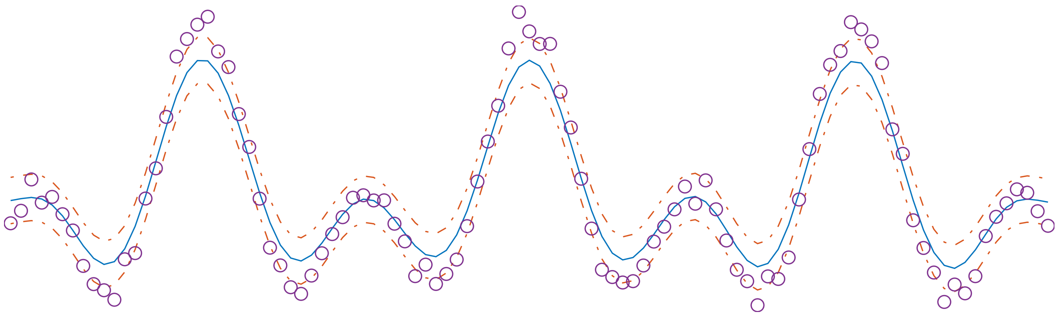
KU LEUVEN

FACULTY OF ENGINEERING

Support Vector Machines Report



Moritz Wolter

Contents

Session 1

Classification

1.1 Geometric construction of a classifier

As a first experiment a classifier will be constructed geometrically. This is done by computing the average sample for each of the two training Gaussian distributed data sets under consideration. Next the midpoint of the two average points is found. The classification boundary is drawn with at a $\pi/2$ angle to the line connecting the two average points. The result is shown in figure 1.1 on the left. A couple of points are misclassified by this approach, which given the statistical nature of the problem must always be allowed. In this case this method produces a decent classifier, this must not be true for other distributions though. If for example many additional samples distributed as shown in figure 1.1 on the right are added the method breaks down. This happened because the new samples moved the average and with it the decision boundary. In this new situation a large portion of the blue set is misclassified. Points far from the decision boundary should not influence the classifier like this. This does not happen to this extend if an optimal margin is sought which maximizes the distance of the classifier to each data set if separable or the best possible separation in terms of classification if the data set is not. Figure 1.2 illustrates this.

1.2 Vapnik Support Vector machines

At the hart of vapnik's theory is the optimal hyperplane algorithm. In the linear the hyperplane condition is given as[1]

$$y_k[\mathbf{w}^T\mathbf{x}_k + b] \geq 1, \quad k = 1, \ldots, N \tag{1.1}$$

With x_k being the input data points and y_k the desired output. Training the machine means finding the high dimensional hyperplane normal vector w and the bias scalar b. Normal means here that the dot product with any vector lying in the plane must be zero. Technically a plane is defined by $\mathbf{w}^T(\mathbf{x} - \mathbf{p}) = 0$ but its only interesting here to evaluate the classifier so $b = \mathbf{p}^T\mathbf{w}$ can be used instead and the displacement within the feature

[1]Support-Vector Networks CORINNA CORTES VLADIMIR VAPNIK,Machine Leaming, 20, (1995) page 291 or Least Squares Support Vector Machines, Suykens et al., page 31.

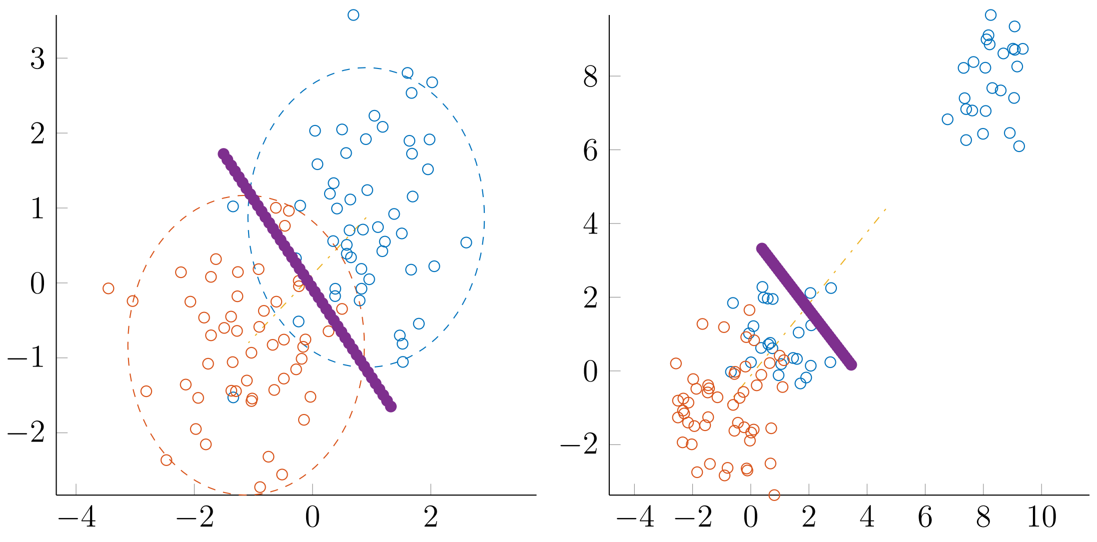


Figure 1.1: Geometrical construction of a linear classification line using average values.

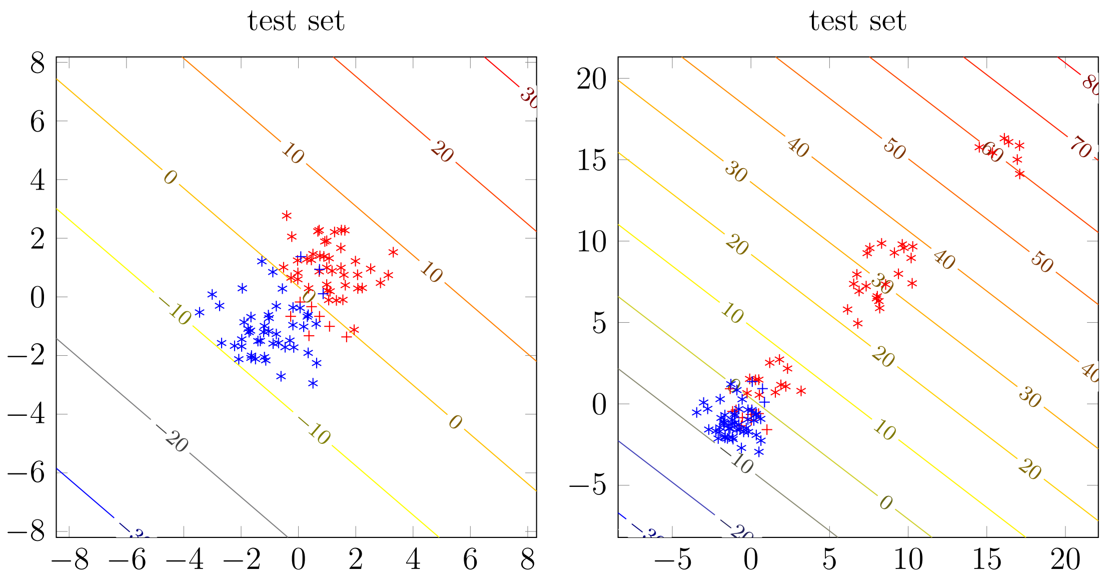


Figure 1.2: Linear support vector machine classification.

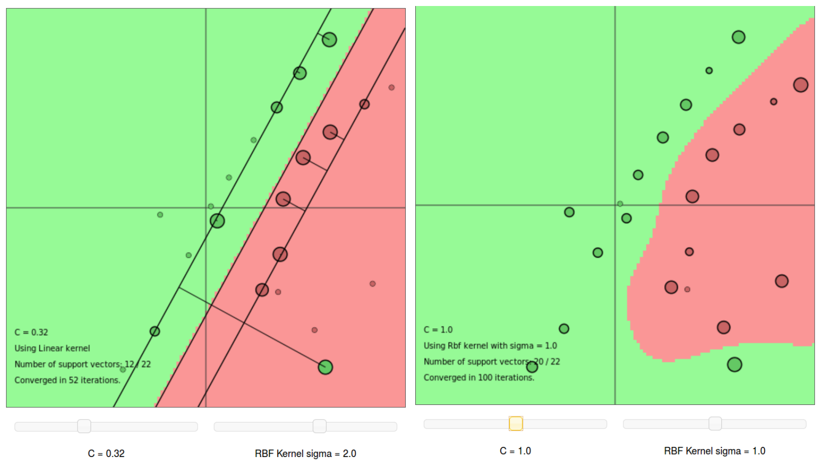


Figure 1.3: Support vector machine classification of an almost linearly separable problem using a linear and an radial basis function kernel

space remains unknown. The next step is to formulate the optimization problem

$$\min_{w,b} \frac{1}{2}\mathbf{w}^T\mathbf{w} \;\text{ such that }\; y_k[\mathbf{w}^T\mathbf{x}_k + b] \geq 1. \tag{1.2}$$

Which is the same as asking to maximize the classification margin. This can been seen from rescaling the discriminant function to $\|\mathbf{w}^T\mathbf{x}+b\| = 1$. For separable problems using the fact that $\|\mathbf{w}^T\mathbf{x} + b\| = 0$, on the boundary can be used to express the classification margin to be $\frac{1}{\|\mathbf{w}\|}$. This shows that the optimization problem formulation maximizes the classification margin by minimizing $\mathbf{w}^T\mathbf{w}$.[2] [3] Formulating the Lagrange dual, and taking the gradient of $\mathcal{L}(\mathbf{w}, b; \alpha)$ with respect to $(\mathbf{w}, b)$ leads to a problem in α. The Lagrange multipliers are called α, if a point $\mathbf{x}_k$ has an associated $\alpha_k > 0$ it is considered a support vector. From an optimization point of view these are points with active set index Lagrange multipliers. A problem reformulation using a mapping to a high dimensional feature space $\varphi(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ as

$$y_k[\mathbf{w}^T\varphi(\mathbf{x}_k) + b] \geq 1 \tag{1.3}$$

allows for different kernel options. The classifier is given by

$$y(\mathbf{x}) = \text{sign}(\sum_{k=1}^{N} \alpha_k y_k K(\mathbf{x}, \mathbf{x}_k) + b). \tag{1.4}$$

The following three kernels will be considered here[4]

$$K(\mathbf{x}, \mathbf{x}_k) = \mathbf{x}_k^T\mathbf{x} \qquad \text{(linear SVM)}, \tag{1.5}$$
$$K(\mathbf{x}, \mathbf{x}_k) = \exp(-\|\mathbf{x} - \mathbf{x}_k\|_2^2/\sigma^2) \qquad \text{(RBF Kernel)}. \tag{1.6}$$

If the problem is not separable, it means that classification cannot be done without error. In the underlying optimization problem slack variables have to be included in the formulation:

$$y_k[\mathbf{w}^T\varphi(\mathbf{x}_k) + b] \geq 1 - \xi_k \tag{1.7}$$

[2]Least Squares Support Vector Machines, Suykens et al., page 30
[3]Support Vector Machines and Kernels for Computational Biology, Asa Ben-HurAsa et al, page 6
[4]Least Squares Support Vector Machines, Suykens et al., page 43.

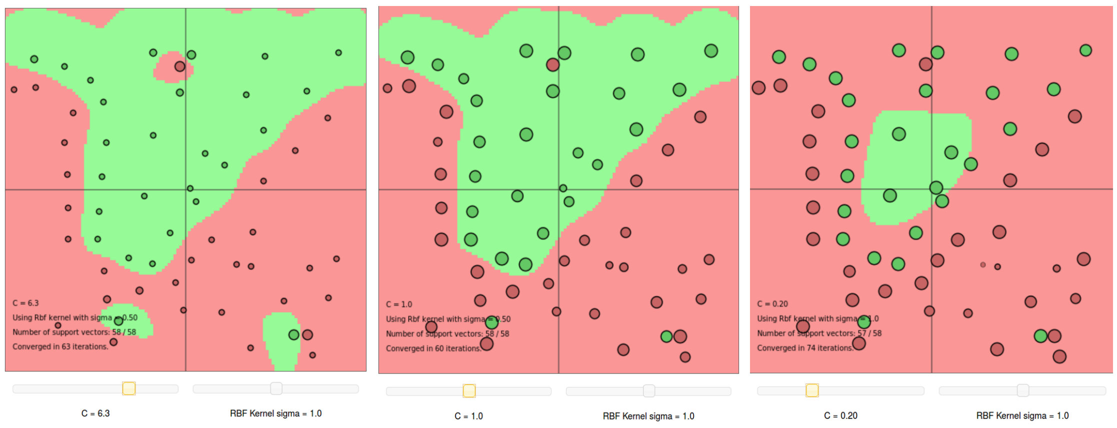

```
C = 6.3
Using Rbf kernel with sigma = 0.50
Number of support vectors: 58 / 58
Converged in 63 iterations.
```
```
C = 1.0
Using Rbf kernel with sigma = 0.50
Number of support vectors: 58 / 58
Converged in 60 iterations.
```
```
C = 0.20
Using Rbf kernel with sigma = 1.0
Number of support vectors: 57 / 58
Converged in 74 iterations.
```

Figure 1.4: The effect of a large mis-classification penalty constant c (left), a good choice for c (middle) and a too small penalty value (right).

Which leads to the optimization problem:

$$\min_{w,b,\xi} J_p(w,\xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + c\sum_{k=1}^{N}\xi_k \tag{1.8}$$

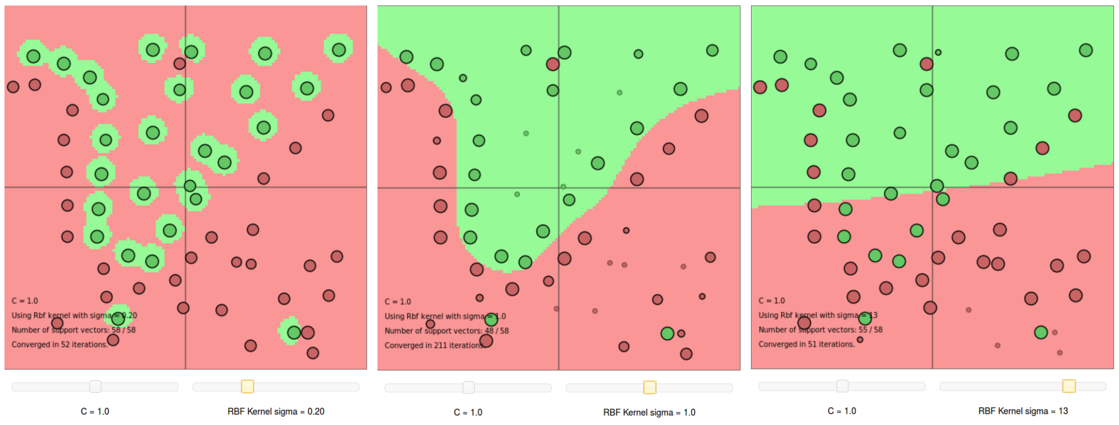
$$\text{such that } y_k[\mathbf{w}^T\varphi(\mathbf{x}_k) + b] \geq 1 - \xi_k, \quad k = 1,\dots,N \tag{1.9}$$

$$\xi_k \geq 0, \quad k = 1,\dots,N \tag{1.10}$$

Where c is the positive real mis-classification penalty constant. High values of c make erroneous classification more costly in terms of the merit function. Low values make those cheaper. Using the `svmjs`[5] package the result of using different values of c is illustrated in figure 1.4. Choosing the right value for c is a balancing act. It is important to choose c not too small. As very small values will result in underfitting the problem. The resulting classifier will ignore important features of the problem, as illustrated on the right in figure 1.4. On the other hand c must also not be too large. If the penalty on incorrect classification is too large the support vector machine will start to memorize noisy details, as illustrated in figure 1.4 on the left. A good choice like the one shown in the middle captures the key points while not falling prey to the noise, while using just as many support vectors as necessary. The effect of changing the kernel density σ is explored in figure 1.5. In this case the classification error increases significantly for very small or very large σ values. Like choosing a good c value, when picking σ under and over-fitting considerations are important. Choosing the kernel too small as shown in figure 1.5 on the left will result in over-fitting. Even tough all points are classified correctly the model misses the general overall geometry of the input data points completely. If σ is too large under-fitting is observed in this case. Results are better in comparison to the very small σ-value but one would expect to see results of similar quality from a simple linear kernel. A good choice such as the one in figure 1.5 in the middle captures the big picture while not using excessive amounts of support vectors.

Figure 1.6 illustrates the effect new data points have on the decision boundary. In the right plot two new red points have been added inside of the formerly entirely green region. The decision boundary changes if no additional mis-classification follows from

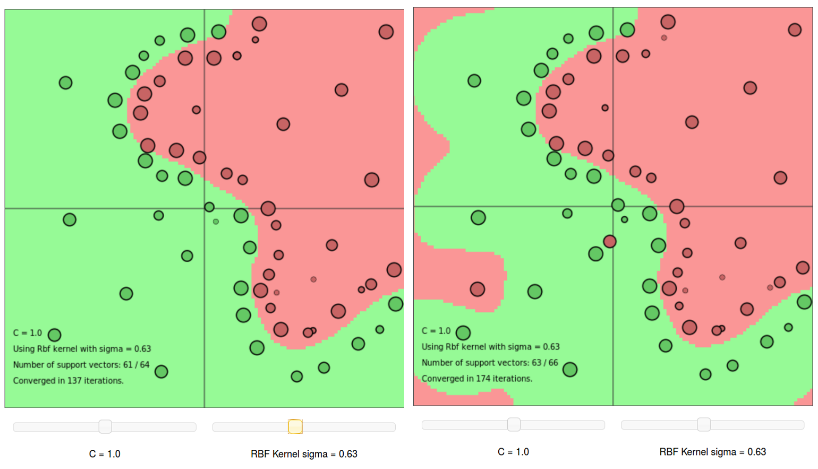
[5] `http://cs.stanford.edu/people/karpathy/svmjs/demo/`


C = 1.0
Using Rbf kernel with sigma = 0.20
Number of support vectors: 58 / 58
Converged in 52 iterations.
C = 1.0
RBF Kernel sigma = 0.20

C = 1.0
Using Rbf kernel with sigma = 1.0
Number of support vectors: 48 / 58
Converged in 211 iterations.
C = 1.0
RBF Kernel sigma = 1.0

C = 1.0
Using Rbf kernel with sigma = 13
Number of support vectors: 55 / 58
Converged in 51 iterations.
C = 1.0
RBF Kernel sigma = 13

Figure 1.5: Using a too small (left), a nicely chosen (middle) and too large (right) radial basis function parameter σ.


