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 28, 2016

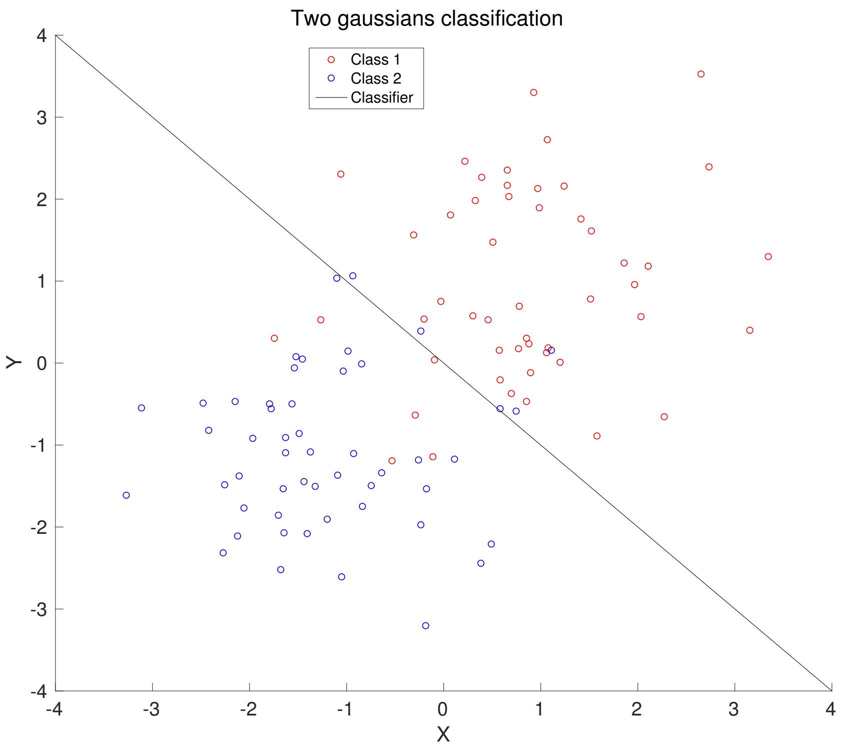
Contents

Context

The analysis presented in this report was produced for the class of “Support Vector Machines: methods and applications” at KU Leuven (Spring 2016). The goal is to display understanding of the principles behind support vector machines and of how to work out good solutions using these techniques. This first report focuses on classification of linearly separable, and non-linearly separable datasets using SVM, and Least-Squares SVM (LS-SVM). The implementation was done using the MatLab environment (v2015a) and the libraries for LS-SVM developed at KU Leuven [1].

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 using a gaussian noise $N(0, 1)$. Since we know the true underlying distribution for both classes (same covariance matrices), we can define a bayes optimal classifier whose decision boundary will consist in the line with equation $y = -x$. This classifier will be valid regardless of the overlap between the distributions of the 2 classes.



2 Support Vector Machine

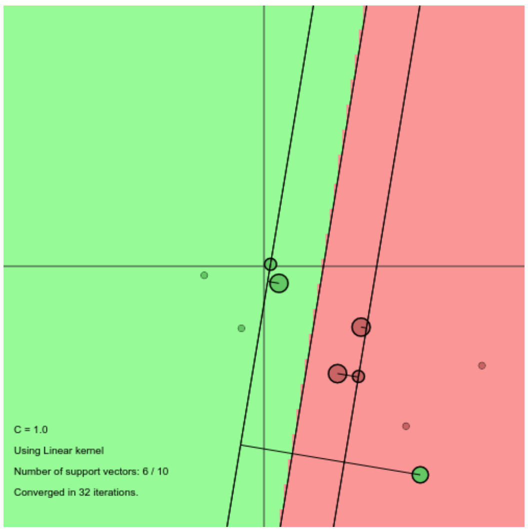
The exercises in this section consisted of exploring the properties of Support Vector Machines via a web application [2]

[1] http://www.esat.kuleuven.be/sista/lssvmlab/
[2] http://cs.stanford.edu/people/karpathy/svmjs/demo/

2.1 Linear kernel (subquestions 1 to 3)

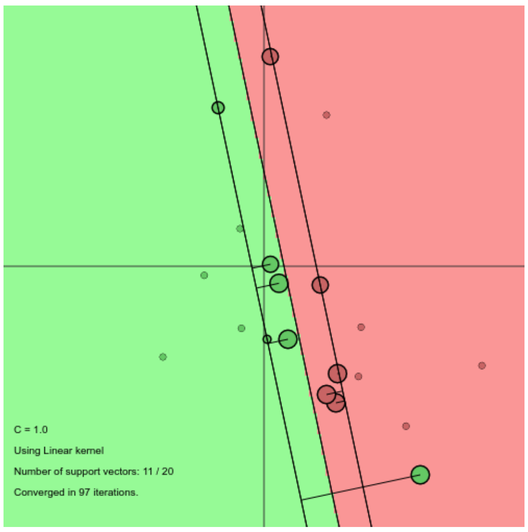
The data with which we are presented in the application is not linearly separable. Adjusting to a linear kernel, we see that there are 5 points of each classes, a total of 6 support vectors, 1 point (green) that is misclassified, and a couple of points inside the margin, indicating the need for slacking variables (see 1a). With the addition of 10 new points, depending on where the points are added, you can preserve the total number of misclassified points (1 green point - see 1b). In general, 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 region shown in the application frame can become classified as being of the over-represented class.



```
C = 1.0
Using Linear kernel
Number of support vectors: 6 / 10
Converged in 32 iterations.
```

(a) Initial setting (linear kernel)



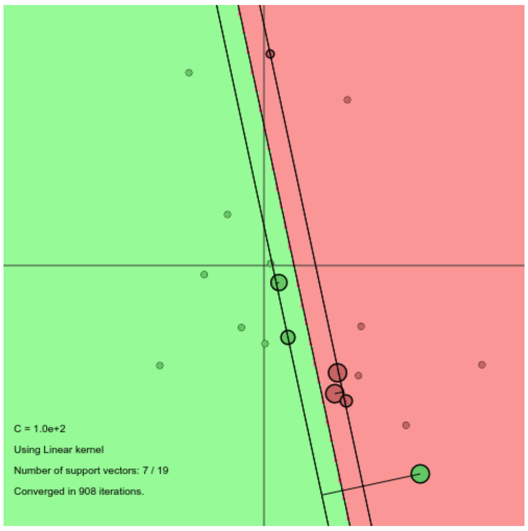
```
C = 1.0
Using Linear kernel
Number of support vectors: 11 / 20
Converged in 97 iterations.
```

(b) After addition of 10 datapoints

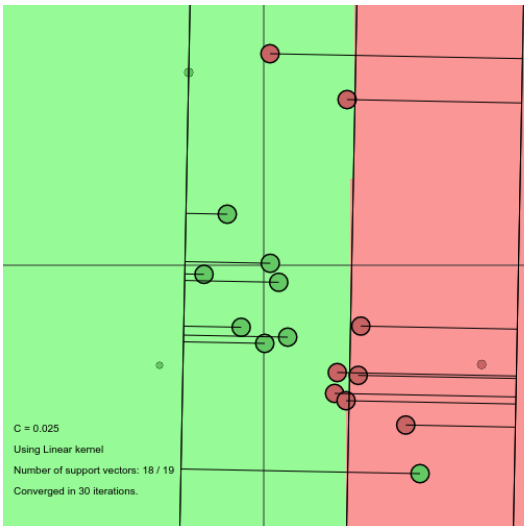
Figure 1: Linear kernel SVM with misclassication

For the same dataset, varying the values of the hyperparameter for the regularization impacts the margins of the classifier as shown on figure 2a and 2b.

