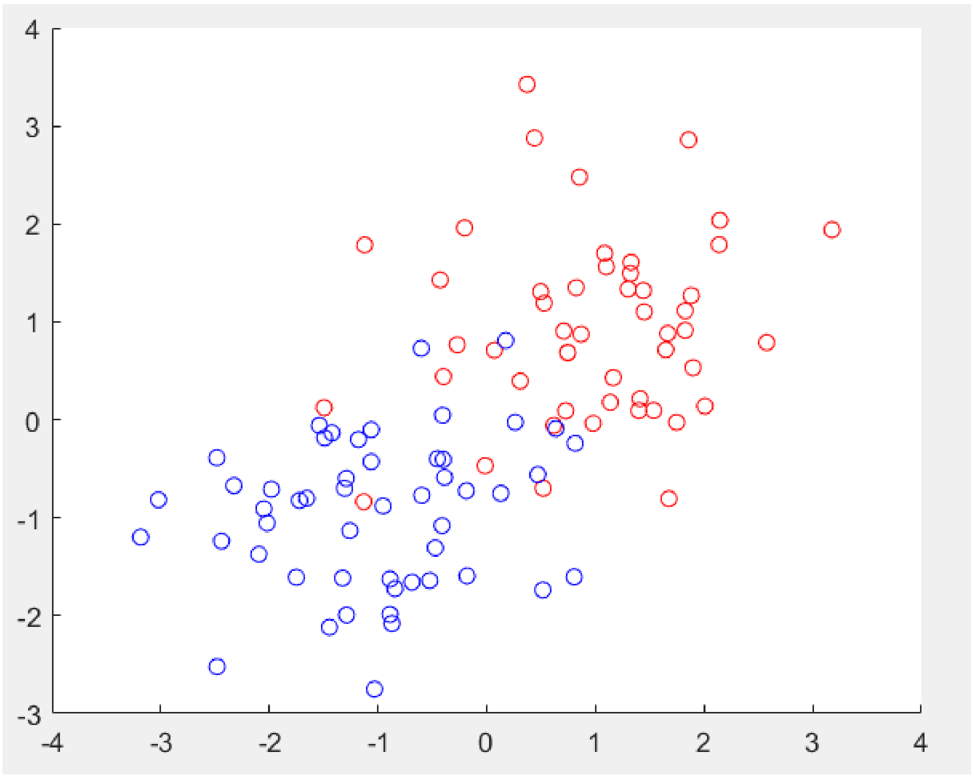
Kenneth Devloo, s0219469

Support vector machines: Exercise session 1

1.1 A Simple Example: Two Gaussians

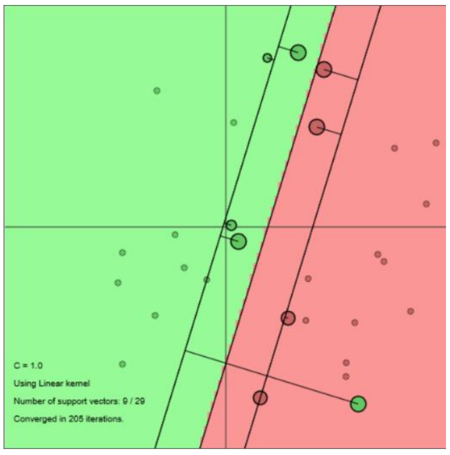
With just a line you can solve this linearly. You will however have misclassifications. A way to measure the cost would be to count the errors.



1.2 The Support Vector Machine

1. Follow the instructions. Switch to using the linear kernel by toggling the "k" button. Adjust the existing datasets to have at least 10 data points for each class. What do you observe when you are adding data points to the classes? How drastically can classification boundaries change?

It will only slightly change the estimation as longs as you remain outside of the margins for SVM


C = 1.0
Using Linear kernel
Number of support vectors: 9 / 29
Converged in 205 iterations.

2. What if you add an outlying datapoint which lies on the wrong side of the classification boundary? How does it affect the classification hyperplane?

One outlier will not change the classification much as long as there are plenty of datapoints on the correct side. However, the new datapoint will be misclassified and this may not be solvable in a linear way. (The error rate will be too high.)

