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.

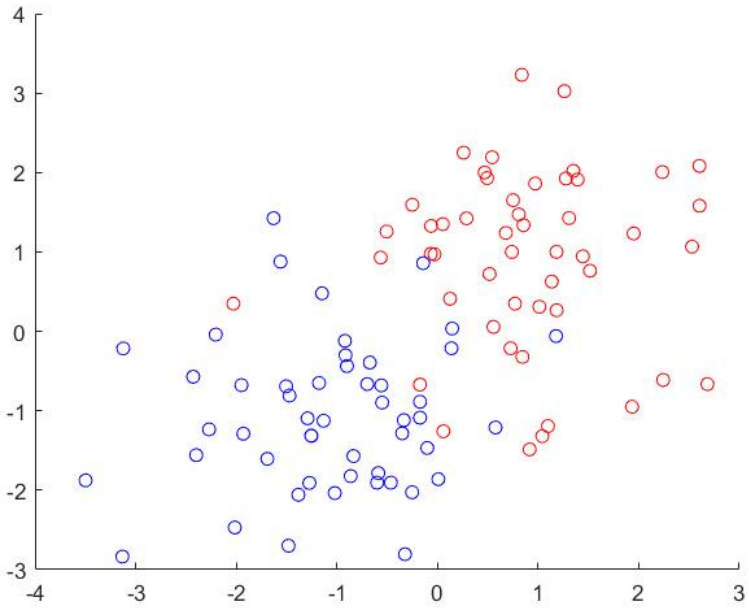


Figure 1: Graphical representation of the binary classification problem.

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.

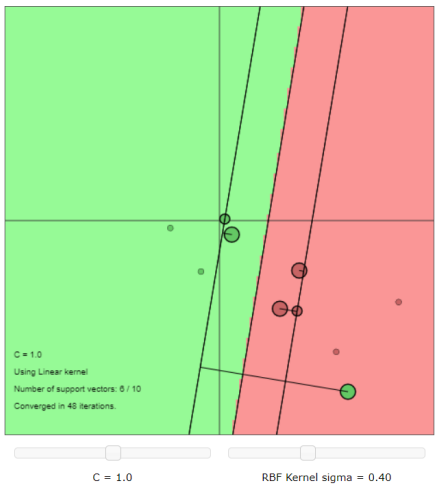
1.2.1 Linear kernel

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.

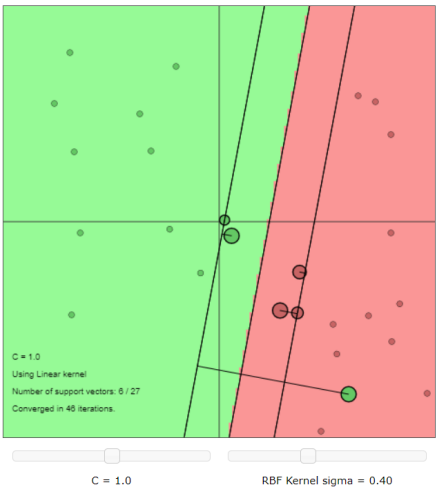
If points are added within the margin, they take the role as support vectors, depicted as big coloured circles in figure 2c. These support vectors are the non zero elements in kernel matrix, obtained when solving the convex QP problem (sparsity property). The decision boundary can be expressed by a these 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

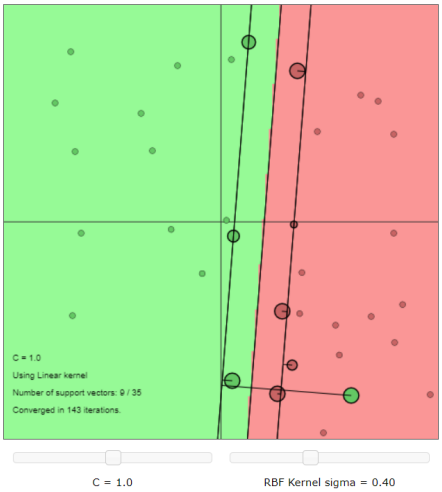
vectors, even when residing outside of the margin. They all have an influence on the width of the margin as well.



