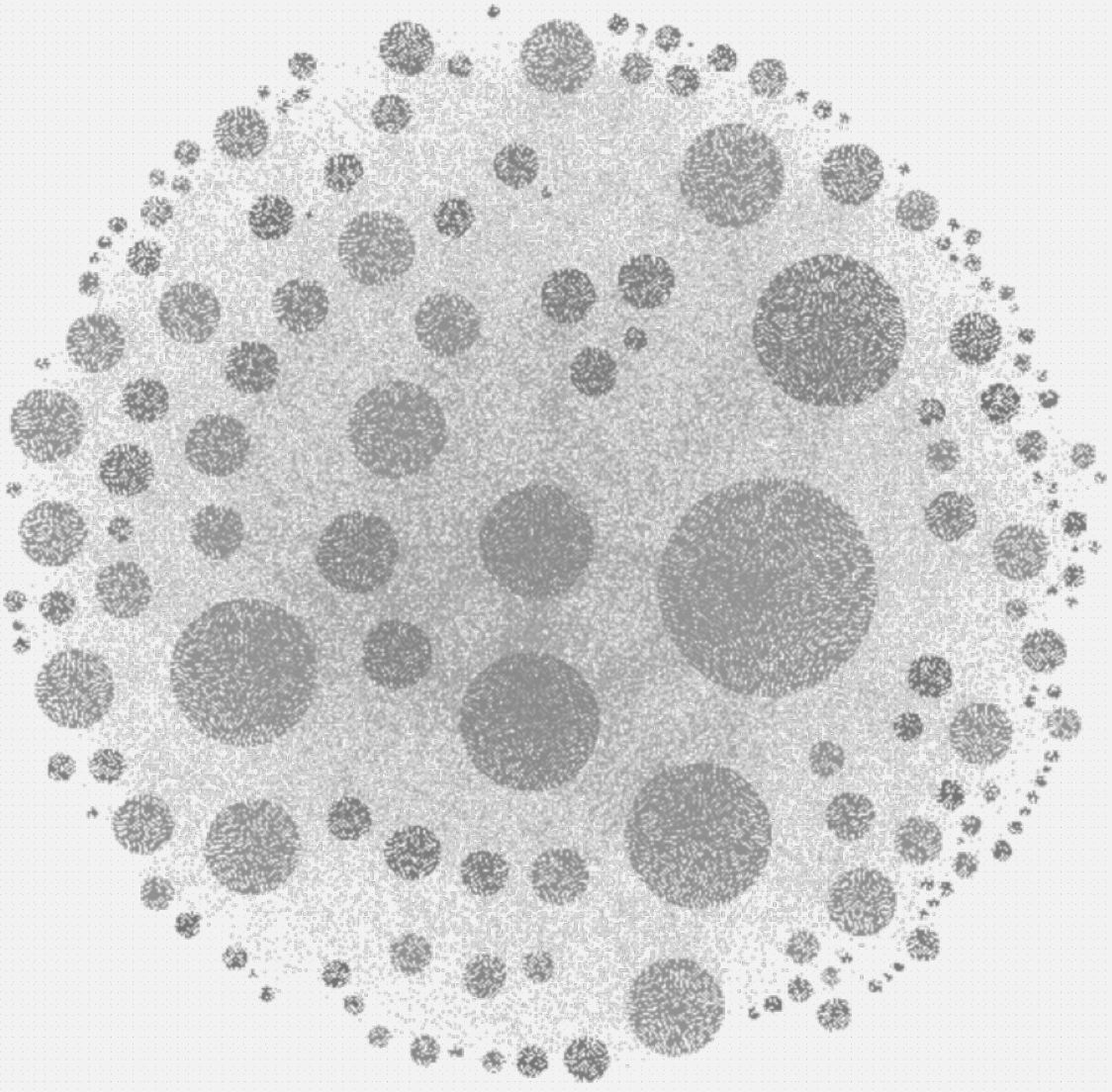
Support Vector Machines

Bruno Vandekerkhove



Academic Year 2020

H02D3A: Support Vector Machines

In this document each exercise session is dealt with in separate sections that are named correspondingly. Each of them is accompanied by a source file which has a code section for each exercise or homework.

Contents

Classification

`/src/session_1/.m`

Two Gaussians

For a binary classifier where the distributions are (assumed or known to be) Gaussian with equal covariance matrices the decision boundary that maximises the posterior probability $P(C_i|x)$ becomes linear. This is independent of the amount of overlap. Trying to get a better boundary would lead to overfitting. In this particular example where $\Sigma_{xx} = \mathbb{I}$ one ends up with a perpendicular bisector of the segment connecting the two cluster means $(-1, -1)$ and $(1, 1)$, which gives $f(x) = -x$ as a decision boundary.

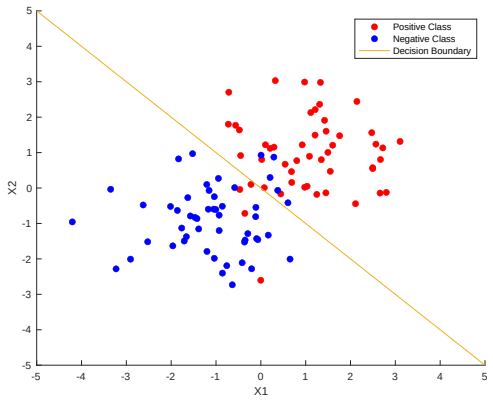


Figure 1: Decision boundary (in orange) for two Gaussian distributions with equal covariance matrices.

Support Vector Machine Classifier

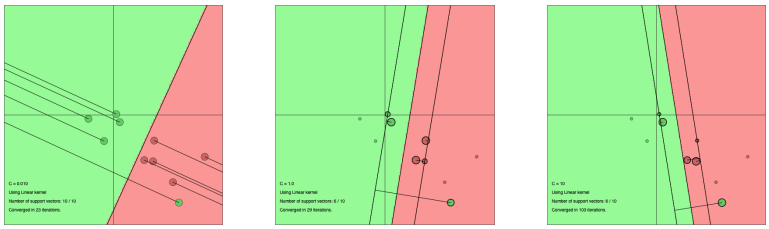
To deal with the non-linearly separable classification problem in the example one solves the following minimisation problem, where the hyperparameter C controls the trade-off between maximising the margin and making sure that the data lies on the correct side of that margin :

$$\min_x \frac{1}{2} \cdot w^T \cdot w + C \cdot \sum_{k=1}^{N} \xi_k \qquad \text{such that } y_k \cdot [w^T \cdot x_k + b] \geq 1 - \xi_k \text{ and } \xi_k \geq 0 \ (\forall k \in \{1, .., N\})$$

A dual form can be expressed by making use of Lagrange multipliers which in this context are also called *support values*. Any data points for which the corresponding support value isn't zero are called *support vectors*. These lie close to the boundary or are misclassified.