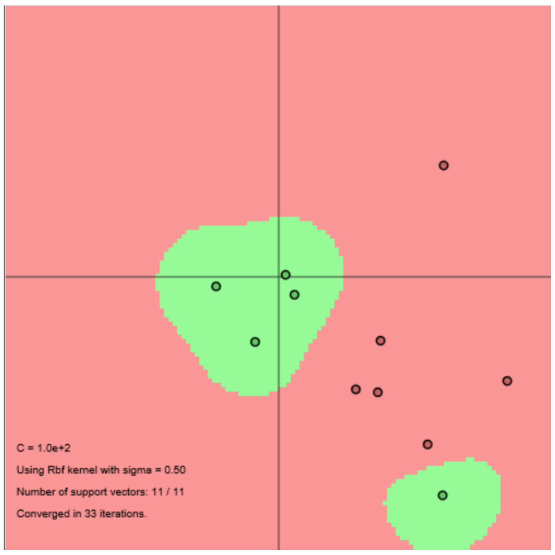
3. Try different values of C regularization hyperparameter. How does it affect the classification outcome? What is the role of it?

Lowering C can in theory help against overfitting. In general, it will also create give bigger margins to account However, a value too low will cause all datapoints to be classified in one class.

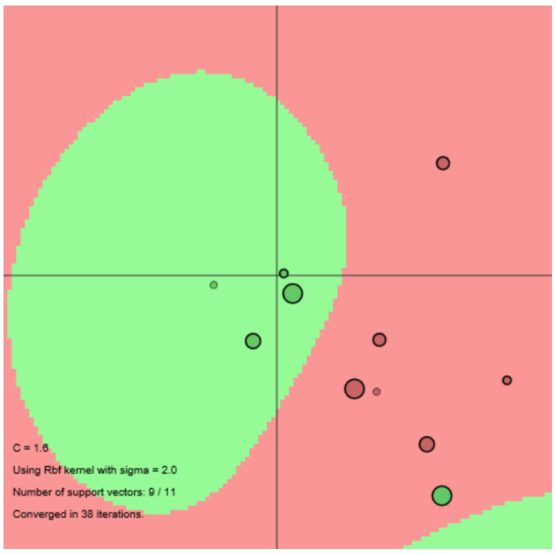
Raising c will give smaller margins to classify more accurate.

4. Follow the instructions and switch back to RBF kernel by toggling the "k" button. Compare to the classification outcome of the linear case.

Sss



```
C = 1.0e+2
Using Rbf kernel with sigma = 0.50
Number of support vectors: 11 / 11
Converged in 33 iterations.
```



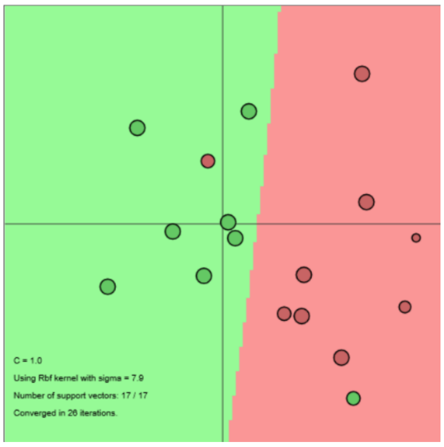
```
C = 1.6
Using Rbf kernel with sigma = 2.0
Number of support vectors: 9 / 11
Converged in 38 iterations.
```

With an rbf kernel, we can have an outlier and classify it right. This is a big advantage over lthe linear case. This actually also makes it sensitive to wrong generalisation. Certainly if that outlier was misclassifid. Experimenting with different values, and trying them on cross validation sets may help to prevent this.

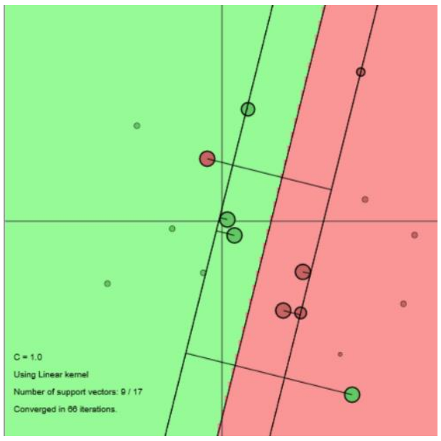
5. Try to change the RBF kernel sigma hyperparameter. What is your intuition? How does it affect the classification boundaries? Now try to change both hyperparameters. What is the right choice of those if your data is almost linearly separable?

Intuitivly, I would want all misclassified points not to be included and raise the kernel size in order to make sure that support vectors have enough weight to outrule outliers.

If the data is almost linearly seperabale I would raise the kernel the size as it will approximate a linear solution: On the left image, classification is done using RBF kernel. It is similar to the right iage wich uses the linear method. In both cases, outliers do not add to the classification results and thus we prevent overfitting. (Low alpha value.)

