Master of bioinformatics

Support Vector Machines

Assignment 1: Classification

Spring 2016

Author:
Cedric Lood

Supervisors:
Dr. Carlos Alaiz
Dr. Emanuele Frandi
Prof. Johan Suykens

May 19, 2016

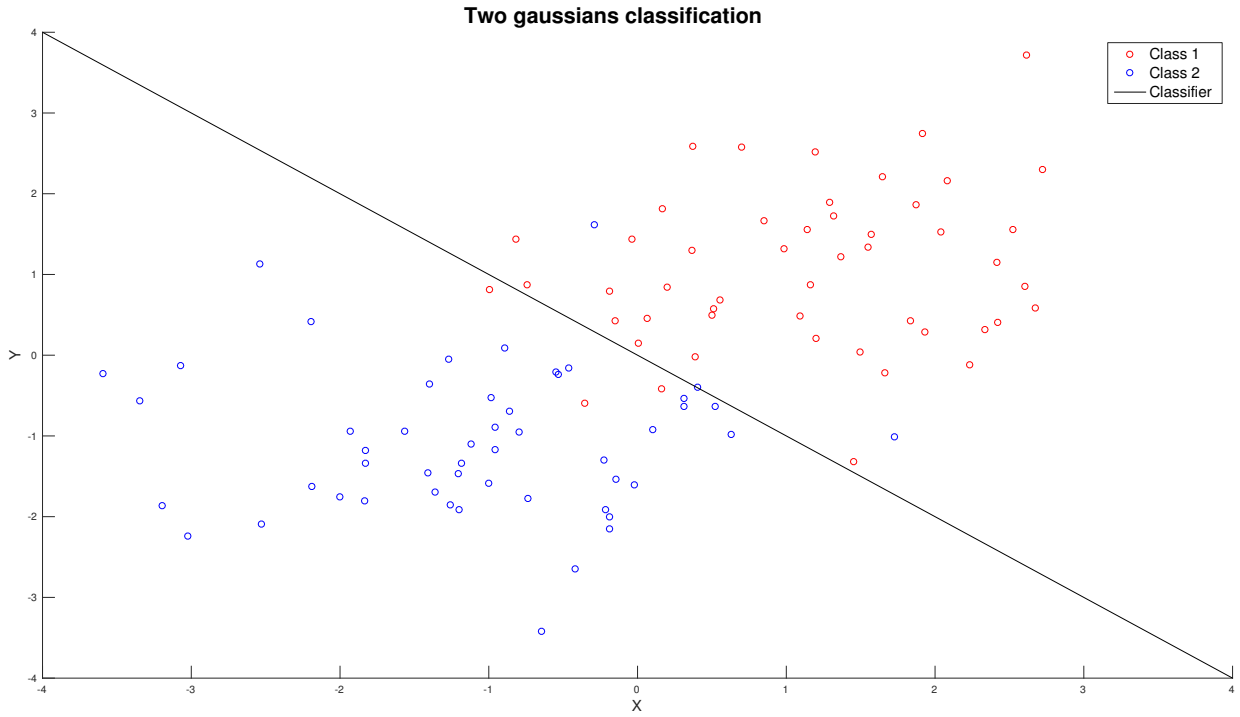
Contents

Context

The analysis presented in this report was produced for the class of "Support Vector Machines: methods and applications" (Spring 2016) at KU Leuven. The goal is to display understanding of the techniques and of their practical use. This first report focuses on classification using SVM, and Least-Squares SVM (LS-SVM). The implementation was done using the MatLab software (v2015a) and the libraries for LS-SVM developed at KU Leuven.

1 Two Gaussians

In this first application, an artificial dataset was generated consisting of 100 points in $\mathbb{R}^2$. Two centroids were defined to generate the points, one at $(1, 1)$ and the other at $(-1, -1)$. For both centers, 50 datapoints were generated by adding a gaussian noise $N(0, 1)$. Since we know the true underlying distribution for both classes, we can define an optimal classifier as per the Bayesian Decision Theory. This classifier, regardless of the overlap between the distributions of the 2 classes, will consist of a line in $\mathbb{R}^2$, and otherwise a hyperplane in higher dimensions.



2 Support Vector Machine

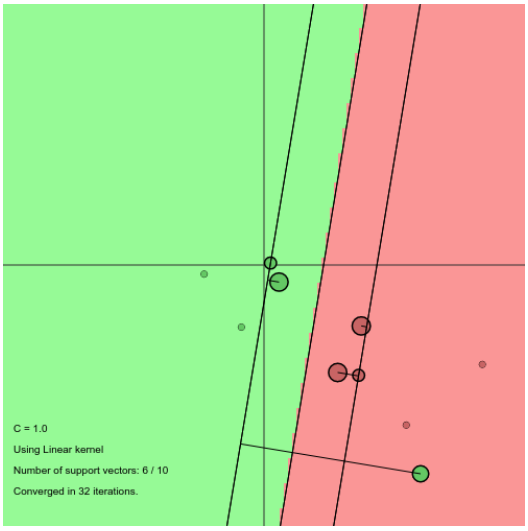
The exercises in this section consisted of exploring the properties of Support Vector Machines via a web application [1]

[1] http://cs.stanford.edu/people/karpathy/svmjs/demo/

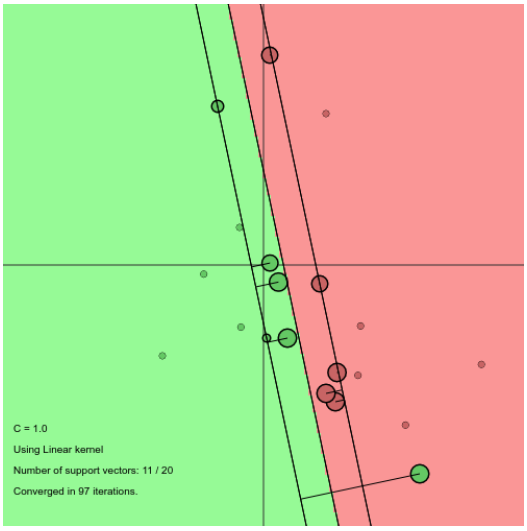
2.1 Linear kernel (questions 1 to 3)

The data with which we are presented in the application is not linearly separable. Adjusting to a linear kernel, we see that there are 5 points of each classes, a total of 6 support vectors, and 1 point (green) that is misclassified (see 1a). Depending on where the points are added, you can preserve the total number of misclassified points. Points added in the same class region, and close to other of the same class tend not to affect the classifier too much.

Misclassified points can change the classifier drastically, potentially switching the regions if one class of points outweighs the other. If the balance of points between the two classes shifts too much, the whole region can become classified as being of only 1 class.

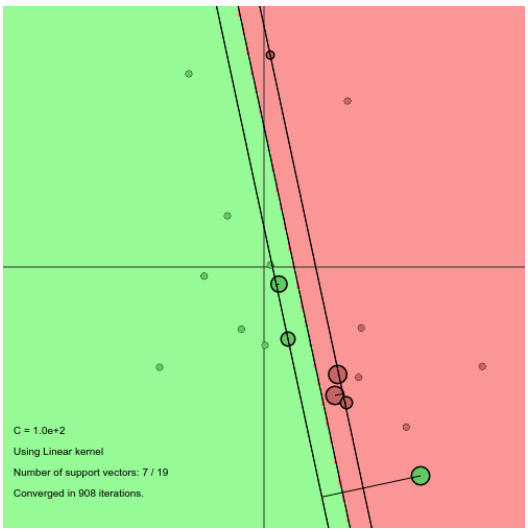


(a) Initial setting (linear kernel)

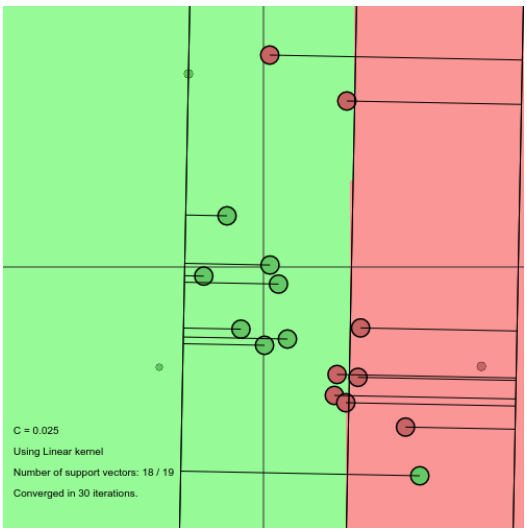


(b) After addition of 10 datapoints

For the same setting, varying the values of the hyperparameter for the regularization impacts the margins of the classifier. Eventually tolerating even more misclassification when C is extremely small.