C = 1.0
Using Rbf kernel with sigma = 0.63
Number of support vectors: 61 / 64
Converged in 137 iterations.
C = 1.0
RBF Kernel sigma = 0.63

C = 1.0
Using Rbf kernel with sigma = 0.63
Number of support vectors: 63 / 66
Converged in 174 iterations.
C = 1.0
RBF Kernel sigma = 0.63

Figure 1.6: The effect of two additional red points on the boundary

this change. For the new red point close to the green boundary this cannot be done without mis-classifying the green points at the boundary, which therefore has to remain unchanged. At this point it is also important to note that [6]

$$\mathbf{w} = \sum_{k=1}^{N} \alpha_k y_k x_x \text{ (linear)}, \tag{1.11}$$

$$\mathbf{w} = \sum_{k=1}^{N} \alpha_k y_k \varphi(x_k) \text{ (non-linear)}. \tag{1.12}$$

Thus not all points contribute equally to the orientation of the decision boundary. In fact only support vectors with $\alpha_k \gg 0$ contribute, for all other points the decision bound hardly changes or remains unchanged, if $\alpha_k = 0$, if the are removed.

1.3 Least Squares Support Vector machines

In contrast to classical support vector machines, their least squares version is defined as:[7]

$$\min_{w,b,e} J_p(w,e) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\sum_{k=1}^{N} e_k^2 \tag{1.13}$$

$$\text{such that } y_k[\mathbf{w}^T\varphi(\mathbf{x_k}) + b] = 1 - e_k,\ k = 1, \ldots, N \tag{1.14}$$

In comparison to the Vapnik formulation the error variable is squared and the constraint is turned into an equality constraint. After training the dual space classifier

$$y(x) = \text{sign}[\sum_{k=1}^{N} \alpha_k y_k K(x, x_k) + b] \tag{1.15}$$

is obtained. Least squares support vector machines (LSSVM) are an attempt to reduce the computational load for very large data sets.
In a first experiment the iris data set[8] is used to test the LSSVM algorithm and learn more about the hyper-parameter space it works well in. The challenge here is to differentiate three types of iris plants *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*, based on the leaf dimensions: sepal length [`cm`], sepal width [`cm`], petal length [`cm`], petal width [`cm`]. Figure 1.7 shows what petal and sepal flower leafs are[9] as well as the data points included in the original iris data set. For a first LSSVM trial run the simplified version shown in figure 1.8 is used. Figure 1.9 shows the effect of three kernels on the classification results. As the merged *Setosa-Virginica* data points surround the *Versicolor* measurements there is no way these could be seperated by a simple line. A linear-svm is therefore bound to fail. It comes as no surprise that the linear-kernel did not deliver meaningful results, as shown in figure 1.9. The radial basis function (rbf) and the polynomial kernel are able to cope with the complexity of the problem. The next section will explore which parameter values should be chosen for both kernel types.

[6]Least Squares Support Vector Machines, Suykens et al., page 32 and 41.
[7]Least Squares Support Vector Machines, Suykens et al., page 72.
[8]`http://archive.ics.uci.edu/ml/datasets/Iris`
[9]`https://en.wikipedia.org/wiki/Sepal`

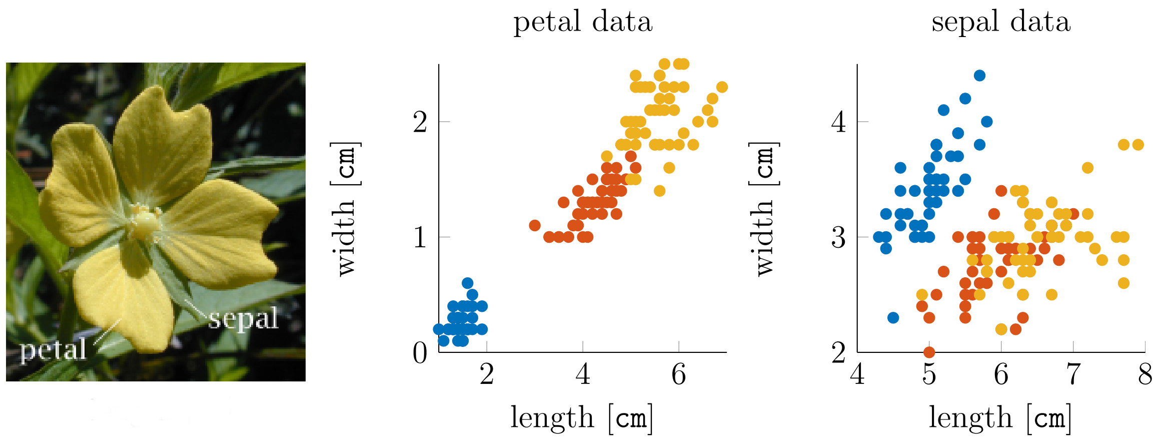


Figure 1.7: Illustration of the difference between petal and sepal flower leafs. As well as an illustration of the data distribution with blue for *Iris-setosa*, red indicating *Iris-versicolor* and yellow *Iris-virginica*.

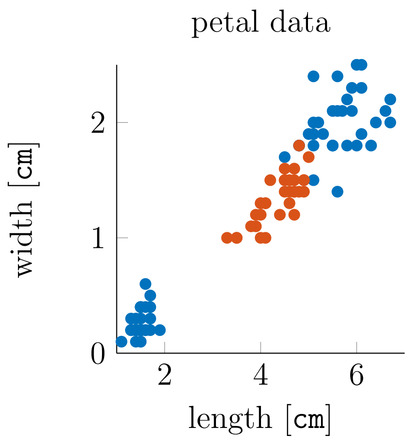


Figure 1.8: Simplified iris data set, only petal leafs and only two species *Iris-setosa-virginica* and *Versicolor* are considered.

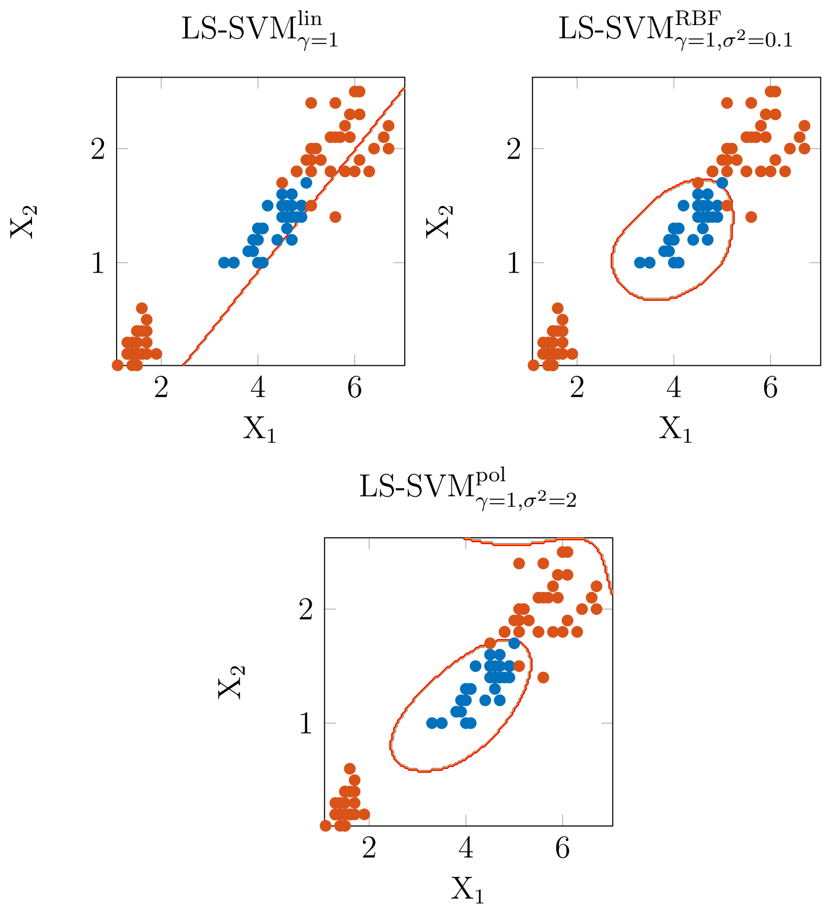


Figure 1.9: Classification of the simplified iris data set using LSSVMs with different kernels using two classes in each case.

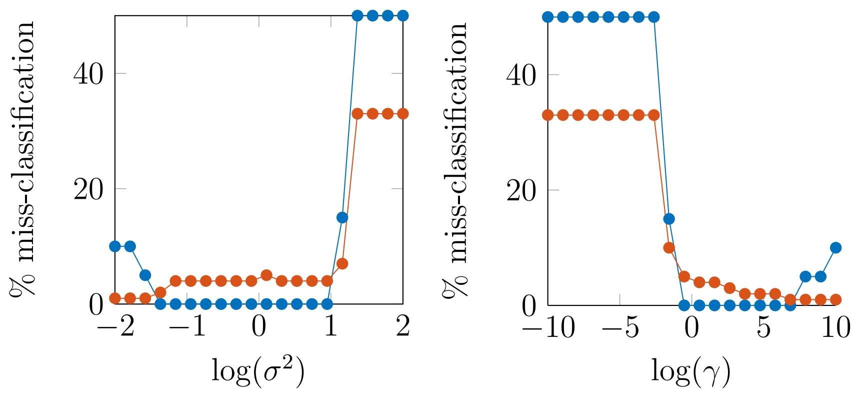


Figure 1.10: Miss-classification of test samples versus σ and γ. The test set performance is shown in blue the training set in red.

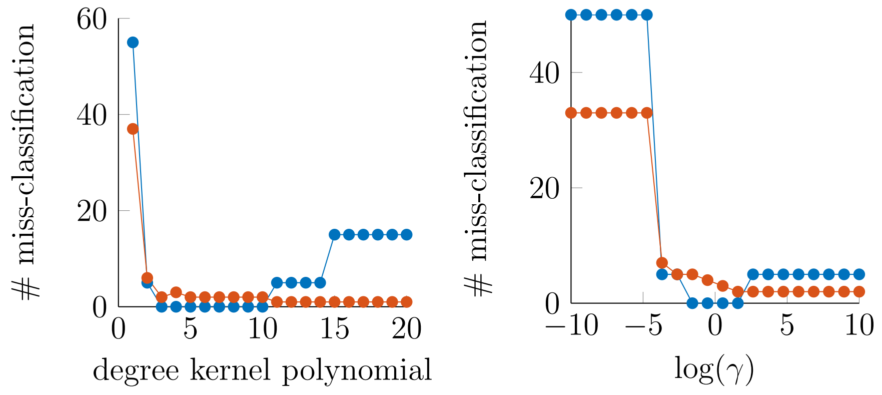


Figure 1.11: Miss-classification of test samples versus polynomial degree and γ. The test set performance is shown in blue the training set in red.

1.3.1 RBF kernel parameter selection for the iris data set

When using a radial basis function kernel the training process depends on the two parameters σ and γ. It was observed earlier that too small values of σ lead to over-fitting and too large ones will lead to an oversimplified classifier. This picture is repeated in figure 1.10, on the left. The second parameter γ is the regularization parameter. If a small value is chosen minimizing of the complexity of the model is emphasized. For a large γ on the other hand, good fitting of the training data points is stressed. This leads to a situation where the too large γ values cause overfitting and to small ones oversimplification as can be seen in figure 1.10.

1.3.2 Polynomial kernel parameter selection for the iris data set

Polynomial kernels follow the similar pattern. Figure 1.11 reveals that when one chooses the degree of the kernel polynomial too large the risk of over-fitting is high. A degree chosen too small will not capture the problem to its full extent. Gamma behaves with polynomial kernels just like it did with rbf-kernels.

1.3.3 The impact of different validation methods

In the previous section the importance of avoiding under- or over-fitting became clear. In this section three different validation methods will be compared. The first procedure randomly splits the one-hundred data points of the simplified iris data set into a twenty point validation and a hundred point training data set. Using the training data a svm is trained and then tested on the validation set. The measured mis-classification using this approach for $\gamma \in [10^{-10}, 10^{10}]$ and $\sigma \in [10^{-2}, 10^{2}]$ is shown in figure 1.12.
Cross-validation is a method which attempts to use a data set more efficiently. During k-fold-cross-validation the data set is split into k parts. While the method runs each part is set aside for validation once. A classifier is then trained using the remaining data points. The obtained model can then be tested on the validation part. During the next iteration another part will be ignored during training and so on. Cross validation has the added benefit, that each set serves as a validation set once, which hopefully gives a more complete picture of classification performance. Finally the leave one out method uses all data points except for one to train the classifier. The remaining data point can then be used to evaluate the classifier found. In a next iteration another point is ignored the model is retrained and so on. The leave one out methods is the most conservative in terms of training set size, but also requires the most computational resources as many iterations are required to get a meaningful idea about the classification performance. Figures 1.12, 1.13 and 1.14 show the validation results of the three methods using the same parameter ranges. For poor choices of σ and γ, correct classification sinks to about fifty percent, which is as good as classification at random. Generally speaking the 80% split method is the most data hungry, and it gives an edgy representation of svm performance. Cross validation is more efficient on the data and results are smoother but still follow a similar pattern. The leave one out methods produces an incorrect blue area in the top right for very small γ and σ values, which does not appear when using other methods. Generally speaking the leave one out methods is suitable for small data-sets due to its computational inefficiency.

1.3.4 Automatic tuning

Using the ls-svm toolbox'[10] `tunesvm` function automatic hyper-parameter selection is explored. Two algorithm pairs are available. The first pair is coupled simulated annealing or randomized directional search. The optimization function can be a simplex-type method or brute force gridsearch. Finally classification cost can be evaluated using k-cross-validation or leave-one-out. When choosing a combination a speed or accuracy trade-off has to be made. The fastest but most unstable combination tried during experiments for this report was the randomized directional search coupled with simplex optimization and cross validation. A histogram showing the γ and σ values found during fifty training processes is shown in figure 1.15. Instead of the randomized directional search coupled simulated annealing can be used. Which is not parallelized but the results if finds are more consistent, as can be seen in figure 1.16. The most stable algorithm combination in terms of predictability was a simulated annealing, grid search, leave-one-out validation combination. A histogram with tuning results is shown in figure 1.17. Only one of fifty iterations produced a large outlier.

[10]`http://www.esat.kuleuven.be/sista/lssvmlab/`

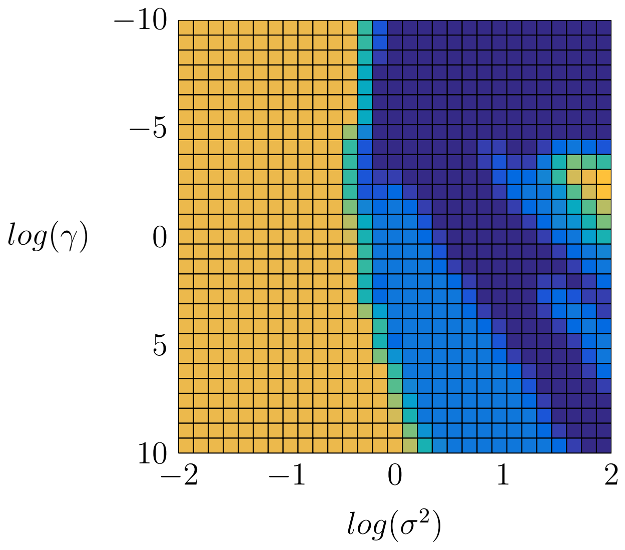


Figure 1.12: Miss-classification of validation data using a 80% training and 20% validation data ratio.

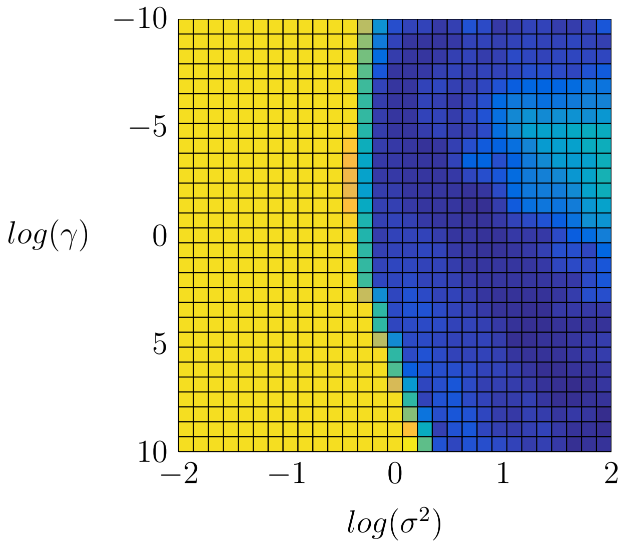


Figure 1.13: Miss-classification using ten fold cross-validation.

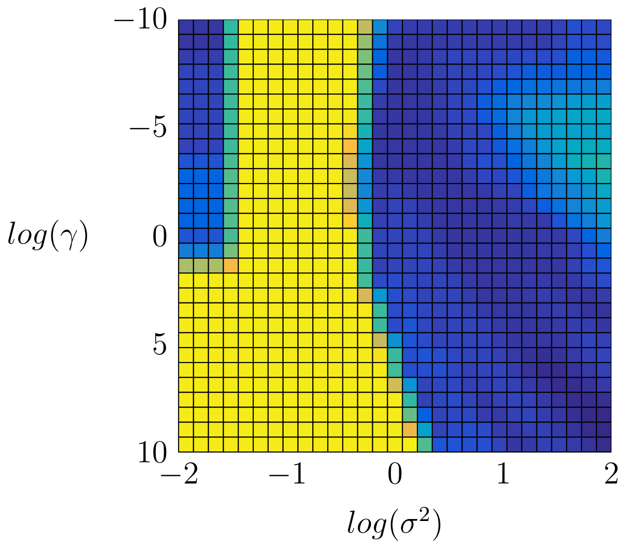


Figure 1.14: Miss-classification using leave one out validation.

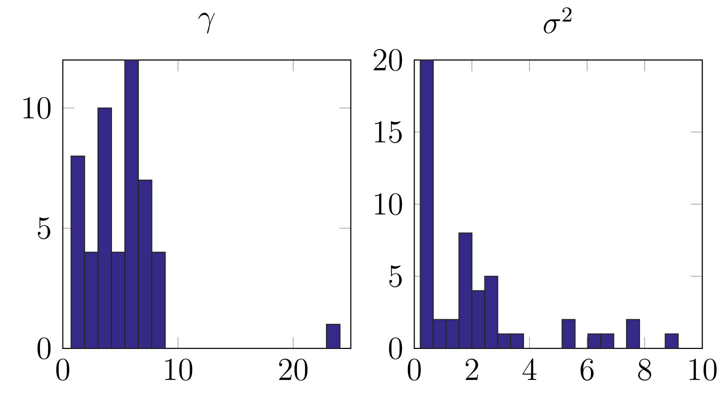


Figure 1.15: Histogram of automatically tuned γ and σ values using randomized directional search, cross-validation together with simplex-optimization.