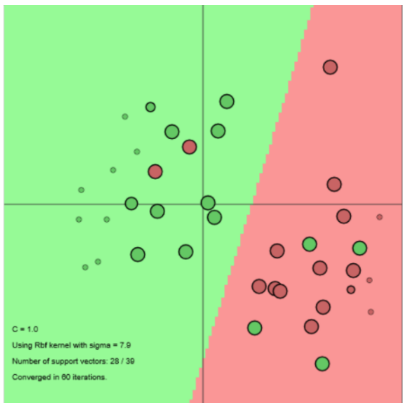


```
C = 1.0
Using Rbf kernel with sigma = 7.9
Number of support vectors: 17 / 17
Converged in 26 iterations.
```

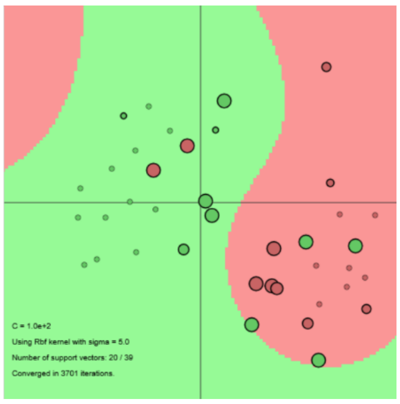


```
C = 1.0
Using Linear kernel
Number of support vectors: 9 / 17
Converged in 66 iterations.
```

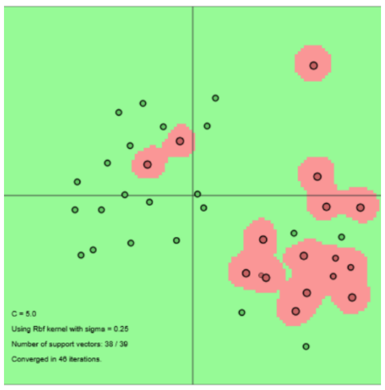
6. Create a linearly non-separable dataset with an overlapping region between classes (e.g. similar to the previous Gaussian clouds). Give comments on the role of the chosen kernel, the regularization parameter (C) and the kernel parameter (sigma).



```
C = 1.0
Using Rbf kernel with sigma = 7.9
Number of support vectors: 28 / 39
Converged in 60 iterations.
```



```
C = 1.0e+2
Using Rbf kernel with sigma = 5.0
Number of support vectors: 20 / 39
Converged in 3701 iterations.
```



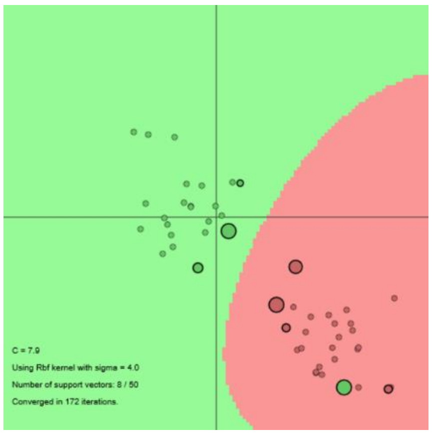
```
C = 5.0
Using Rbf kernel with sigma = 0.25
Number of support vectors: 38 / 39
Converged in 46 iterations.
```

A big value for sigma will still make this act close to a linear classifier. (1st and 2nd picture.) The classifier will try to fit the data better. Support vectors will have a different weight here. Lowering sigma (3rd picture), wil allow us to classify all training data correctly. However, if these outliers are incorrect, we will have bad generalisation. What comes to my mind here is that we would need mre data to do the classification correct. (Higher dimensional feature space.)

7. What is the role of Support Vectors? Change the data-sets to make the number of support vectors increase/decrease. When does a particular datapoint become a Support Vector?

Intuitivly, datapoits become support vectors when they are close to the other class since these have to make the border between two classes. Havng outliers seems to make more support vectors. See previous question fo that. Thick circles show support vectors.

Here is also an example without outliers and two nicely seperated groups to show we do not need much support vectors. We see that outliers also become support vectors.

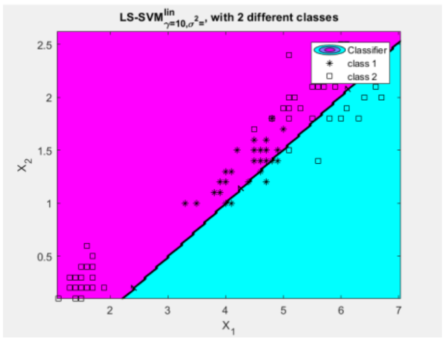


```
C = 7.9
Using Rbf kernel with sigma = 4.0
Number of support vectors: 8 / 50
Converged in 172 iterations.
```

8. When does the corresponding importance2 of a Support Vector change?

This happens when other support vectors already account for most of the classification.

1.3 Using LS-SVMlab



Shown is a first plot of the test data and the classification by the LS-SVM. We see the error rate here is high.