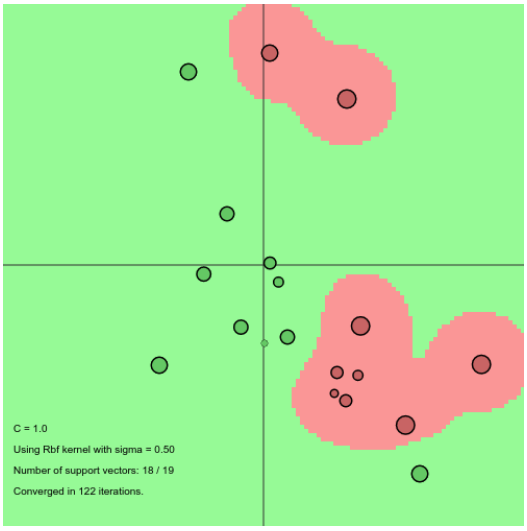
(a) Value of C: 100



(b) Value of C: 0.025

2.2 RBF kernel (questions 4 and 5)

In this section, we were asked to switch away from a linear kernel and opt for the Radial Basis Function kernel (RBF). RBF are localized function, centered at a particular value, and use a gaussian fit with a certain spread around the center.
The same dataset as defined above can be seen on the graphics on the right. Here, one can observe that all points have been correctly classified, but the area defined are very wobbly, displaying clear signs of overfitting.



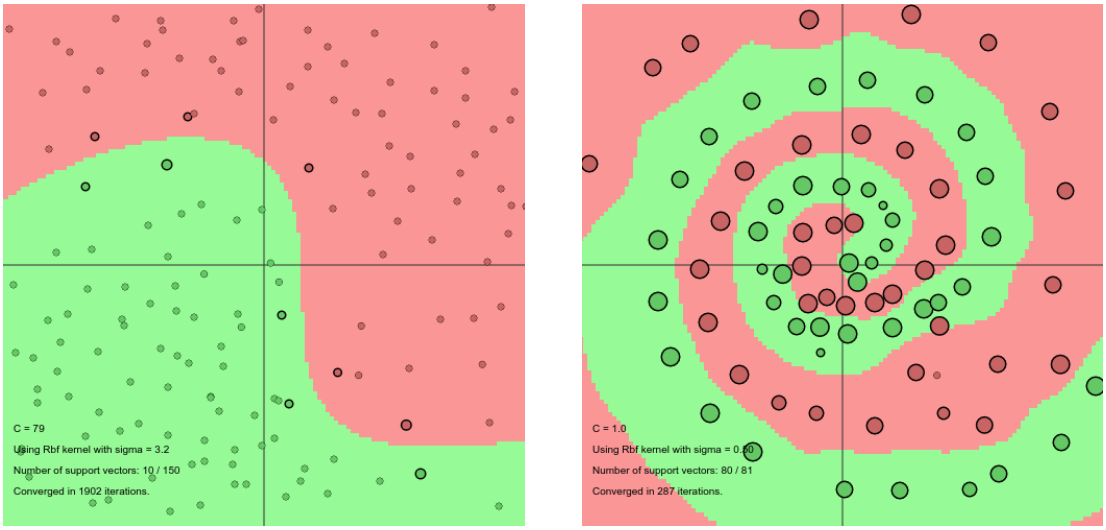
(a) RFB kernel

2.3 Kernel (question 6)

2.4 Support vectors (question 7 and 8)

Support vectors are vectors in the data space (p-dimensional) that support the maximal margin hyperplane. If those particular vectors are moved, then the resulting margin moves as well. Importantly, the classifier's hyperplane depends only on these support vectors, hence the classifier can be expressed using them. This can result in an advantage in terms of compression, as the dataset containing many vectors can be mostly thrown away, saving only those support vectors that are important for the classifier's prediction power.

In the figure 4a below, one can see such an effect of compression. Originally, 150 datapoints were present, but only 10 of them are necessary to construct a satisfying classifier. In this example, I played around with the parameters to obtain the reduced number of support vectors. In figure 4b however, the entire dataset consists of the support vectors. It is not possible to reduce the amount of support vectors by tuning the parameters without impacting the boundaries of the classifier.



(a) Support vectors reduce dataset *(b) No compression*

Figure 4: Support vectors

3 Least-Squares Support Vector Machine

3.1 Iris dataset

3.1.1 LS-SVM sample script exploration

For this section, I explored the iris dataset using the LS-SVM library developed at KU Leuven. The dataset contains 2 distinct classes that are not linearly separable.

The figure 6 below summarizes visually the classifiers built using linear and polynomial kernels for the classification task. For the polynomial kernel, I used degree 2, 3, and 4. When using a polynomial of degree 1, I obviously get exactly the same results as when using a linear kernel (comforting though). In the table 1, you can see the error rate on the validation set dropping as the flexibility of our model is raised. The very wobbly region defined by the kernel of degree 4 seem to display signs of overfitting as the region in the top-rigth corner.

Since the dataset consists of regions that are not linearly separable, the linear kernel performs terribly.

It is not obvious to me that there is a connection between the σ parameter of the RBF kernel, which controls the spread around the center in the gaussian, and the degree of the polynomial kernel. I would expect that overfitting would occur in tandem as you lower the value of *sigma*, and increase the degree of the polynomial.

	# Missclassified	% Error
Linear kernel	11	55%
Polynomial degree 2	5	5%
Polynomial degree 3	0	0%
Polynomial degree 4	0	0%

Table 1: Missclassification summary (polynomial kernel)