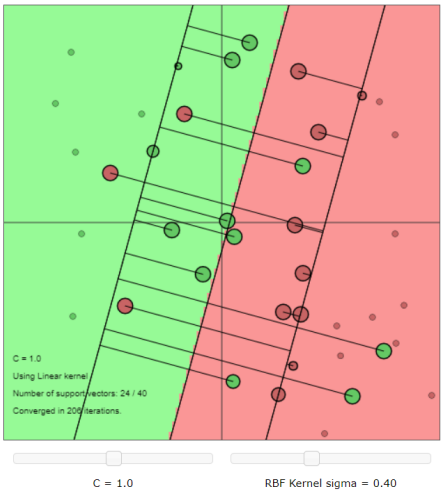
(a) Starting position



(b) Added points outside margin



(c) Added points inside margin



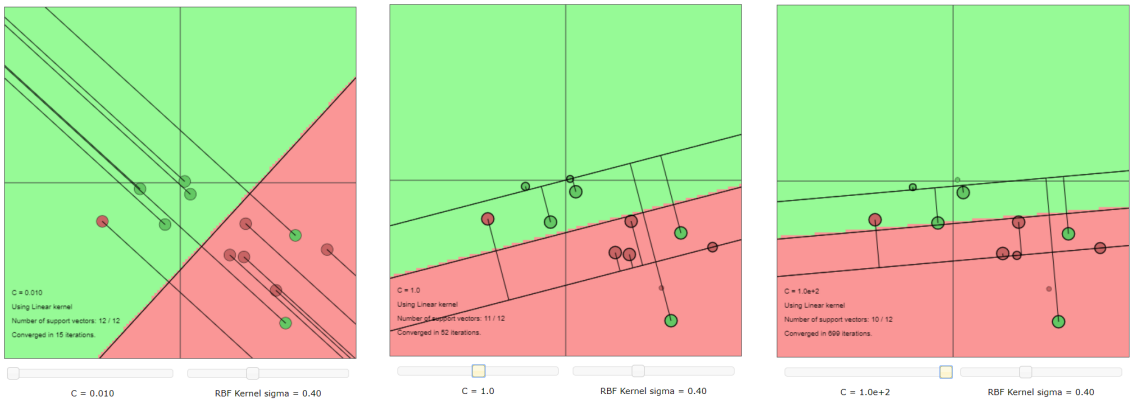
(d) Added points on the wrong side of the dicision boundary

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.

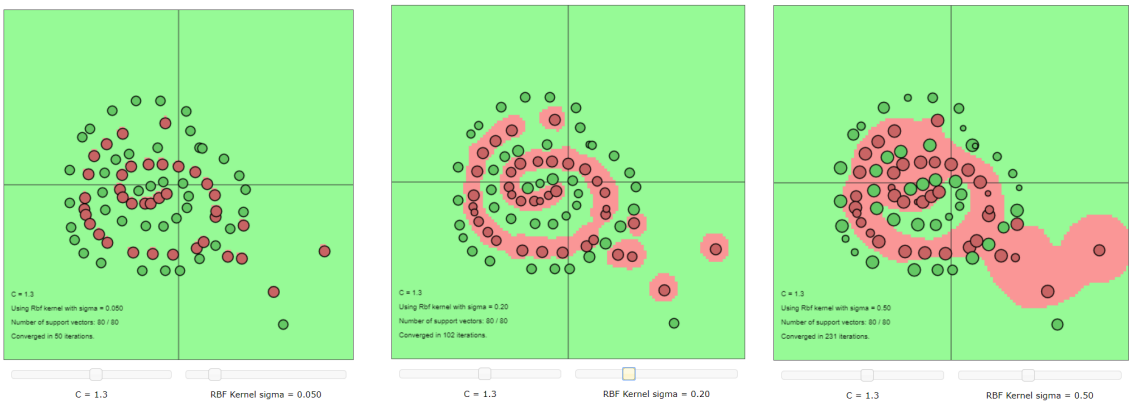
$$y(x) = sign[\sum_{k=1} \alpha_k y_k \exp\left(-\frac{||x - x_k||_2^2}{\sigma^2}\right) + b] \tag{2}$$