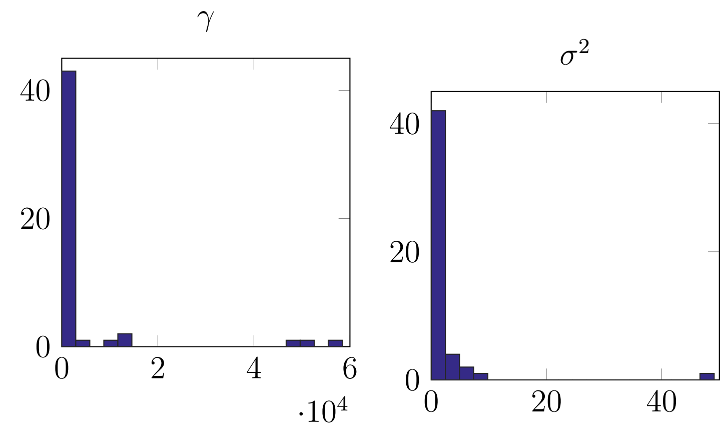


Figure 1.16: Histogram of automatically tuned γ and σ values using coupled simulated annealing, cross-validation together with simplex-optimization.

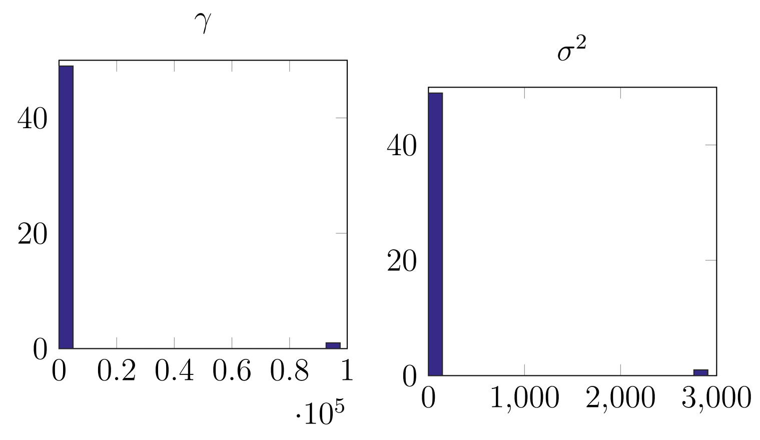


Figure 1.17: Histogram of automatically tuned γ and σ values using coupled simulated annealing, leace-one-out-validation together with a grid search algorithm.

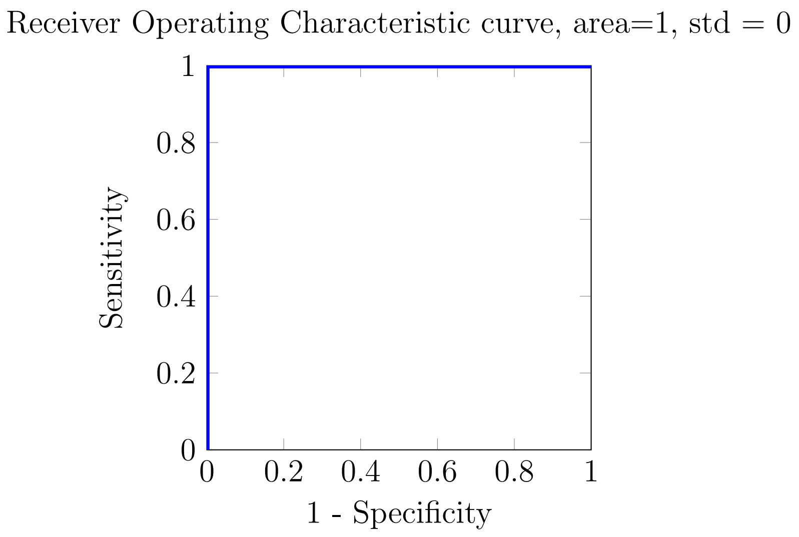


Figure 1.18: ROC curve of an ideal classifier.

1.3.5 The roc-curve

The receiver operating characteristic (roc)-curve is a way to asses the performance of a binary classifier as its discrimination threshold is varied. Defining[11]

$$\begin{aligned}
\text{Sensitivity} &= \frac{\text{TP}}{\text{TP}+\text{FN}} && (1.16)\\
\text{Specifity} &= \frac{\text{TN}}{\text{FP}+\text{TN}} && (1.17)\\
\text{False pos. rate} &= 1-\text{Specifity} = \frac{\text{FP}}{\text{FP}+\text{TN}} && (1.18)
\end{aligned}$$

With TP = “true positive”, FP = “false positive”and TN = “true negative”. The sensitivity is thus the ratio of correctly classified positives over the sum of true positives and false negatives, which is the total number of positive cases that should have been identified as such. Ideally one would aim for a sensitivity of one. Similarly the specificity is the number of true negatives over the total of cases that should have been classified as negative. Going trough possible decision threshold values the roc-curve is drawn. The area under the curve describes the efficency of the classifier. If it is one, a perfect classifier has been found, such as the one in figure 1.18. If the area is one half, the classifier has no added value over classification at random.

1.3.6 Full complexity iris data set classification

There is no reason not to subject least squares support vector machines to the full four dimensional classification problem as seen in figure 1.7. An experiment is done where 30 of the 150 iris plant data points are set aside for validation. The remaining values serve as training data. Automatically trained linear and rbf-kernel svm did not mis-classify a single iris flower and got optimal roc-curves such as the one shown in figure 1.18.

[11]Least Squares Support Vector Machines, Suykens et al., page 19.

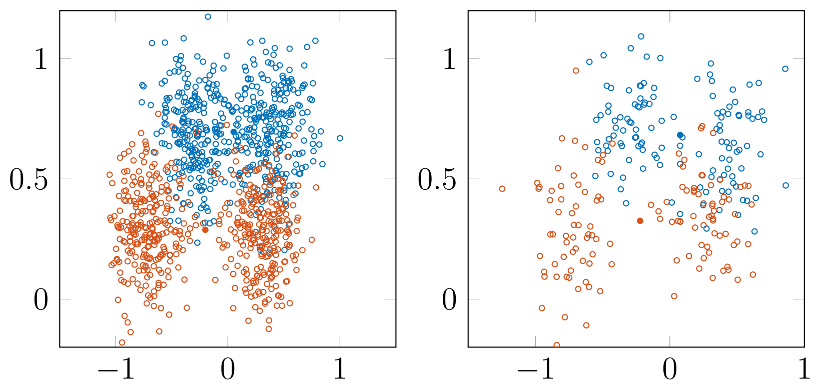


Figure 1.19: Visualization of the binary classification Ripley training and validation data set. The filled dots indicate the position of the average point for each set.

1.4 The Ripley Data-Set

The ripley data set[12] is a well know set used to test machine learning algorithms. The training and validation data are shown in figure 1.19. A linear and radial basis function kernel will be used to classify the validation data set. Figure 1.20 shows the linear classifier on the training data set and the validation data based roc-curve. In this experiment it was observed that the linear classifier got 16.8% or 42 of the 250 validation data points wrong. Due to the non-liner nature of the problem the radial basis function kernel was able to outperform its linear counterpart, only 2.4% or 6 validation data points where incorrect. This performance difference is somewhat reflected in the roc curves shown on the right of figures 1.20 and 1.21, the curve associated with the rbf-svm covers more area and has a lower standard-deviation.

1.5 Breast Cancer Data-set

The uci breast cancer set[13] contains 596 multidimensional-data points with tissue sample information such as smoothness, compactness, concavity, etc. The data is annotated, with labels indicating healthy or cancerous samples. 400 training and 169 validation measurements are included. Data points have more then 3 dimensions it is this not possible to visualize them using conventional methods. Therefore different classifiers will be trained and evaluated using their roc-curves. The roc-curves of automatically tuned linear and rbf-classifiers are shown in figure 1.22. Both classifiers perform equally well, only around two to five percent of samples are mis-classified using automatically tuned machines with either of the two kernels.

[12]Pattern recognition and Neural Networks B.D. Riplely, Cambridge University press, `http://www.stats.ox.ac.uk/pub/PRNN/`

[13]`http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29`

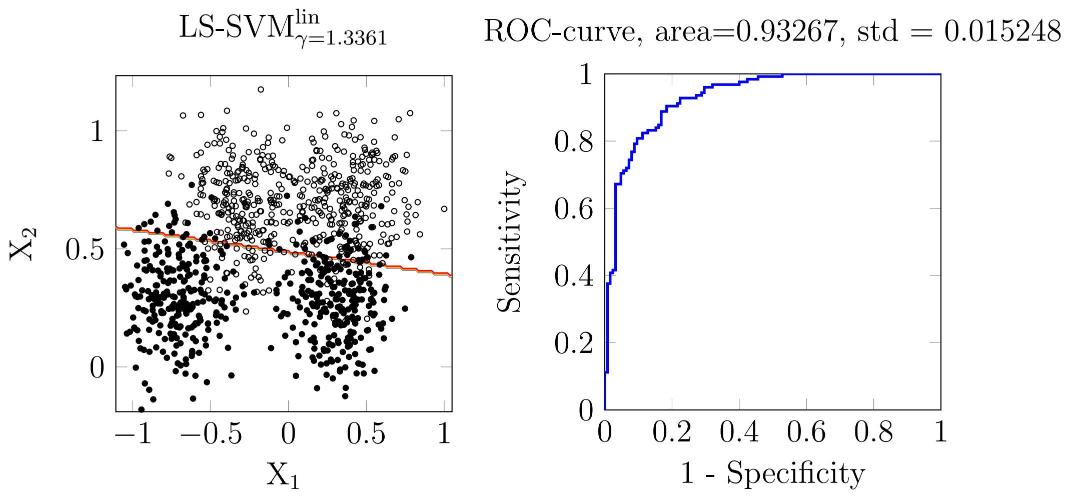


Figure 1.20: Linear least squares support vector machines visualization and validation based roc curve.

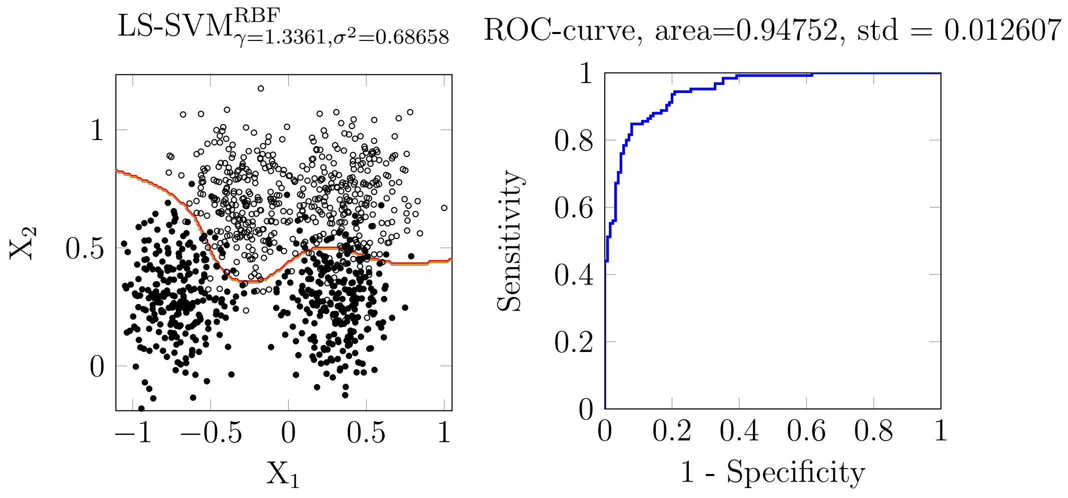
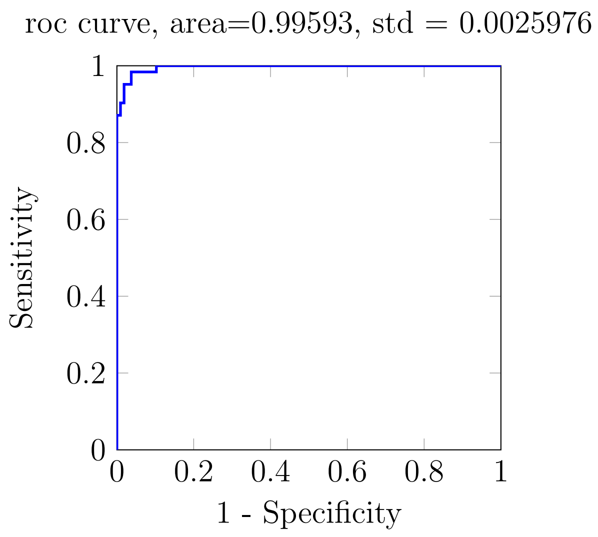


Figure 1.21: Radial basis function least squares support vector machines visualization and validation based roc curve.



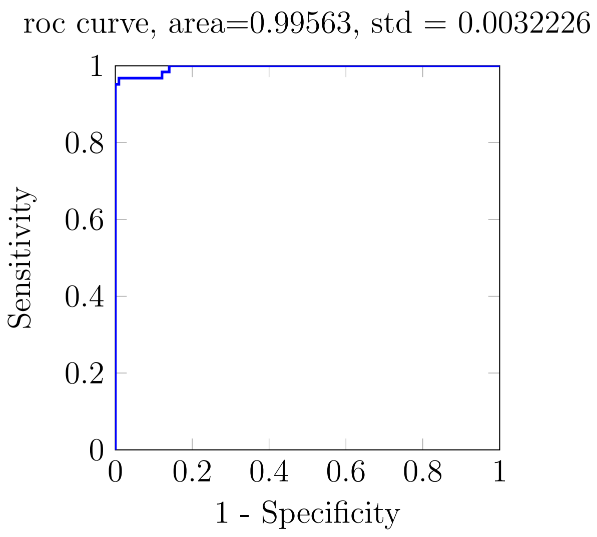


Figure 1.22: Linear and radial basis function lssvm classifier roc-curves for the breast-cancer data set.

1.6 Diabetes Database

In this example data from a study of diabetes cases among female Pima indians [14] is used. The data set consists of 300 training data points and 168 validation samples. Each point consists of information such as the number of past pregnancies, body mass index, plasma glucose concentration and so on. Again for multi-dimensional data sets such as this one visualization is not trivial. Thus classifiers will once more be compared using their roc-curves. A linear and radial basis function svm has been trained. Their roc-plots are shown in figure 1.23. Again the two classifiers perform similarly with the linear classifier being correct in 74% of cases and the radial basis function delivering correct classification in 77.8% of all validation cases.

[14] `http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.names`

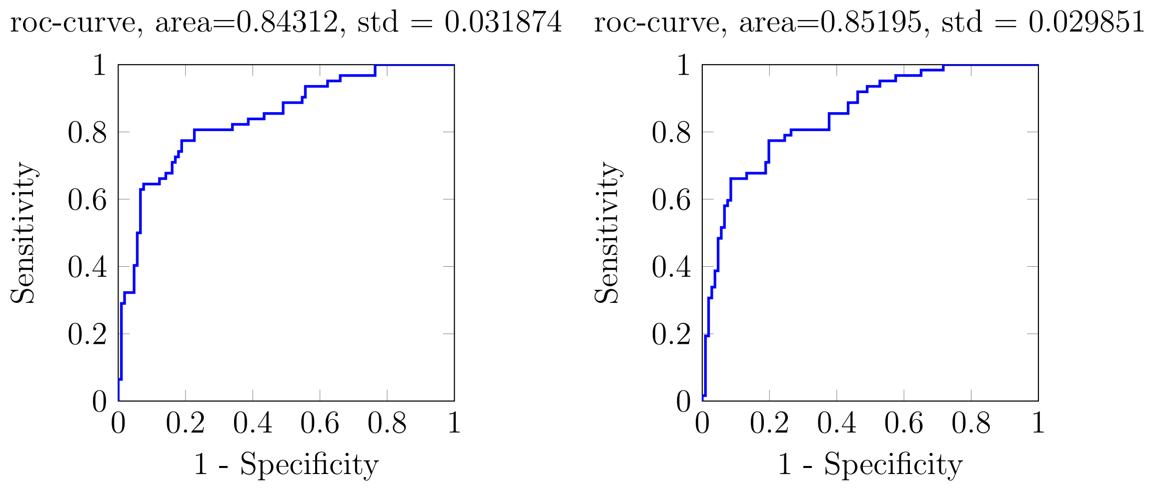


Figure 1.23: Linear and radial basis function lssvm classifier roc-curves for the diabetes data set.

Session 2

Function Estimation and Time-series Prediction

2.1 Regression Support Vector Machines.

The support vector methodology is not limited to classification problems. [1] In the primal space a model funcion of the form: [2]

$$f(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + b \tag{2.1}$$

With annotated training samples $\{x_k, y_k\}$ and a nonlinear mapping $\varphi(\cdot) : \mathbb{R}^n \to R^{n_h}$, which is only implicitly defined. This definition leads to the primal optimization problem:

$$\min_{w,b,\xi,\xi^*} J_p = \frac{1}{2}\mathbf{w}^T\mathbf{w} + c\sum_{k=1}^{N}(\xi_k + \xi_k^*) \tag{2.2}$$

$$\text{such that } y_k - \mathbf{w}^T\varphi(\mathbf{x}_k) - b \leq \epsilon + \xi_k, k = 1, \ldots, N \tag{2.3}$$

$$\mathbf{w}^T\varphi(\mathbf{x}_k) - b - y_k \leq \epsilon + \xi_k^*, k = 1, \ldots, N \tag{2.4}$$

$$\xi, \xi_k \geq 0, k = 1, \ldots, N \tag{2.5}$$

After taking and solving the dual the representation

$$f(\mathbf{x}) = \sum_{k=1}^{N}(\alpha_k - \alpha_k^*)K(\mathbf{x}, \mathbf{x}_k) + b \tag{2.6}$$

of the model is obtained. The σ^2, c and ϵ are user parameters, which do not follow from a QP, but should be determined by cross-validation for example. A closer look at ϵ is taken in figure 2.1, which has been generated using the `uiregress` function. [3] Here c and $\sigma^2 = 0.1$ are kept constant, while ϵ is varied. In the leftmost plot $\epsilon = 0.1$ is used, then increased to $\epsilon = 0.25$ and finally raised to $\epsilon = 0.5$. Being similar to the ϵ-tube for Lipschitz continuous functions the ϵ-parameter sets up a tube around the original function. In this case the tube is not used for measuring continuity but to set up a tolerance region around

[1] Support Vector Machines: Methods and Applications, Suykens et al., page 53

[2] Support Vector Machines: Methods and Applications, Suykens et al., page 54

[3] SVM toolbox `www.isis.ecs.soton.ac.uk/isystems/kernel/svm.zip`

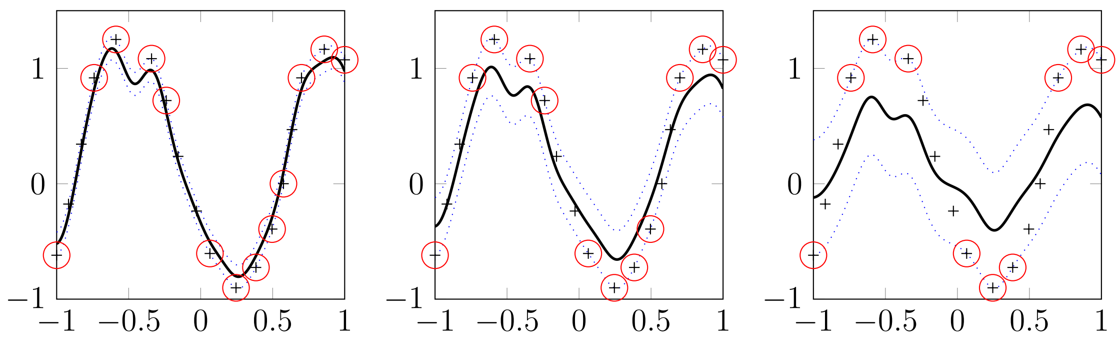


Figure 2.1: RBF kernel based funtion esimation with $\sigma^2 = 0.1$, $\epsilon = 0.1$, $\epsilon = 0.25$ and $\epsilon = 0.5$.