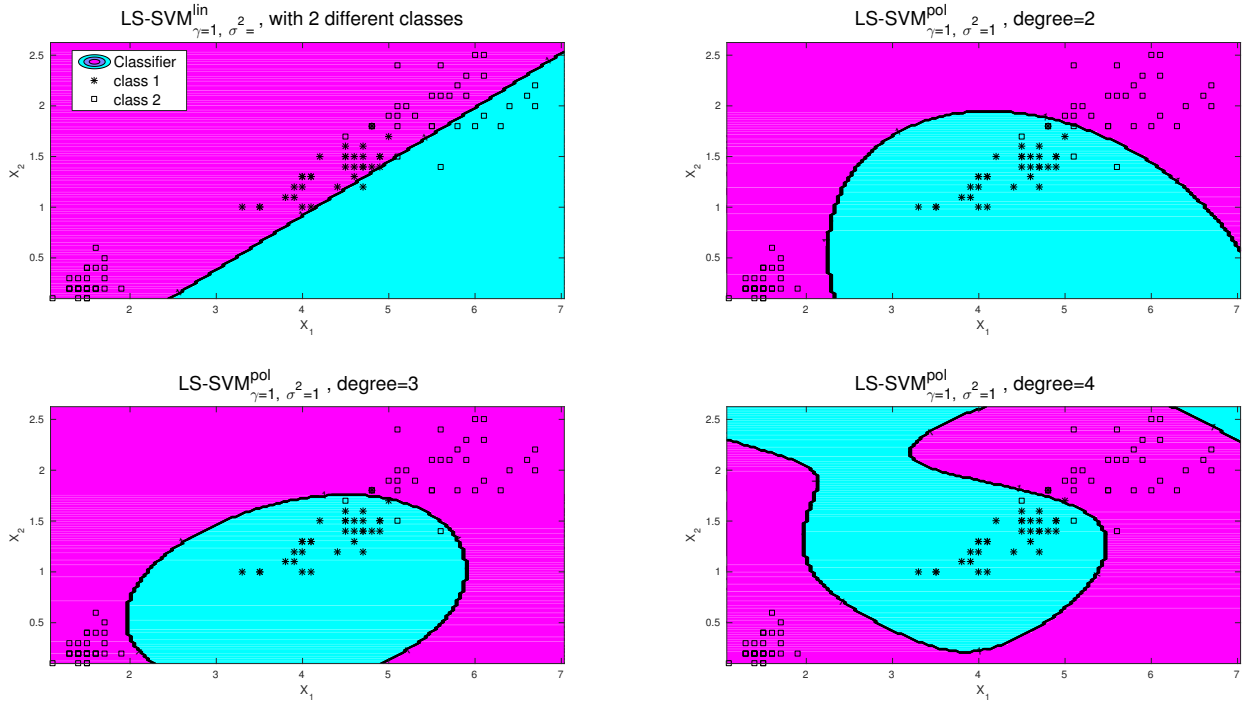


Figure 5: Linear and polynomial kernel classifiers

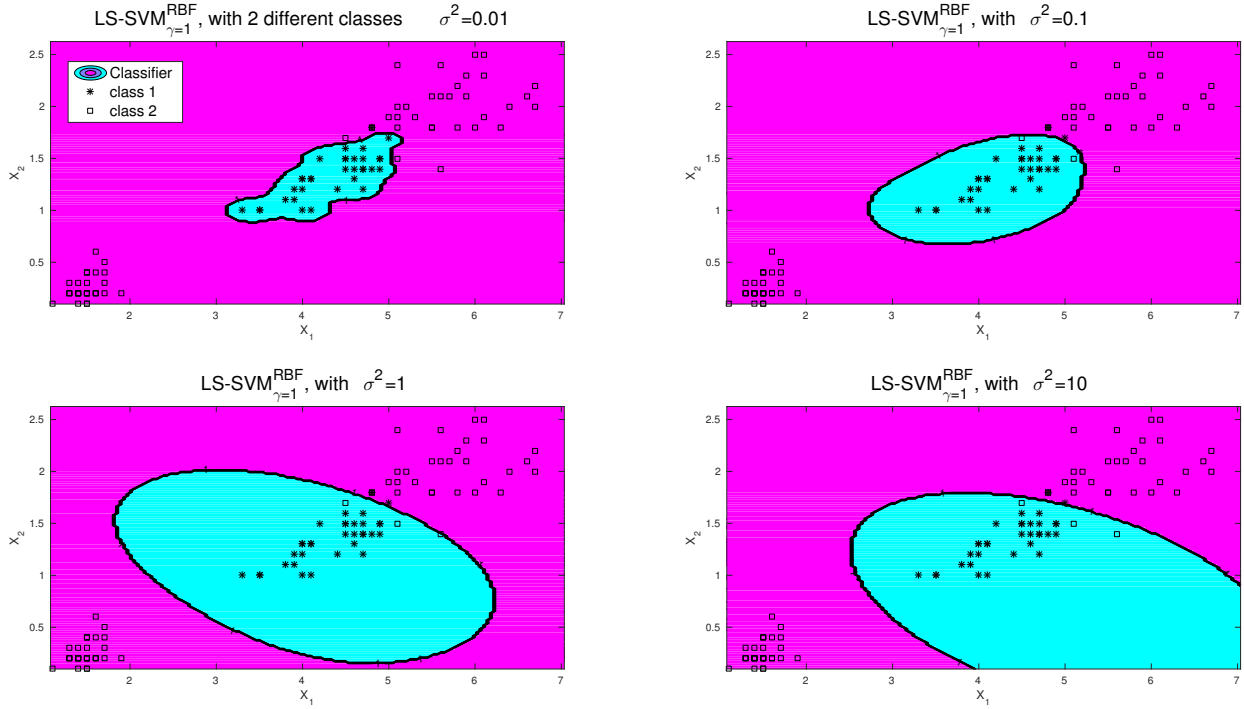


Figure 6: RBF classifier with different values of sigma

This last figure 7 illustrate a bit one of the extremes of the bias-variance tradeoff and overfitting. As seen in the previous graphs, with less flexible models, and especially in this case with a linear model (high bias), you run the risk of not capturing the pattern in your dataset correctly and get terrible results on the validation set. On the other end of the

	# Missclassified	% Error
RBF $\gamma = 1$, $\sigma = 0.01$	2	10%
RBF $\gamma = 1$, $\sigma = 0.1$	0	0%
RBF $\gamma = 1$, $\sigma = 1$	0	0%
RBF $\gamma = 1$, $\sigma = 10$	0	0%

Table 2: Missclassification summary (RBF kernel)

spectrum, with very flexible models, your variance increases too much and although the performance on the training set are close to perfection, the results on the validation set are off **??**. Typically, you have *memorized* your dataset, but fail to obtain good generalization.

	# Missclassified	% Error
RBF $\gamma = 1$, $\sigma = 0.001$	6	30%
RBF $\gamma = 1$, $\sigma = 0.01$	2	10%
Polynomial degree 15	1	5%
Polynomial degree 20	4	20%

Table 3: Missclassification summary (Overfitting)

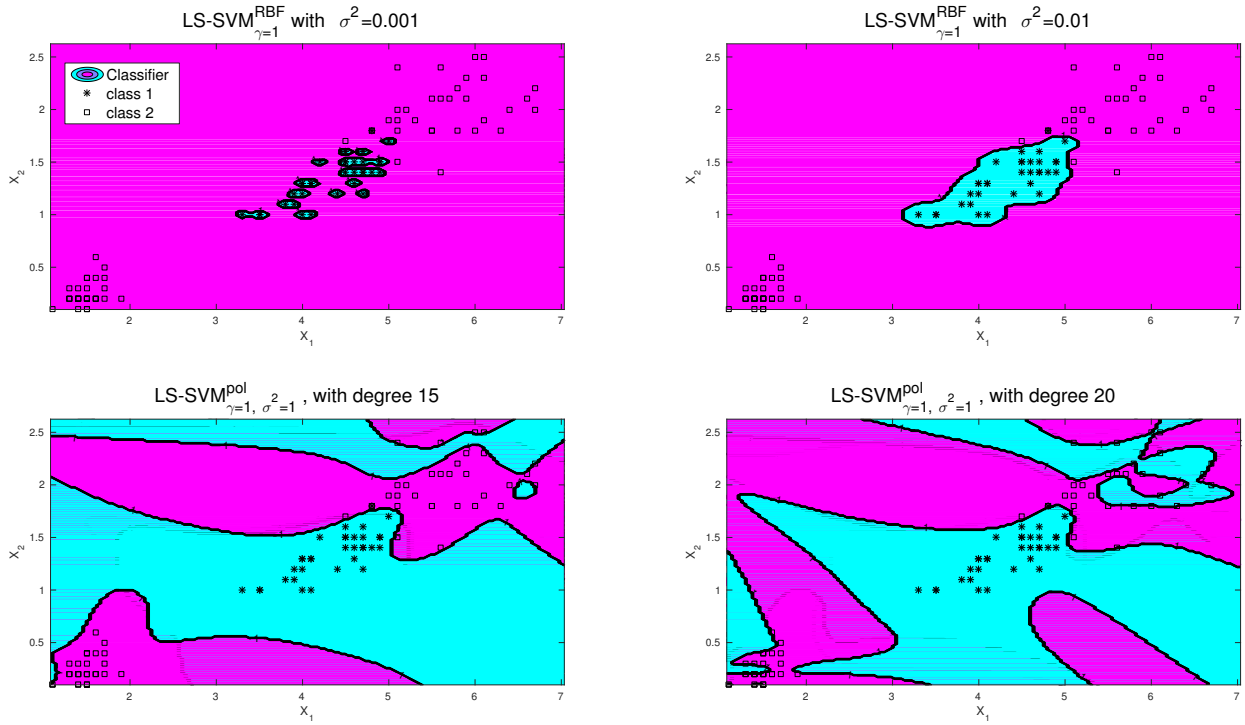


Figure 7: Overfitting with polynomial and RBF kernels

3.1.2 RBF kernel: sigma

In this section, we explore further the effect of the bandwidth parameter σ^2 for the RBF kernel and its effect on generalization as measured on a test set.

Below is the corresponding matlab script and figure 8 used to establish the interval out of which to pick a reasonable σ^2. The idea here is to fix the value of γ to 1, and establish a

logarithmic grid of possible values for σ^2, systematically building models using the training set and evaluating the performance on the test set.