For the given toy dataset it can readily be seen in figure 2 (top row) that for a decreasing value of C the margin does become larger and that less *'slacking'* is allowed for. By adding data points close to the boundary or at

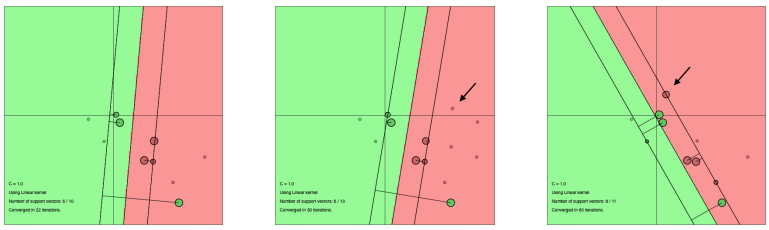
the *'wrong'* side of it the margin usually changes a lot and the new data points nearly always become support vectors.



(a) $C = 0.01$ (large margin, lots of *'slacking'*.)

(b) $C = 1$

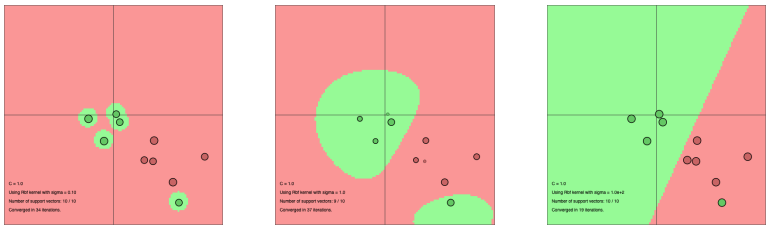
(c) $C = 10$ (small margin, little *'slacking'*.



(d) Initial solution.

(e) Negative data points added at the right side of the margin.

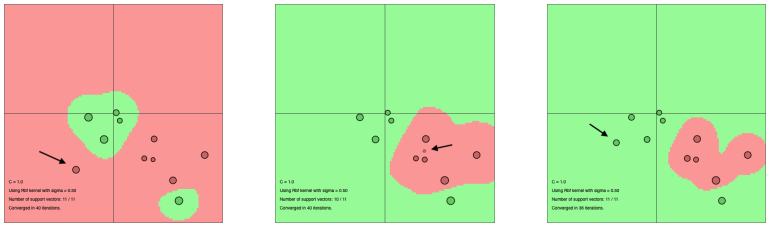
(f) Negative data point added at the other side of the margin.



(g) $\sigma = 0.1$

(h) $\sigma = 1.0$

(i) $\sigma = 100$



(j) Negative data point added far away from the boundary.

(k) Positive data point added inside the boundary.

(l) Positive data point added inside the boundary.

Figure 2: Visualisation of various experiments with C (top row), addition of points using the linear kernel (second row), σ (third row) and addition of points using the RBF kernel (last row). The number of iterations (tends to) increase as C gets larger or as the number of data points increases.

The σ^2 hyperparameter in the case of an RBF kernel controls the locality of each data point. As it increases the feature space mapping becomes smoother, producing simpler decision boundaries (figure 2, third row). This relates to the bias-variance tradeoff. For σ^2 small enough you can separate any two classes - the Gaussian kernels are universal. But for very large bandwidths linear separability in the infinite-dimensional feature space cannot be achieved anymore such that the decision boundary becomes near-linear (separating the high-density areas of each class) or one class may even overtake the other. As for C, it works the same as before, prioritising either larger margins or lower misclassification rates. When σ^2 is large a bigger value of C can make the model more complex again.

The RBF kernel approach can generate models having misclassification rates that are lower than the classic linear kernel approach does as decision boundaries that are nonlinear in the input space can be learned. It also tends to use more data points as support vectors. This makes the model less compact (computationally

efficient) and if the data is linearly separable it is unnecessary. Deciding whether the model generalises better can be done through evaluation on a test set.

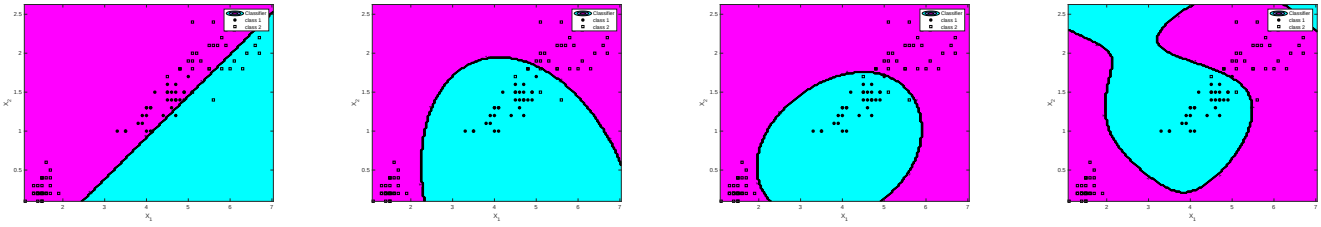
Least-squares support vector machine classifier

Figure 3 depicts results with various polynomial kernels where $t = 1$. This t is called the *constant* and it controls the importance of higher - versus lower-order terms in the polynomial as can be deduced from the application of the binomial theorem to the definition of the kernel :

$$(x^T \cdot y + \tau)^d = \sum_{i=0}^{d} \binom{d}{i} (x^T \cdot y)^{d-i} \cdot \tau^i$$

The effect of t is not very noticeable for small values of d while for the higher degrees it tends to make the decision boundary more complex.

When the degree δ is 1 the feature map is linear which is equivalent to the classic non-linear SVM problem while for higher degrees a boundary of increasing complexity is learned such that for $\delta \in \{3, 4\}$ no data points are misclassified. To make it less likely that the model overfits a lower δ is likely to be preferable (this corresponds to the application of Occam's razor).



(a) $\delta = 1, \rho = 0.55$ (b) $\delta = 2, \rho = 0.05$ (c) $\delta = 3, \rho = 0.0$ (d) $\delta = 4, \rho = 0.0$

Figure 3: Visualisation of the results of experiments with polynomial kernels on the iris dataset (δ is the degree of the polynomial, ρ the misclassification rate on the test set, τ and γ are fixed to 1). As the degree of the kernel increases, the misclassification error drops.

In the case of RBF kernels one can see in figure 4 that for a γ value of 1 any σ^2 between 0.1 and 1 performs well. The same interval works for γ when $\sigma^2 = 1$. This corresponds to the results of the provided sample script where the base used in the semilog plot happens to be the natural logarithm (instead of using base 10). For large σ values (say, 25) one class may overtake the other one as discussed before. The experiment is of limited value as only few parameter combinations were tried.