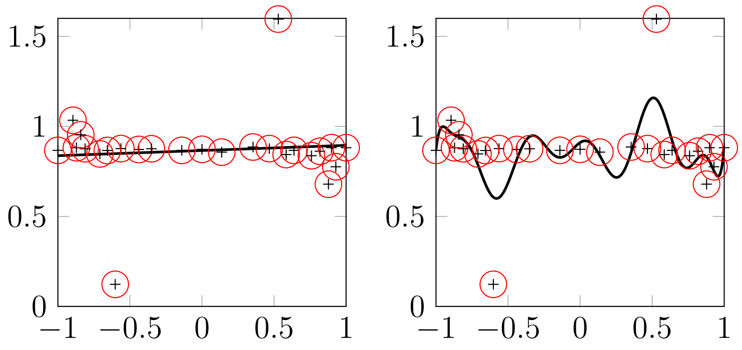


Figure 2.2: On this data set the linear kernel is able to outperform a radial basis function kernel, which is more susceptible to the noisy outliers.

a function which leads to the approximation following the data less closely. A look at the optimization problem constraint function formulation confirms this. Only for small ϵ does the problem become infeasible and the slack variables necessary. Errors within the ϵ-tube don't count into the error term of the cost function. Figure 2.1 also suggests that larger values for ϵ reduce the number of support vectors. Figure 2.2 shows an example of an approximately line-like function with some outliers. In this case a simple linear kernel is a good choice, as it is unable to track the added complexity of the noise. An rbf-function kernel is able to integrate the noise into the model, but in this example one would like to avoid this. However it is possible to fundamentally improve the rbf kernel results if the parameters σ, c[4], and ϵ are changed more towards regularization. Most notably the parameter c, which determines the tolerance for deviation from the desired ϵ-accuracy can be used to improve the performance of the rbf-kernel. If c is set to a comparatively large value with ϵ kept small, results improve. A good choice will use a moderate value for ϵ to reduce the amount of support vectors and thereby increase the sparsity of the solution vector[5], while using c to make sure the solution is not to prone to deviations introduced by outliers.

[4] c is the regularization parameter. It is called γ in svm toolbox functions.

[5] Support Vector Machines: Methods and Applications, Suykens et al., page 33

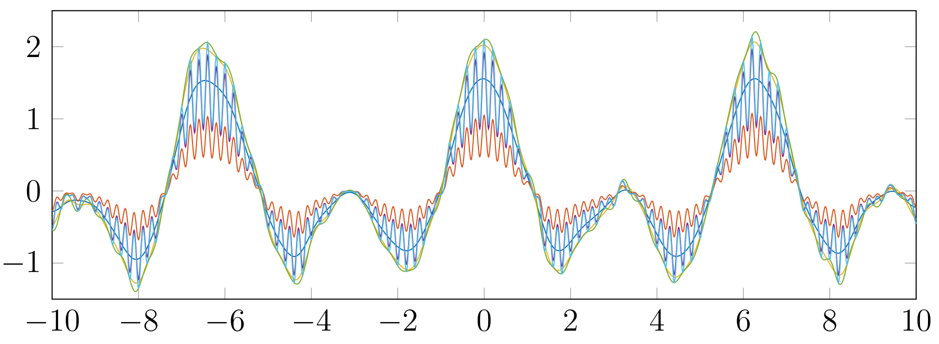


Figure 2.3: Estimation results using all combinations of $\sigma^2 \in \{0.0018738, 0.0001\}$ and $\gamma \in \{1.0, 10.2353, 10000\}$ as inputs to function estimation svms.

2.1.1 Sum of cosines

As an initial noise free example function:

$$f(x) = \cos(x) + \cos(2x) \tag{2.7}$$

will be considered. Finding a good set of hyper-parameters by hand can be tricky as figure 2.3 illustrates. The approximation tends to oscillate if σ^2 is chosen very small. This can be explained using equation 2.6, which states that the estimator is created by superposing weighted and shifted kernel functions. In this case Gaussians are added up. If their width (σ^2) becomes small their sum must oscillate, because the Gaussians will not overlap sufficiently. The height of the gaussians is controlled by the regularization parameter γ, as it acts like a box constraint when the dual is formed. In the primal gamma is the weighting term of the deviations from the original function values. Thus, when γ is chosen too small the kernel functions can not be tall enough to approximate the function. To make a good choice, a hundred by hundred grid in $\sigma^2 \in [10^{-5}, 10^0]$ and $\gamma \in [10^0, 10^{10}]$ has been searched using the two norm of the error vector $(\mathbf{y}_{est} - \mathbf{y})$ as cost function. The error function on the described input space as well as the estimator with the lowest error is shown in figure 2.4. The grid search led to the optimal parameter pair $\sigma^2 = 0.0019$ and $\gamma = 10.2353$. Estimation results are shown in the middle of figure 2.4 training set performance is plotted on the left. Next a noisy version of $f(x)$ is considered:

$$f(x)_n = \cos(x) + \cos(2x) + x_n \tag{2.8}$$

With the values for x_n drawn from the normal distribution $\mathcal{N}(0, 0.1^2)$. The changed situation is shown in figure 2.5. When noise is added on average the optimal values for σ^2 must be chosen larger to avoid over-fitting.

2.1.2 Hyper-parameter tuning

Just like it was possible to automatically search the hyper-parameter space for good tuning parameters in the classification setting, it is possible in the function estimation case. One option is certainly to use a brute fore grid based approach. Figure 2.7 shows the cost function within the space $\sigma^2 \in \{0.0018738, 0.0001\}$ and $\gamma \in \{1.0, 10.2353, 10000\}$ on a hundred by hundred grid. The cost function $\log_{10}(\|\mathbf{y}_{\text{est}} - \mathbf{y}\|_2)$ is shown logarithmically on the z-axis. The plot reveals that this problem might not be convex. Several possible

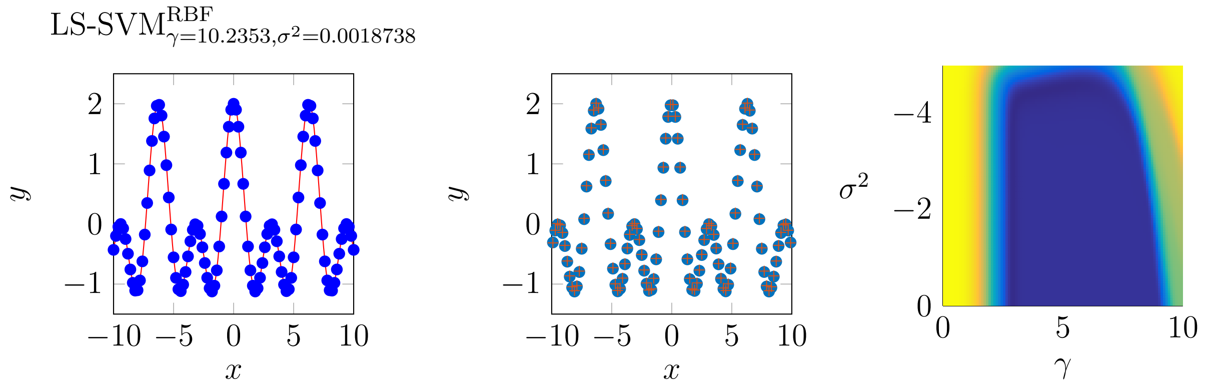


Figure 2.4: Plot of the optimal estimation of the noise-free function $f(x)$. The right plot shows the approximation on training data, the middle one performance on validation data. The blue dots are function values the red crosses show approximation values. Finally the plot on the right shows the explored hyperparameter-space and corresponding two norms of the error vector $(\mathbf{y}_{est} - \mathbf{y})$.

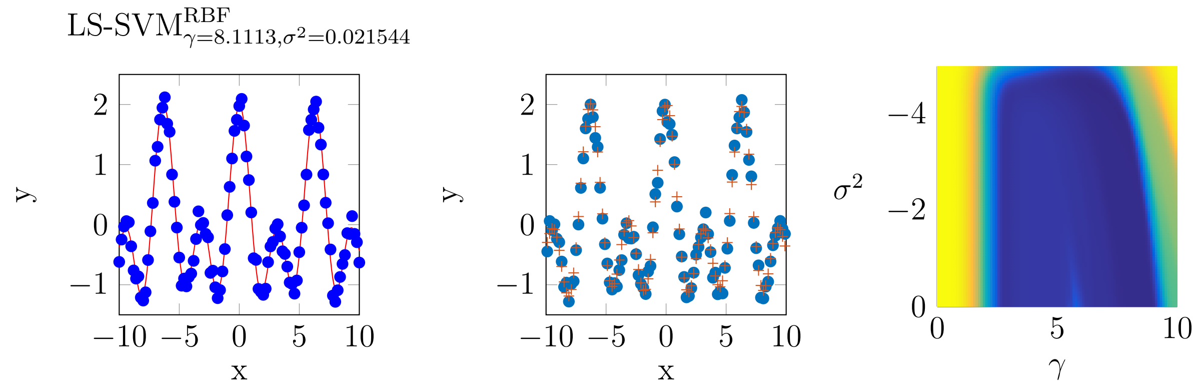


Figure 2.5: Plot of the optimal estimation of the noisy function $f_n(x)$. The right plot shows the approximation on training data, the middle one performance on validation data. Finally the plot on the right shows the hyper-parameter-space and corresponding two norms of the error vector $(\mathbf{y}_{est} - \mathbf{y})$.

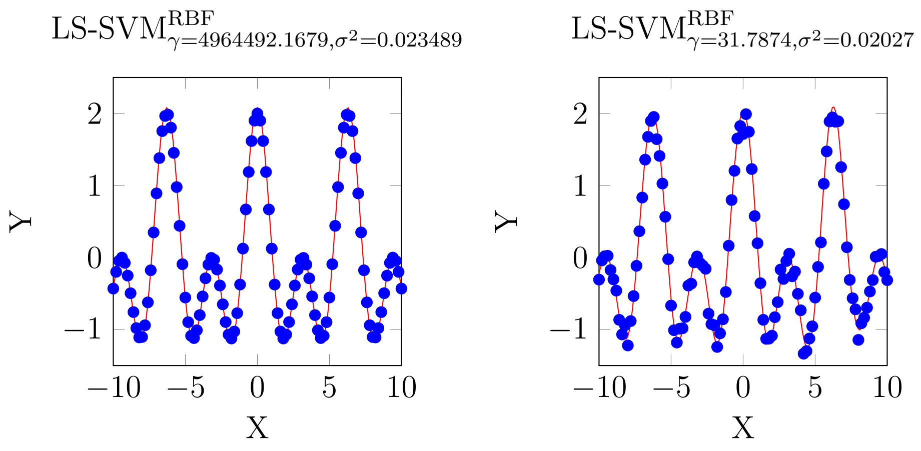


Figure 2.6: Exemplary tuning results on noise free and noisy data.

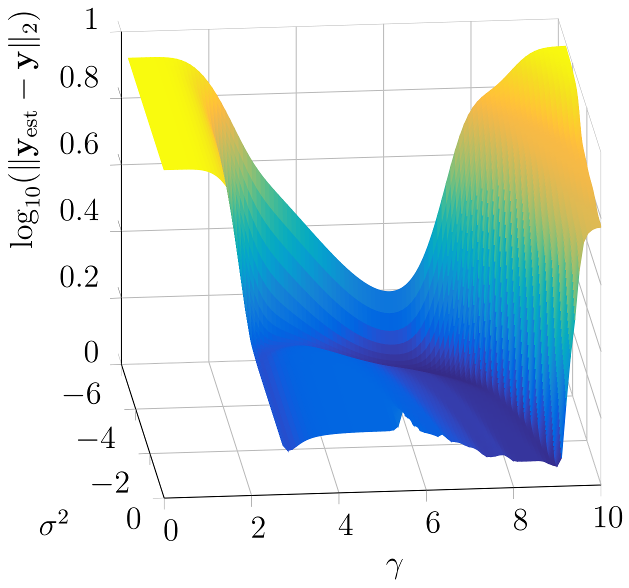


Figure 2.7: A 3d logarithmically scaled plot of the cost in the hyper-parameter space of the noisy function estimation problem.

local minima exist where a pure gradient based optimization approach could get stuck. Therefore a global optimization method is needed as a remedy, locally standard methods like the simplex method can be employed. In a first experiment the global optimization algorithm Coupled Simulated Annealing and local simplex-optimization are used. If noise is present the optimization algorithm must choose the regularization constant smaller to stress model complexity reduction. If no noise is present good fit of the model can be stressed, as can be seen in figure 2.6.

Table 2.1 shows the average time of 10 different tuning function runs. It can be said that simplex optimization is faster then a grid based approach and randomized directional search is faster then simulated annealing (if the startup time of the parallel pool is neglected). In terms of accuracy of the found solution no significant differences could be found on average, see table 2.2. Which means that all method combinations find good parameters in this case in general.

	csa	ds
simplex	0.6765	0.5944
grid	0.9788	0.9062

Table 2.1: Average time of a single tuning function execution. The average was computed from running each combination ten times on the noisy cosine function estimation problem.

	csa	ds
simplex	0.0018	0.0019
grid	0.0019	0.0018

Table 2.2: Average cross-validation-cost of a the parameters found by the tabulated algorithm combinations. Ten folds have been used for cross validation and the mean squared error served as cost function.

2.1.3 The Bayesian Framework

Bayesian inference is a layered process. For classification the modified primal problem,

$$\min_{w,b,e_c} J_p(w, e_c) = \mu \frac{1}{2} \mathbf{w}^T \mathbf{w} + \zeta \frac{1}{2} \sum_{k=1}^{N} e_{c,k}^2 \tag{2.9}$$

$$\text{such that } y_k[\mathbf{w}^T \varphi(\mathbf{x}_k) + b] = 1 - e_{c,k}, k = 1, \ldots, N \tag{2.10}$$

is used which eventually leads to the dual space classifier[6]

$$y(x) = \text{sign}[\frac{1}{\mu} \sum_{k=1}^{N} \alpha_k K(x, x_k) + b] \tag{2.11}$$

It follows that the primal weight space parameters w, b the hyper-parameters μ, ζ and finally the kernel parameters (if any) have to be determined. The hyper-parameters μ, ζ are a different way to express γ, which was used in previous formulations. The relation $\gamma = \frac{\zeta}{\mu}$ holds [7], which can be checked by substituting $\mu = 1$. The unknowns will be found following a three layered process. The first layer focuses on the weight vector and the bias terms defining $\mathcal{D} = \{x_k, y_k\}_{k=1}^{N}$ and using $\mathcal{H}_i$ to denote different models i.e. radial basis function kernels with different width. The first level uses the equation:[8]

$$p(\mathbf{w}, b|\mathcal{D}, \mu, \zeta, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mathbf{w}, b, \mu, \zeta, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\sigma)} p(\mathbf{w}, b|\mu, \zeta, \mathcal{H}_\sigma) \tag{2.12}$$

With $p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\sigma)$ denoting the evidence which is determined by integrating over all possible values of $\mathbf{w}, b$ and normalizing the result. $\mathbf{w}, b$ follow from the first level, which is then used in the second layer to find μ and ζ by maximizing:

$$p(\mu, \zeta|\mathcal{D}, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mathcal{H}_\sigma)} p(\mu, \zeta|\mathcal{H}_\sigma). \tag{2.13}$$

[6]Support Vector Machines: Methods and Applications, Suykens et al., page 119
[7]Support Vector Machines: Methods and Applications, Suykens et al., page 118
[8]Support Vector Machines: Methods and Applications, Suykens et al., page 122

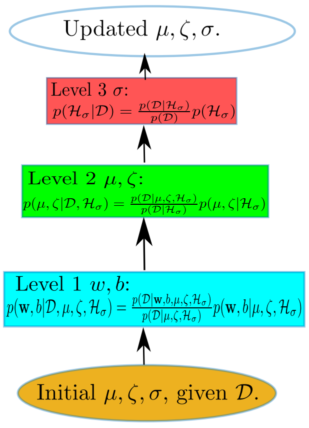

Updated μ, ζ, σ.
Level 3 σ:
$p(\mathcal{H}_\sigma|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_\sigma)}{p(\mathcal{D})} p(\mathcal{H}_\sigma)$
Level 2 μ, ζ:
$p(\mu, \zeta|\mathcal{D}, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mu,\zeta,\mathcal{H}_\sigma)}{p(\mathcal{D}|\mathcal{H}_\sigma)} p(\mu, \zeta|\mathcal{H}_\sigma)$
Level 1 w, b:
$p(\mathbf{w}, b|\mathcal{D}, \mu, \zeta, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mathbf{w},b,\mu,\zeta,\mathcal{H}_\sigma)}{p(\mathcal{D}|\mu,\zeta,\mathcal{H}_\sigma)} p(\mathbf{w}, b|\mu, \zeta, \mathcal{H}_\sigma)$
Initial μ, ζ, σ, given $\mathcal{D}$.