The role of the C parameter is to control the balance in the SVM model between, on one hand the amount of slacking tolerated, and on the other hand the maximization of the margin. It is akin to a regularization hyperparameter, common in statistics (such as ridge regression). In short, a high value of C results in a smaller tolerance for datapoints on the wrong side of the margin and results in a margin with a small width, while a smaller value of C results in a larger margin width and a higher tolerance for datapoints on the wrong side of the margin.



(a) Value of C: 100

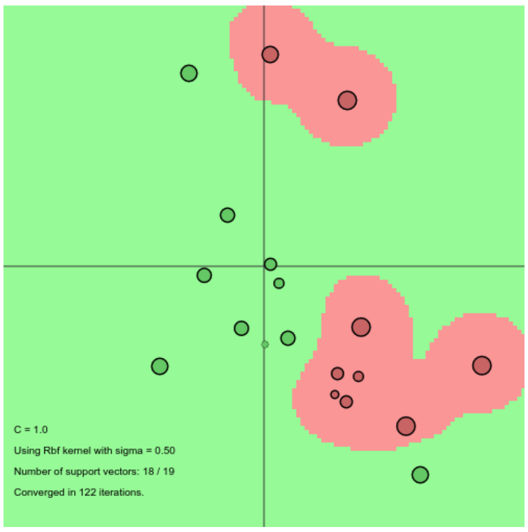


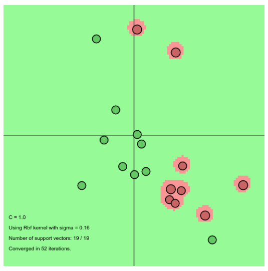
(b) Value of C: 0.025

Figure 2: Impact of regularization on the linear classifier boundaries

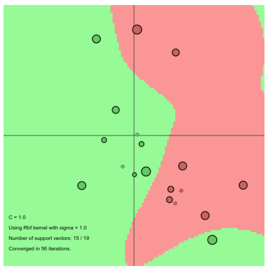
2.2 RBF kernel (subquestions 4 and 5)

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.
This can be tuned by changing the value of σ, which controls the bandwidth of the gaussian function, see figure 4. Larger values of σ leading to more linear decision boundaries.

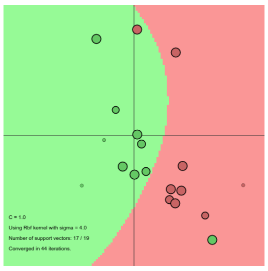




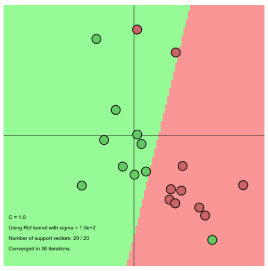
(a) Value of $\sigma = 0.16$



(b) Value of $\sigma = 1$



(c) Value of $\sigma = 4$



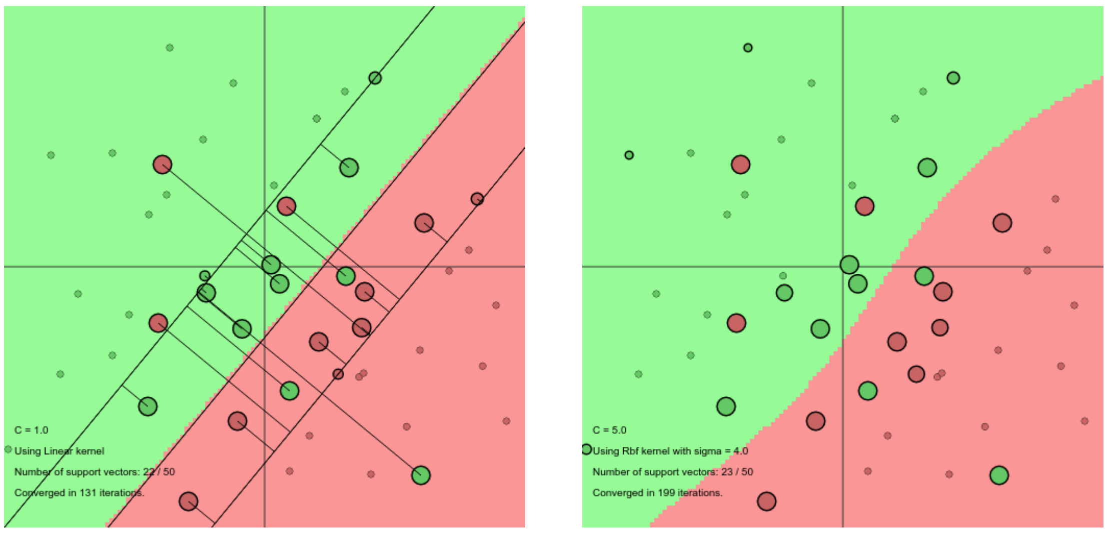
(d) Value of $\sigma = 100$

Figure 4: Impact of σ on the RBF classifier boundaries (C=1.0)

From the experimentation with the application, I would tend to think that in the case of an almost linearly separable dataset, a large value for σ is preferable, with a value for C of 1.0 (possibly optimized depending on the amount of misclassified points).

2.3 Kernel (subquestion 6)

For this question, I generated a balanced dataset of points (25 points for each class, hence 50 points in total), with the goal in mind of having a linear classifier work best. I was able to obtain a good definition of the decision surface using both the linear kernel, but also the RBF kernel after some tuning. However, the linear kernel was straightforward to use, as the C parameter did not have much impact except for extreme values. The minimum number of support vectors achievable in both cases was also similar.