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



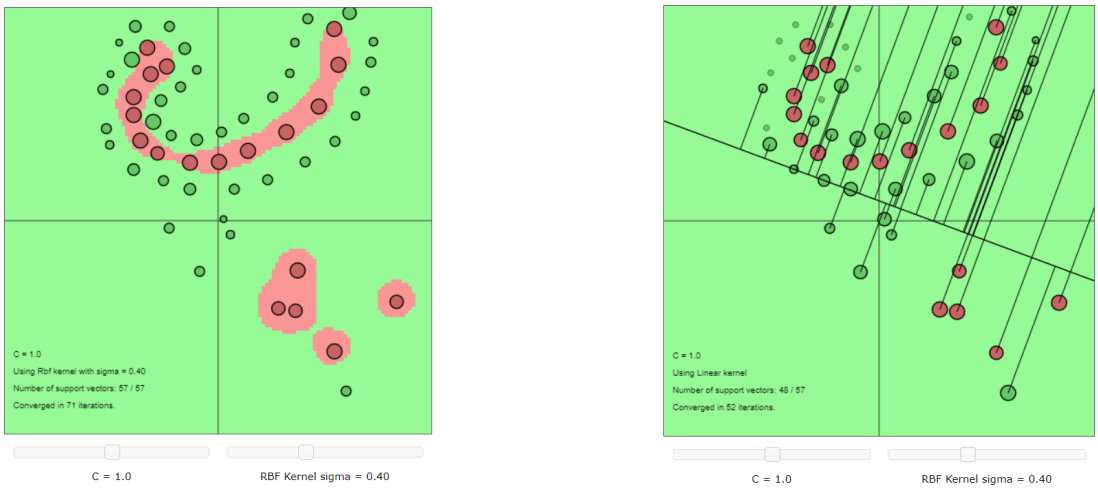
(a) σ=0.05 (b) σ=0.2 (c) σ=0.5

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

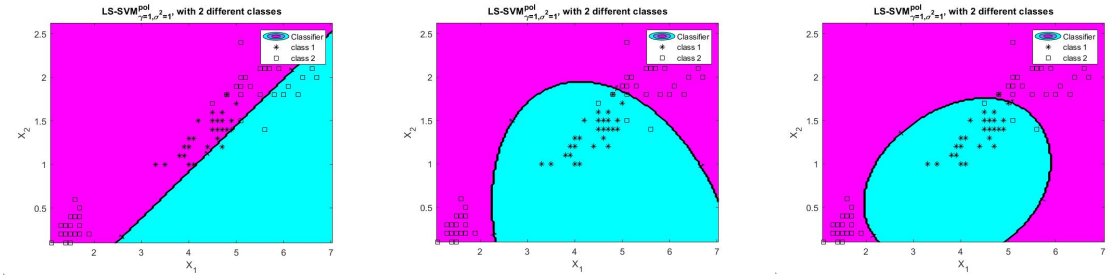
Figure 6: Highly non-linear dataset classified using different kernels.