Figure 2.8: Bayesian inference schematic.

Which leads to the lowest third level where the kernel parameters are found:

$$p(\mathcal{H}_\sigma|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_\sigma)}{p(\mathcal{D})} p(\mathcal{H}_\sigma). \tag{2.14}$$

For function estimation the picture is similar, but a couple of small modifications have to be made. For example assuming a hyper-parameter vector $\zeta_{1 \ldots N} = [\zeta_1, \ldots, \zeta_N]$. Using Bayesian inference also makes it possible to compute error bounds. The model found from the initial parameters $\sigma^2 = 0.01$, $\gamma = 10$ is shown in figure 2.8. It is not perfect but all training points lie indeed inside the confidence interval. Choosing an initial point closer to the global optimum will probably results in better parameters σ and γ.
A good example for binary classification is the simplified iris data set. Using bayesian inference the posterior class probabilities can be estimated. This has been done for a classifier trained using $\gamma = 5$ and $\sigma^2 = 0.75$. The result is shown in figure 2.10. The purple indicates that the classifier will place points in this area in the positive class, while data in the blue area will be placed in the negative class. Like it was observed earlier the regularization parameter reduces model complexity for low values and stresses good fit to the data for high values. The kernel density on the other hand stresses fit for small values and model complexity reduction for larger widths.

Bayesian inference can also be used for input relevance detection. This is done using radial basis function kernels with a different width for each kernel on the third level of inference. Inputs which end up with small kernel width are successively removed as small density is a strong indication for data memorization. Figure 2.11 shows three different random inputs, while the first one was used to compute the noisy target function f_n. Clearly it is the most relevant input. And Baysian inference comes to the same conclusion. Input selection can also be done using a layered cross-validation approach. On way to do it is eliminate one input from the data and compute the value of the cross-validation mean squared error. If each input is neglected once the most important one will be the input for which the cost rose most when eliminated. In this example the cost values $1.2928, 1.0031, 1.0083$ where found when the first, second, and third input where eliminated. It can be concluded that the first input is the most important is its absence leads to higher cost as if any of the other inputs are missing. Another method is to only keep the input in question, in this case the cost values $0.5132, 1.2910, 1.2578$ where found. This confirms what was found earlier.

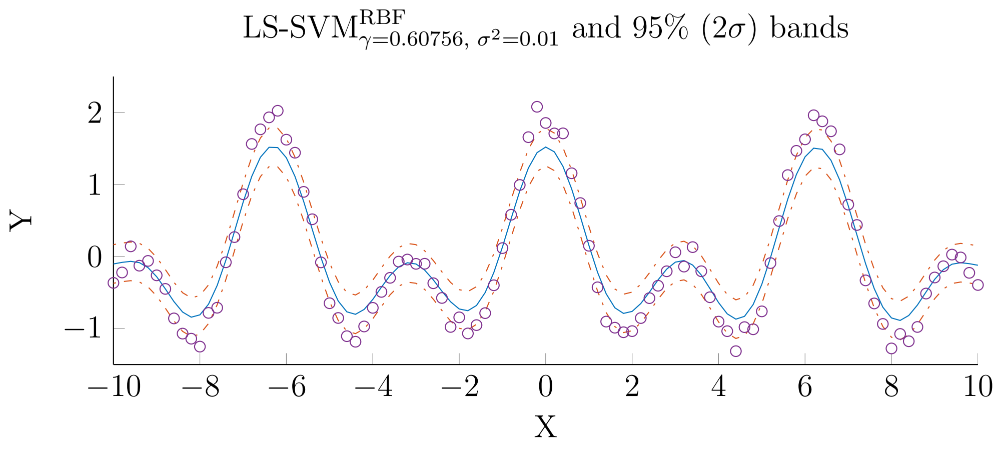


Figure 2.9: Function estimation with parameters from Baysean inference and confidence bounds.

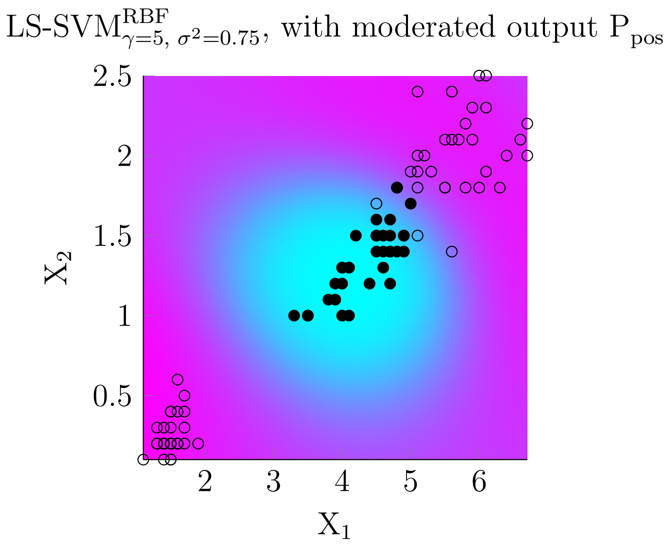


Figure 2.10: Posterior class probabilities of an iris data classification svm

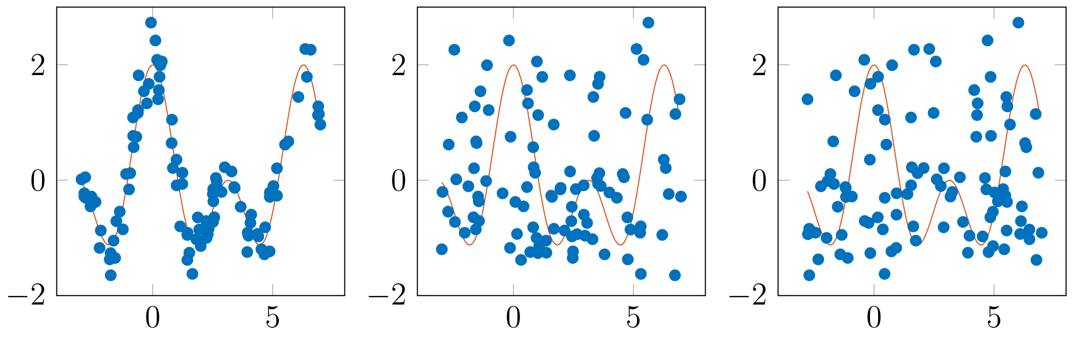


Figure 2.11: Plots of three different input sequences and their corresponding output value. With the generating function overlaid.

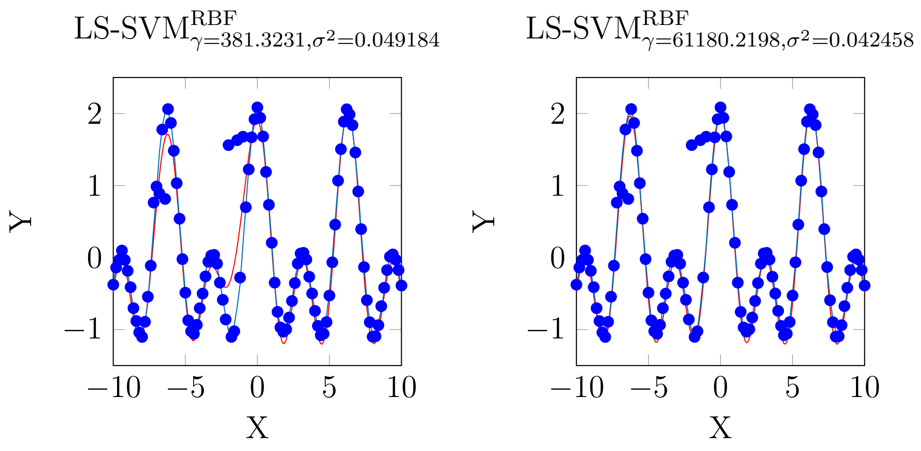


Figure 2.12: Standard svm estimation and its robust version using Huber weights.

2.1.4 Robust Regression

In order to make svm regression more robust with respect to outliers in the data, a weighted approach is used.[9] The primal optimization problem is slightly modified:

$$\min_{\mathbf{w}^*,b^*,e^*} J_p(\mathbf{w}^*,e^*) = \frac{1}{2}\mathbf{w}^{T^*}\mathbf{w} + \gamma\frac{1}{2}\sum_{k=1}^{N} v_k e_k^{*2} \tag{2.15}$$

Using the weight function[10]

$$v_k = \begin{cases} 1, & \text{if } |e_k/\hat{s}| \le c_1 \\ \frac{c_2-|e_k/\hat{s}|}{c_2-c_1}, & c_1 \le \text{if } |e_k/\hat{s}| \le c_2 \\ 10^{-4}, & \text{otherwise} \end{cases}. \tag{2.16}$$

The constants c_1 and c_2 are typically chosen as $c_1 = 2.5$ and $c_2 = 3$ in the literature. [11] Finally $\hat{s}$ must be a robust estimate of the standard deviation and e_k is given as $e_k = \alpha_k/\gamma$. Figure 2.12 shows the difference between the standard approach and the more robust weighted one, which handles the noise better. Other weight functions such as logistic- or hampel-weights perform similarly.

2.2 Santa Fe laser Time series prediction

The santa fe laser competition data stems from a 1989 paper, physicists measured the intensity of a unidirectional far infrared NH_3 laser[12]. In physics these types of lasers are modeled using so called Lorenz-Haken models. These models are chaotic systems, with their Lyapunov exponent determining the speed by which trajectories starting at similar initial conditions diverge. In practice the quality any approximation made using Lorenz-Haken models depends on the initial condition and the constants used in the model. Figure 2.13 shows a screen-shot from the original paper and a plot of the part of

[9] Support Vector Machines: Methods and Applications, Suykens et al., page 154
[10] Support Vector Machines: Methods and Applications, Suykens et al., page 155
[11] Support Vector Machines: Methods and Applications, Suykens et al., page 155
[12] U. Hübner et al, Dimensions and entropies of chaotic intensity pulsations in a single-mode far-infrared NH3 laser, `http://journals.aps.org/pra/pdf/10.1103/PhysRevA.40.6354`

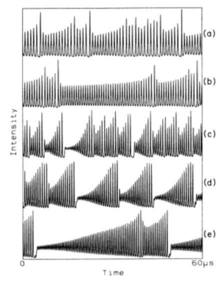


FIG. 2. Spiral-type pulsing of the laser intensity. The pressure varied for cases (a)–(e) from 8 to 10 Pa and the pump intensity was about 14 times above threshold. Resonator tuning was kept as constant at line center as experimentally possible. 1500 samples were plotted per trace. The average pulsing frequency increases from about 1.05 MHz (about 25 samples per pulse) for trace *a* to about 1.7 MHz (about 15 samples per pulse) for trace *e*. Successive traces are offset vertically for clarity.

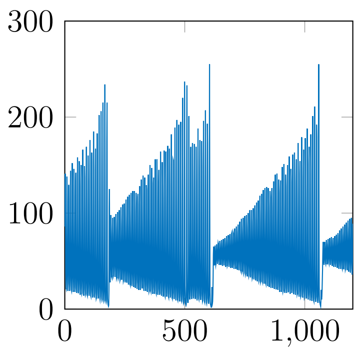


Figure 2.13: Data set measured by Hübner et al. and the subset selected for the Santa Fe competition.

the measurements selected for the santa-fe competition. Instead of trying to figure out a good initial condition, as well as reliable parameters for a Lorenz-Haken model a ls-svm in recurrent mode will be used to model this problem. In the autonomous case of a recurrent model;

$$\hat{y}_k = f(\hat{y}_{k-1}, \hat{y}_{k-2}, \hat{y}_{k-3}, \dots) \tag{2.17}$$

a recurrent support vector machine can be used to model the system dynamics, given a starting value and windowed training data. Windowing the data means placing it into a Hankel matrix, where the rows represent shifted versions of the input. The ls-svm approach then uses the optimization problem: [13]

$$\min_{\mathbf{w},b,e} J_p(\mathbf{w}, e) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \gamma\frac{1}{2}\sum_{k=p+1}^{p+N} e_k^2 \tag{2.18}$$

$$\text{such that } y_k - e_k = \mathbf{w}^T\varphi(y_{k-1_k-p} - e_{k-1_k-p}) + b \quad k = k \dots N. \tag{2.19}$$

The problem states that the approximation error $e_k = y_k - \hat{y}_k$ should be reduced while enforcing the recurrent model dynamics.

In order to choose the window size the sample autocorrelation function shown in figure 2.14 is considered. In the plot the autocorrelation values start to come close to the confidence bounds around a lag of fifty samples. Therefore a window size of fifty will be used as this shift covers the most significant samples. The plot on the right of figure 2.14 confirms this choice as reasonable when a window size of fifty samples is reached the error has already fallen over three decades. From fifty to hundred the error falls only by about one more decade. Thus choosing fifty as window size covers the most important part of the information present.

Figure 2.15 shows recurrent svm approximation using automatically tuned hyper-parameters found trough a coupled simulated annealing, simplex optimization algorithm pair. Ten fold cross-validation was used in order to evaluate the absolute error cost function. Results using this cost function have been significantly better then using a mean square error function or infinity norm based cost. A look at figure 2.16, verifies the hyper-parameter choice determined by global optimization. It also reveals that 10 fold cross-validation is a good indicator of prediction performance quality in this case. Another way to obtain a

[13]Support Vector Machines: Methods and Applications, Suykens et al., page 225

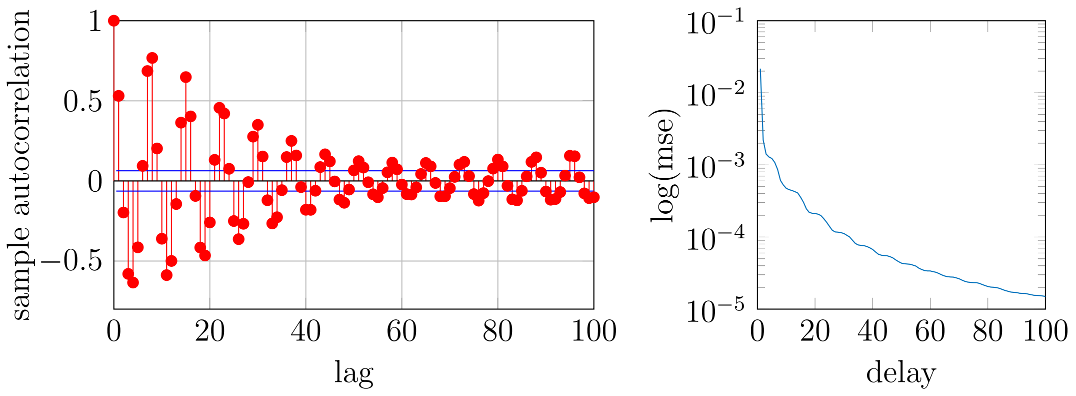


Figure 2.14: Autocorrelation analysis of the training time-series shown with 95% confidence bounds (left). Training set error using an svm trained using $\gamma = 75.5, \sigma^2 = 27.8$ and a linearly increasing window size.

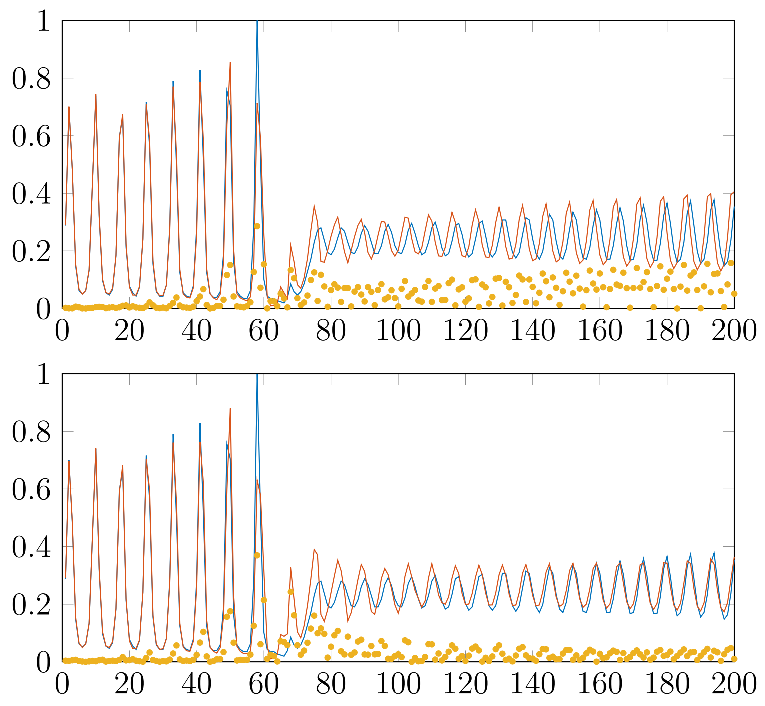


Figure 2.15: Recurrent ls-svm approxmation of scaled Santa-Fee using the hyper-parameters $\gamma = 158.5795, \sigma^2 = 23.35559$ (top) and $\gamma = 75.4764, \sigma^2 = 27.7826$ (bottom) found by using 10 fold cross-validation and numerical optimization techniques. The blue curve shows the validation data set, the red one the svm-approximation. Yellow dots indicate the error at any given point.

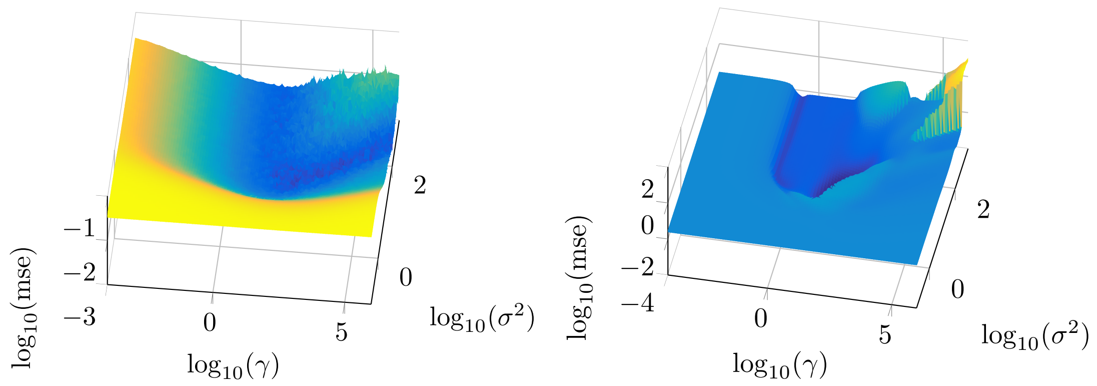


Figure 2.16: Plot of 10 fold cross-validation mean squared cost (left) and prediction mean squared error (right) in the hyper-parameter space.

