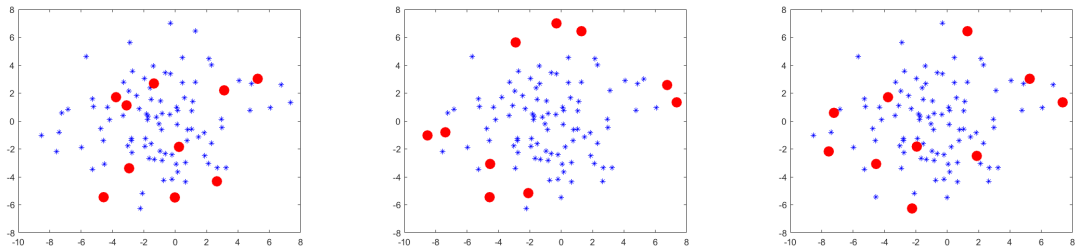
$$H_R = -log\frac{1}{M^2}\sum_{ij}\Omega_{(M,M)_{ij}} \tag{6}$$

where $\Omega_{(M,M)_{ij}}$ is the kernel matrix of the M space. The eigenvalue decomposition of this matrix is shown in equation 7.

$$\Omega_{(M,M)}\bar{U} = \bar{U}\bar{\Lambda} \tag{7}$$

where $\bar{\Lambda}$ contains the eigenvalues corresponding to the eigenvector matrix $\bar{U}$. In essence, working with the subset M involves working with an approximation of the original feature map, obtained through the Nyström method (downsampling of the original NxN to the MxM space). It must be noted that with the Nyström method, we lose the sparsity property.

In figure 4a, the entropy of the converged subset of the data is shown, for different σ^2 values. Low σ^2 values again impose that the similarities between two points (that could be very close to each other) are extremely low (kernel matrix full of zeros). The entropy is therefor quite rapidly maximal and does not give a good estimation of how much entropy the subset actually contains, as shown on figure 3a. High σ^2 values cannot differentiate between points that are close or distant to one another as they are all regarded to as similar (kernel matrix full of ones). Because the subset of the points in this case will lie at the boarders of the original data set, maximal entropy is low (shown on figure 3a). Picking a low value of σ^2 seems to be favorable based on the previously mentioned figure. Even though we reach an optimal entropy, the results are not really good for extremely low values for σ^2. Trough the iterations, the maximum entropy is reached quite rapidly (see figures 4b and 4c), in comparison with higher σ^2 values. The model is too quickly happy with its results and is not very flexible to possibly better points that are in later iterations being proposed. The best σ^2 values are thus the ones that are still maximizing the entropy while being as high as possible, more specifically in the range of $[0.1, 1]$, as exemplified in figure 3c. Using the



(a) Result after 100 iterations for $\sigma^2 = 0.000001$

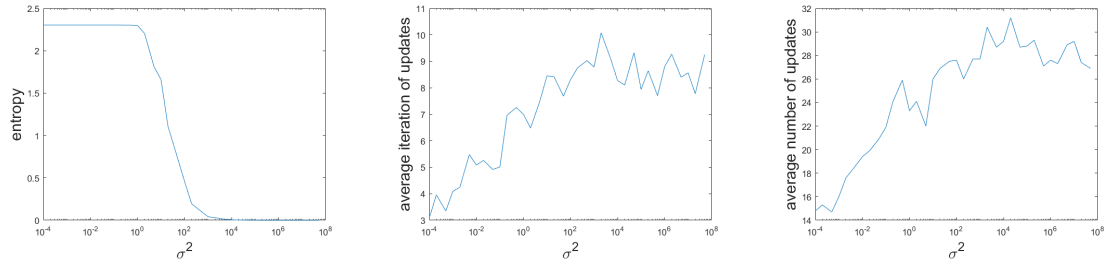
(b) Result after 100 iterations for $\sigma^2 = 100000$