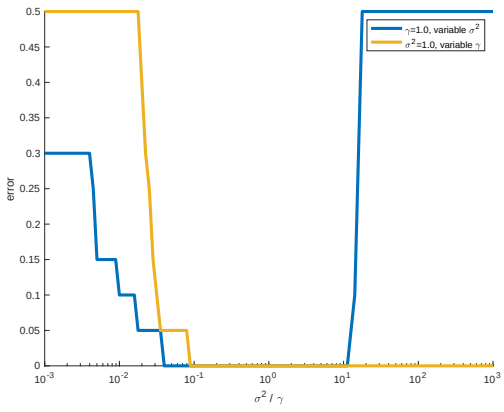


Figure 4: Performance of various RBF kernels on the iris dataset. The orange curve denotes the performance of the model for varying γ and fixed $\sigma^2 = 1.0$, the blue curve does the opposite ($\gamma = 1.0$, varying σ^2).

To properly find good parameter values a more systematic approach is in order. The idea is to search through the parameter space and evaluate the results on a validation set rather than on the test set (which should be used for nothing but the evaluation of the finished model). A validation set can be constructed in a few ways the results of which are illustrated in figure 7, where error estimates in the parameter space are shown. Random splitting i.e. splitting the input data randomly in a training - and a validation set is a way that isn't particularly robust. An improvement upon it is k-fold validation where the input data is randomly split into k folds which are considered as a validation set in turn such that all the input data can be used for parameter tuning. When $k = N$ with N the number of input samples this is called leave-one-out cross-validation.

To some extent deciding what k value to use parallels previous discussions about other parameters since there's a bias-variance tradeoff ; while larger k values should provide a better estimate of the error they also suffer from higher variance as the result depends more on how representative the input data is. This becomes more

important when the number of input data is small. Finally, k shouldn't be too small (such that the training sets remain large enough), it should preferably be a divisor of N (though this is not of primary importance) and if computational expense is an issue it cannot be large as many models would have to be generated.

Automated tuning uses these validation methods in conjunction with a search procedure to find useful combinations of parameters. This search strategy has to deal with a non-convex problem and can be a simple grid search (performing validation for every combination of parameters specified by a grid partitioning the parameter space) or the Nelder-Mead method. The latter aims to find the minimum of a function without needing its derivative by considering a simplex which is updated iteratively until it wraps around the minimum. It tends to execute faster than grid search especially when the number of parameters is high, though to address this one could also consider a variant of grid search that starts with a grid of limited granularity until it finds a promising region in the parameter space on which grid search is performed again.

In the LS-SVM toolbox a technique called *coupled simulated annealing* is used. This is a global optimisation technique which couples together several simulated annealing processes (inspired by coupled local minimisers' effectiveness compared to multi-start gradient descent optimisation). Its output is subsequently provided to either a grid search - or a Nelder-Mead algorithm. Results are the following :

Method	γ	σ^2	*cost*
Grid search	16983.8488 ± 100324.6730	10.5373 ± 83.0560	0.0339 ± 0.0063
Nelder-Mead	85111.5137 ± 75855.5677	12.8189 ± 71.4858	0.0349 ± 0.0069

Table 1: Results of automated tuning strategies (averaged over 100 runs).

Simulated annealing is a stochastic process such that the parameters end up varying quite a bit. The costs do not since they are minimised. A histogram of the results gives a more complete picture and shows that there are a few outliers that made the average γ and σ^2 large :

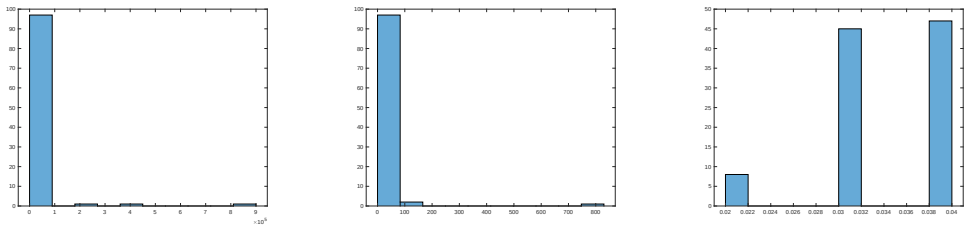


Figure 5: Histograms for 100 γ, σ^2 and costs returned by the automated tuning process.

Results of the automated tuning process correspond to the contour plots given in figure 7. As for the runtimes, the grid search ended up being twice as slow as the simplex method.

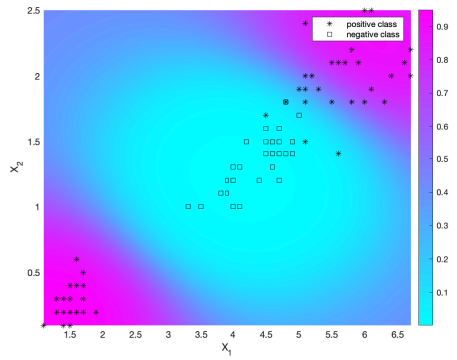


Figure 6: Posterior probability estimates using a Bayesian approach ($\gamma \approx 0.038, \sigma^2 \approx 0.56$). The hue indicates the probability that the data point at a location in the 2-dimensional plane belongs to one or the other class. A smaller σ^2 can lead to overfitting and makes for a more clearly defined, smaller blue area.

An ROC curve can be generated for any model. One calculates the result of the model applied on every test data point and uses the results as thresholds for which the *true positive rate (TPR)* is plotted in function of the *false positive rate (FPR)*. The *area under the curve (AUC)* can then be used to gauge the effectiveness of the classifier. For $\gamma = 0.037622, \sigma^2 = 0.559597$ this happened to be 1 indicating a perfect classifier. One can note that if the *sign* function had been used to classify test data there would be a misclassification error (the FPR would be 0.1).

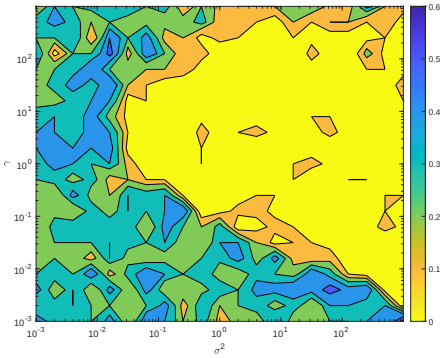
Using a Bayesian approach which requires a prior denoting the probability of occurrence of each class it is possible (by applying Bayes' theorem) to estimate the probability that a given data point belongs to the positive or the negative class. A corresponding plot of these posterior probabilities for the same model used to calculate the ROC curve is shown in figure 6. The prior is taken to be 0.5 (equal probability of each class).