(a) Linear classifier *(b) RBF kernel classifier*

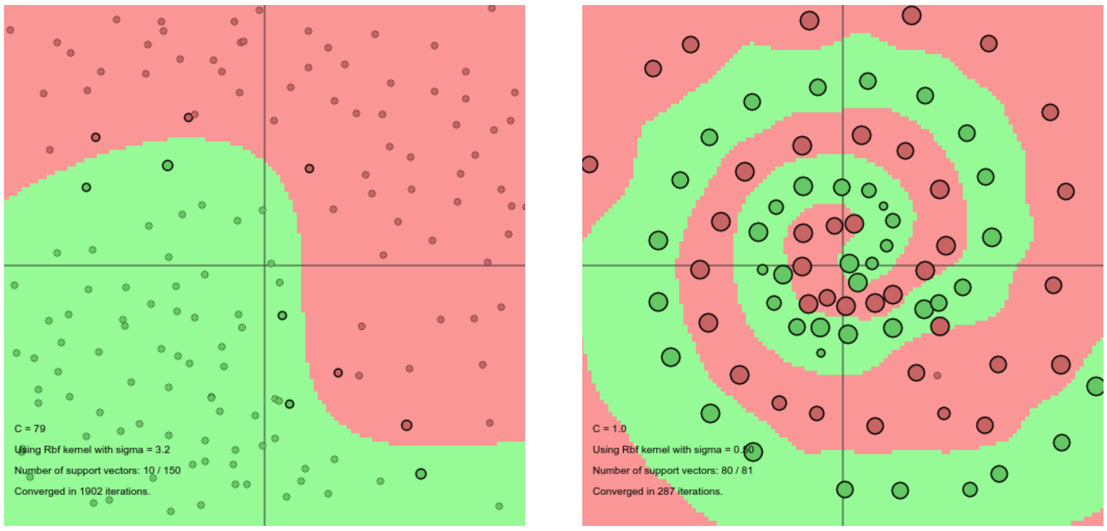
Figure 5: Classification using linear SVM and RBF kernel SVM

2.4 Support vectors (subquestion 7 and 8)

Support vectors are vectors in the data space (p-dimensional) that support the maximal margin hyperplane (in the feature space using the kernel trick for the RBF). If those particular vectors are moved, then the resulting margin moves as well. Importantly, the classifier's decision boundary 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 can be trimmed down after training, saving only those support vectors that are necessary for the classifier.

In the figure 6a 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 6b however, the entire dataset (highly non-linear) consists of the support vectors. It is not possible to reduce the amount of support vectors by tuning the parameters without impacting with much detriment the boundaries of the classifier.

A datapoint becomes a support vector when it is in proximity of the margin of the classifer, or if it is misclassified. From the experimentation on the web application, we can see that anytime a point is added in the wrong region, or within the margin, it becomes a support vector.


C = 79
Using Rbf kernel with sigma = 3.2
Number of support vectors: 10 / 150
Converged in 1902 iterations.

C = 1.0
Using Rbf kernel with sigma = 0.50
Number of support vectors: 80 / 81
Converged in 287 iterations.

(a) Few support vectors needed (10/150)　　*(b) No compression*

Figure 6: 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 8 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. In the table 1, you can see the error rate on the test 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. Overfitting typically leads to poor generalization.

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^2 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 $sigma^2$, 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)