What is the performance on the test set Xt and Yt?

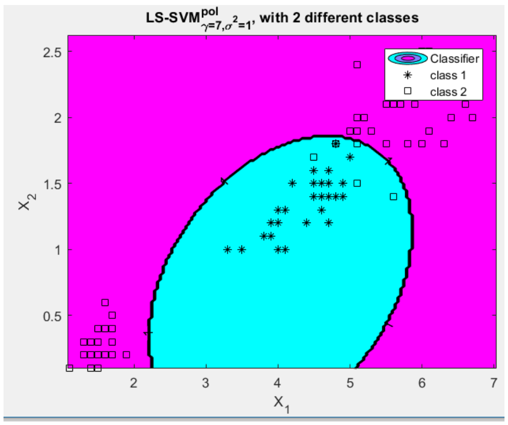
The error rate was 55%. Playing a bit with the regularization parameter did not help here.

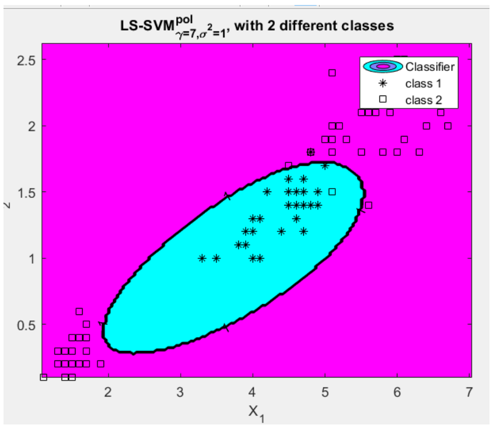
Try out the polynomial kernel with degree = 1,2,3... and t= 1.

What happens when you are changing the degree of a polynomial kernel? Explain the obtained results. Does it correspond to the changes in sigma hyperparameter of the RBF kernel in the previous example?

Accuracy will greatly improve. With degree 1 it will work like a linear kernel. However, with degree 2 we already obtain a 5% error rate on the test set. With degree 3 and higher we have a 0% error rate.

This is because the higher degree allows us to use express the result in function of x1 and x2 better. This is best illustrated with below 2 figures where degree 2 and 3 were used. In the feature space, it puts the border in more variable shapes. The higher the degree, the more complex we can go.

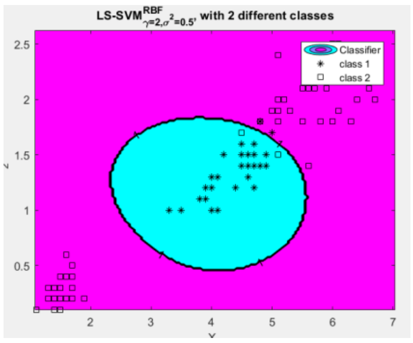


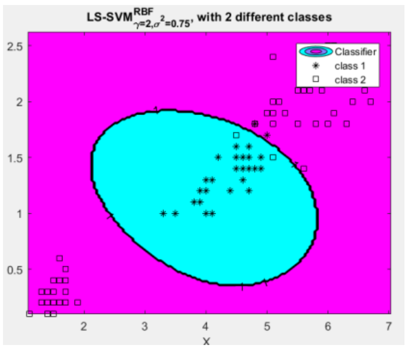


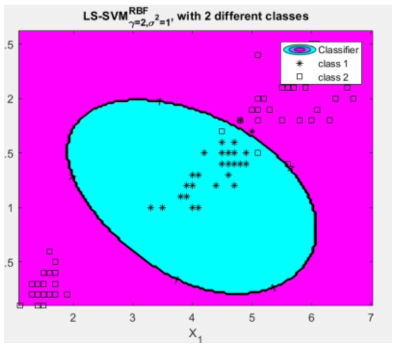
This gives a result that looks a bit like RBF kernels. Note however That usually RBF kernels will also try to classify outliers as it is able classify in different regions in the feature space. Lowering sigma will also make the data fit better. But as shown in the images in 1.2, this acts differently by changing the reach the individual support vectors have on the results.

2. Let us focus on the RBF kernel with kernel parameter the bandwidth σ 2. Try out a good range of different sig2's as kernel parameters. For each individual value of sig2, the corresponding LS-SVM is evaluated on the test set. Make a figure of the sig2's with their corresponding test set performance.