(c) Result after 100 iterations for $\sigma^2 = 0.1$

Figure 3: analysis of fixed size LS-SVM

Nyström approximation method, the mapping of data to the feature space can also be evaluated explicitly. This 'Automatic Feature Extraction by Nystrom method' gives the features that one can use for a linear regression or classification. The decomposition of the mapping to the feature space relies on the eigenvalue decomposition of the kernel matrix. In figures 5a and 5b, the mapping is shown in the first three dimensions of the feature space, calculated for the extracted points/features (that's why the red dots are different from the origin, as opposed to many other points/features which are not extracted). The extracted features are the result of picking 10 random points of the data set in the first image, and the points in the original space of the second space are the ones calculated with the entropy method described above.

As mentioned already, a serious drawback of LS-SVM models is the lack of sparseness, as nearly all patterns become support vectors. Two kinds of solutions have been looked into; 1) pruning after

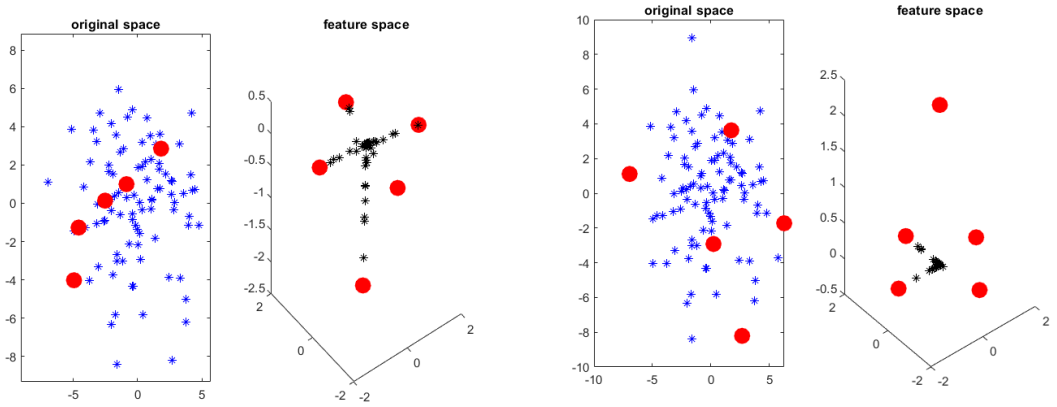


(a) Maximum entropy reached after hundred iterations for different σ^2 values

(b) Average iteration on which updates occurred for 100 total iterations, averaged over 10 runs

(c) Average amount of updates for hundred iterations, averaged over 10 runs

Figure 4: analysis of fixed size LS-SVM



(a) 10 random points

(b) The 10 obtained points after using the fixed size method with the quadratic Renyi entropy

Figure 5: Visualising the feature space

training and then retraining, and 2) enforcing sparseness from the beginning. An advantage of the first solution is that it is not guaranteed that the numberof SVs will be greatly reduced. In the second solution, the L_0-norm has been receiving increasing attention by the Machine Learning community. It counts the number of non-zero elements of a vector. Therefore, minimizing it leads to very sparse models. Unfortunately, this cannot be done in a straightforward manner, since the resulting problem is NP-hard [1]. To handle this problem, some approximations to it are looked into [2].

In this method, the parameter k is a factor used to determine the number of representative points M by heurisitic $k\sqrt{N}$ where N = dataset size. For the comparison of the results of fixed-size LS-SVM to the SV_L_0_norm-approximation in terms of test errors, number of support vectors and computational time, I'd like to refer to the homework problems on Fixed-size LS-SVM's.