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

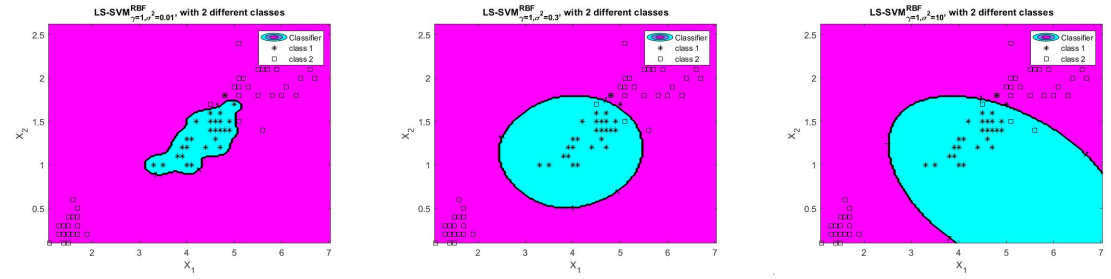
Figure 7: Graphical results for the polynomial kernel for different degrees.

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



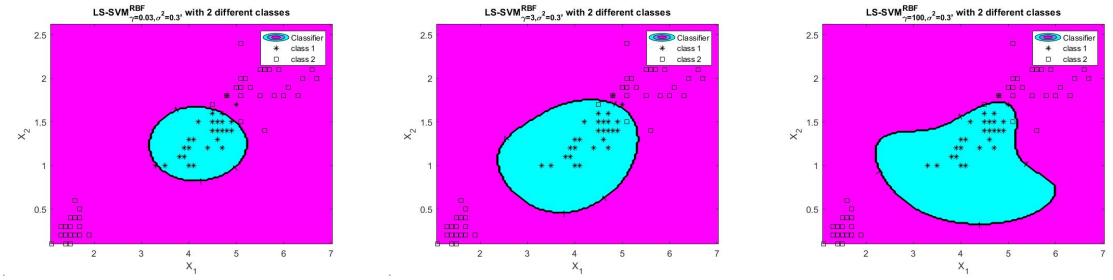
(a) $\sigma^2 = 0.01$ (b) $\sigma^2 = 0.3$ (c) $\sigma^2 = 10$

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.

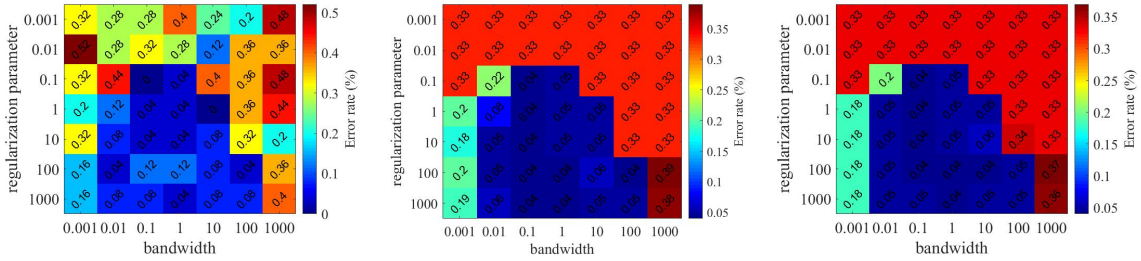


(a) $gam = 0.03$ (b) $gam = 3$ (c) $gam = 100$

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

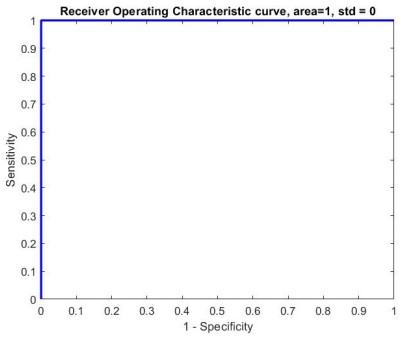
Both algorithms return different results for the hyperparameters each time, of which each of them corresponds to the results shown in figures 10a, 10b, and 10c. For both algorithms, the cost is always equal to 0.02, 0.03 or 0.04. Simulated annealing is a local search method and returns the parameters 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