sqjfsqld fsqdf kqdsfj qsldkfj dsql jfqsd jfqsdlj flqsdj lfqjsdflk sqjfl kqsj

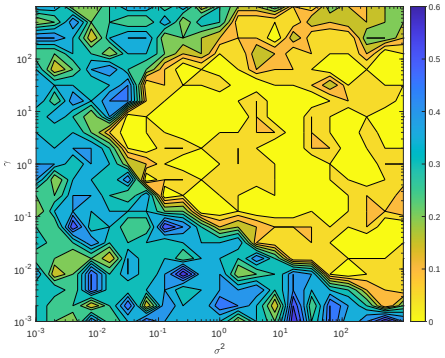
Ripley dataset

Wisconsin breast cancer dataset

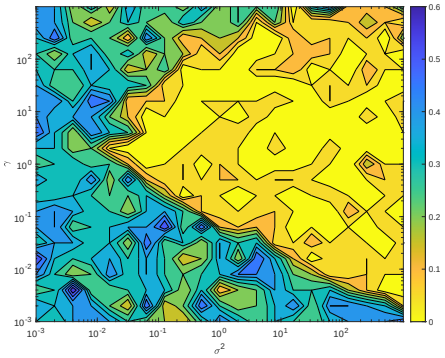
Diabetes dataset



(a) Random split validation.



(b) k-fold validation ($k = 10$).



(c) Leave-one-out validation.

Figure 7: Grid search in conjunction with various validation techniques. The results differ and the results for the random split strategy varies between runs. k-fold crossvalidation appears to be a better approximation of the error but may generalise less well. Reasonable parameters lie in the yellow zone where the error reaches a maximum of about 10%. A small bandwidth of between say, 0.1 and 1.0 and a regularization constant of 1.0 seems reasonable.

Function Estimation and Time Series Prediction

`/src/session_2/.m`

Support vector machine for function estimation

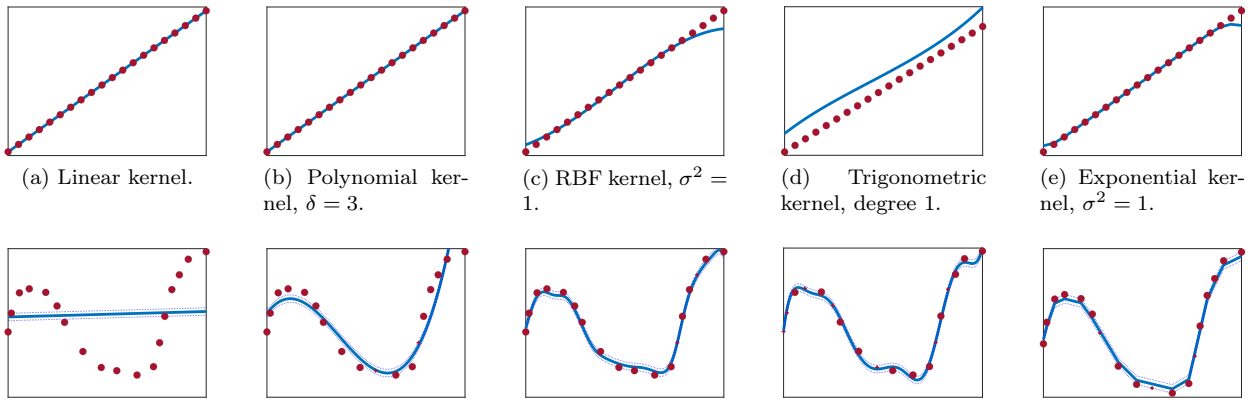
Two datasets are experimented with in figure 8. The first exists of 20 points lying on a slope. Therefore one can expect the linear kernel (which generates a linear model) to outperform the others. The second (non-linear) dataset is more challenging such that other kernels which enable the SVM to model the non-linearities have to be used instead.

From the problem statement :

$$\min_{w,b,\xi,\xi^*} \frac{1}{2} w \cdot w^T + C \cdot \sum_{k=1}^{N} \xi + \xi^*$$

where ξ, ξ^* are slack variables representing deviation from the so-called ϵ-*insensitive tube* it can be deduced that the C (the *bound*) plays the usual role of a regularisation parameter, prioritising either the smoothness of the model or how much it fits the data (i.e. minimising the slack variables). $C = 0$ leads to simple horizontal lines for the linear kernel which express the view that the two input features x_1 and x_2 aren't related. The ϵ parameter controls the width of the tube in which the data points are made to lie. A larger value decreases the number of support vectors and makes the model less accurate.

The formulation resembles that of a least squares fit with Thikonov regularisation but the ϵ-insensitive tube makes for a different loss function that encourages sparsity and since the problem is turned into a constraint optimisation problem a dual form can be expressed for which one can apply the kernel trick to be able to model any nonlinearities.



(a) Linear kernel. (b) Polynomial kernel, $\delta = 3$. (c) RBF kernel, $\sigma^2 = 1$. (d) Trigonometric kernel, degree 1. (e) Exponential kernel, $\sigma^2 = 1$.

(f) Linear kernel. (g) Polynomial kernel, $\delta = 3$. (h) RBF kernel, $\sigma^2 = 0.1$. (i) Linear b-spline kernel, degree 5. (j) Exponential kernel, $\sigma^2 = 1.0$.

Figure 8: Visualisation of the results of experiments with various kernels and parameters for two different datasets. In the first row $\epsilon = 0$ and `bound` $= 10$, in the second one $\epsilon = 1$ and `bound` $= 1000$. Some of the models are likely to be overfitted though these are just toy examples without any test set.