2 Homework problems

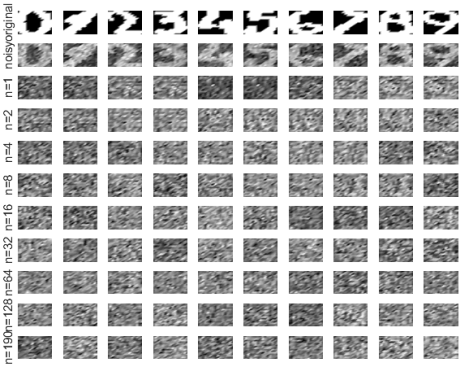
2.1 Kernel principal component analysis

Densoising digits on which Gaussian noise is added consists of the extraction of the main components of a pattern and not including the less significant ones. More technically, this corresponds to the matrix decomposition into eigenvectors and eigenvalues of a covariance matrix. In the case of a LS-SVM approach to a kernel PCA, the matrix to be diagonilised is the one of equation 1. Each element of the matrix Ω_c is a relative measure of the distance between two data-points x_a and x_b compared to the rest of the data-set (see also equation 5). This value will thus be high if x_a and x_b are closer to each other (due to the negative exponent of e). Ω_c can thus be interpreted as an enhanced covariance matrix of the data. Analogous to the reasoning in subsection 1.2, if the σ^2 is very low, the resulting measure of distance will be very strict. In order to detect correlation between two data points, the data points should be extremely close to one another. This will result in very few correlated point and an overall constant Ω_c. After calculating the matrix decomposition, the small number of principal components with a non-zero loading will correspond to very specific features that could very well not be shared by all the digits of the same class. We should thus be aware that σ^2 should not be chosen too low.

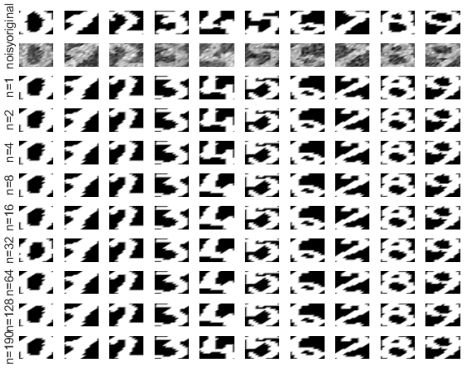
On the other hand, when the σ^2 is too high, all point will be considered close to each other, without much discrepancy. The resulting Ω_c's values will thus be quite constant and high. The resulting principal components will correspond to very broad features often shared by many classes of digits. In this case, these PC's will not be able to specify the important features over the unimportant ones (such as noise). In the figures entailed by figure 6, different σ^2 values are wielded for a noise factor of 0.3. Indeed, when decreasing the σ^2 value, only very clear components of the digits are found, while other less pronounced patterns are neglected. Decreasing σ^2 even more results in the distance measure being so strict that no patterns are found and the reconstructed images consist of random noise only (also, Matlab sputters : "Starting value changed"). Increasing σ^2 will over-generalize and eventually result in noisy reconstructed digits as well, as the KPCA cannot keep the importance of the noise and the underlying non-noisy features from each other.

In second investigation, regular PCA and KPCA are put next to eachother, for a gaussian noise factor of precisely 1. Results of the reconstruction of the digits using PCA are shown in figure 7a, KPCA's reconstruction results are shown in 7b. In both cases, 1, 2, 4, 8, 16, 32, 64, 128, and 190 PC's were used. It's clear that KPCA manages to "memorize" the digit's features without the noise. Due to the high noise, reconstructing the digits does not always happens correctly (seven is reconstructed as a two and three as a five for a higher number of PC's). It is also perceived that increasing the number of PC's for KPCA makes the picture more and more clear. Though it must be noted that the first 8 PC's in this case of linear PCA seem to depict some general digit's features and not the noise, it never manages to reconstruct the images well enough. The rule of thumb that was wielded to get to the value of σ^2 is equal to a constant (0.7) times the number of dimensions (240) times the mean of variances for the pixel values for each digit in the data set, which results in $\sigma^2 = 35.9078$.