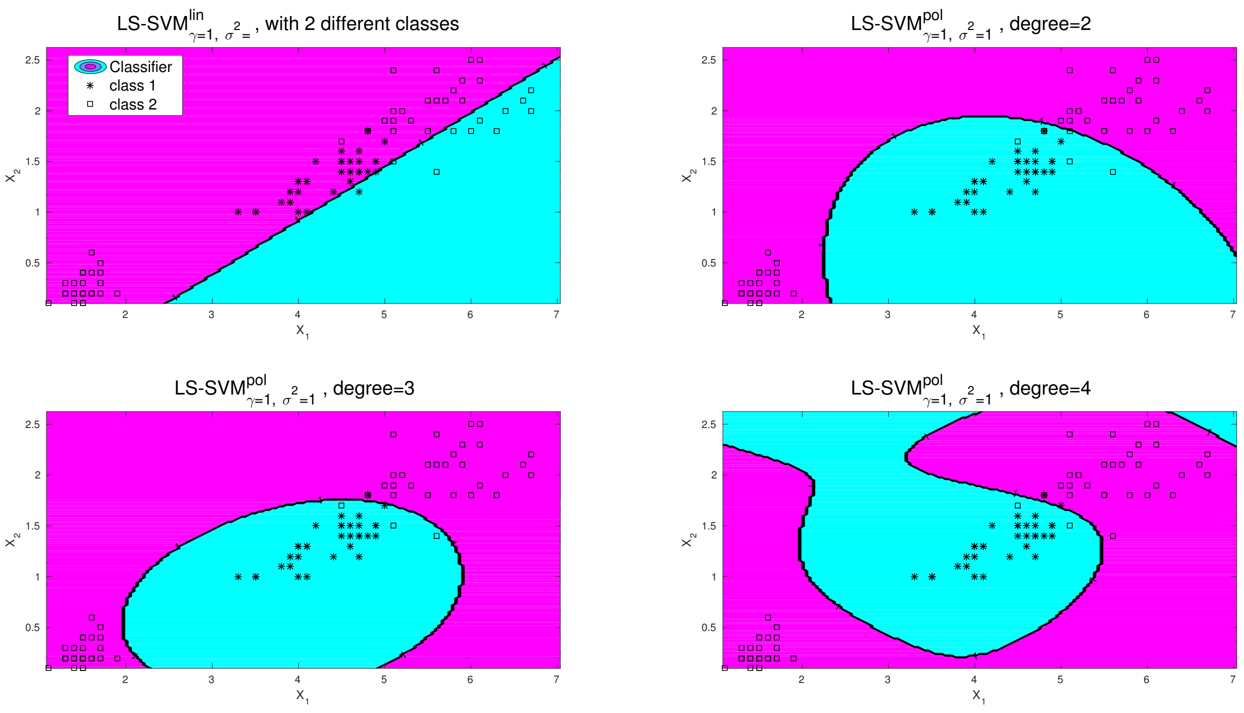


Figure 7: Linear and polynomial kernel classifiers

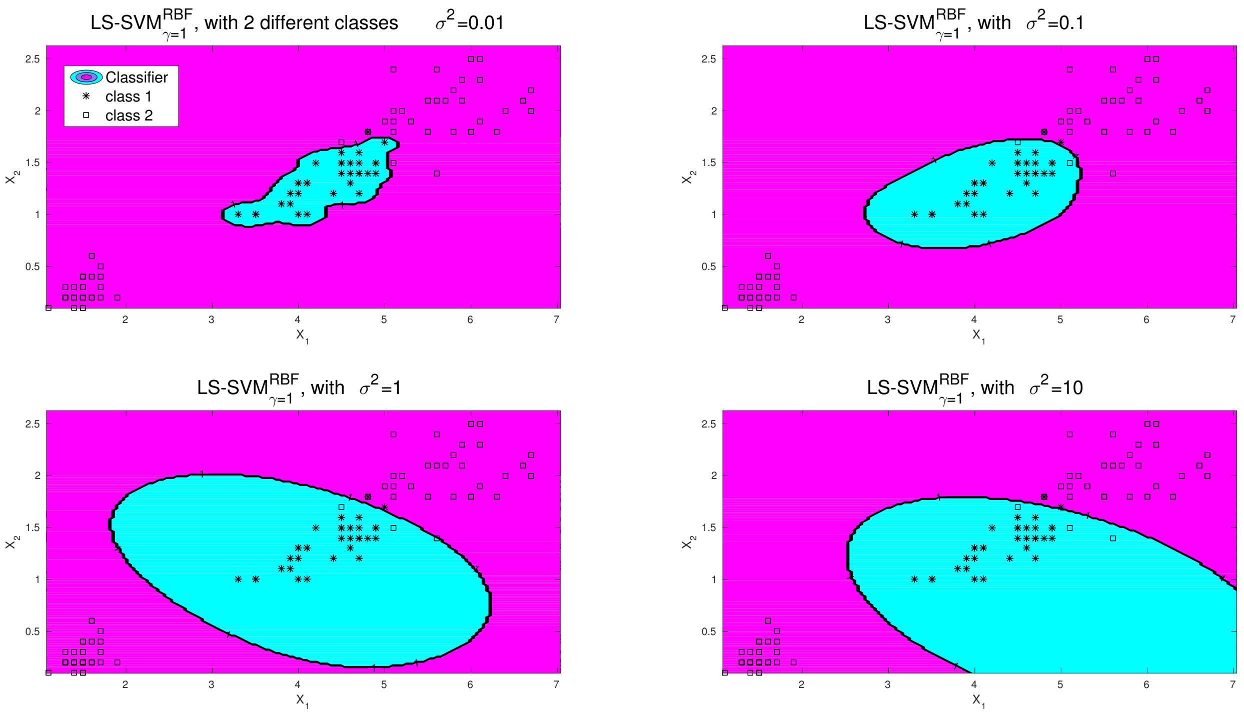


Figure 8: RBF classifier with different values of sigma

This last figure 9 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 test set. On the other end of the spectrum,

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

Table 2: Misclassification summary (RBF kernel)

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 test set are off (see table 3). Typically, you have *memorized* your dataset, but failed to obtain good generalization.

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

Table 3: Misclassification summary (Overfitting)

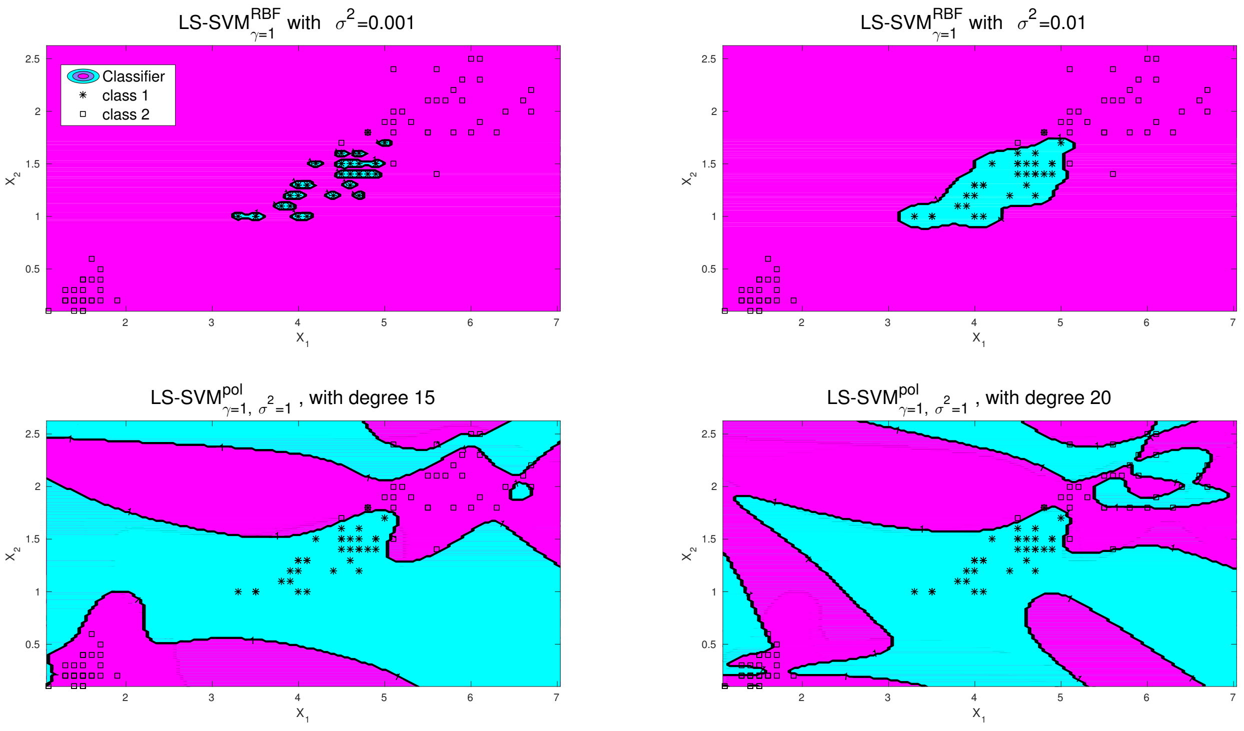


Figure 9: Overfitting with polynomial and RBF kernels

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 code and figure 10 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 %');
```

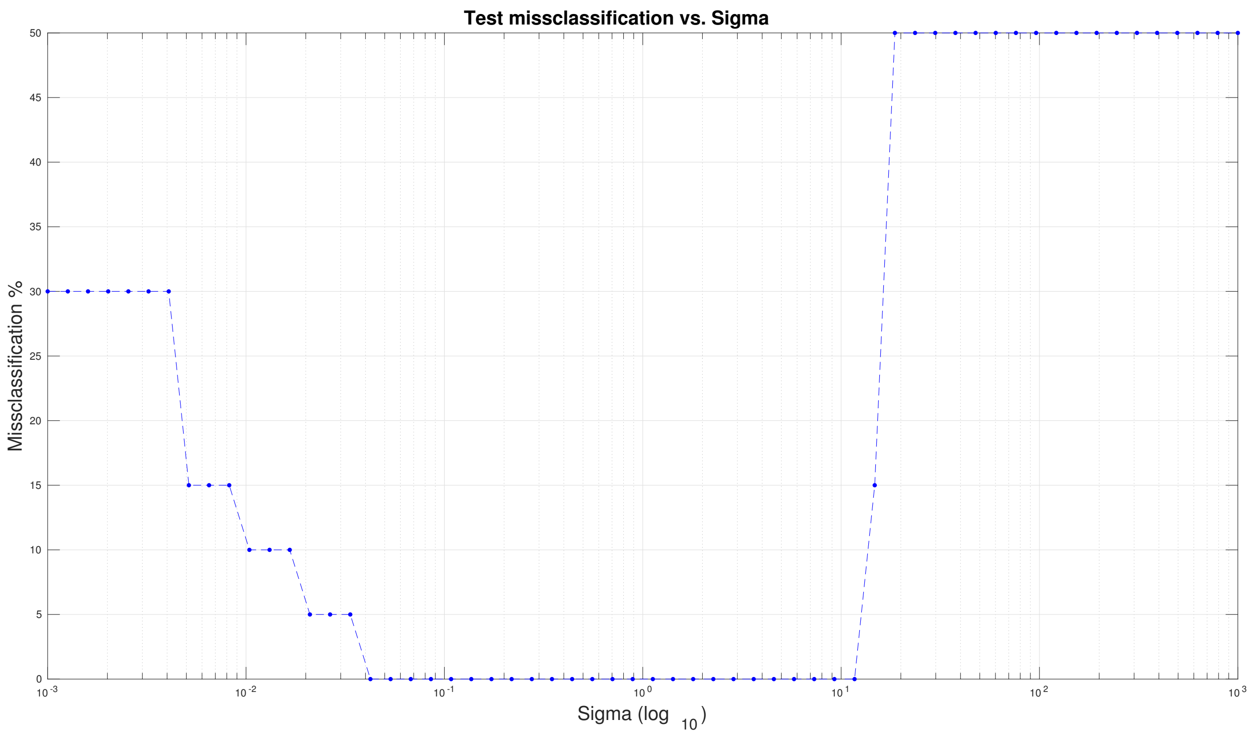


Figure 10: Tuning of parameter σ^2 for fixed $\gamma = 1$

3.1.3 RBF kernel: regularization constant

In this section, we explore further the effect of the tuning parameter γ, used for regularization (to control the amount of slack variables for non-separable data), 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 %');
```