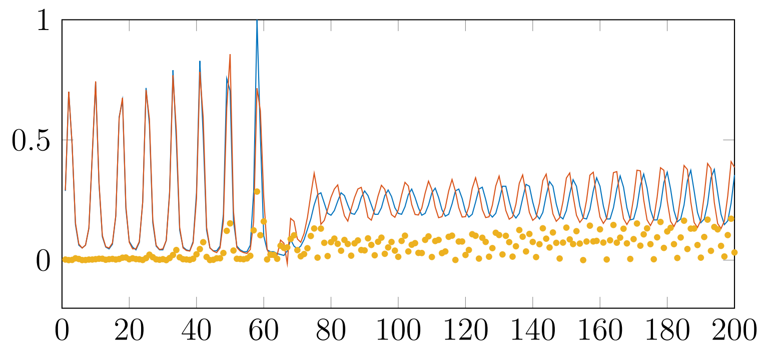


Figure 2.17: Prediction performance using the set of hyperparameters $\{\gamma = 270.0, \sigma^2 = 30\}$ as well as support vectors and bias term found using Bayesian inference.

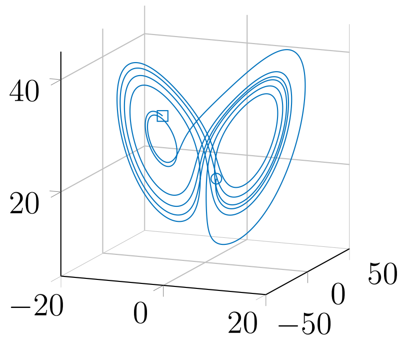


Figure 2.18: Runge-Kutta (`ode45`) simulation of lorenz' strange attractor.

good set of hyper-parameters is using Bayesian inference. Starting from an initial guess of $\gamma = 1000$ and $\sigma^2 = 30$ and improving these values going trough all three layers of inference the parameters $\gamma = 270.0$ and $\sigma^2 = 30$ as well as support vectors and bias term are obtained. The performance of an svm using these values is shown in figure 2.17.

2.3 Lorenz equation estimation

In the previous experiment svms where able to predict about 200 samples with reasonable accuracy. But the question remains what happens when more samples are predicted. Will the solution deteriorate or will the svm continue to produce behavior that is characteristic for the dynamical system under consideration? In order to answer this question the lorenz-haken data used during the previous experiment is replaced by data from simulating the well known Lorenz equations [14]

$$\dot{X} = a(Y - X) \tag{2.20}$$
$$\dot{Y} = X(b - Z) - Y \tag{2.21}$$
$$\dot{Z} = XY - cX \tag{2.22}$$

With the constants $a = 10.0$, $b = 28.0$, and $c = 8.0/4$. Using the initial condition $(2\ 1\ 20)^T$ leads to the solution shown in figure 2.18 after Runge-Kutta simulation with a maximum time step of 0.01, which will serve as a data source. Before svm training the input data points are flattened into a vector of the form $\mathbf{v}_{\text{in}} = x_1, y_1, z_1, x_2, y_2, \dots,$ which is then windowed with a 333 sample sized window. The window size was determined by looking at the autocorrelation function as described before. An svm with parameters $\gamma = 100$ and $\sigma^2 = 10$ has been used. The parameters have been determined by trail and error. Figure 2.20 shows the evolution of the solution for 3333 flattened data points or 1111 three dimensional points. The start and end of each curve are denoted by a circle and square respectively. The ls-svm prediction does diverge from the Runge-Kutta solution, but continues to exhibit typical Lorenz equation behavior. Figure 2.21 shows the same experiment with an svm trained using $\gamma = 146040$ and $\sigma^2 = 10$, which where found using Bayesian inference. The larger γ places lesser emphasis on regularization, which leads to a longer period of accurate prediction, at the cost of an complete breakdown of the prediction process at roughly 1500 flattened samples.

[14]Wikipedia, `https://de.wikipedia.org/wiki/Lorenz-Attraktor`

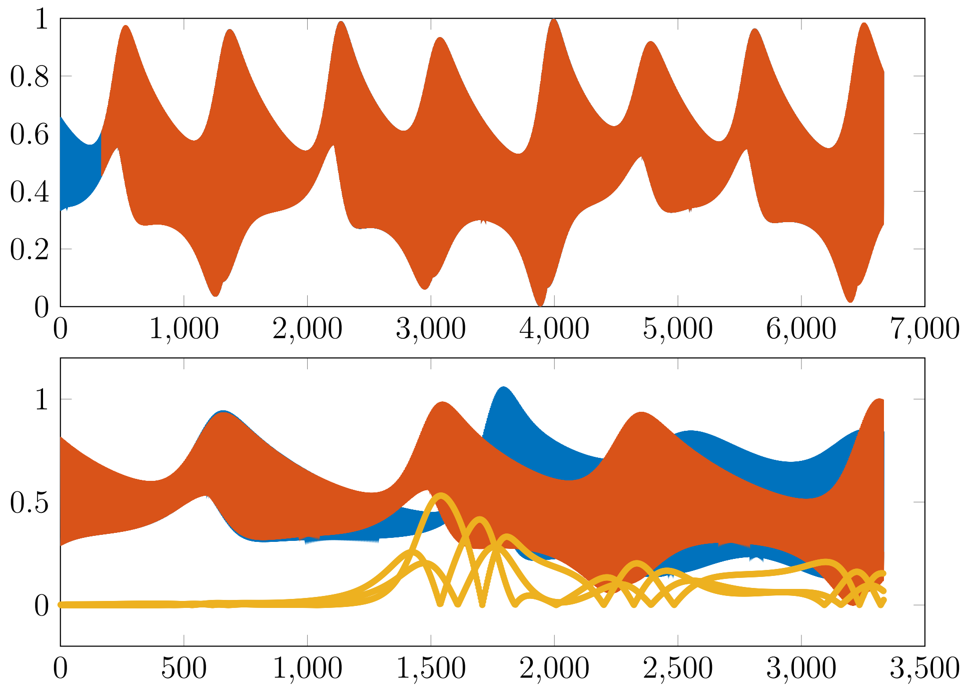


Figure 2.19: LS-SVM fit to training data and recurrent prediction results in flattened form. Note how the three dimensionality of the input appears in the error.

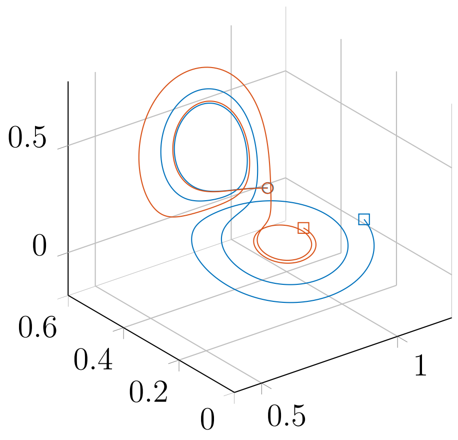


Figure 2.20: LS-SVM prediction (red) and Runge-Kutta simulation data (blue).

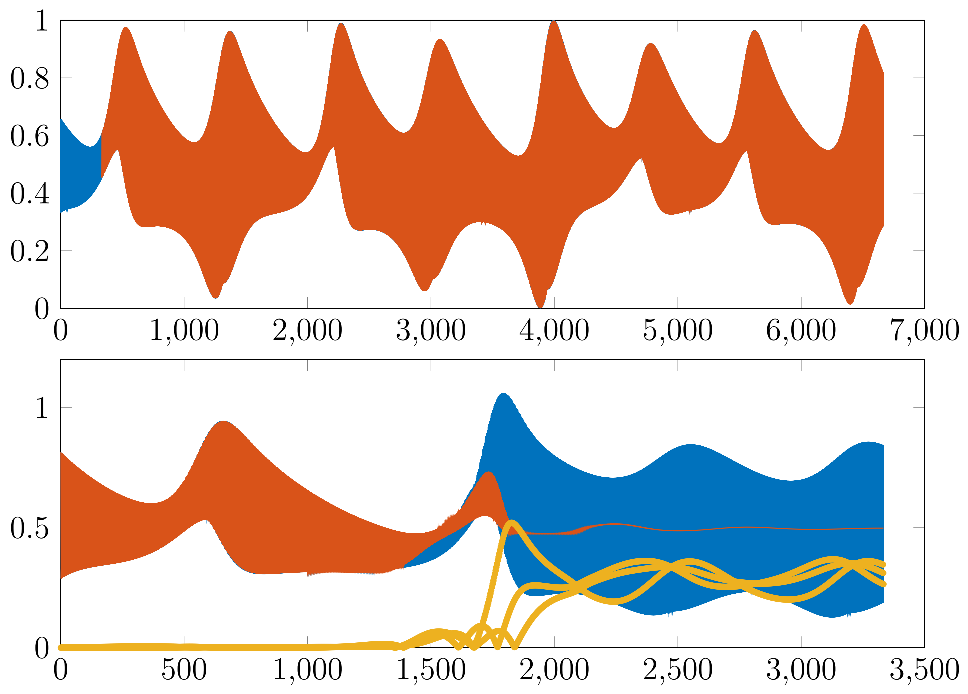


Figure 2.21: Bayesian inference parameter LS-SVM fit to training data and recurrent prediction results in flattened form. Note how the three dimensionality of the input appears in the error.

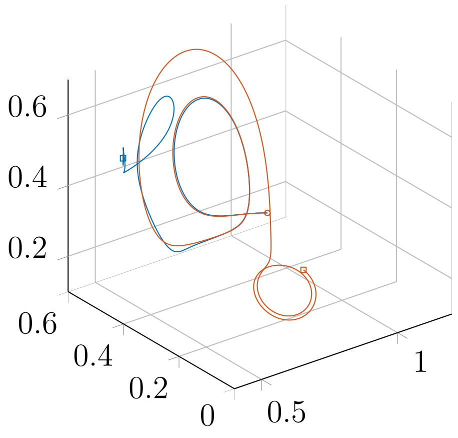


Figure 2.22: Prediction results with Bayesian inference parameters.

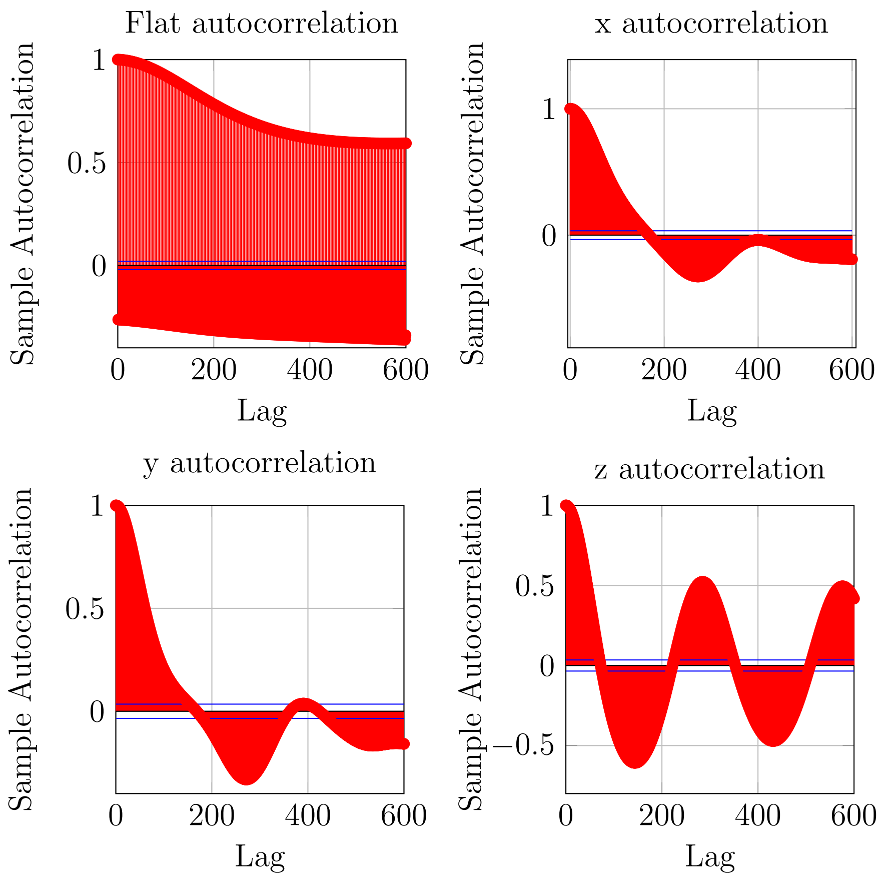


Figure 2.23: Autocorrelation of the flattened three dimensional as well as x,y and z related training data sets.

2.3.1 A Lorenz SVM commitee

The comparison of the flattened autocorrelation with the single dimension couterparts shown in figure 2.23 reveals that possibly the three dimensions are best treated separately. Therefore an attempt to train a Lorenz svm committee-network has been made. One svm per dimension is trained, which means one svm per Lorenz equation. The current version of the `lssvm toolbox` requires an `X : N x d matrix with the inputs of the training data` as well as an `Y : N x 1 vector with the outputs of the training data.` [15] Which means no tensor inputs are allowed. As the columns of the input are required for the windowing a flattened version of the 3 dimensional input has to be used as a workaround. In order to meet the length requirement the unwanted dimensions of the output have been padded with zeros. The flattened predictions have then been added up and reshaped into their original three dimensional form. But unfortunately this approach did not improve results. This is probably due to the fact that zero padding the unwanted output dimensions introduces new false information, from which the Committee-network does not recover.

[15]LS-SVMlab Toolbox Users Guide version 1.8, K. De Brabanter et al. page 104, `http://www.esat.kuleuven.be/sista/lssvmlab/downloads/tutorialv1_8.pdf`

Session 3

Unsupervised Learning and sparsity

3.1 Kernel principle component analysis

Linear component analysis works by finding the main axes which describe a given data set best. Linear combinations of these principal component vectors can then be used to recreate each point in the original data set. In practice linear PCA operates on the mean:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k. \tag{3.1}$$

Next from each data point's deviation from the mean the covariance matrix is computed:[1]

$$\Sigma = \frac{1}{N-1}\sum_{k=1}^{N}(\mathbf{x}_k - \bar{\mathbf{x}})(\mathbf{x}_k - \bar{\mathbf{x}})^T \tag{3.2}$$

The linear principle components are defined to be the eigenvectors $\mathbf{u}$ of the covariance matrix Σ;

$$\Sigma\mathbf{u} = \lambda\mathbf{u}. \tag{3.3}$$

If the eigenvectors are stored in a matrix U,

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{Ub} \tag{3.4}$$

can be used to recombine the prinipal components to match points close to the original data set. By enforcing limits on the weight vector $\mathbf{b}$ the closeness of remaps to the original data can be controlled. [2] Another important application of linear PCA is dimensionality reduction. An reduction of input dimension is achieved by considering the remapped data,

$$\mathbf{z}_i = \mathbf{u}_i^T(\mathbf{x} - \bar{\mathbf{x}}_i) \;\; i = 1, \ldots, m. \tag{3.5}$$

With $m < N$, the dimension is reduced by neglecting the smallest eigenvalues and their corresponding eigenvectors. This method allows to reduce the input data complexity, while keeping the information loss as small as possible. Figure 3.1 shows a classical coordinate system consisting of the principal axes centered at the mean. The data set

[1]Support Vector Machines: Methods and Applications, Suykens et al., page 20

[2]Active Shape Models - Their training and Application, Cootes et al, pages 43,44 and 49

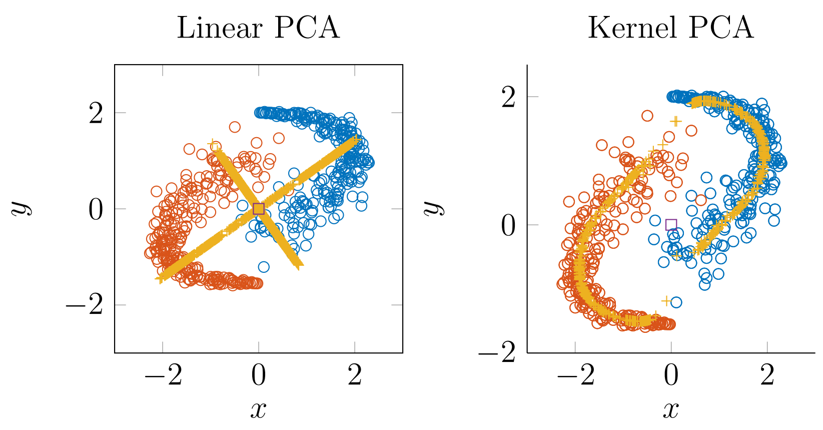


Figure 3.1: Linear PCA showing two principal components(left) and kernel PCA (right) . The mean sample value $\bar{\mathbf{x}}$ is indicated by a square at $(0, 0)$ in both plots.

under consideration is nonlinear, therefore the linear approach fails to capture essential information present in the data.

On the right of figure 3.1 the result of using a non-linear approach is shown. The kernel-PCA is able to track the nonlinear lines despite the presence of progressively increasing noise. The linear PCA can also be formulated as an optimization problem [3]:

$$\max_{\mathbf{w}} \operatorname{Var}(\mathbf{w}^T\mathbf{x}) = \operatorname{Cov}(\mathbf{w}^T\mathbf{x}, \mathbf{w}^T\mathbf{x}) \simeq \mathbf{w}^T C\mathbf{w} \tag{3.6}$$

which is valid, when considering zero mean data and using $\mathbf{C} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{x}_k\mathbf{x}_k^T$. Taking into account the constraint $\mathbf{w}^T\mathbf{w} = 1$ leads to the Lagrangian:

$$\mathcal{L}(w; \lambda) = \frac{1}{2}\mathbf{w}^T\mathbf{C}\mathbf{w} - \lambda(\mathbf{w}^T\mathbf{w} - 1) \tag{3.7}$$

The solution of the problem will be found where the gradient of the Lagrangian is zero $\bigtriangledown\mathcal{L} = 0$. Using this equation yields the eigenvalue problem,

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}. \tag{3.8}$$

Which can be solved as discussed before. The kernel case is an extension of the linear case [4] the mean equivalent is defined by,

$$\hat{\mu} = \frac{1}{N}\sum_{k=1}^{N} \varphi(\mathbf{x}_k). \tag{3.9}$$

The ls-svm approach to the problem looks at:

$$\max_{\mathbf{w},\mathbf{e}} J_p(\mathbf{w}, \mathbf{e}) = \gamma\frac{1}{2}\sum_{k=1}^{N} e_k^2 - \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{3.10}$$

$$\text{such that } e_k = \mathbf{w}^T(\varphi(\mathbf{x_k} - \hat{\mu})), \quad k = 1, \ldots, N \tag{3.11}$$

[3]Support Vector Machines: Methods and Applications, Suykens et al., page 202
[4]Support Vector Machines: Methods and Applications, Suykens et al., page 211

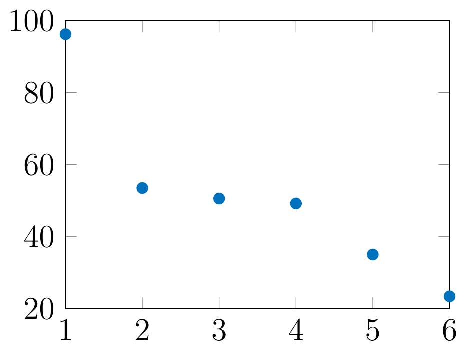


Figure 3.2: Plot of the first six eigenvalues of the centered kernel matrix.