This allows us to define an interval for the choice of the parameter: $\sigma^2 \in [0.1, 10]$

```
gam = 1; type='c'; sig2list=logspace(-3,3,60); errlist=[];

for sig2=sig2list,
    [alpha,b] = trainlssvm({X,Y,type,gam,sig2,'RBF_kernel'});
    % Obtain classification of test set using trained classifier
    [Yht, Zt] = simlssvm({X,Y,type,gam,sig2,'RBF_kernel'}, {alpha,b}, Xt);
    err = sum(Yht≠Yt); errlist = [errlist; err];
end

figure('Color',[1 1 1]);
semilogx(sig2list, errlist./20.*100, 'b-');
title('Test missclassification vs. Sigma');
xlabel('Sigma (log_{10})'); ylabel('Missclassification %');
```

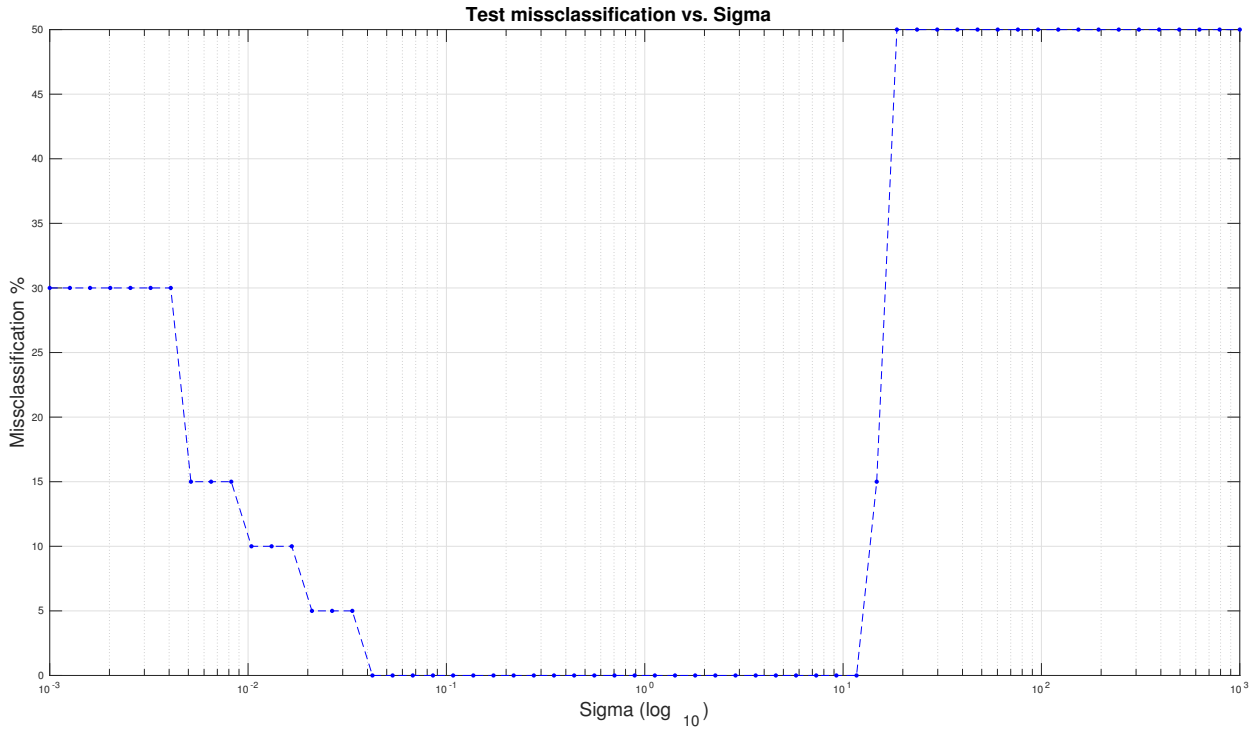


Figure 8: Tuning of parameter σ^2 for fixed $\gamma = 1$

3.1.3 RBF kernel: regularization constant

In this section, we explore further the effect of the tuning parameter γ, used for regularization, and its effect on generalization as measured on a test set.

The procedure to select the value of the hyperparameter is the same as described in the previous section. The result indicates to chose $\gamma \in [0.15, 80]$

```
sig2 = 0.1; gamlist=logspace(-3,3,60); type='c'; errlist=[];

for gam=gamlist,
```

```
    [alpha,b] = trainlssvm({X,Y,type,gam,sig2,'RBF_kernel'});
    % Obtain classification of test set using trained classifier
    [Yht, Zt] = simlssvm({X,Y,type,gam,sig2,'RBF_kernel'}, {alpha,b}, Xt);
    err = sum(Yht≠Yt); errlist = [errlist; err];
end

figure('Color',[1 1 1]);
semilogx(gamlist, errlist./20.*100, 'b-');
title('Test missclassification vs. Gamma (fixed \sigma^2=0.1)');
xlabel('\gamma (log_{10})'); ylabel('Missclassification %');
```

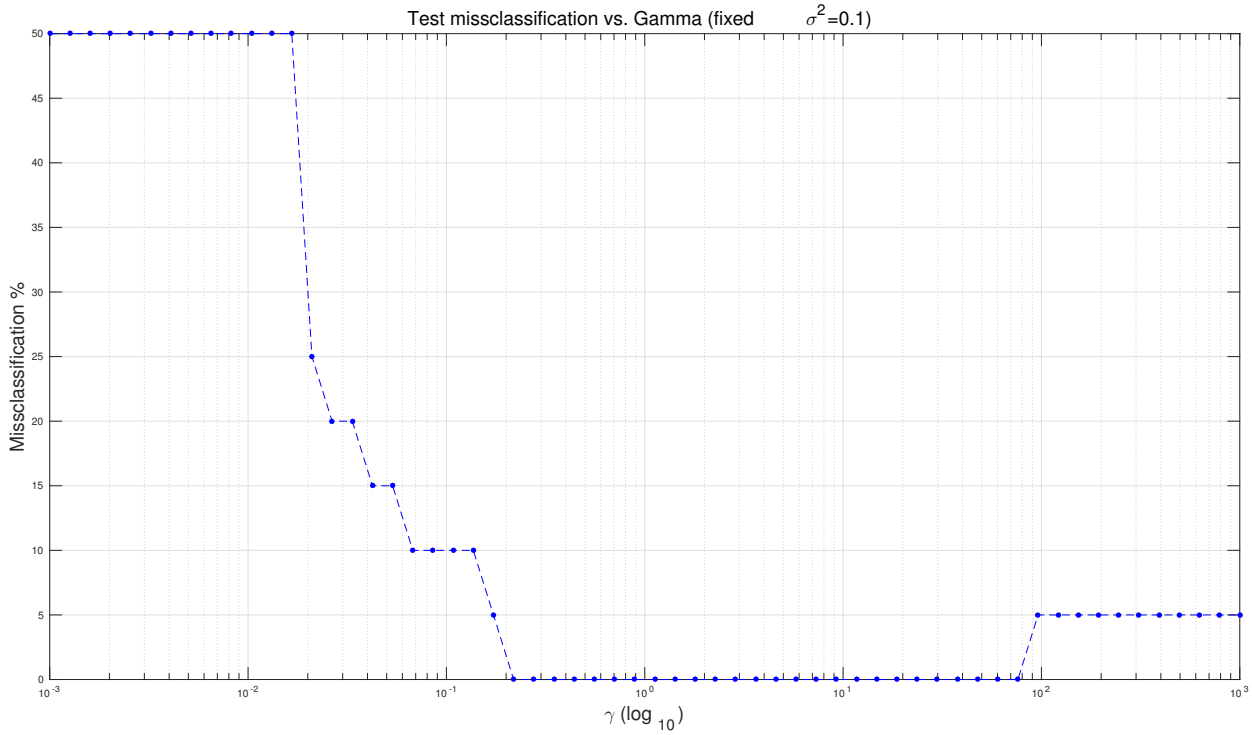


Figure 9: Tuning of parameter γ for fixed $\sigma^2 = 0.1$

3.2 Choice of hyper-parameters

3.2.1 Validation set

For this section, I created a validation set for the Iris dataset consisting of a fifth of the total number of observations used in the previous section to train the classifier. Here is the code for that purpose:

```
idx = randperm(size(X,1));
Xtrain = X(idx(1:80),:);
Ytrain = Y(idx(1:80),:);
Xval = X(idx(81:100),:);
Yval = Y(idx(81:100),:);
```

In order to choose appropriate values for the hyper parameters, I trained classifiers using a log grid of σ and γ, and computed the test error on the validation dataset as follow:

```
% Searching sigma space for a good value (good generalization on test set)
```

```
sig2list=logspace(-3,3,60); errsiglist=[]; gam = 1;
for sig2=sig2list,
    [alpha,b] = trainlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'});
    % Obtain classification of test set using trained classifier
    estYval = simlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, ...
        Xval);
    err = sum(estYval≠Yval); errsiglist = [errsiglist; err];
end

% Searching sigma space for a good value (good generalization on test set)
sig2 = 0.1; gamlist=logspace(-3,3,60); type='c'; errgamlist=[];
for gam=gamlist,
    [alpha,b] = trainlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'});
    % Obtain classification of test set using trained classifier
    estYval = simlssvm({Xtrain,Ytrain,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, ...
        Xval);
    err = sum(estYval≠Yval); errgamlist = [errgamlist; err];
end
```

The figure 10 gives an overview of the misclassification rate for different values of the hyperparameters. One can observe that the "optimal" range in which to select the parameters if slightly different than that of the previous section. Since we have defined a different set to test the performance and given that we have few observations (<100), we can expect variance in the evaluation of test error depending on the split of the dataset between training and validation.