A simple example: the sinc function

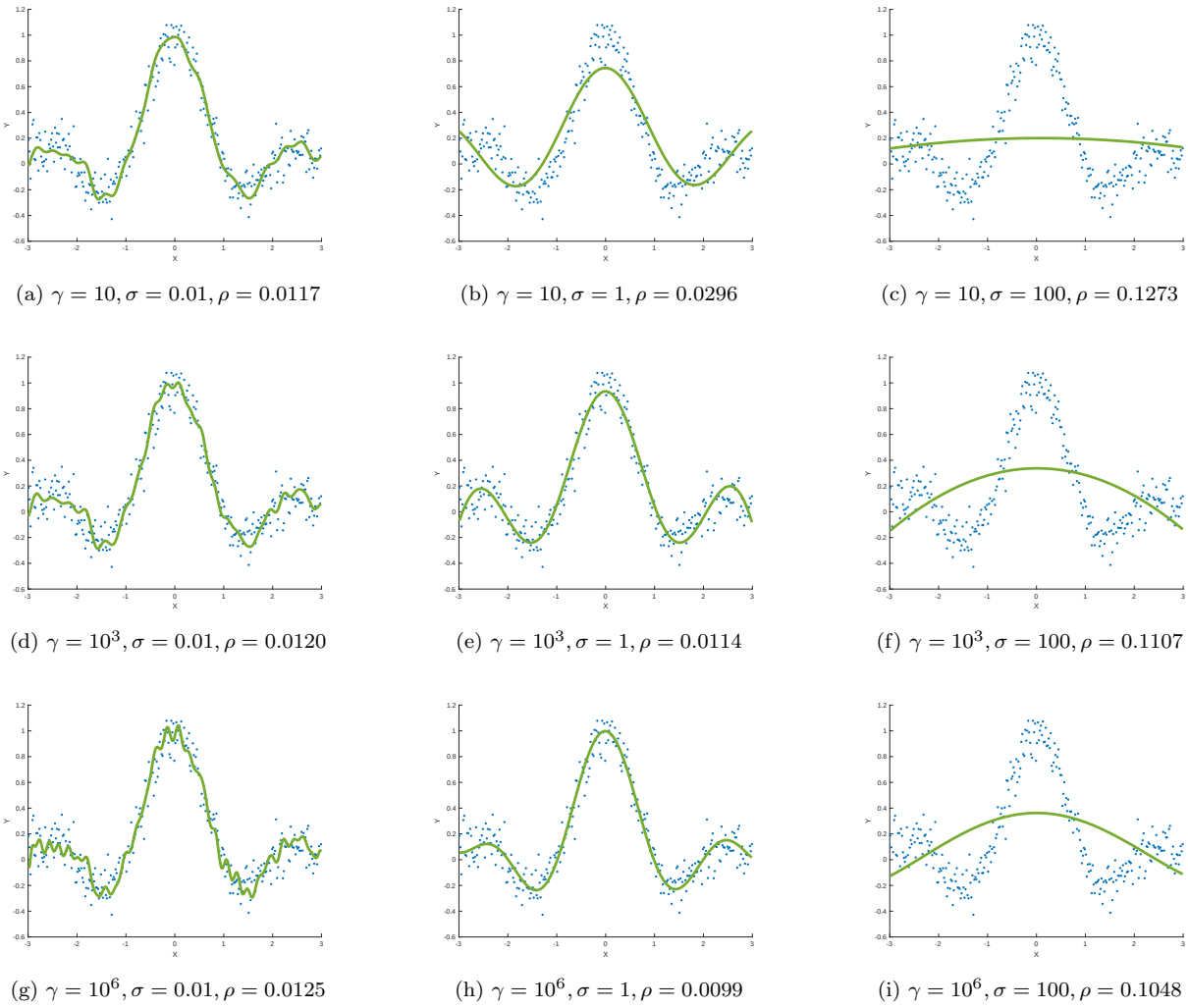
Figure 9 shows the results of LS-SVM regression applied on the `sinc` function. The mean squared error is lowest for $\sigma^2 = 1.0, \gamma = 10^6$. The smaller σ^2 value (0.01) leads to overfitting. In this case the underlying function is known and it seems reasonable to take $\sigma \in \{0.1, 1.0\}$ and, say, $\gamma = 10^3$. However, an optimal parameter cannot be found as it depends strongly on the training set and even for a given training set there's no unique parameter combination that works best.

Results of automated tuning are shown hereunder :

Method	γ	σ^2	*mean squared error*
Grid search	5528319.4668 ± 54371026.5606	0.1239 ± 0.1513	0.0104 ± 0.0001
Nelder-Mead	2329.7378 ± 2318.2696	0.1049 ± 0.0892	0.0104 ± 0.0001

Table 2: Results of automated tuning strategies (averaged over 40 runs).

The results are comparable to those obtained previously in the context of classification, with a few outliers for γ. Again, grid search appeared to be slower (185 seconds versus 110 for the simplex method). Some of the models seem to overfit the data a bit.



(a) $\gamma = 10, \sigma = 0.01, \rho = 0.0117$ (b) $\gamma = 10, \sigma = 1, \rho = 0.0296$ (c) $\gamma = 10, \sigma = 100, \rho = 0.1273$

(d) $\gamma = 10^3, \sigma = 0.01, \rho = 0.0120$ (e) $\gamma = 10^3, \sigma = 1, \rho = 0.0114$ (f) $\gamma = 10^3, \sigma = 100, \rho = 0.1107$

(g) $\gamma = 10^6, \sigma = 0.01, \rho = 0.0125$ (h) $\gamma = 10^6, \sigma = 1, \rho = 0.0099$ (i) $\gamma = 10^6, \sigma = 100, \rho = 0.1048$

Figure 9: Function estimation experiments with the `sinc` function. Noisy samples are fed to an LS-SVM. The green line is the estimated model, the blue dots represent the test data. ρ is the mean squared error. Small σ values make the model fit the noise.

When making use of a validation set that very set cannot be used for training. Making use of a Bayesian framework is an alternative and allows one to infer appropriate parameter combinations while making full use of the dataset. Depending on the kernel that is being used either 2 or 3 levels of inference are applied. At each of these level Bayes' theorem is used to infer parameter values. The equations are :

$$p(\mathbf{w}, \mathbf{b}|\mathcal{D}, \mu, \zeta_{1\_N}, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|w, b, \mu, \zeta_{1\_N}, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mu, \zeta_{1\_N}, \mathcal{H}_\sigma)} \cdot p(w, b|\mu, \zeta_{1\_N}, \mathcal{H}_\sigma) \tag{1}$$

$$p(\boldsymbol{\mu}, \boldsymbol{\zeta_{1\_N}}|\mathcal{D}, \mathcal{H}_\sigma) = \frac{p(\mathcal{D}|\mu, \zeta_{1\_N}, \mathcal{H}_\sigma)}{p(\mathcal{D}|\mathcal{H}_\sigma)} \cdot p(\mu, \zeta_{1\_N}|\mathcal{H}_\sigma) \tag{2}$$

$$p(\boldsymbol{\mathcal{H}_\sigma}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{H}_\sigma)}{p(D)} \cdot p(\mathcal{H}_\sigma) \tag{3}$$

As indicated by the colourings the evidence at any level equals the likelihood in the next level. At the first level one can take the logarithm of the product to see the relation with the primal form in the LS-SVM problem specification ; while the prior corresponds to the regularisation term the likelihood corresponds to the least squares cost term.

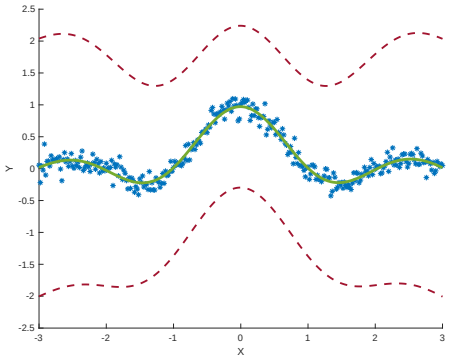


Figure 10: Results of Bayesian inference applied on training data based on the `sinc` function with some added white noise (blue). The function estimate is coloured in green, 95% error bars are indicated in red.