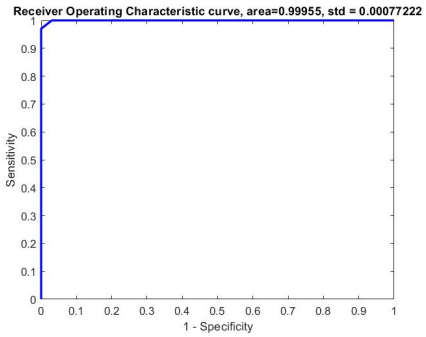
ROC curves are a measure of performance of your model. The goal is to maximize the area under the curve (AUC). This AUC is determined by the True Positive Ratio (TPR) and the False Positive Ratio (FPR). For a particular class i, TPR is the number of outputs whose actual and predicted class is class i, 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

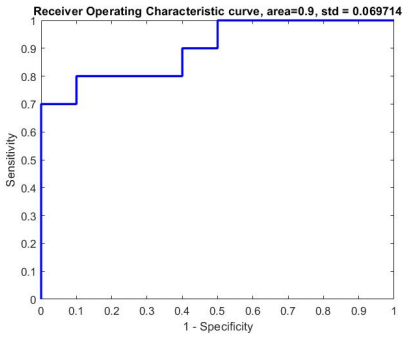


(a) ROC on test set

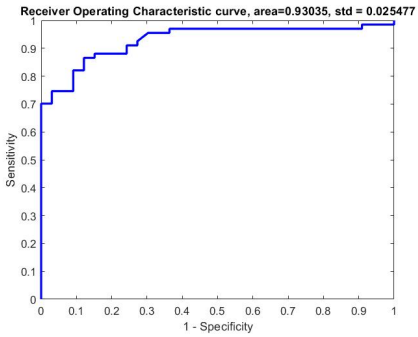


(b) ROC on training set

Figure 11: ROC's with tuned parameters.



(a) ROC on test set

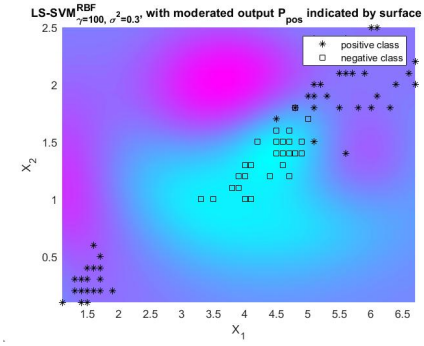


(b) ROC on training set

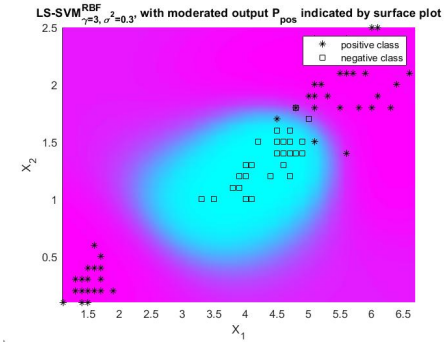
Figure 12: ROC's with a bad parameter combination.

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

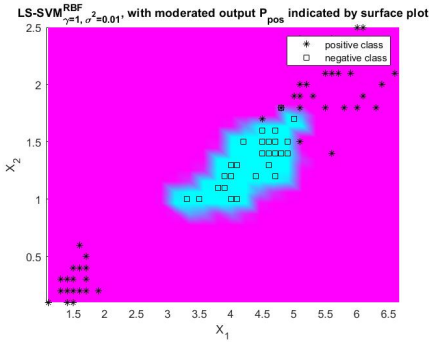
The color purple denotes high probabilities of points in that area belonging to the crosses' class. Light blue denotes a probability of zero percent of the point belonging to the cross class (or 100 % that the point belongs to the square class). For high valued regularization parameters and higher bandwidths, 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.



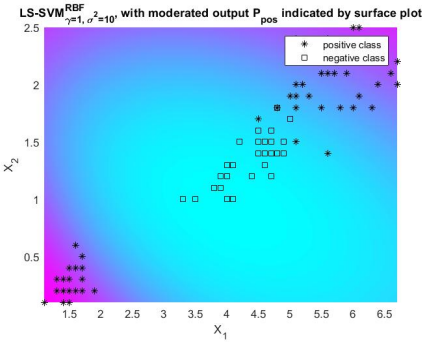
(a) Tuned parameters.