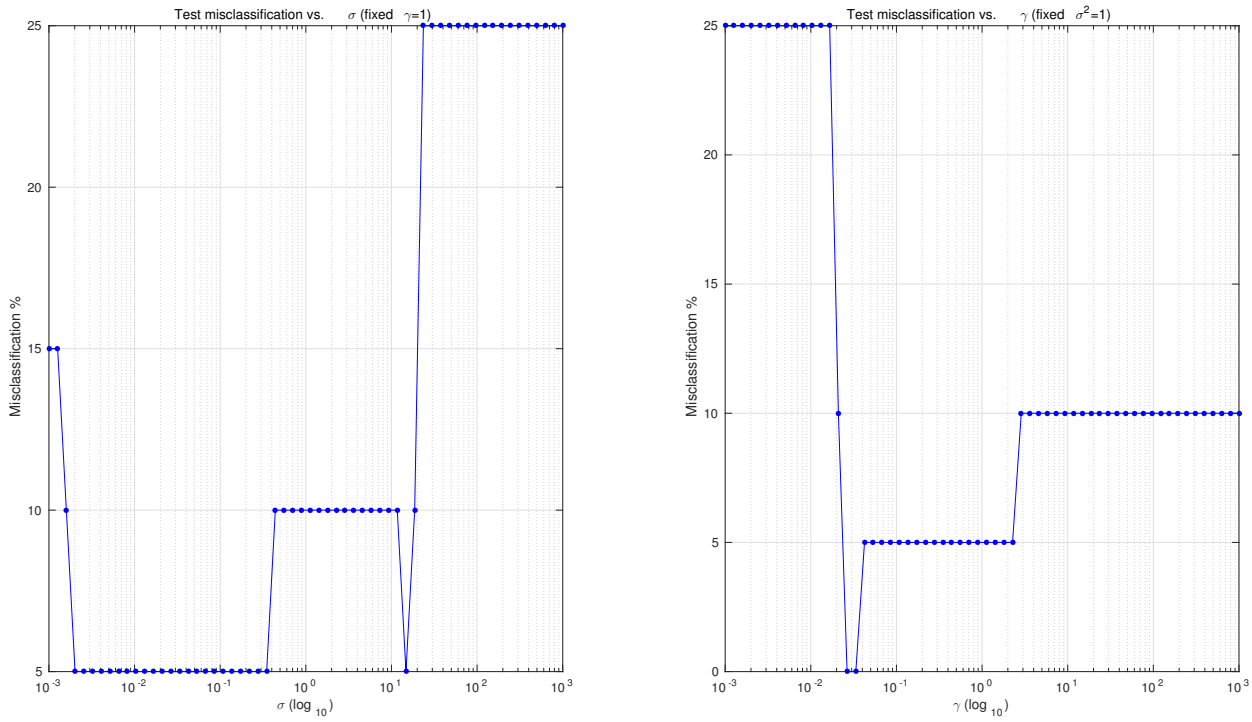


Figure 10: Tuning of parameter γ and σ^2 based on the validation set

3.2.2 Cross validation

Using cross validation is a common technique in statistics to help remove the variance due to the random split of the dataset between training set and validation set. In smaller datasets, high variance can ensue from validating a model given one split, or another split. By

systematically building multiple splits, training and validating the models, one can obtain a more average test error.

In this section, we used 10 fold cross validation, which consists in creating 10 different splits of the original dataset. For each split, 90 percent of the data is used to train a model, and 10 percent is used for validation. At each new split, there is no overlap of the test cases with previous test cases, so the whole dataset is surveyed. The test errors are averaged accross the 10 splits to obtain the average test error on the whole dataset.

For the LOOCV, the procedure is the same, but only a single datapoint is kept at each split to test the model. The procedure is thus repeated n times (n being the number of observations).

These are the results for the Iris dataset:

```
>> crossvalidate({X, Y, 'c', gam, sig2, 'RBF_kernel'}, 10, 'misclass');
    0.0600

>> leaveoneout({X, Y, 'c', gam, sig2, 'RBF_kernel'}, 'misclass');
    0.0500
```

The LOOCV can be very computationally expensive on large datasets, though there exists exceptions to that (eg, in the case of linear regression, one can compute the LOOCV using a simple analytical formula). The LOOCV is also known to have more bias since it uses systematically "almost" the whole dataset. It can however be useful on datasets containing very few observations.

3.2.3 Optimization techniques

In this section, we try to optimize the hyperparameters using global optimization techniques. The results are reported in 4. One thing to note is the stochasticity of the search process. Accross multiple runs (3 illustrated in the table), one obtains widely variable results in terms of the selected hyperparameters. The hyperparameter σ^2 however falls into the range previously established. γ is a bit higher in magnitude than expected using the previous approach.

The combination of all optimization approaches seem to converge rapidly to a minimum cost.

```
% coupled simulated annealing
model_csa = {X, Y, 'c', [], [], 'RBF_kernel', 'csa'};
[gam1, sig21, cost1] = tunelssvm(model_csa, 'simplex', 'crossvalidatelssvm', ...
    {10, 'misclass'});
[gam2, sig22, cost2] = tunelssvm(model_csa, 'gridsearch', ...
    'crossvalidatelssvm', {10, 'misclass'});
% randomized directional search
model_ds = {X, Y, 'c', [], [], 'RBF_kernel', 'ds'};
[gam3, sig23, cost3] = tunelssvm(model_ds, 'simplex', 'crossvalidatelssvm', ...
    {10, 'misclass'});
[gam4, sig24, cost4] = tunelssvm(model_ds, 'gridsearch', ...
    'crossvalidatelssvm', {10, 'misclass'});
```

		Run 1			Run 2			Run 3		
		γ	σ^2	Cost	γ	σ^2	Cost	γ	σ^2	Cost
CSA	simplex	2.14	1.32	0.04	3670	0.06	0.03	123.84	0.03	0.03
	gridsearch	176.24	0.02	0.03	0.65	0.09	0.04	0.11	0.73	0.04
DS	simplex	0.93	6.66	0.04	7.60	2.37	0.05	4.54	1.27	0.05
	gridsearch	0.05	0.93	0.03	41.96	0.21	0.03	0.05	0.21	0.03

Table 4: Hyperparameters tuning

3.2.4 ROC

Receiver Operating Characteristics are a very popular tool to assess the quality of a classifier in terms of predictive power. They can be thought of as continuously evaluated (by changing the decision treshold) contingency tables.

One should not build ROC using the training dataset for the same reason that you don't assess the quality of a model based on the training dataset. Many techniques systematically minimize the error between the model and the training observations (eg, least mean squares). Thus the ROC curve can be made really good by just "memorizing" the dataset, without any regard for good generalization.

4 Applications

4.1 Ripley dataset

4.1.1 Data visualization

A scatterplot of the training dataset is shown on figure 11. The 2 categories seem to be overlapping quite a bit, and are separated (though not clearly) on the vertical axis, with the upper part of the scatterplot consisting mostly of one category (here "1"), and the bottom part of the scatterplot, with the other category (here "-1"). Further, it would seem that the "-1" category has 2 centroid, one around X=-0.75, and the other around X=0.4. The same observation can be made for the category "1", though the separation between the 2 centroids is less pronouced.