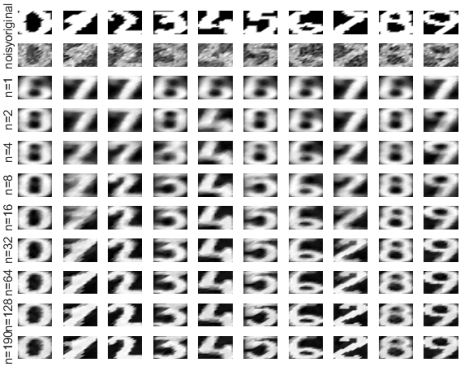
Tweaking of the parameter σ^2 could also be done based on the RMSME on the training and validation set. The RMSME is the RMSE where the error is equal to the mean of the errors for every pixel of an image. In figures 8a, 8b and 8c, the results on test and training set are depicted for different σ^2 values, for a noise level of 0.3 and taking the first 4, 16 and 64 PC's into account, respectively. The optimal σ^2 tends to increase and the RMSME tends to decrease with an increasing number of components. Again, we perceive that for low values of σ^2, there is overgeneralisation, which is why the training set error is quasi non-existent. For too high σ^2 values, training and validation set rise, as the reconstructed images are extremely noisy. The best points are reached in the minima of the validation set. For the case of 64 PC's, the reconstructions are shown in figure 9a, and seems to be of plus minus equal quality, in comparison with the thumb rule.