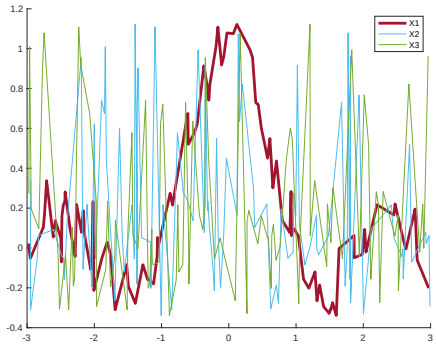


Figure 11: Plot of 3-dimensional input data of which only the first dimension models the `sinc` function, the other 2 being white noise.

One nice facet of this approach is that one ends up with error bars indicating uncertainty of the prediction. These error bars can be seen in figure 10 where the method is applied on the `sinc` function.

Automatic Relevance Determination

By modifying the kernel function for the LS-SVM dual formulation it becomes possible to disregard irrelevant or noisy input dimensions when these don't contribute to the final classification (a form of dimensionality reduction). In the case of an RBF kernel by using a (preferably diagonal) matrix S as a weighted norm a bandwidth is associated with each of the input dimensions. Level 3 of the Bayesian framework discussed before is then used to infer the elements of S and any input dimensions associated with small bandwidths are iteratively disregarded until no additional improvement is obtained. For the toy dataset the algorithm found the x_1 to be the sole relevant one which is also illustrated in figure 11. An other, crude way of doing relevance determination can be done by considering all possible non-empty subsets of the input features and applying cross-validation on the resulting data to figure out which subset performs best. For the given data the conclusion remained the same as any subset excluding the first (non-noisy) dimension performed badly ($mse > 0.1$) and the subset excluding all but the first dimensions performed best ($mse \approx 0.012$).

Robust regression

Two downsides of LS-SVMs are the loss of sparsity and the lack of robustness. Outliers can increase the variance of the decision boundary quite a bit due to the use of the least squares loss, which penalises outliers more than the *mean absolute error* does (such that there is more perturbation). If the noise on the data set is Gaussian then the least squares loss is appropriate, but in general the distribution of the noise is not known and in the case of the toy dataset dealt with here the noise isn't normally distributed. The idea, then, is to find an LS-SVM model as per usual and apply methods from robust statistics on the resulting parameters. Specificially, a weight is assigned to each data point based on its error. Some possible weight functions are the following : After

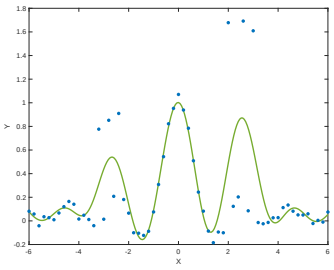
	Huber	*Hampel*	*Logistic*	*Myriad*
Weight function $W(r)$	$\begin{cases} 1 & \text{if } \lvert r\rvert < \beta \\ \frac{\beta}{\lvert r\rvert} & \text{if } \lvert r\rvert \geq \beta \end{cases}$	$\begin{cases} 1 & \text{if } \lvert r\rvert < \beta_1 \\ \frac{b_2 - \lvert r\rvert}{b_2 - b_1} & \text{if } \beta_1 \leq \lvert r\rvert \leq \beta_2 \\ 0 & \text{if } \lvert r\rvert > \beta_2 \end{cases}$	$\frac{tanh(r)}{r}$	$\frac{\delta^2}{\delta^2 + r^2}$

Table 3: Possible weight functions for use in robust regression with LS-SVMs.

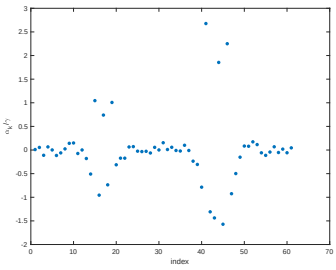
assigning the weights a weighted LS-SVM is trained and this process may be repeated as desired. Results with 4 weight functions are compared with a non-robust LS-SDM approach in figure 12.

Introduction: time series prediction

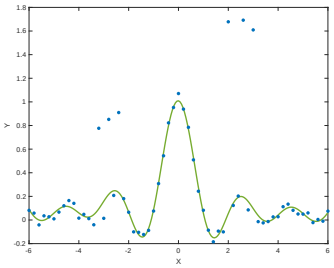
aaa



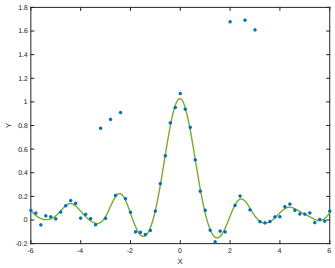
(a) Non-robust regression.



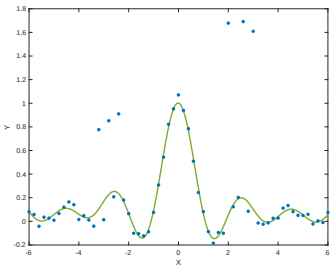
(b) Error values for the non-robust regression ($\frac{\alpha_k}{\gamma}$). After robust regression the errors decrease for the non-outliers which means that aside from those 6 outliers at the top the remaining values are centered around the x-axis.



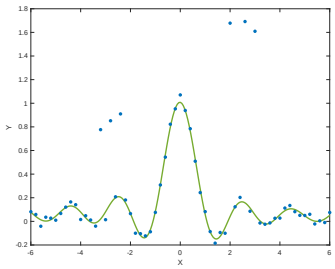
(c) *Huber* weight function.



(d) *Hampel* weight function.



(e) *Logistic* weight function.



(f) *Myriad* weight function.

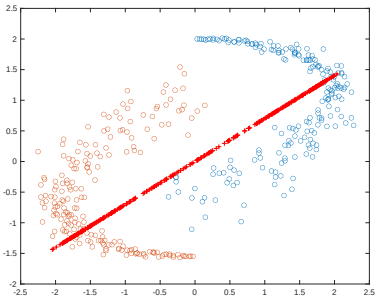
Figure 12: Comparing non-robust regression to robust regression. A variety of weight functions are considered. The smallest *mse* on the test set was obtained with the *Hampel* weight function. The noise is not normally distributed in contrast with the datasets shown before.

Logmap dataset

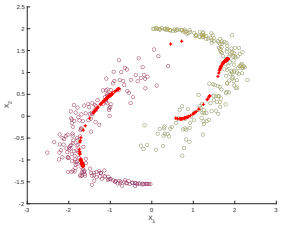
Santa Fe dataset

Kernel principal component analysis

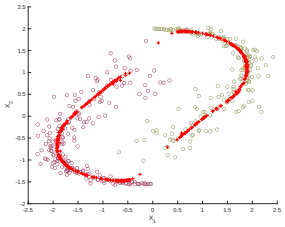
The point of *linear principal component analysis (PCA)* is to find a linear transformation (a projection onto an orthogonal basis) such that the variance of the projected data points is maximised. It's an old method introduced by Pearson. The transformed variables are called principal components. By disregarding some of these it is possible to achieve a dimensionality reduction while still capturing most of the important information carried by the input data. This can be used for denoising purposes.