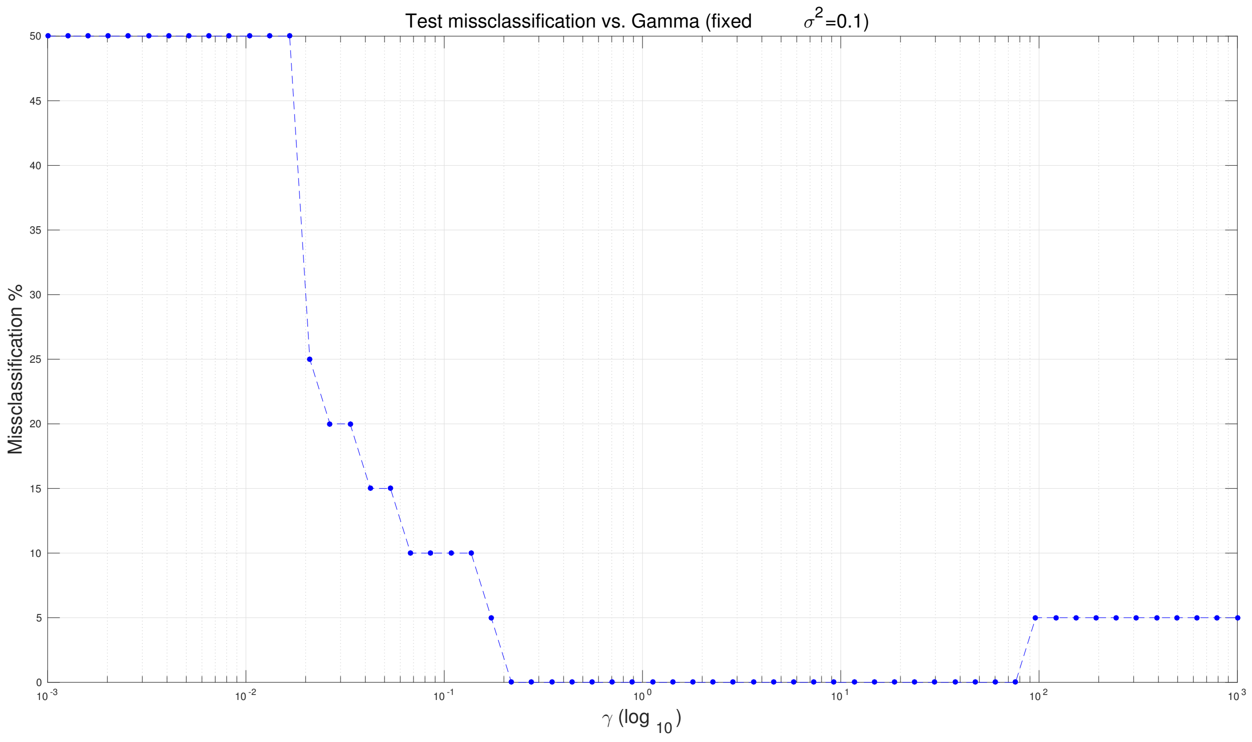


Figure 11: Tuning of parameter γ for fixed $\sigma^2 = 0.1$

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^2 and γ, and computed the test error on the validation set as follow:

```matlab
% 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 12 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. The specific values obtained being contingent upon the random split of the dataset between training and validation.

Importantly, one should not re-use the validation set to estimate the generalization of the model but rather should set aside a test set (untouched during the whole training). Indeed using the same validation set would likely induce an optimisticly biased generalization estimation (memorization of the dataset).

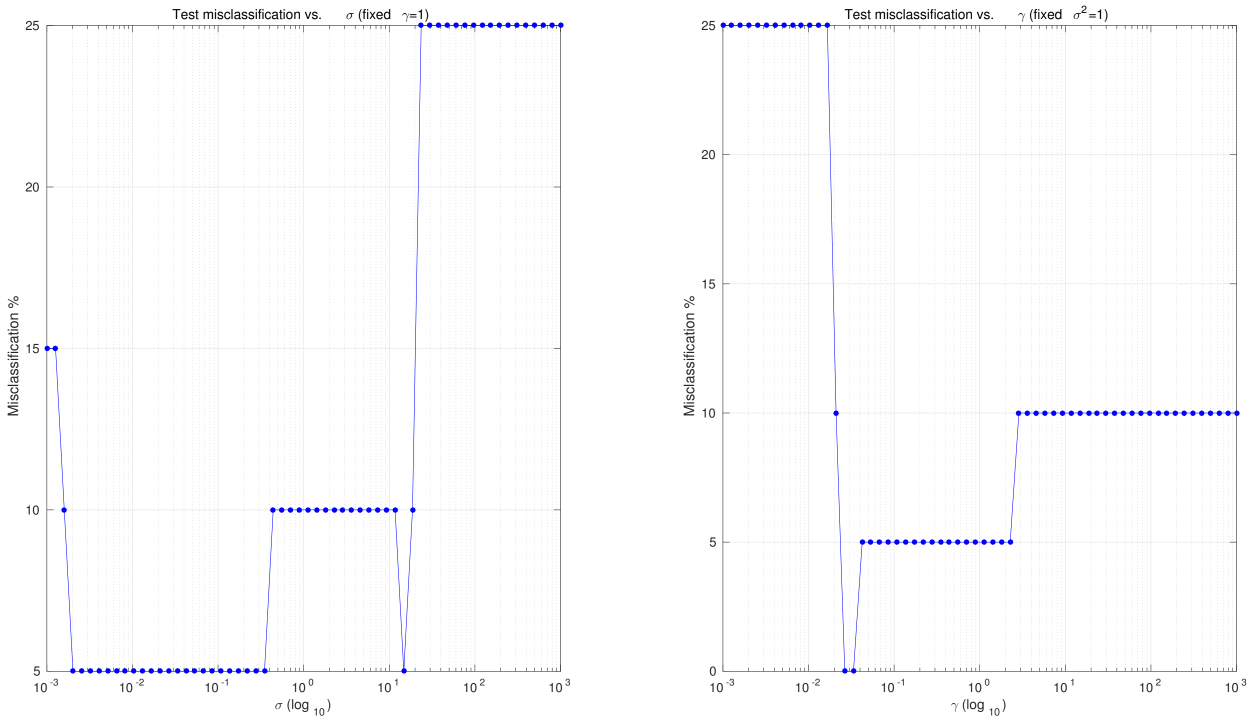


Figure 12: Tuning of parameter γ and σ^2 based on the validation set

3.2.2 Cross validation

Cross validation is a common technique in statistics to help remove the variance in the parameters estimation 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 averaged 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 validation cases with previous validation cases, so the whole training dataset is surveyed. The validation errors are averaged accross the 10 splits to obtain the average validation error on the whole training dataset.

For the LOOCV, the procedure is the same, but only a single datapoint is kept at each split to validate 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 computationally expensive on large datasets, though there exists exceptions to that (eg, in the case of linear regression, it is possible to 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), the results are widely variable 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.

Another important distinction to make here, is that although the formulation of the SVM problem is indeed convex, we now are faced with a problem of hyperparameter selection, which is non-convex, and many local minima are now possible. Hence the technique *csa* and *ds* are global optimization techniques that can help with the search (though without garantees of locating a global minimum). In a first step, those techniques will act to select in the landscape a place to start the search, which is then finalized using a *simplex* approach, or a pre-defined *searchgrid*. This explains the variability in the returned *optimized* parameters.