Trough an analysis of the optimality conditions once more an equivalent eigenvalue problem can be found,

$$\Omega_c \alpha = \lambda \alpha. \tag{3.12}$$

With $\lambda = \frac{1}{\gamma}$ and $\alpha_k = \gamma e_k$. And the kernel matrix defined by,

$$\Omega_{c;k,l} = (\varphi(\mathbf{x}_k) - \hat{\mu})^T (\varphi(\mathbf{x}_l) - \hat{\mu}). \tag{3.13}$$

Which can be found by using the Kernel trick, after that the analysis can proceed like it did in the linear case. The projected space is given by:

$$z(\mathbf{x}) = \mathbf{w}^T (\varphi(\mathbf{x}) - \hat{\mu}). \tag{3.14}$$

This space is also called target space an is interpreted as en error where the target is zero. A plot of the kernel matrix eigenvalues associated with the toy-problem under consideration is shown in figure 3.2. The plots in 3.3 show the projections of the kernel matrix onto the first six principal components. It can be observed that the projection quality decreases alongside the relevance of the associated eigenvalue. The actual de-noising is done using:

$$\bar{\mathbf{x}} = h(z) \tag{3.15}$$

Where the function h must minimize:

$$\min \sum_{k=1}^{N} \|\mathbf{x}_k - h(z_k)\|^2. \tag{3.16}$$

The function h is generally an MLP trained using Bayesian learning. [5] But solving an the unconstrained optimization problem using a specified kernel function is also possible.

3.2 Handwritten digit denoising

To learn more about the two techniques, both will be applied to a digit recognition problem using the supplied script `digitsdn`. The original data set consists of handwritten numbers with approximately 20 records per number from 0 to 9. Each image has a resolution of 16 by 15. Two test sets with one image per numeral are also included. The images are

[5]Support Vector Machines: Methods and Applications, Suykens et al., page 213

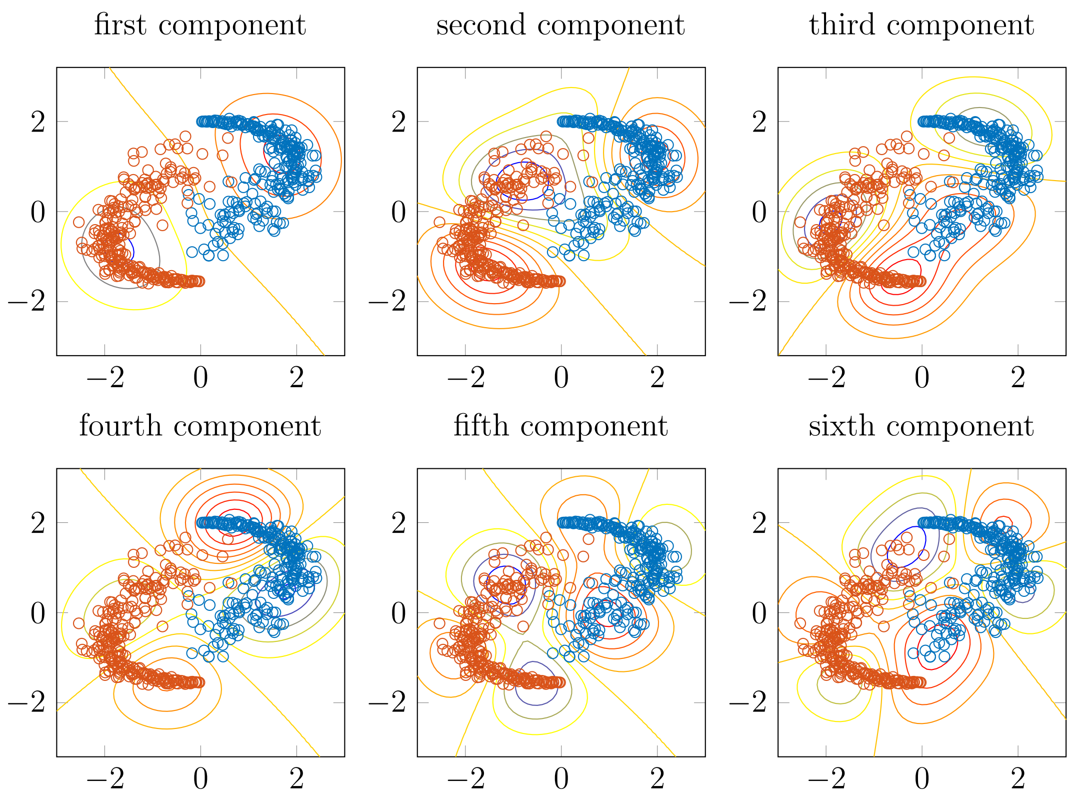


Figure 3.3: Plots of the kernel matrix projections using each of the six computed principal components. The quality of the plots decreases together with the relative importance of the associated eigenvalue.

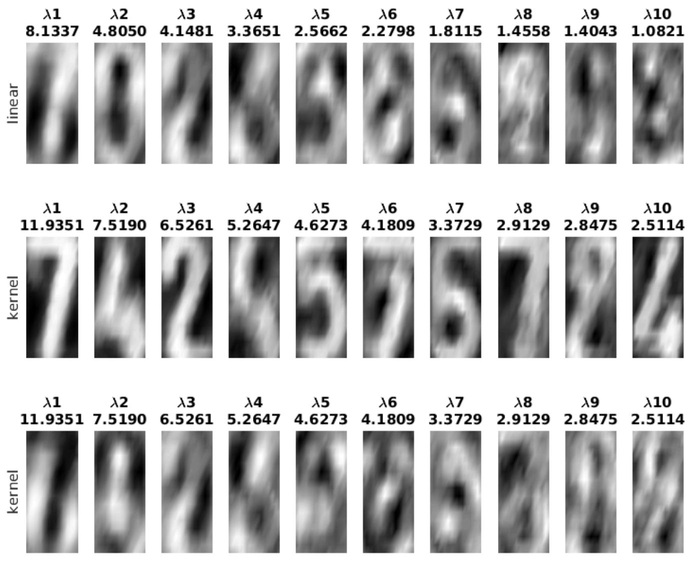


Figure 3.4: The principal components using linear PCA (first row). Kernel PCA principal components (second row) and projections using the first principal components (third row).

flattened into row vectors and stored in an input matrix with one image The top rows of the plots in figure 3.5 show the noise free data of the first test set. The second row noisy versions of the same hand written numbers. In the non-linear case the rbf-kernel-function density is initialized using the approximation dimension times the mean of the variance in the training set. Figure 3.4 shows the principal components found by using linear and kernel pca on the large training data set. These principal components will be used later to clean the noisy numerals in the test set. The plots reveal that a seven shaped principal component is dominant in the kernel as well as the linear case. All rows after the second row in figure 3.5 show efforts to reconstruct de-noised versions of the input using an increasing number of principal components. If few components are used the dominant seven shaped vector takes over the reconstruction quite often. If a more complete model with more components is used this does not happen anymore in the linear as well as the nonlinear case. The crucial difference between the two cases is that in the linear case the noise reduction decreases when more principal components are used, resembling the original image more and more. In the kernel case noise reduction gets better each time moving more towards a prototypical representation of the input. This behavior probably stems from the fact that the kernel case is able to adapt better to non-linear number shapes. The mis-representation of the number seven in the can be fixed by increasing the rbf-kernel width, the result is shown in figure 3.6. Unfortunately increasing the width also increases the noise, so a good trade off is required.

3.3 Spectral clustering

In this section a problem consisting of two interlocked rings as shown in figure 3.7 will be analyzed. Two the human eye the grouping of points into two independent rings is

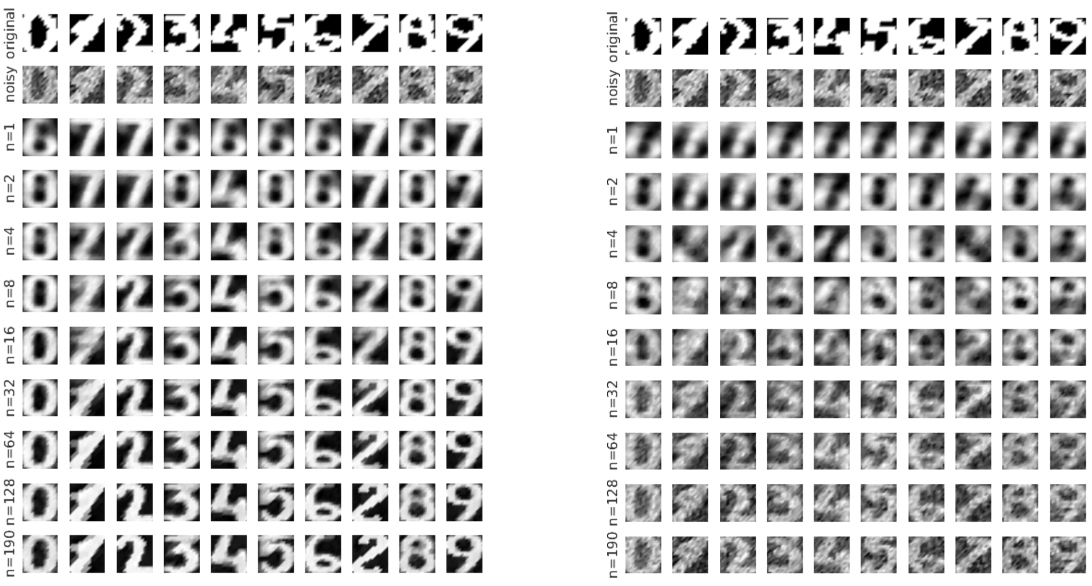


Figure 3.5: Digit de-noising using kernel (left) and linear PCA (right).

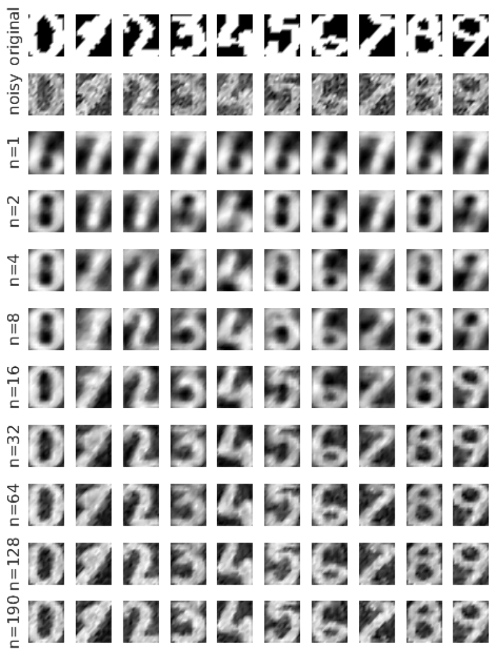


Figure 3.6: The kernel pca process using an rbf function with larger width σ^2.

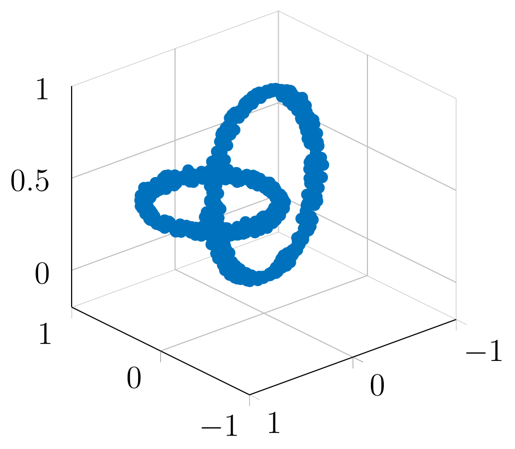


Figure 3.7: The training data set consisting of two interlocked rings.

immediately apparent. Spectral clustering is an algorithm, with which a computer can find these two classes. As the data set does not have annotations, this is an unsupervised learning algorithm. Spectral clustering methods use a kernel function as a similarity measure. The largest eigenvectors of a rescaled kernel-matrix are computed. The eigenvectors corresponding to the three largest eigenvalues are computed the second largest will contain the clustering information in the sign. If more then three eigenvectors are found the one containing the clustering information will be the smallest with an even number, if one starts counting with one at the largest vector. It must be noted that in this case the eigenvalues are all very close to one, so this observation cannot necessarily be generalized. Figure 3.10 shows the results of using classification information from the second largest eigenvector when computing the first three eigenvectors using Lanczos' iterative method. It can be observed that for the two smaller values $\sigma^2 = 0.001$ and $\sigma^2 = 0.01$ the method is able classify the two rings correctly. For the larger value $\sigma^2 = 0.1$ the rbf-kernel becomes to large, which results in incorrect linear looking classification.

3.4 Fixed-size LS-SVM

As the number of data points increases working in the dual space becomes harder and harder, because the dimension of the unknown support vectors depends on the number of input data points $\alpha \in \mathcal{R}^N$. The primal problem is better suited for many input data points as the unknown primal weights vector length is determined by the dimension of the input data $\mathbf{w} \in \mathcal{R}^n$. In other words large data set problems should be solved in the primal space, while the dual space should be used if the input data is high dimensional. [6] When SVMs are trained in the primal space, they are called fixed size svms. For the primal space case the kernel trick made it possible to train svms without explicitly knowing which non-linear function mapped to the feature space. When working in the primal space $\varphi(\mathbf{x})$ must be evaluated. This is only simple for linear classifiers, where $\varphi(\mathbf{x}) = \mathbf{x}$. In the non-linear case the Nyström method is required to approximate the nonlinear mapping $\varphi(\mathbf{x})$, the general idea is to choose a a fixed subsampled kernel matrix size M. Typically M is a lot smaller then the true Kernel-matrix size $M \ll N$[7] This smaller kernel matrix

[6]Support Vector Machines: Methods and Applications, Suykens et al., page 174
[7]Support Vector Machines: Methods and Applications, Suykens et al., page 175

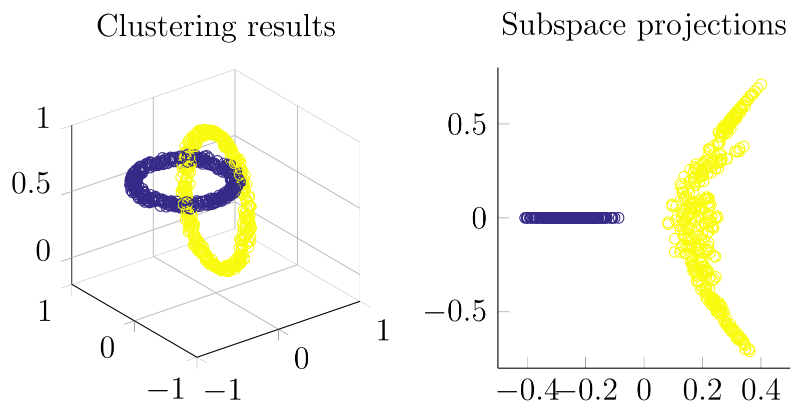


Figure 3.8: Spectral clustering and subspace projections using $\sigma^2 = 0.001$.

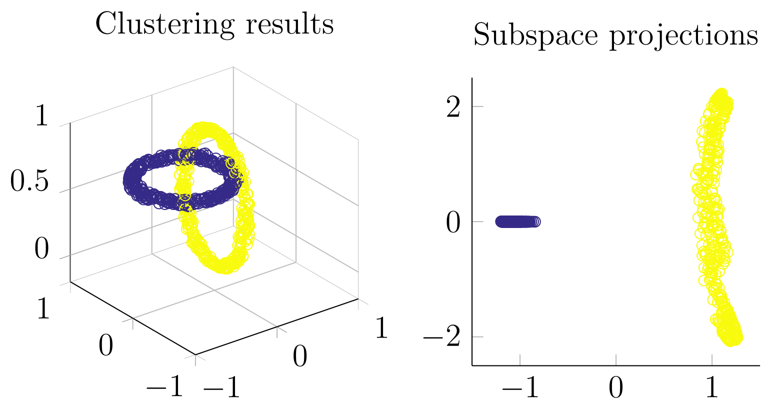


Figure 3.9: Spectral clustering and subspace projections using $\sigma^2 = 0.01$.

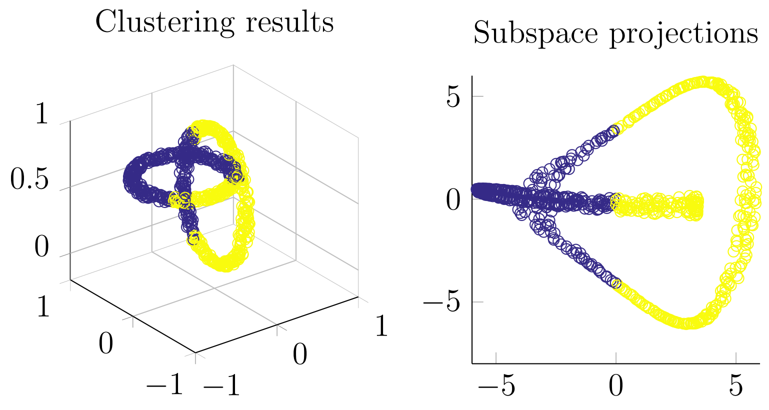


Figure 3.10: Spectral clustering and subspace projections using $\sigma^2 = 0.1$.