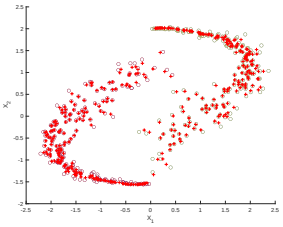
(a) De-noising with linear PCA.



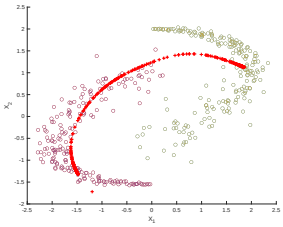
(b) $\sigma^2 = 0.4, n_h = 2$.



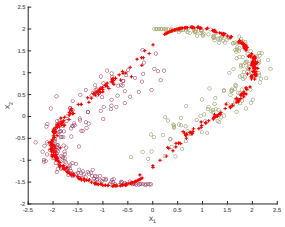
(c) $\sigma^2 = 0.4, n_h = 6$.



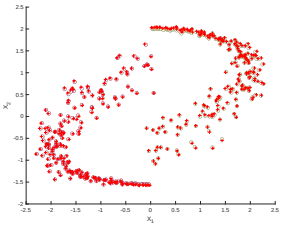
(d) $\sigma^2 = 0.4, n_h = 20$.



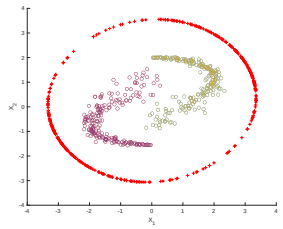
(e) $\sigma^2 = 1.0, n_h = 2$.



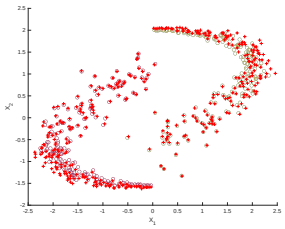
(f) $\sigma^2 = 1.0, n_h = 6$.



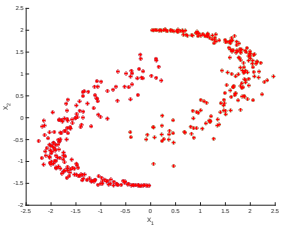
(g) $\sigma^2 = 1.0, n_h = 20$.



(h) $\sigma^2 = 10.0, n_h = 2$.



(i) $\sigma^2 = 10.0, n_h = 6$.



(j) $\sigma^2 = 10.0, n_h = 20$.

Figure 13: De-noising a toy dataset consisting of two spirals. The number of data points equals 400, the data point dispersion 0.3. n_h is the number of components. Retaining too few principal components leads to a poor result while retaining too many of them prevents any de-noising.

Like *Fisher discriminant analysis (FDA)* it is possible to relate this to LS-SVMs and formulate it as a constraint optimisation problem :

$$\max_{w,e} \quad \mathcal{J}_P(w,e) = \gamma \frac{1}{2} \sum_{k=1}^{N} e_k^2 - \frac{1}{2} w^T w \qquad \text{such that } e_k = w^T \cdot x_k \quad (k = 1, \ldots, N)$$

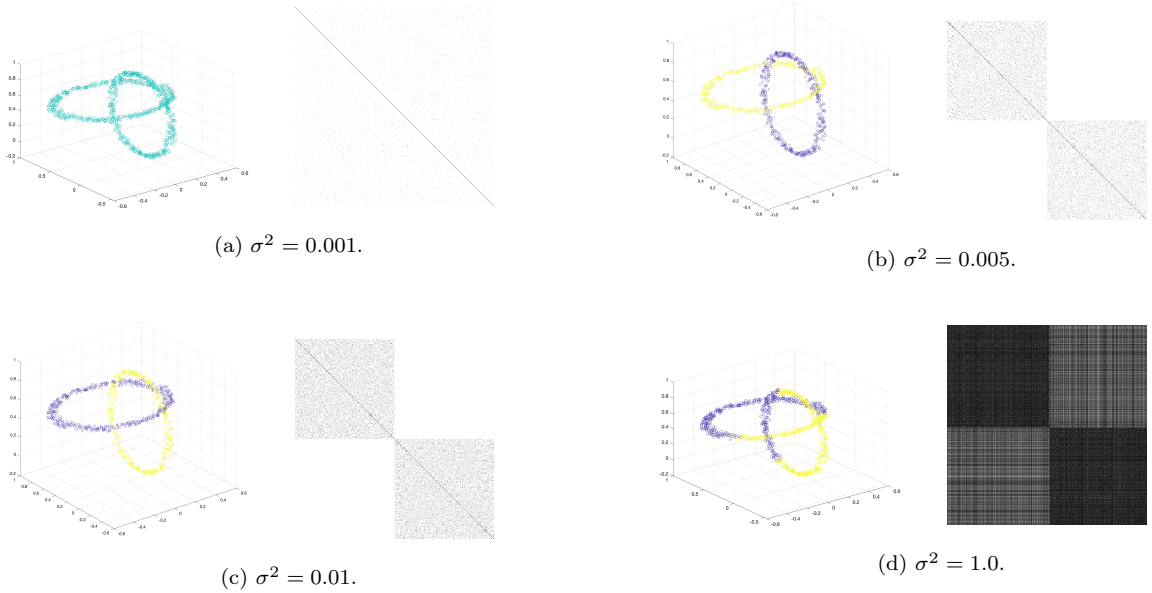
Here too, a dual form can be introduced using Lagrange multipliers. This makes it possible to apply the kernel trick which can make for a non-linear approach (in which case PCA is applied in the feature space). Additionally an increase in the number of dimensions becomes possible when the number of training samples is larger than the number of dimensions. The bias term is responsible for centering the kernel matrix automatically rather than having to do this beforehand. This is especially interesting in the case of *kernel spectral clustering (KSC)* which is considered in the next paragraph.

Because the alternative PCA formulation is model-based it becomes possible to do so-called out-of-sample extensions - the solution is not limited to the input data. It also becomes possible to tune models and find the right parameters that fit the data. However, because of the unsupervised nature of PCA this is not trivial. If the resulting model is meant to be used as a sort of preprocessing step e.g. for use in classification or in the context of a reconstruction problem (assuming an information bottleneck) then tuning can be done on the basis of the related loss function by making use of validation sets (as demonstrated previously). Otherwise it is possible to use common cross-validation techniques for linear PCA to pick an appropriate number of components followed by cross-validation based on pre-images to find useful values of the kernel parameters. These pre-images are approximations of inverse transformations of kernel PCA outputs back to the input space and can be used to calculate the reconstruction error for any point in the validation set (which consists of one data point in the case of LOOCV).