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



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



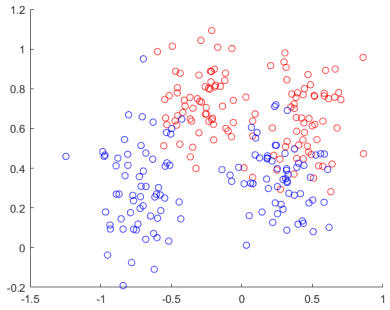
(d) $\gamma = 1$ and $\sigma^2 = 10$

Figure 13: Probability plots of different settings.

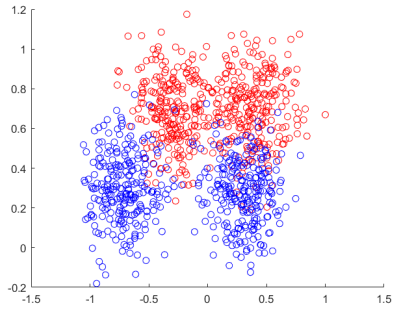
2 Homework problems

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

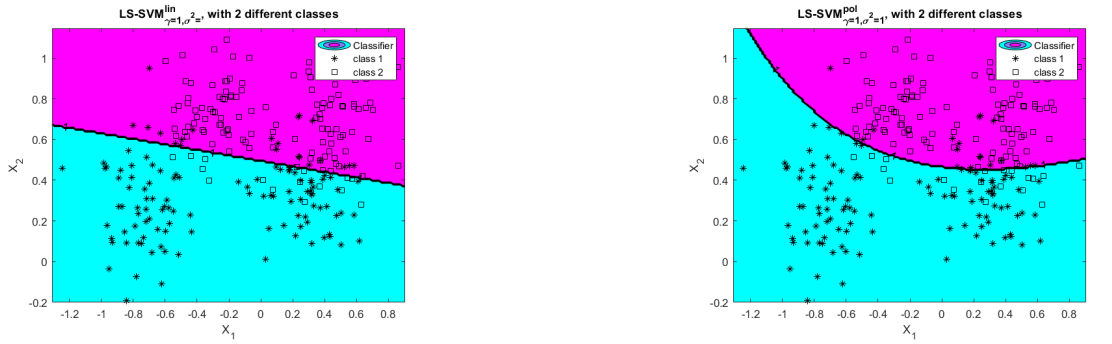


(b) data set

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.

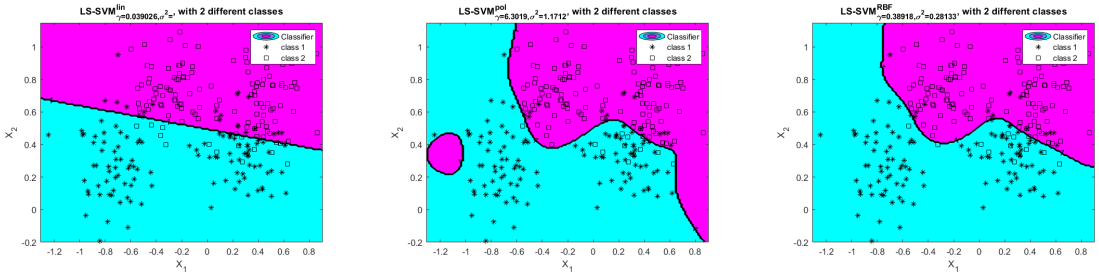


(a) Linear kernel

(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.