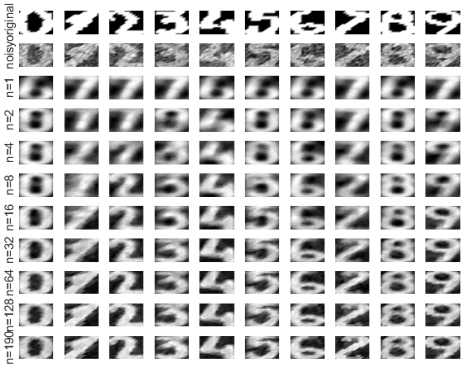
(a) $\sigma^2 = 0.01$



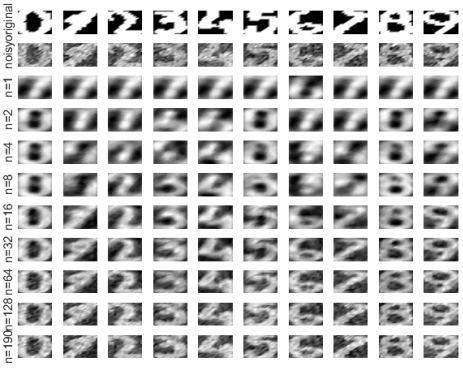
(b) $\sigma^2 = 1$



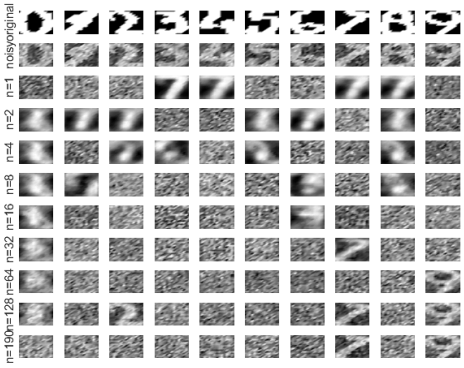
(c) $\sigma^2 = 35.9078$



(d) $\sigma^2 = 100$



(e) $\sigma^2 = 1000$



(f) $\sigma^2 = 10000$

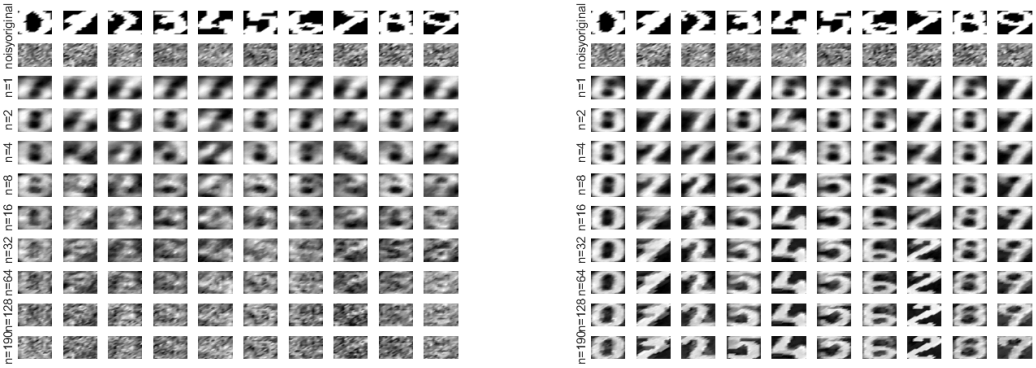
Figure 6: Reconstructing noisy digits using KPCA for different σ^2 values

2.2 Fixed-size LS-SVM

2.2.1 Shuttle (statlog)

The Shuttle data set containst 58000 tuples with 9 explanatory numerical variables. There are 7 classes in total and approximately 80% of the data belongs to class 1. Therefore the default accuracy is about 80%. The aim proposed by the authors of the data set here is to obtain an accuracy of 99 - 99.9%.

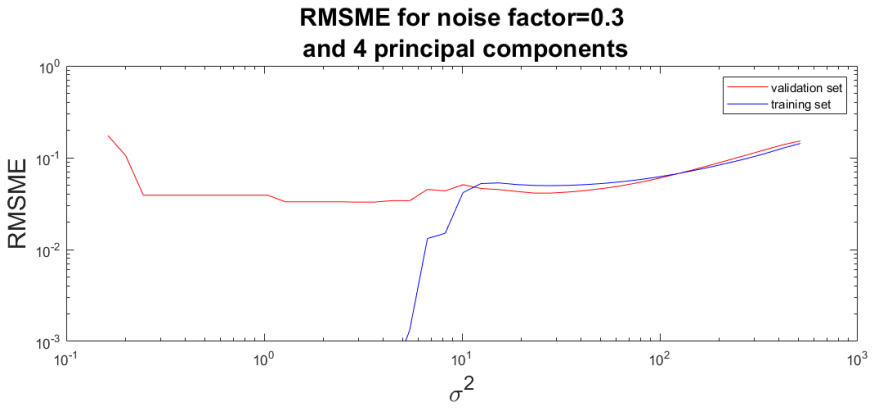
As this problem is a multi-classification problem, and the FS-LSSVM toolbox only implements



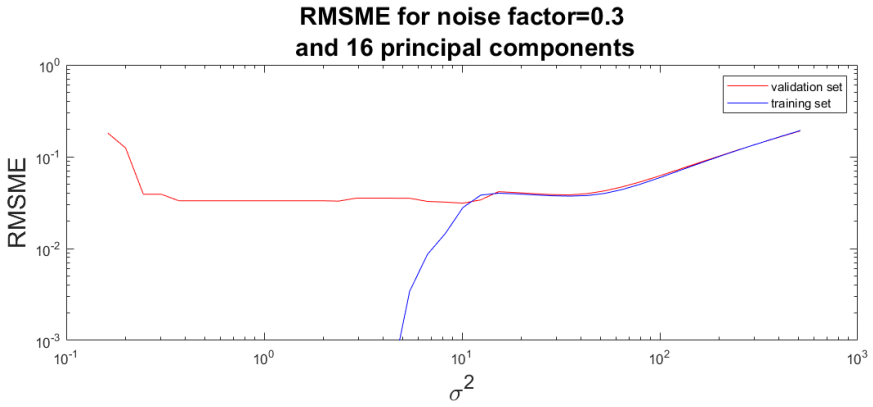
(a) PCA (b) KPCA

Figure 7: Reconstructing noisy digits (gaussian noise=1)