```matlab
% 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 13. 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.

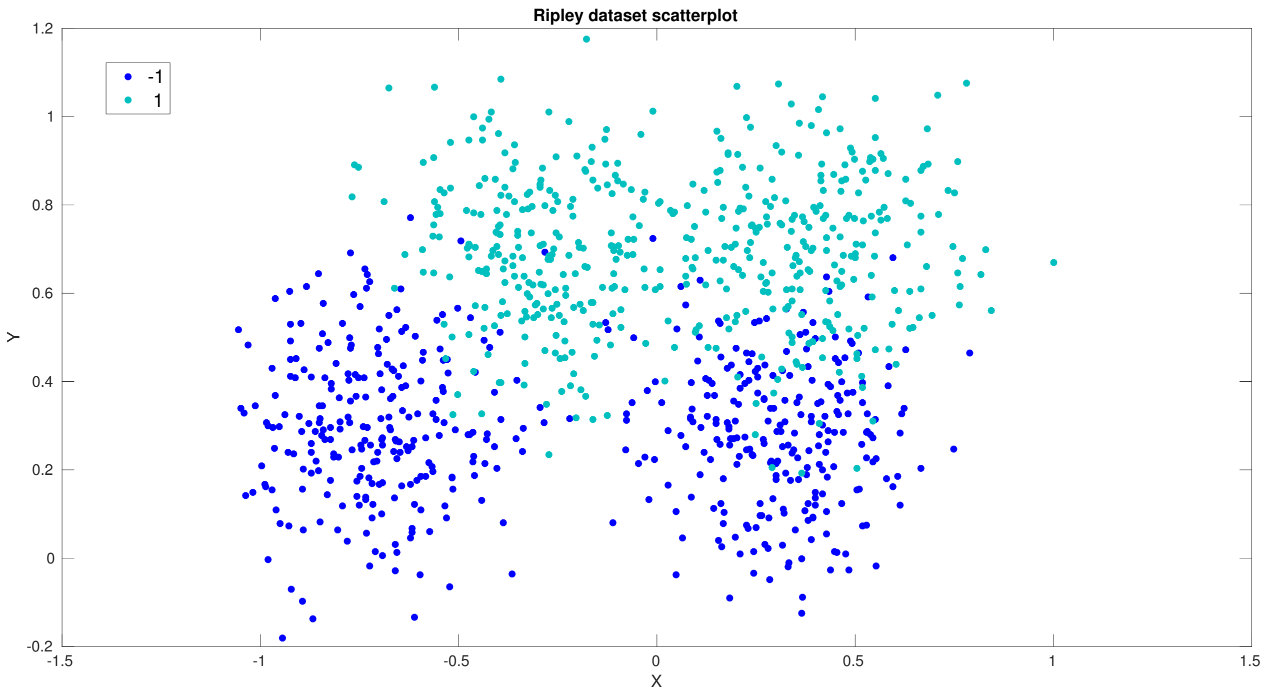


Figure 13: Ripley dataset scatterplot by group

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 14

```
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 14