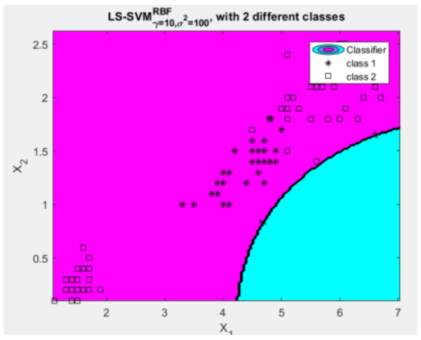
Most values of sig2 actually do fine on this simple test set. If you look at the test set. You will notice class 2 being more the default class. Raising sig2 will give the support vectors of class 1 more weight in the empty space as there is nothing to counter this. Most of the time, I had 0% error rate, as long as I used a good value of gamma. To illustrate this, I have included plots for sigma =0.5, 0.75 and 1.





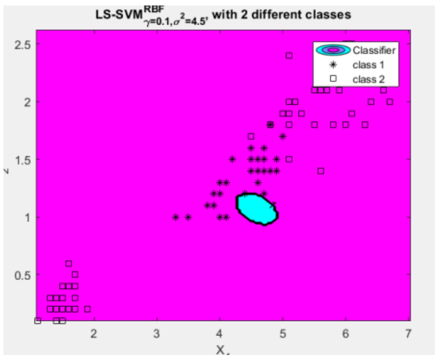


Also added below is the
interesting case where sig2 is extremely high. This fails because the data is not linearly separable.



3. Now, take a look at the regularization constant gam. Fix a reasonable choice for the sig2 of the RBF kernel and again compare a range of gam's by plotting the corresponding test set performances. What is a good range for gam?

Putting this parameter too high or low will cause a solution that is overfitting or underfitting respectively. For this test set, the performance is good as long as it is not too low. A combination of gam <=0.1 performs bad when sig2 is too high. For example, sig2=5.



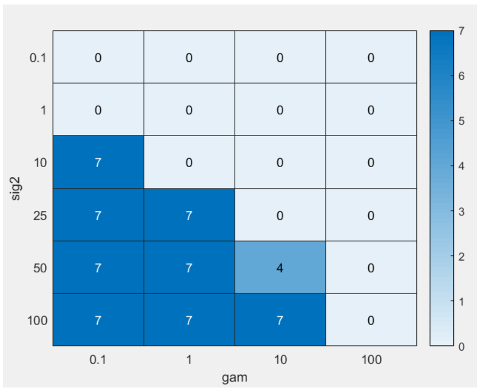
Compare your results with the sample script SampleScript_iris.m on the website of the course on Toledo.

The script on Toledo also show good results with RBF kernels. It is hard to compare both since the iris set is quite simple for this case.

1.3.1 The Choice of the Hyper-parameters

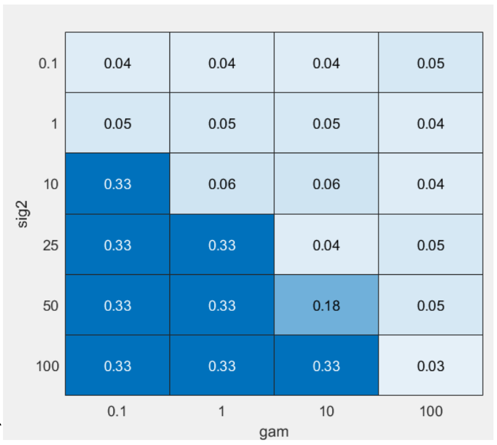
Calculate the performance in terms of misclassification error on the validation set (using estYval and Yval). Make a plot of the performance with respect to several values of gam and sig2 (e.g. gam= 1, 10, 100, sig2= 0.1, 1, 10). Give comments about the results.

Since the training set was quite simple, cross-validation doesn't always show improvement when changing variables as the error rate is 0 as is visible on the right. It completely depends on what set is taken. Using the suggested test values and running the code multiple times, a logical conclusion to be drawn is that we should not use a big kernel with a low regularisation parameter. Having determined good parameters does not mean the features or training data are good. That should be determined with the test set.

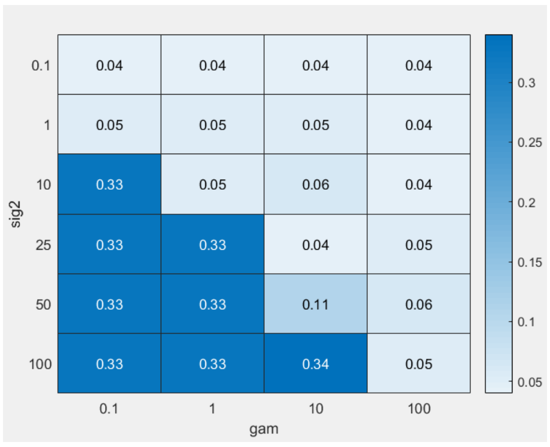


Perform crossvalidation using 10 folds. Think about a clear and intuitive way to represent this technique. Why should one prefer this method over a simple validation? Change crossvalidate procedure for leaveoneout (removing the 10). Is it giving better results? In which cases one would prefer each?