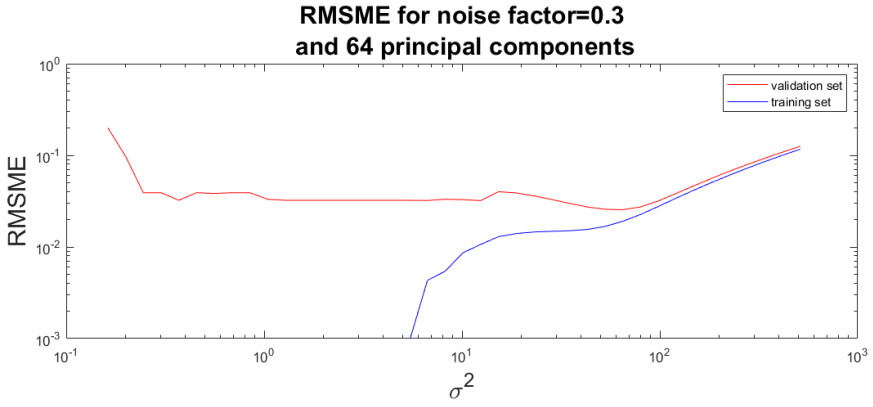
binary classification it is not ideal. I felt that implementing the multi-classification problem to the toolbox was not part of the scope of this assigment, but rather stating and interpreting the differences between the FS-LSSVM and the SV_L_0_norm. The SV_L_0_norm performs an L_0_norm over the Nyström approximated support vectors and incorporates information about the entire dataset in the kernel matrix while just using the support vectors for training. I changed the classes so that the biggest class (1) forms a class on its own, and all others (2-7) are also combined. For computation time purposes, the linear kernel was wielded and only 10000 points were used. 80% thereof were randomly assigned to be part of the training set whilst the other 20% are part of the validation set. In this case, I opted for the linear kernel and Randomised Directional Search as the latter runs in parallel and the combination would benefit computation times, which did not seem to work as the *ds* option crashed the lssvm toolbox. Instead, I used the linear kernel in combination with Coupled Simulated Annealing (which runs sequential). Figures 10, 11, and 12 show the results for computation times, error rates and number of support vectors for both methods. Computation times rise with the number of k, as increasing k imposes a bigger data set that needs processed. The number of support vectors is equal for both norms. Computation times do not differ much between the two methods and it seems that FS-LSSVM is superior as it involves a lower error rate. For different k values, only the error rate of the SV_L_0_norm seems to change slightly (highest error rate of 3.6% for k=5).



(a) 4

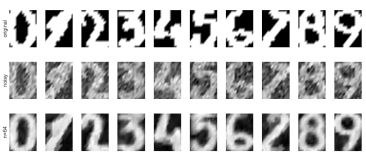


(b) 16

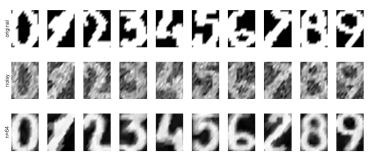


(c) 64

Figure 8: RMSME for different number of PC's

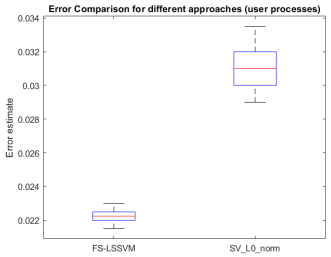


(a) Mimima of validation set for 64 PC's

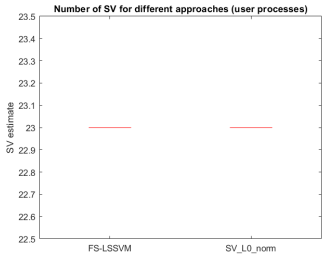


(b) Thumb rule

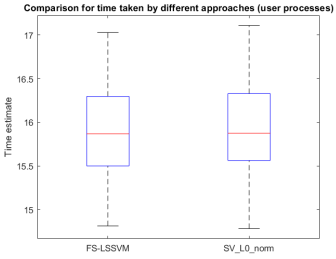
Figure 9: reconstruction for different values of σ^2