Intuitively, it would seem that a linear classifier may do a good job at classifying observations that are further appart to the Y=0.5 axis, but will perform poorly close to that region due to considerable overlap.

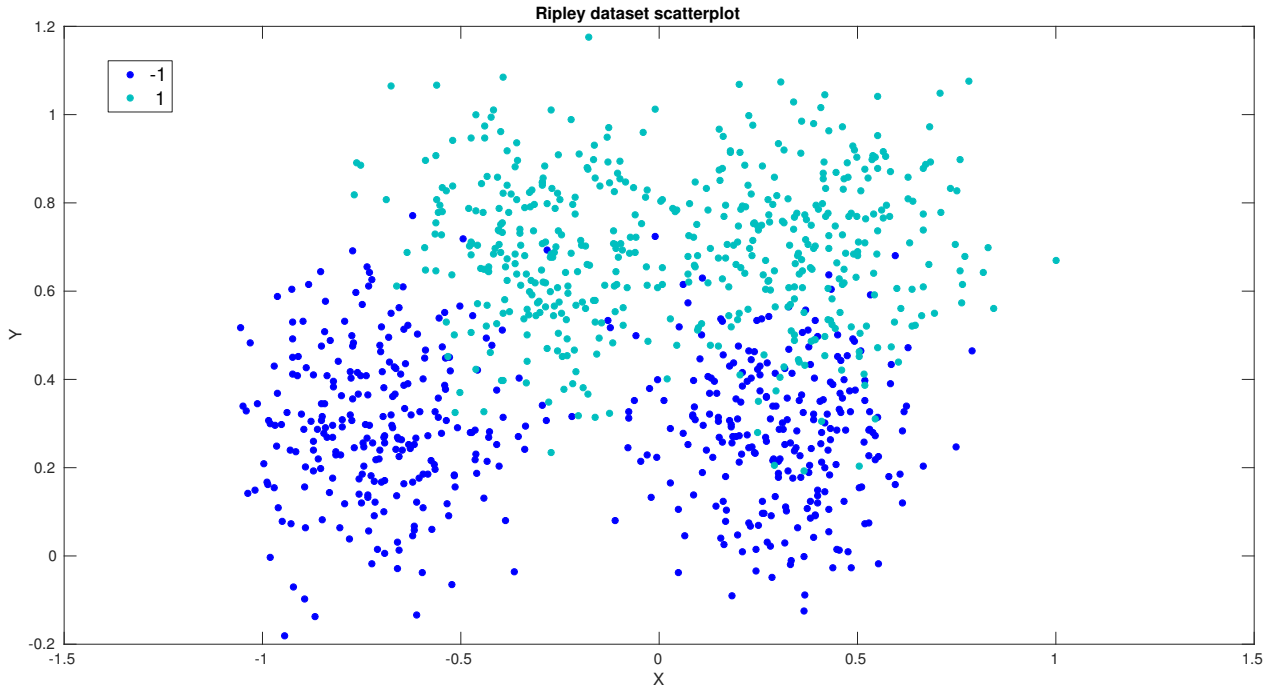


Figure 11: Ripley dataset scatterplot by group

4.1.2 Linear model

In order to test the intuition about the linear separability, I built a LS-SVM model that has a polynomial kernel of degree 1. The resulting classifier can be seen on the left-hand side of figure 12

```
gamlist=logspace(-4,3); type='c'; errlist=[];

for gam=gamlist,
    [alpha,b] = trainlssvm({Xt,Yt,type,gam,[],'lin_kernel'});
    [Yht, Zt] = simlssvm({Xt,Yt,type,gam,[],'lin_kernel'}, {alpha,b}, X);
    err = sum(Yht≠Y); errlist = [errlist; err];
end
[min, index] = min(errlist);
plotlssvm({Xt,Yt,type,gamlist(index),[],'lin_kernel','preprocess'},{alpha,b});
```

4.1.3 RBF model

The resulting classifier can be seen on the right-hand side of figure 12

```
model_csa = {Xt, Yt, 'c', [], [], 'RBF_kernel', 'csa'};
[gam, sig2, cost] = tunelssvm(model_csa, 'simplex', 'crossvalidatelssvm', ...
    {10, 'misclass'});

[alpha,b] = trainlssvm({Xt,Yt,'c',gam,sig2,'RBF_kernel'});
estYval = simlssvm({Xt,Yt,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, X);
err = sum(estYval≠Y);
plotlssvm({Xt,Yt,type,gam,sig2,'RBF_kernel','preprocess'},{alpha,b});
```

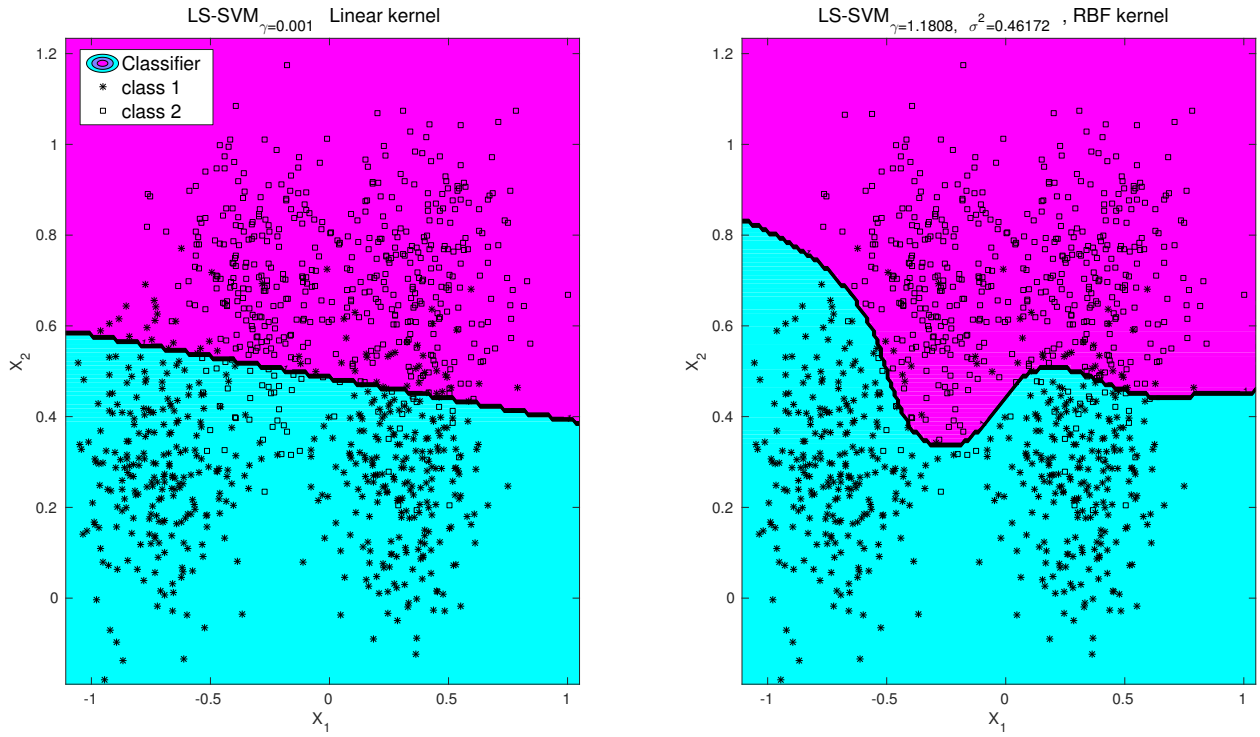


Figure 12: Ripley dataset, left linear kernel, right RBF kernel

4.1.4 ROC curves

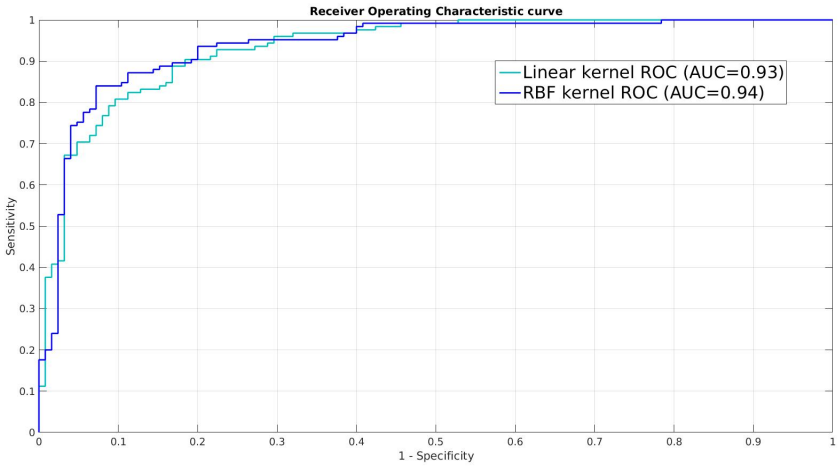


Figure 13: ROC of both classifiers

4.1.5 Final model selection

There really isn't much support for and RBF kernel based modelling for this dataset. As illustrated by the ROC curves, the performance on the validation set of the RBF classifier is only marginally superior.