On the left are the results for the 10fold, on the right for leaveoneout. Both have similar results on this simple test set. Leaveoneout does require to build more models. Leaveoneout will validate with as much details as possible, but it is expensive. With bigger training sets, 10fold crossvalidation should be ok.



`

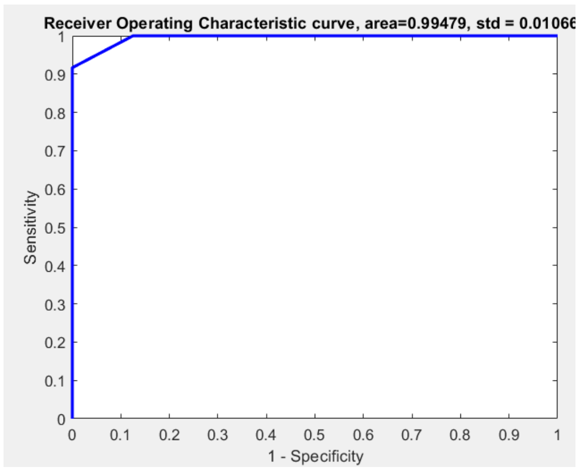


Use tunelssvm procedure to optimize the hyperparameters. Try to change different parameters like 'csa' (Coupled Simulated Annealing) vs. 'ds' (Randomized Directional Search) and 'simplex' (Nelder-Mead method) vs. 'gridsearch' (brute force gridsearch). What differences do you observe? Why in some cases the obtained hyperparameters differ a lot?

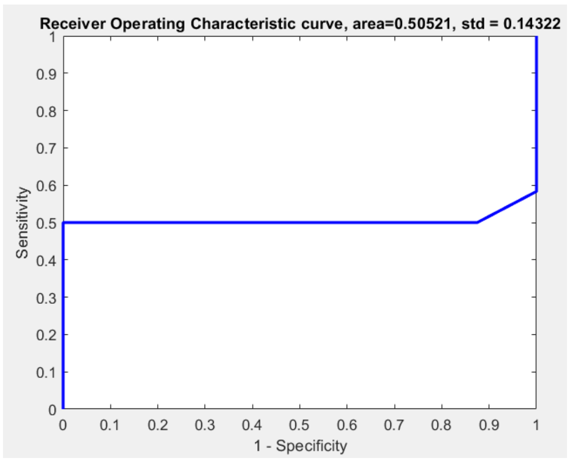
Values come with a cost of 0,04, It can be seen in the above heatmaps that the possibilities are quite broad there. The brute force algorithms do slightly better in results since they search broader and simplex converges to one maximum. Csa or ds determine the start parameters. But since these are random, convergence could happen in any minimum.

One alternative way to judge a classifier is by using the Receiver Operating Characteristic (ROC) curve of a binary classifier.

What matters here is that we have as much as possible space under the line. On the left is a curve for a good model, on the right for a bad model. The ROC curves are made with the validation data. It should not be done with training data to avoid overfitting.



Receiver Operating Characteristic curve, area=0.99479, std = 0.0106
Sensitivity
1 - Specificity

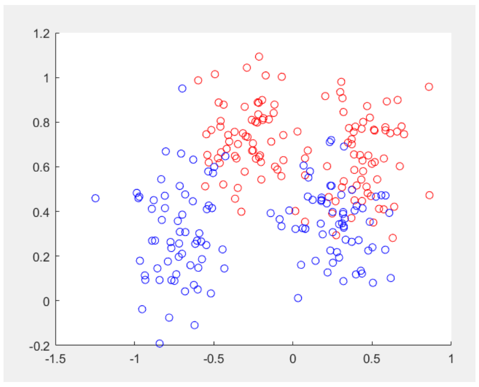


Receiver Operating Characteristic curve, area=0.50521, std = 0.14322
Sensitivity
1 - Specificity

1.4 Homework Problems

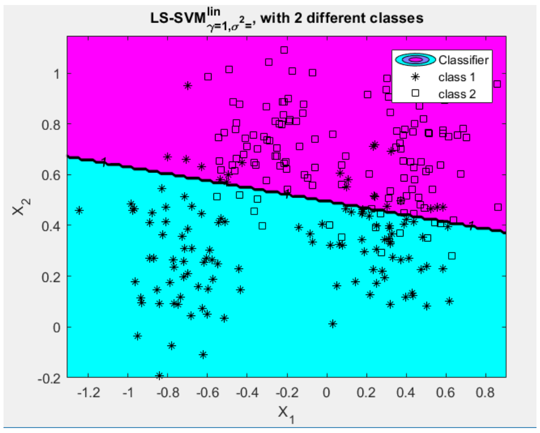
Ripley

The data:



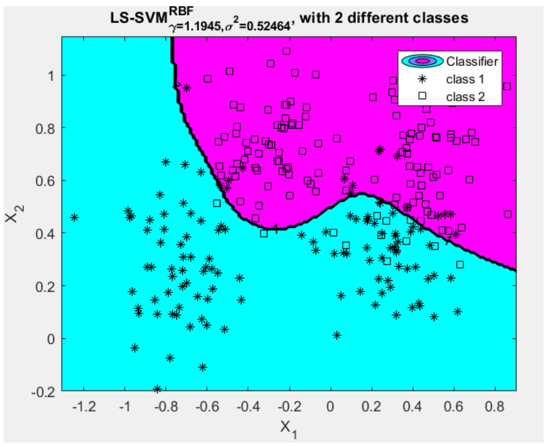
It can be seen that there is an overlap between both classes, so we should expect some error here. A linear kernel does not seem perfectly fit due to this. It doesn't look like a transformation on the features would change this. More ideally there would be more features available. Certainly polynomial kernels will have better results as they add polynomial features or RBF kernels that allow for very flexible borders.

The linear kernel:



The error rate here is 10,8%. There are still some clear areas where improvement seems possible. As noted above, and RBF kernel will probably do better.

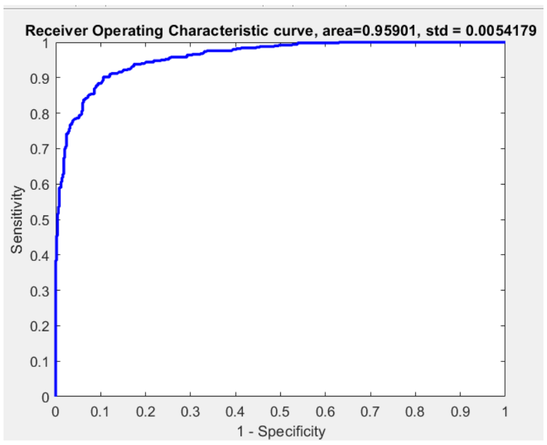
The RBF kernel:

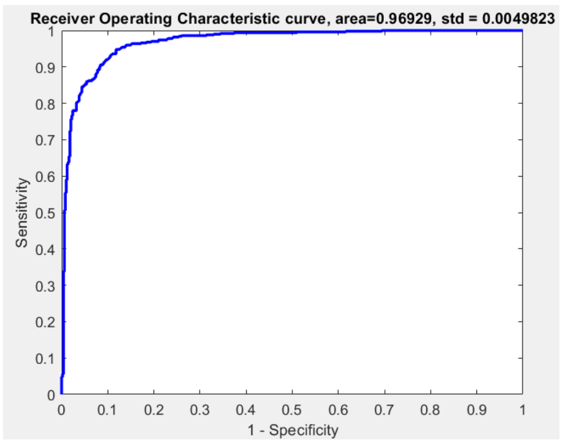


Doing tunelssvm actually improves the parameters on multiple runs. The reason here is that the search is done randomly. With only one iteration. Doing this multiple times gets the error rate just below 10%. In the example here, we get 9,3% error rate.

ROC curve analysis:

On the left is the curve for the linear solution, on the right or RBF.





The RBF kernel has a slight advantage over the linear kernel in the area below the line. Although it is smaller than expected. We might want to prefer the linear svm since it is computationally much cheaper.

Conclusion:

With an error rate of about 10% on the test set, it is still not very accurate. It might be preferred to look for more features to distinguish the classes or to find a better method.

If a choice between linear and rbf would have to be made, linear could be a better choice since it is computationally cheaper and he error rates are similar.

Breast cancer

Data:

The description here is not clear on what the features stand for, so all of them will be used. Important here is that we can't have too many false negatives as in this application you would be better safe and have a doctor look at. Preprocessing was used because we don't know how important all features are.