(a) error rates

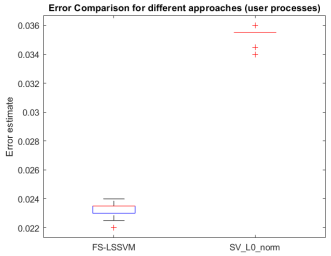


(b) number of support vectors

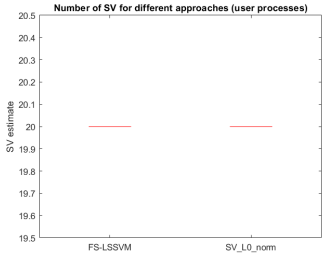


(c) computation times

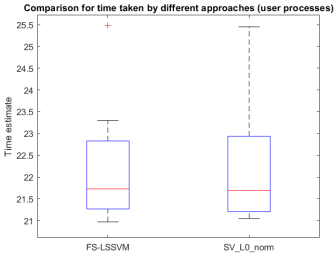
Figure 10: Results when k equals 2



(a) error rates

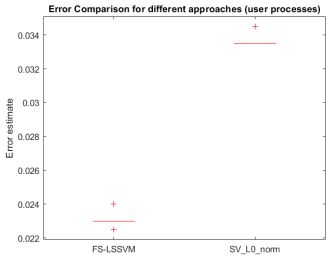


(b) number of support vectors

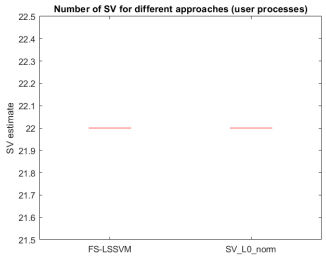


(c) computation times

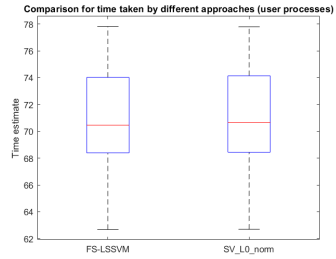
Figure 11: Results when k equals 5



(a) error rates



(b) number of support vectors

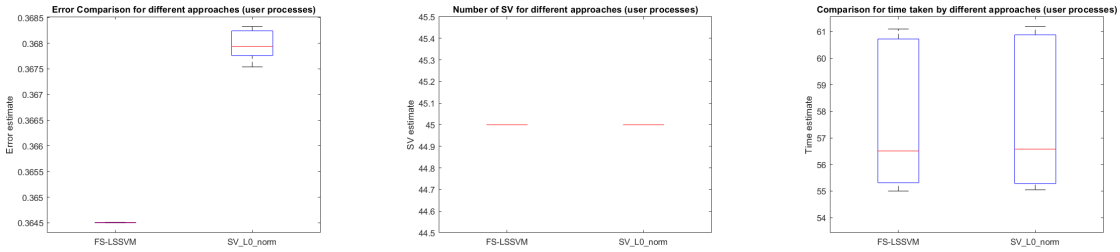


(c) computation times

Figure 12: Results when k equals 10

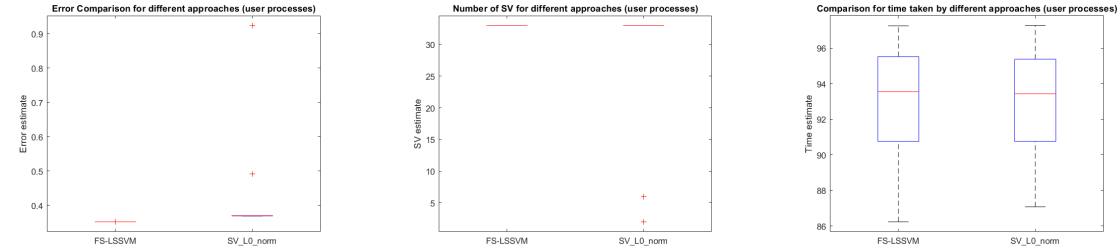
2.2.2 California

The regression problem of the California Housing dataset, which contains 20460 tuples, is stated as finding the median house prices (more specifically ln(median house value)) for California districts derived from the 1990 census. There are 8 continuous explanatory variables and one binary variable which is omitted in the dataset we've gotten. The eight explanatory variables include longitude, latitude, house median range, total number of rooms, total number of bedrooms, population, households, median income, and median house value. Again, 80 percent of the data is used as training set, the remainder as validation set. CSA was again used as search