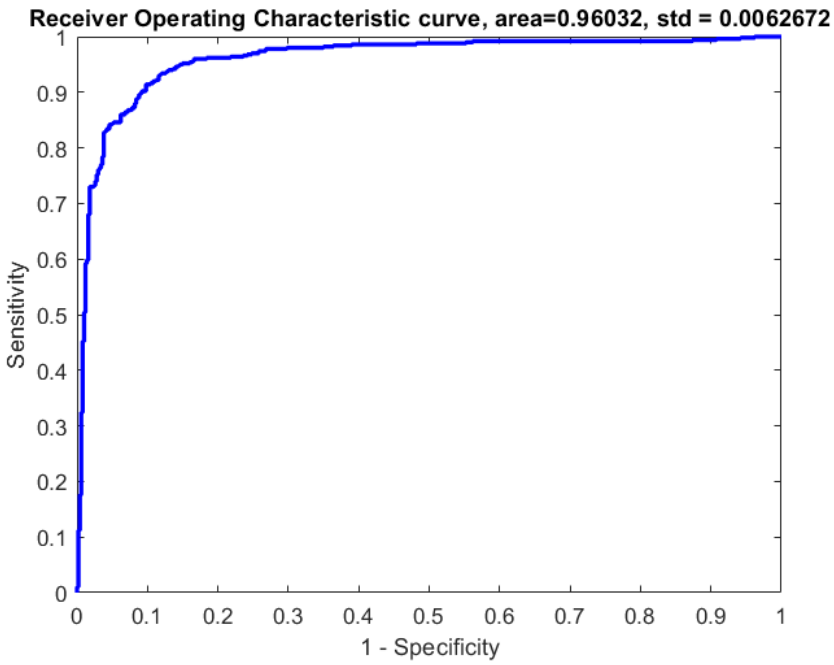
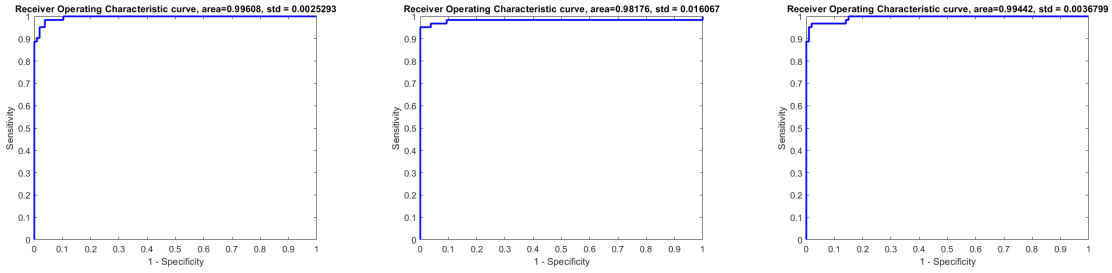


Figure 17: ROC of the best RBF kernel.

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.

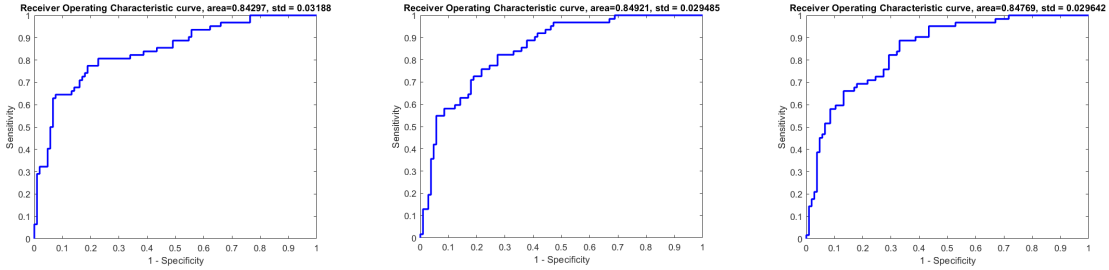
It could be interesting to have extra features in the LSSVM toolbox with regard to misclassification, so that not only the error rate of all classes can be used as a metric to run the automatic parameter tuning alogirthms upon, but also misclassifications of specific classes (in this topic the malignant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