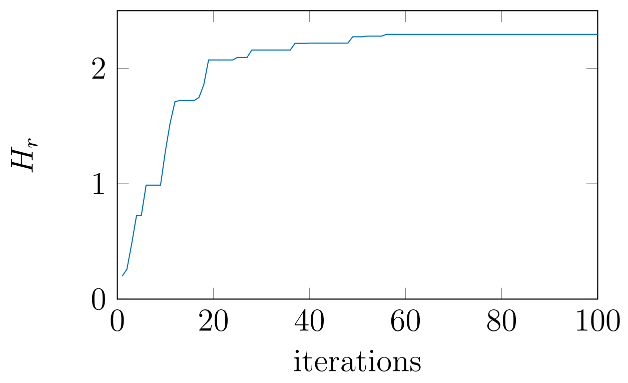


Figure 3.11: The value of the entropy function for an optimization process of a subset of 10 values drawn from the normal distribution $\mathcal{N}(0, 2^2)$

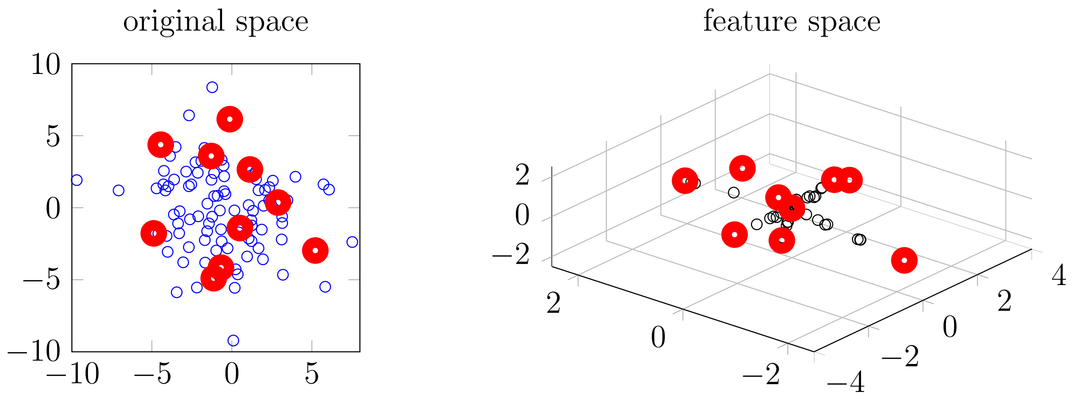


Figure 3.12: Feature space reconstruction

is then approximated using a subset of the input data. The computed eigenvalues and eigenvectors found from this set are then used as an approximation to the true large version of the matrix. Instead of choosing these support vectors randomly the entropy function,[8]

$$H_r = -\log \int p(\mathbf{x})^2 d\mathbf{x} \tag{3.17}$$

$$\int p(\mathbf{x})^2 d\mathbf{x} = \frac{1}{N^2}\mathbf{1}^T \Omega \mathbf{1} \tag{3.18}$$

is used. Starting from a random fixed size pool of support vectors a selected vector is replace with a value from the training set. If the entropy increases the datum is kept in the support vector set. If the entropy function does not increase the new value is rejected and the old one is kept in the set. This procedure is repeated until the entropy function does not increase sufficiently anymore or a maximum number of iterations is reached. The reduced kernel matrix can be determined from the fixed set. After estimating its eigenfunction $\mathbf{w}$ and b are determined. Figure 3.11 shows the entropy function over hundred iterations of a ten vector subset from a normally distributed data-set. Given this optimized subset the nonlinear mapping can be approximated as shown in figure 3.12

[8]Support Vector Machines: Methods and Applications, Suykens et al., page 181

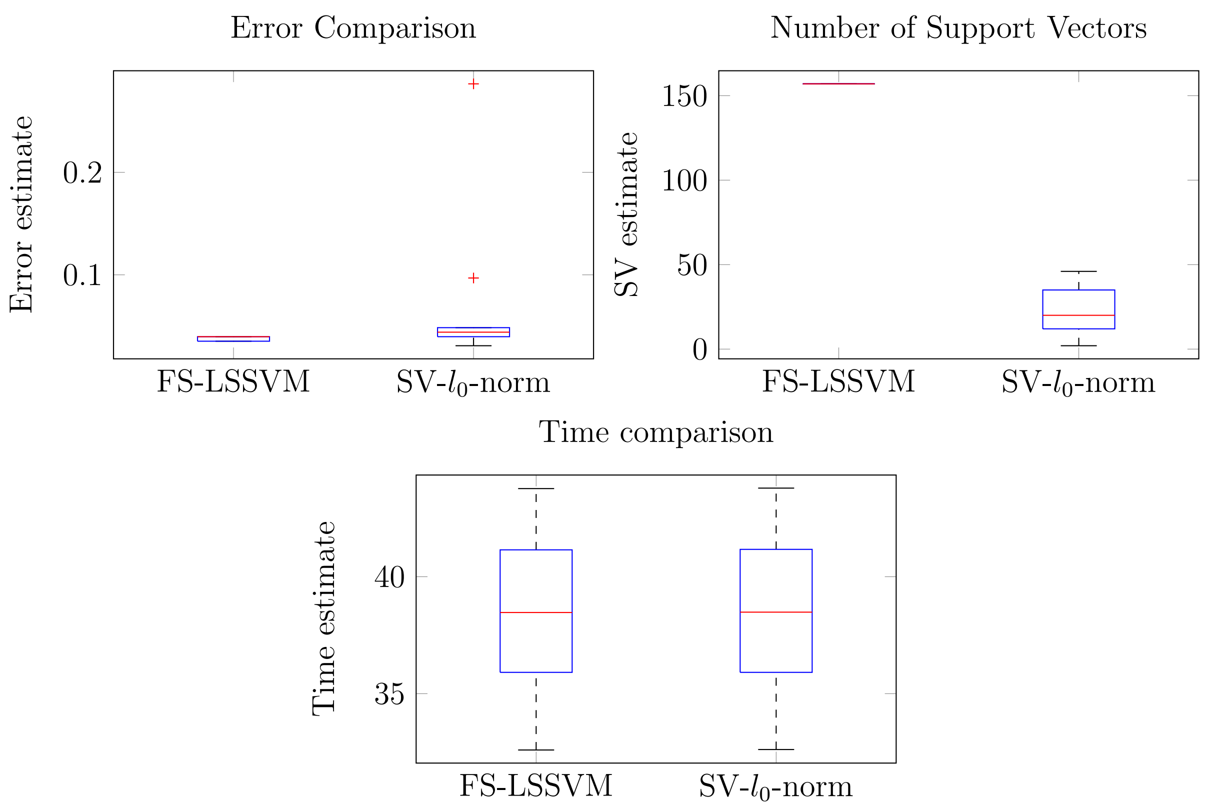


Figure 3.13: Classical fixed size svm and l_0 reduced version comparison for classification on the Wisconsin breast cancer dataset.

using the Nyström method, if alongside the selected inputs a kernel function and its parameters are chosen.

3.4.1 Sparsity and the l_0-norm

Sparsity is a desirable property of trained support vector machines. In the dual space an ls-svm classifier is sparse if many support vectors are zero. [9] When evaluating the classifier only the non-zero vectors have to be taken into account, making the computation more efficient. For fixed size ls-svms the primal problem is considered, the notion of sparsity translates into a smaller representation of the basis given by:

$$\mathbf{w} = \sum_{N}^{k=1} \alpha_k \varphi(\mathbf{x}_k) \tag{3.19}$$

A good method of determining suitable subset of the input space $\{\varphi(\mathbf{x}_k)\}_{k=1}^{N}$, could for example be the entropy selection method discussed earlier. The determined subset could then serve as a way to approximate a sparse $\mathbf{w}$. This goal is formulated using the l_0 norm

$$\min_{\mathbf{w}} \|\mathbf{w}\|_0 = \|w_1\|^0 + \|w_2\|^0 + \cdots + \|w_N\|^0 \tag{3.20}$$

[9]Support Vector Machines: Methods and Applications, Suykens et al., page 33

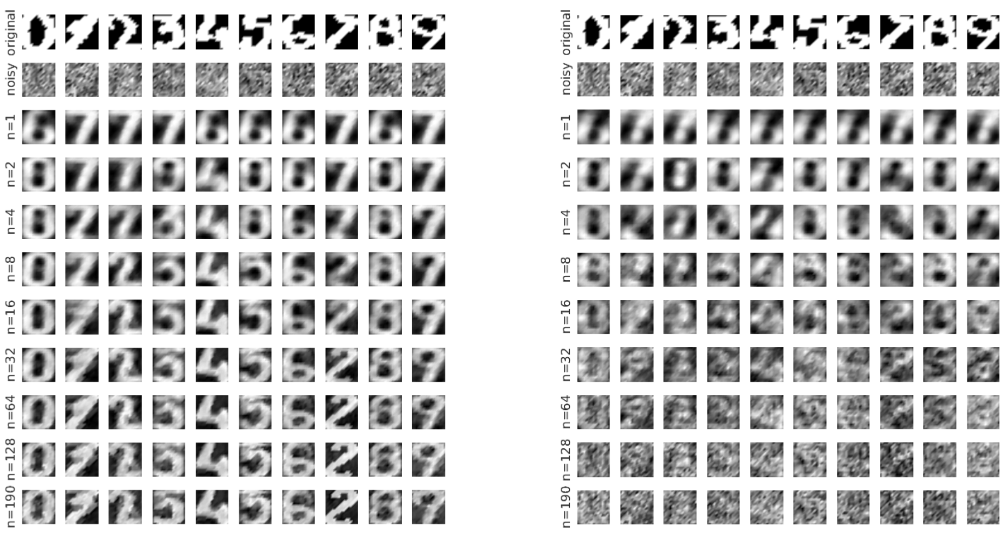


Figure 3.14: Kernel based and linear pca de-noising of very noise data.

which is equivalent to minimizing the count of non-zero elements in the vector.[10] A first experiment using this method of producing sparsity is use for classification of the comparatively small Wisconsin data set. Results from running `fslssvm_script` are shown in figure 3.13. The error comparison shows a slightly elevated median for the sparse version, probably due to the random nature of the input set reduction process bad outliers in terms of error exist. The figure also reveals, that the l_0 norm minimization process significatly reduced the number of support vectors, while it does not lead to a significant increase in training time.

3.5 Kernel and linear PCA-de-noising on high noise data

In this section an extreme noise example is considered, where the human eye has trouble identifying the characters correctly. Figure 3.14 shows the input data as well as the performance of a low sigma kernel-pca in comparison to a linear one. The trade off that came with choosing the kernel width σ^2 was observed earlier. A too large σ^2 led to correct predictions but little noise reduction. Smaller width sometimes ended up clear but incorrect letter representations. In the case shown in figure 3.14 a small σ^2 had to be chosen in order to deal with the very noisy input. Given the low quality of the input the kernel-PCA does an incredibly good job at de-noising. It does confuse 3 with 5 and 6 with 2 however. In the linear case this does not happen but it does not come close in terms of noise reduction.

[10]David Wipf and Bhaskar Rao, l_0-norm Minimization for Basis Selection

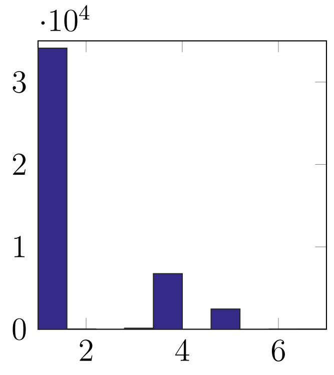


Figure 3.15: Histogram of the statlog dataset's training data class distribution.

3.6 Shuttle dataset-analysis

The statlog/shuttle dataset, contains 58000 ten dimensional row data vectors. With the last entry in each row indicating the class to which the data point belongs. Traditionally the first 43500 data rows are used for training and the last 14500 data points are retained for testing. Figure 3.15 shows the shuttle-dataset's class distribution. About eighty percent of the data belongs to the first class. A modified version of the `fslssvm_script` has been used with the shuttle data set as an input. The aim here is again classification, but its must be noted that the shuttle dataset is much larger than the Wisconsin-cancer set, it has 58000 data points while the cancer set only contains 682 data points. The results of the machine architecture comparison are shown in figure 3.16. On this larger data set the difference in terms of the error estimate become negligible. Quite a significance difference in terms of sparsity persist however, while the training time does not increase significantly.

3.7 California dataset-analysis

The California housing regression problem. Using windowed input data with a delay of 1032, which equals a data size to delay ratio of 20. The same ratio has been successfully applied to the Santa-fe estimation problem. The same experiment is repeated for a regression scenario using the California housing data set. Figure 3.17 shows the findings. In this case the a more sparse solution can only be found in some cases. Success in finding a sparse representation probably depends on luck with the random subset the entropy selection method works with.

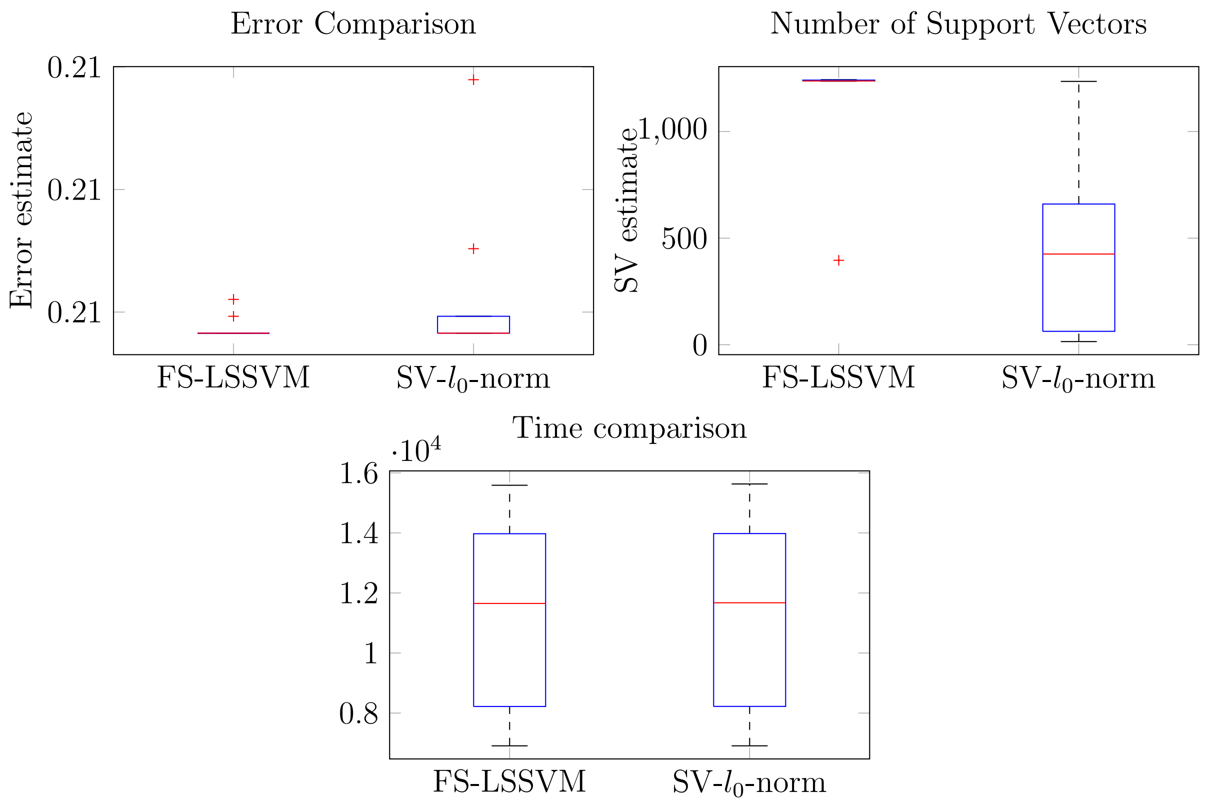


Figure 3.16: Fixed size svm and l_0 svm comparison for classification on the nasa shuttle dataset.

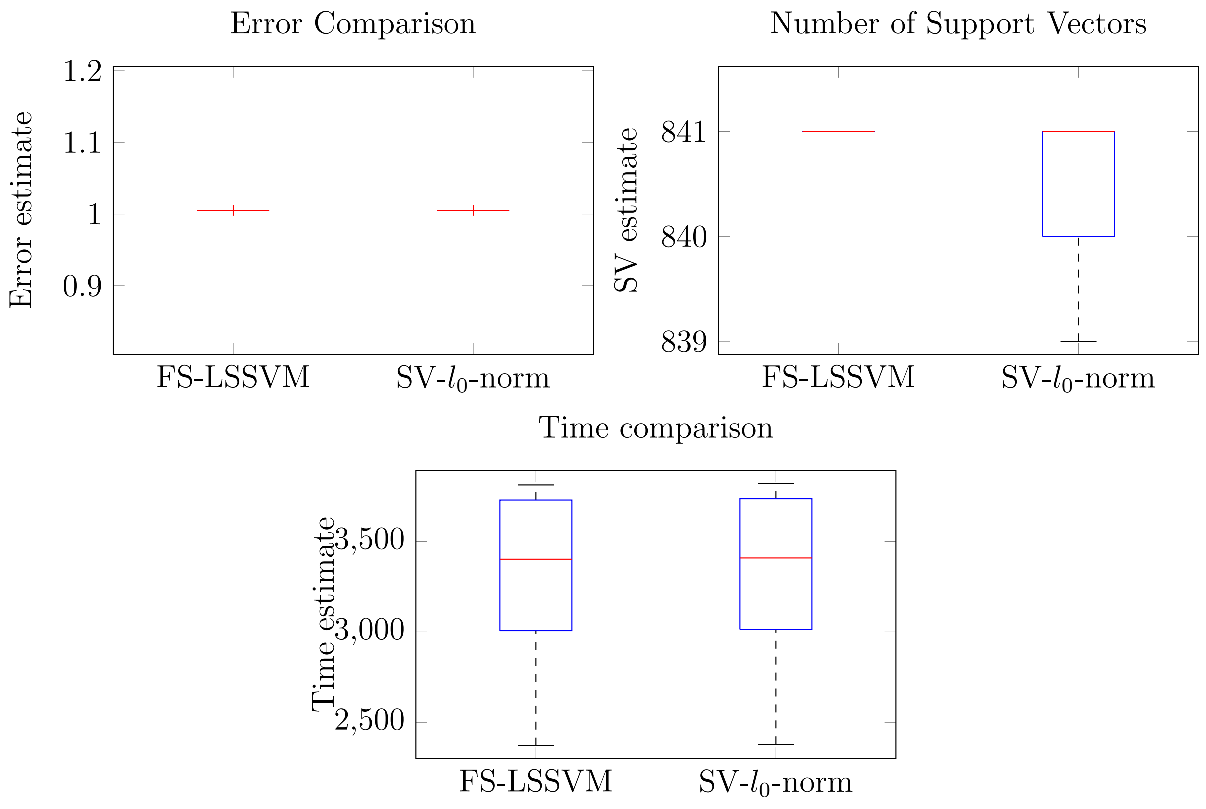


Figure 3.17: Fixed size svm and l_0 svm comparison for regression on the california-housing dataset.