method. Data normalization was carried out as the ranges of the features are quite different. For the linear kernel, the results were not of superior quality (see figures 13 and 14). Therefor, also the polynomial kernel was looked into, which involves higher computation times but improves the accuracy (see figures 15 and 16). There seems to be no difference anymore in number of SV's in these cases.



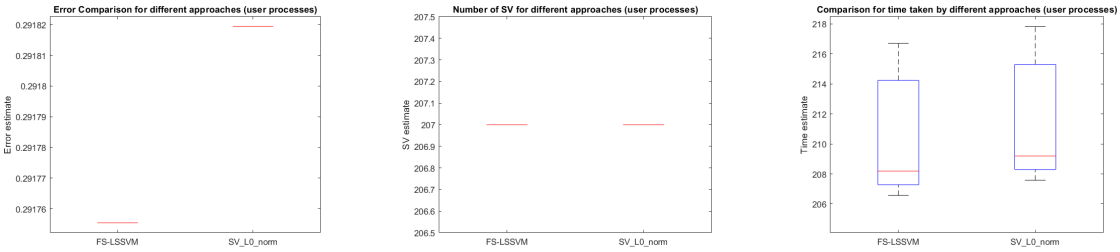
(a) error rates (b) number of support vectors (c) computation times

Figure 13: Results when k equals 4, linear kernel



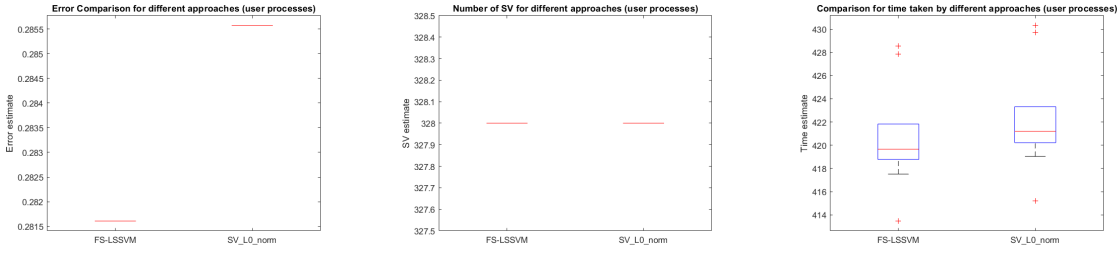
(a) error rates (b) number of support vectors (c) computation times

Figure 14: Results when k equals 8, linear kernel



(a) error rates (b) number of support vectors (c) computation times

Figure 15: Results when k equals 2, polynomial kernel



(a) error rates (b) number of support vectors (c) computation times

Figure 16: Results when k equals 4, polynomial kernel

References

[1] Jorge Lázaro, K De Brabanter, José Dorronsoro, and Johan Suykens. Sparse lssvms with l0-norm minimization. *ESANN*, pages 189–194, 01 2011.

[2] Jason Weston, André Elisseeff, Bernhard Schölkopf, and Mike Tipping. Use of the zero-norm with linear models and kernel methods. *Journal of machine learning research*, 3(Mar):1439–1461, 2003.