The Ripley dataset consists of overlapping gaussian distribution, so using a simpler model that incorporates a linear classifier makes sense here.

4.2 Breast cancer dataset

4.2.1 Data visualization

This dataset contains significantly more variables than the previous one and cannot be easily plotted for interpretation. A quick boxplot of the 30 variables reveals a lot of variance in the scale, so I performed dimensionality reduction using Principal component analysis with scaling using variance. The results are shown on figure 14. From the importance of principal component plot, one can see that the first 3 PC explain a large amount of the variability in the data. Linear separability seems to appear when plotting the data using 3PC and 2PC.

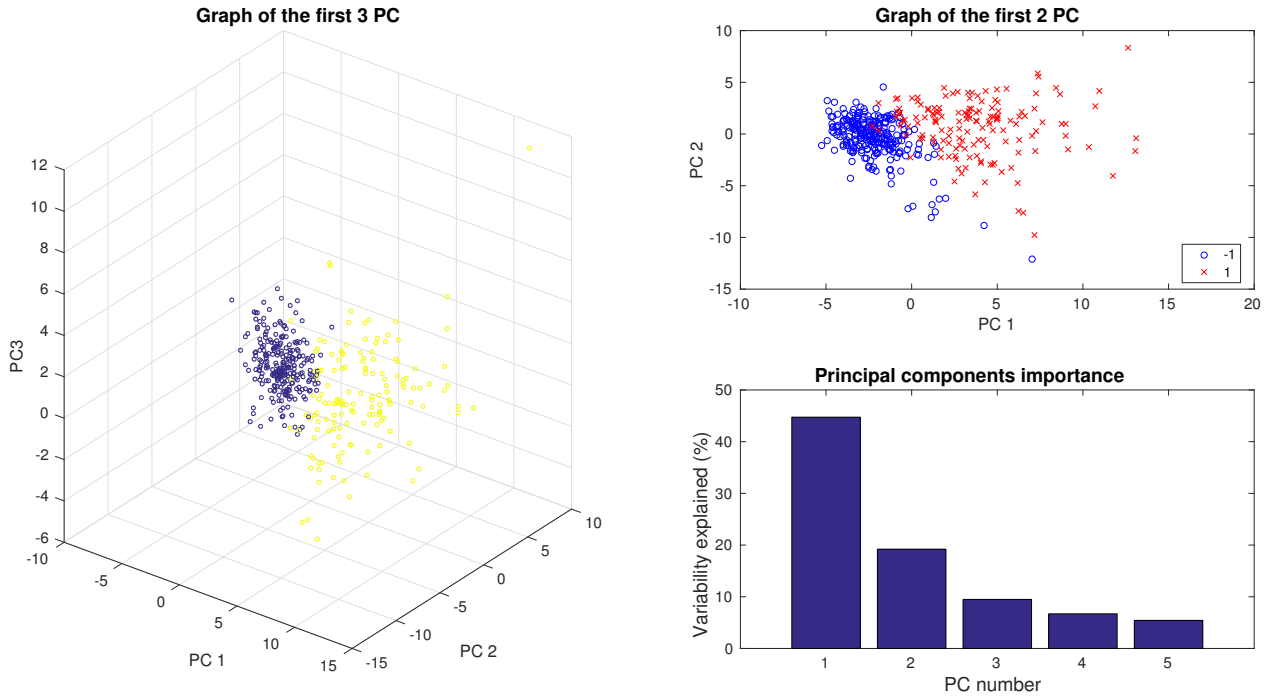


Figure 14: Principal components analysis and visualization

4.2.2 Linear model

The reported performance of the classification for the test set is: 96.4%

```
gamlist=logspace(-3,3); type='c'; errlist=[];

for gam=gamlist,
    [alpha,b] = trainlssvm({trainset,labels_train,type,gam,[],'lin_kernel'});
    [Yht, Ylin] = simlssvm({trainset,labels_train,type,gam,[],'lin_kernel'}, ...
        {alpha,b}, testset);
    err = sum(Yht≠labels_test); errlist = [errlist; err];
end
[min, index] = min(errlist);
[Yht, Ylin] = ...
    simlssvm({trainset,labels_train,type,gamlist(index),[],'lin_kernel'}, ...
    {alpha,b}, testset);
perf_lin = (1-errlist(index)/numel(labels_test))*100;
```

4.2.3 RBF model

The reported performance of the classification for the test set is: 97.6%

```
model_csa = {trainset, labels_train, 'c', [], [], 'RBF_kernel', 'csa'};
[gam, sig2, cost] = tunelssvm(model_csa, 'simplex', 'crossvalidatelssvm', ...
    {10, 'misclass'});

[alpha,b] = trainlssvm({trainset,labels_train,'c',gam,sig2,'RBF_kernel'});
[estYval, YRBF] = ...
    simlssvm({trainset,labels_train,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, ...
    testset);
err = sum(estYval≠labels_test); perf_rbf = (1-err/numel(labels_test))*100;
```

4.2.4 ROC curves

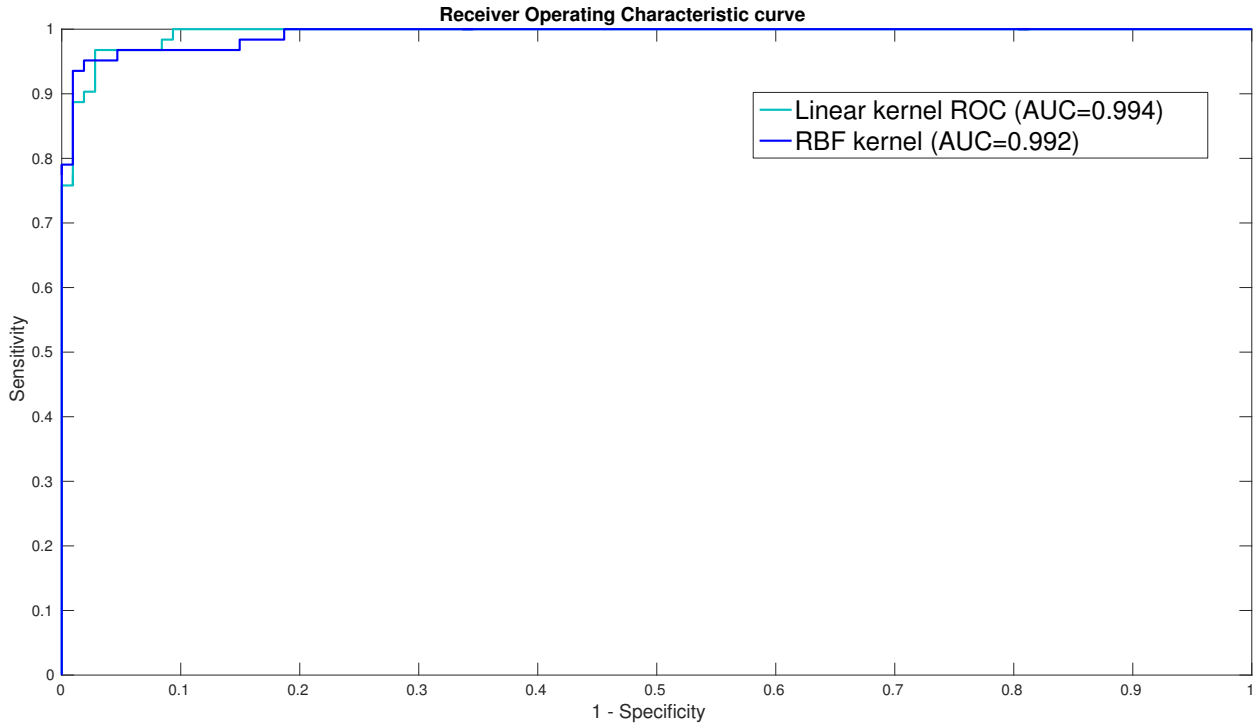


Figure 15: ROC of both classifiers

4.2.5 Final model selection

Similarly to section 4.1.5, we see that isn't much support for and RBF kernel based modelling for this dataset. As illustrated by the ROC curves, the performance on the validation set of the RBF classifier is only marginally superior.