```
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});
```

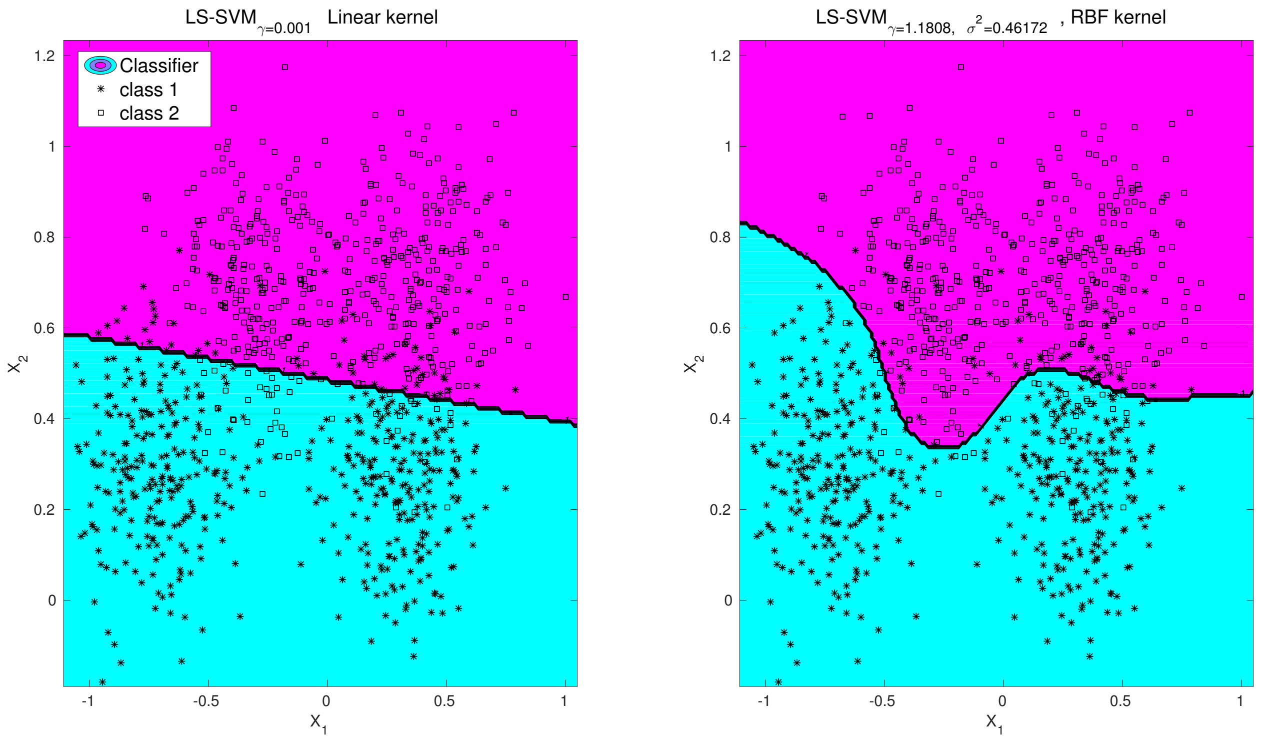


Figure 14: Ripley dataset, left linear kernel, right RBF kernel

4.1.4 ROC curves

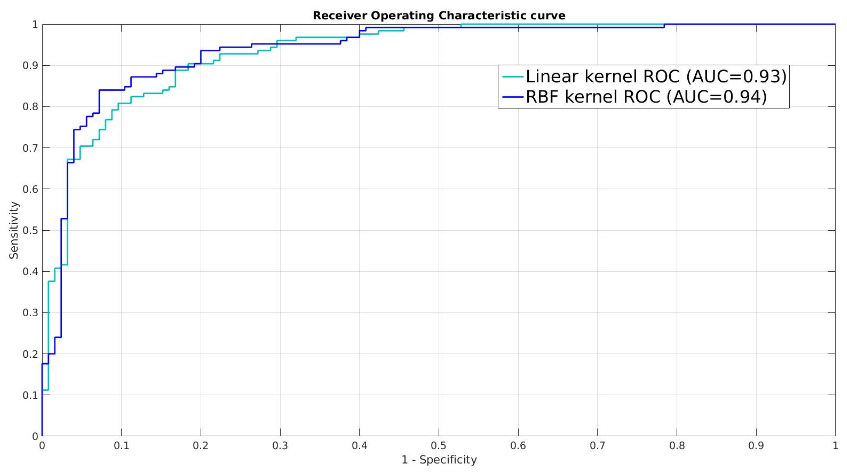


Figure 15: ROC of both classifiers

4.1.5 Final model selection

As illustrated by the ROC curves, the performance on the test set of the RBF classifier is only marginally superior. The Ripley dataset consists of overlapping gaussian distributions, so using a simpler model that incorporates a linear classifier could make sense here, but I would still favor the RBF kernel model given the knowledge of the dataset's structure.

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 16. 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.

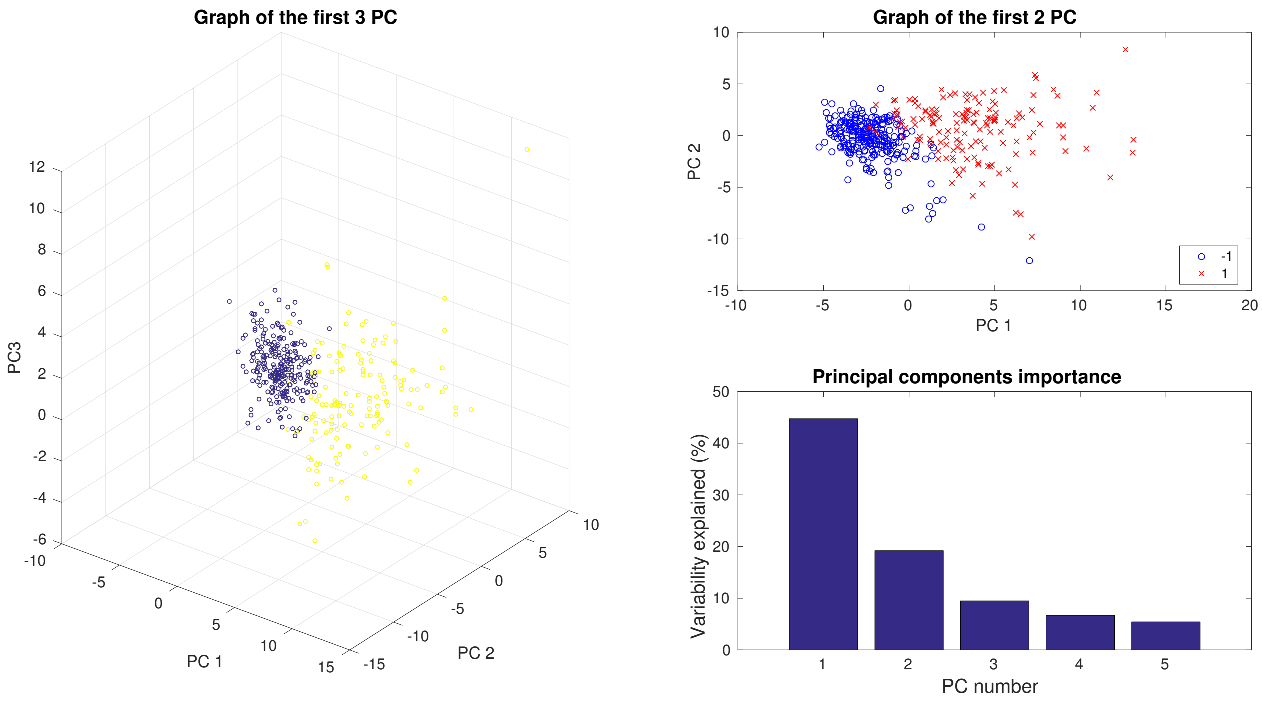


Figure 16: Principal components analysis and visualization

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

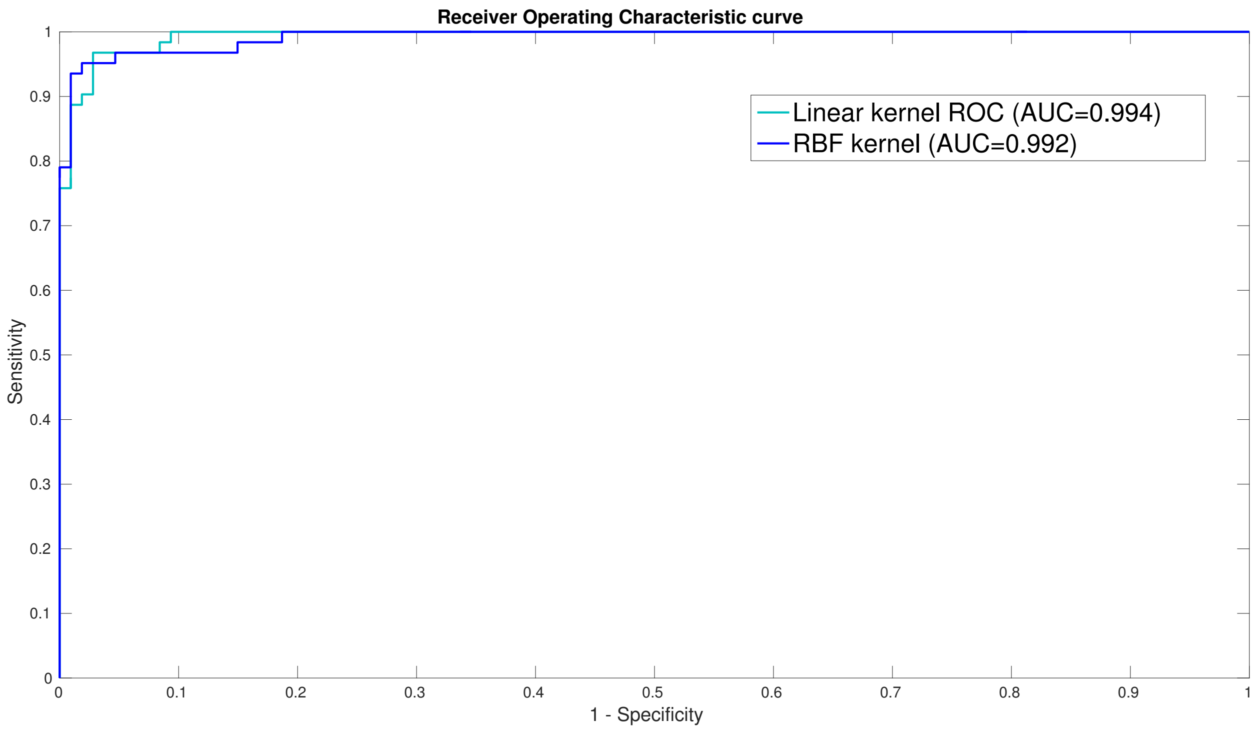


Figure 17: ROC of both classifiers

4.2.5 Final model selection

Similarly to section 4.1.5, we see that there 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 we can see a linear separability in the principal components plot.

4.3 Diabetes database

4.3.1 Data visualization

Using the same approach as in the previous section for the visualization with a PC decomposition, we cannot see a clear linear separability between the datapoints (see figure

18).

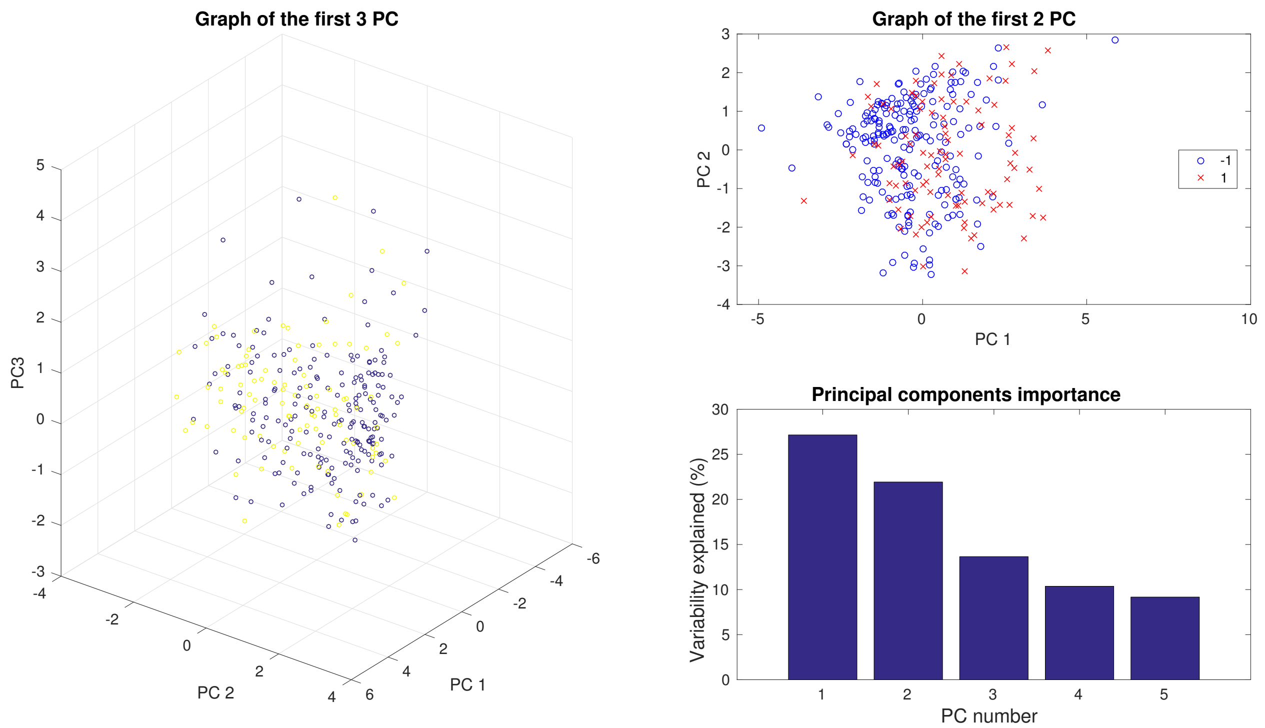


Figure 18: Principal components visualization

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'});

[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;
```