Linear model:

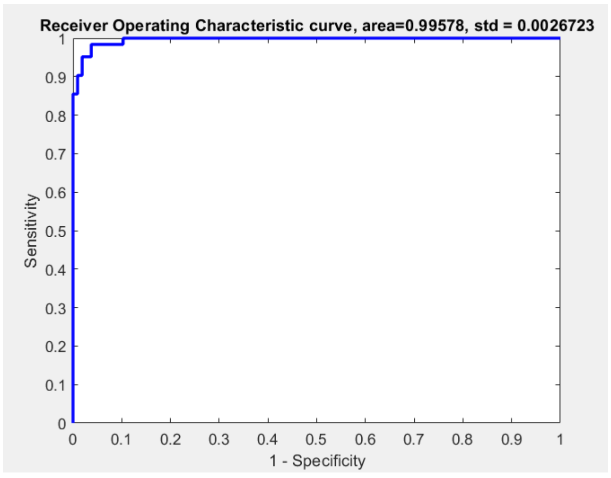
In the linear case, using tunelssvm, error rate was 4.14%. Most of these errors come from false negatives (6/7 errors).

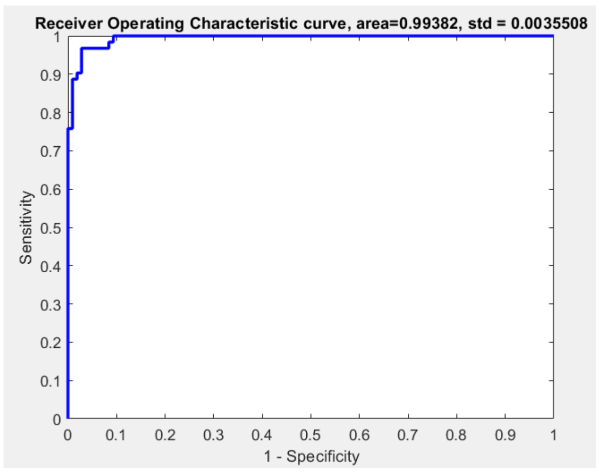
RBF kernel:

For RBF kernels, the error rate and the amount of false negatives is 2.73% (4 samples). Running tunelssvm did improve the performance a lot.

Roc curves:

On the left it is shown linear and on the right for RBF kernels.





Both methods have a good performance and both do similarly well for this case.

Conclusion:

Performance here is very similar. That's in the advantage of the computationally less expensive linear SVM. But since we are talking about a medical application here, I would rather go for the RBF kernel.

Diabetes

Data:

This is also a medical application. Therefore, we will also look at false negatives. No detailed info on the data was available since the website does not work.

Linear model:

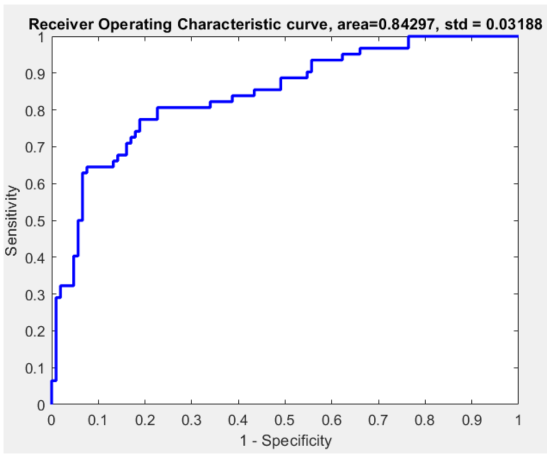
The error rate is 21,43%. The false positive rate was 17,26%.

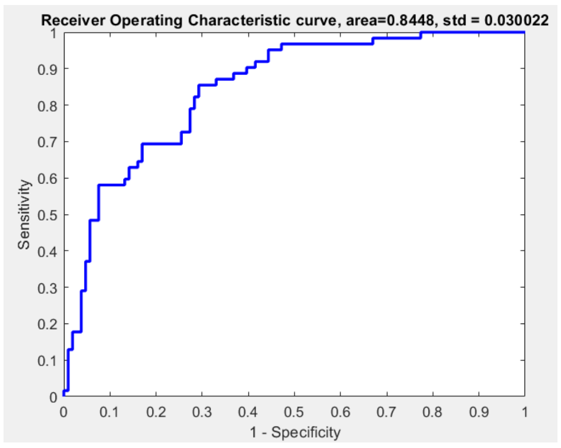
RBF kernel:

The error rate is 22,62%. There are 15,48% false negatives. Changing parameters for tunelssvm only slightly helped.

ROC curves:

On the left is the ROC curve for the linear model, on the right for the RBF version. On both curves, the area is similar under the line.





Conclusion:

These models don't seem the be fit for this application, due to the high error rates. In this case, the difference between RBF and linear is small again. If computational time is what matters, the linear model should be better. Adding more features to discriminate better could also help.

Conclusion

RBF kernels don't always outperform linear kernels when we look at factors like time to compute. It is often sufficient to use a linear model.

In the example datasets, we did not always get the desired performance. Especially the last dataset gave difficulties to discriminate between classes. Adding more features would help here.