This analysis is comforted by the fact that visually one can see a linear separability in the principal components plot.

4.3 Diabetes database

4.3.1 Data visualization

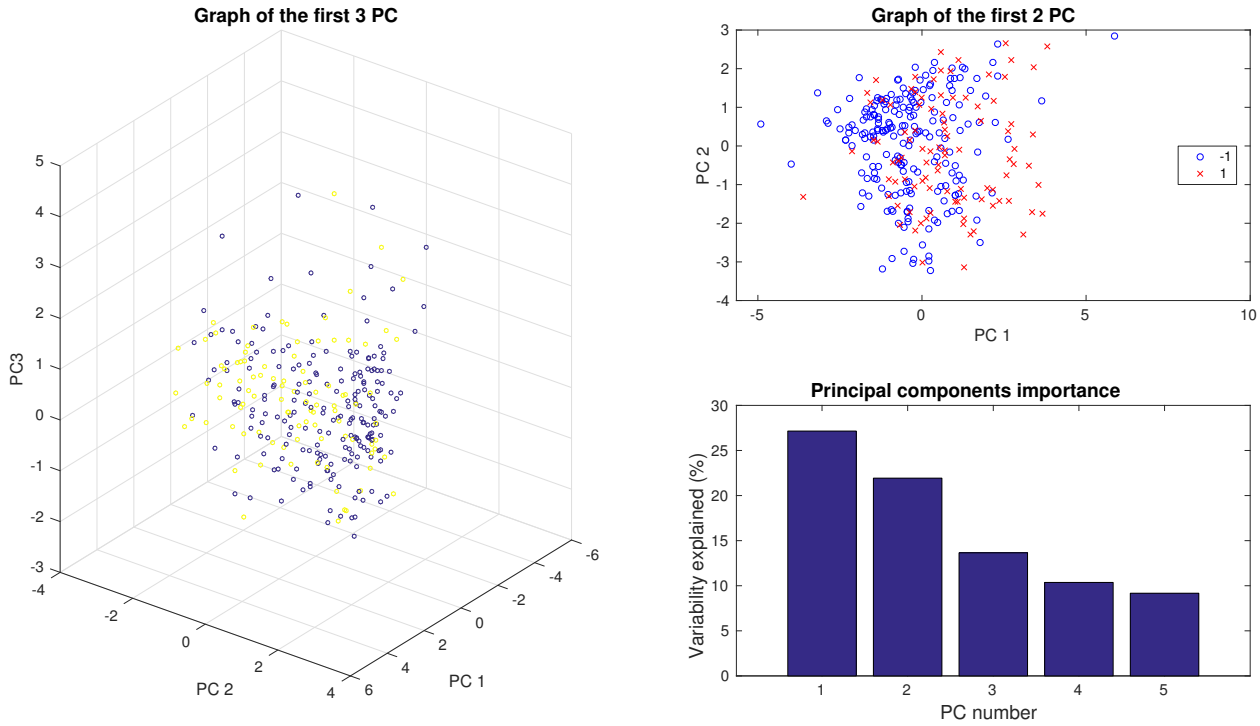


Figure 16: Principal components visualization

4.3.2 Linear model

The reported performance of the classification for the test set is: 79.8%

```
gamlist=logspace(-3,3); type='c'; errlist=[];

for gam=gamlist,
    [alpha,b] = trainlssvm({trainset,labels_train,type,gam,[],'lin_kernel'});
    [Yht, Ylin] = simlssvm({trainset,labels_train,type,gam,[],'lin_kernel'}, ...
        {alpha,b}, testset);
    err = sum(Yht≠labels_test); errlist = [errlist; err];
end
[min, index] = min(errlist);
[Yht, Ylin] = ...
    simlssvm({trainset,labels_train,type,gamlist(index),[],'lin_kernel'}, ...
    {alpha,b}, testset);
perf_lin = (1-errlist(index)/numel(labels_test))*100;
```

4.3.3 RBF model

The reported performance of the classification for the test set is: 76.1%

```
model_csa = {trainset, labels_train, 'c', [], [], 'RBF_kernel', 'csa'};
[gam, sig2, cost] = tunelssvm(model_csa, 'simplex', 'crossvalidatelssvm', ...
    {10, 'misclass'});
```

```
3
4 [alpha,b] = trainlssvm({trainset,labels_train,'c',gam,sig2,'RBF_kernel'});
5 [estYval, YRBF] = ...
      simlssvm({trainset,labels_train,'c',gam,sig2,'RBF_kernel'}, {alpha,b}, ...
      testset);
6 err = sum(estYval≠labels_test); perf_rbf = (1-err/numel(labels_test))*100;
```

4.3.4 ROC curves

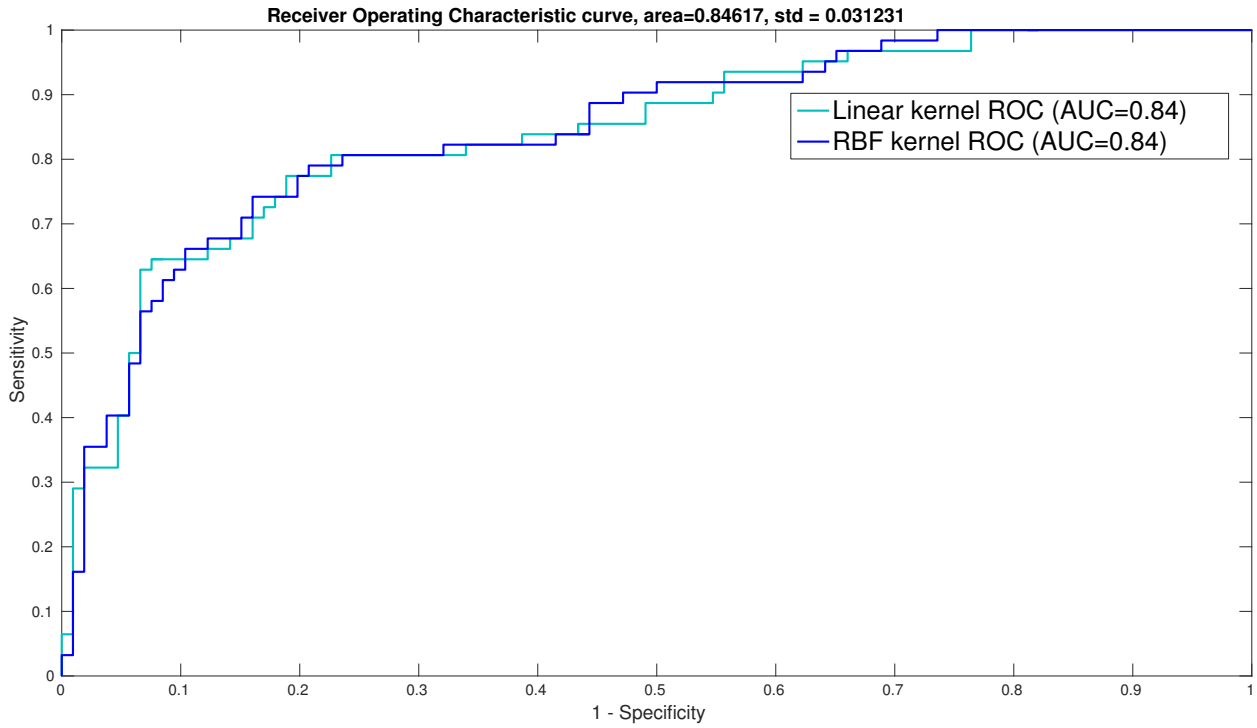


Figure 17: ROC of both classifiers

4.3.5 Final model selection

For this dataset, the performances are slightly worse than we had for the previous ones. I couldn't see evidences of linear separability in the PCA plot with 3 PC, but that does not exclude that one exists. The performance of both kernels are very similar, so I would advise to go with the simplest method, which is the linear kernel.

References