As an example of this method, consider the de-noising of a toy dataset the result of which are shown in figure 13. Tuning is done manually. When the number of principal components increases the reconstruction improves which is not surprising. More meaningfully, retaining a smaller number of principal components results in good de-noising performance. As per usual the linear approach (classical PCA) is not able to capture the non-linearities in the input data leading to poor results for this particular dataset. The σ^2 parameter in the Gaussian kernel can lead to excessive de-noising when set too low (and vice versa, as seen in figure 13).

Spectral clustering

Classification problems were considered in the first section of this report. In a classification setting a model is constructed which is made to predict what class some data point belongs to. These models are trained in a supervised way i.e. there's an annotated training set and test set which are used to determine appropriate parameters for the model and for evaluating it. In a clustering setting the number of classes k is not necessarily set beforehand but can be tuned. A model is trained in an unsupervised manner such that it clusters data points on the basis of some similarity measure.



(a) $\sigma^2 = 0.001$.

(b) $\sigma^2 = 0.005$.

(c) $\sigma^2 = 0.01$.

(d) $\sigma^2 = 1.0$.

Figure 14: Clustering of a toy dataset consisting of two rings. KSC applied using a kernel with varying σ^2 values. When this bandwidth is too large the reach of influence of each data point becomes too large and the clustering becomes subpar.

Spectral clustering in particular deals with finding a minimal cut of a similarity graph (a weighted & undirected graph) rather than finding a decision boundary which separates classes. Since finding a balanced yet minimal

cut is NP-hard a relaxed version is considered instead. The solution of the relaxed version is based on the eigendecomposition of the Laplacian matrix of the graph such that the data points can be embedded in an eigenspace on which a clustering algorithm like *k-means* can be applied.

The Laplacian matrix captures a lot of information about the graph such as the number of connected components (which equals the multiplicity of the eigenvalue $\lambda = 0$). Related to this number is the property that if the data is sorted according to cluster membership the similarity matrix becomes block-diagonal. Such depictions are used in the figures above.

Kernel spectral clustering (KSC) reformulates spectral clustering as an LS-SVM, essentially boiling down to a weighted version of kernel PCA. Again, out-of-sample extensions become possible and one can apply it on large-scale problems.

Experiments with an RBF kernel are shown in figure 14. Since the similarity matrix is the Kernel function and σ^2 happens to control the contrast in similarity between data points a σ^2 value which is small leads to a single class as all points are considered similar while a larger σ^2 leads to poor clustering results.

Fixed-size LS-SVM

Sometimes the amount of data that is being dealt with is too large such that some of the matrices like the kernel matrix involved in the dual formulations cannot be stored in memory. It becomes desirable, then, to solve the problem in the primal rather than in the dual space (it's the other way round when the number of input dimensions is high). The problem with this is that the feature map $\phi(x)$ is generally not known, only the kernel function representing dot products is made explicit.

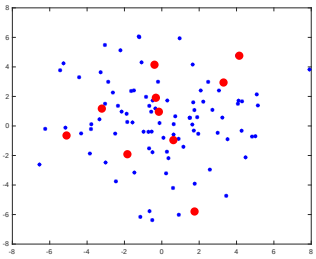
While decomposition can be used for this the approach that is considered in the case of fixed-size LS-SVMs is the Nyström method which is traditionally used for integral approximation. In the current setting it means that one takes a random subsample of the input data (of size $M \ll N$) and then uses this to approximate the feature map $\phi(x)$ such that a parametric solution can be found. For fixed-size LS-SVMs the subsample starts randomly and is then updated iteratively as long as the Renyi entropy improves. This metric can be approximated by taking the sum of the elements of the kernel.

Examples of subsamples obtained through optimisation of the Renyi entropy are shown in figure 15. The kernel is a Gaussian one, the input data is normally distributed. It looks as though a larger value for σ^2 leads to a selection of data points that are more spread out.

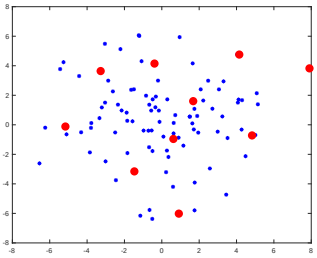
In figure 16 a fixed-size LS-SVM approach is compared with a variant which applies an ℓ_0-penalty after generating a fixed-size LS-SVM solution. This post-processing does not influence the total runtime because its complexity is $\mathcal{O}(M^3)$ while that of the fixed-size LS-SVM is $\mathcal{O}(NM^2)$ j. While the error increases the number of support vector is much lower for the latter method. This is not surprising since the ℓ_0-norm counts the number of non-zero values which means that it aims for sparse representations. The error drops as well, presumably because the model generalises better.

Kernel principal component analysis

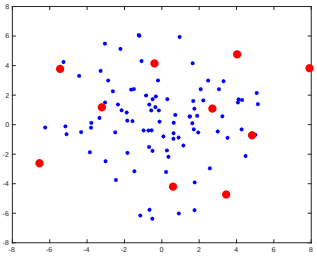
Fixed-size LS-SVM



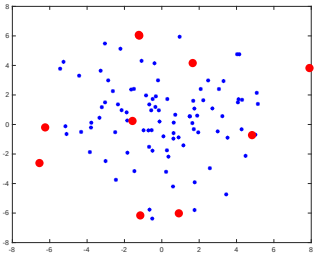
$\sigma^2 = 0.01$



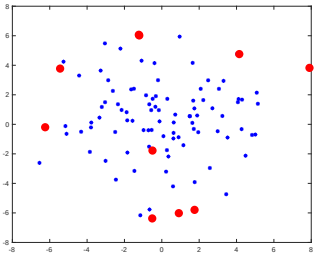
$\sigma^2 = 0.1$



$\sigma^2 = 1.0$

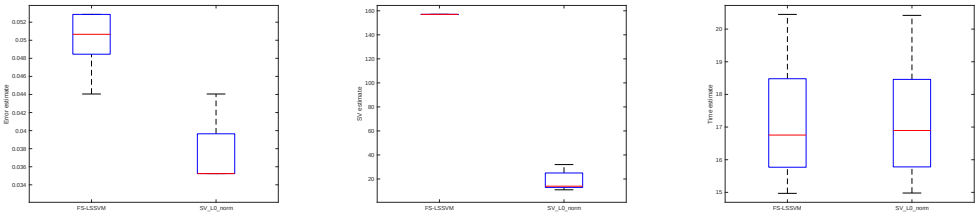


$\sigma^2 = 10.0$



$\sigma^2 = 100.0$

Figure 15: Subsample of input data points obtained for a fixed-sized LS-SVM using an RBF kernel of varying σ^2.



(a) Error estimation. (b) Number of support vectors. (c) Time taken for generating the models.

Figure 16: Comparison of the standard fixed-size LS-SVM approach with a modified approach applying an ℓ_0-penalty to obtain a sparser representation.