4.3.4 ROC curves

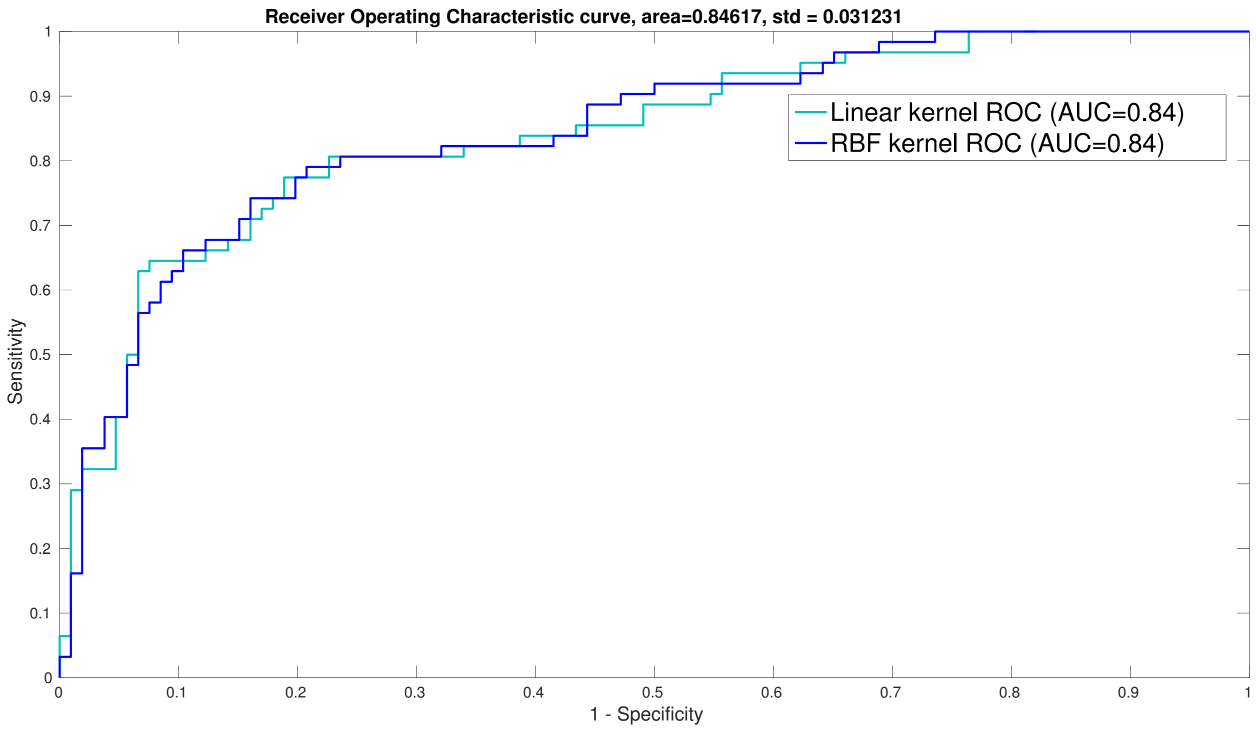


Figure 19: ROC of both classifiers

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.