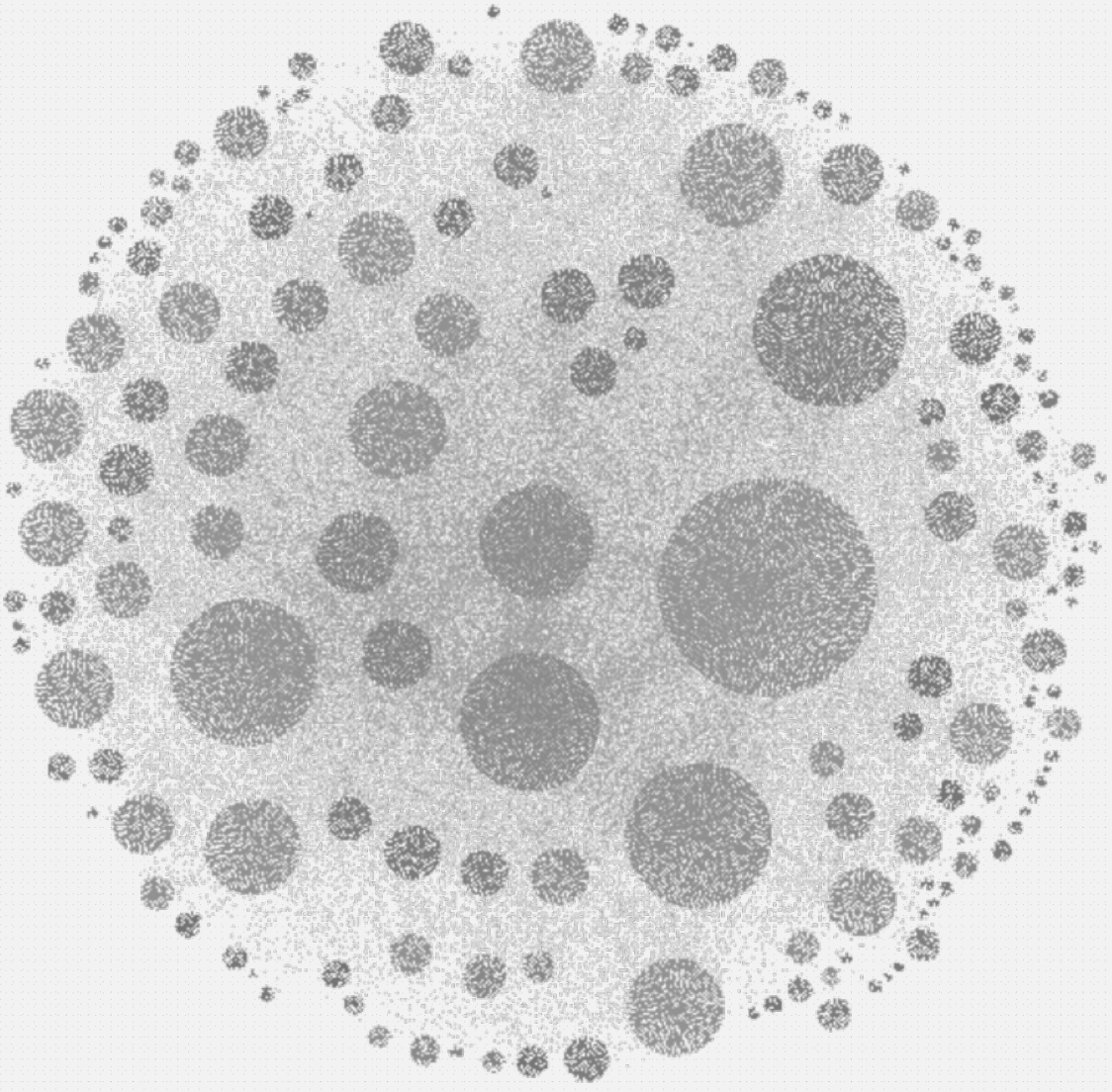
Support Vector Machines

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.

Contents

Classification

`/src/session_1.m`, `/src/homework_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.

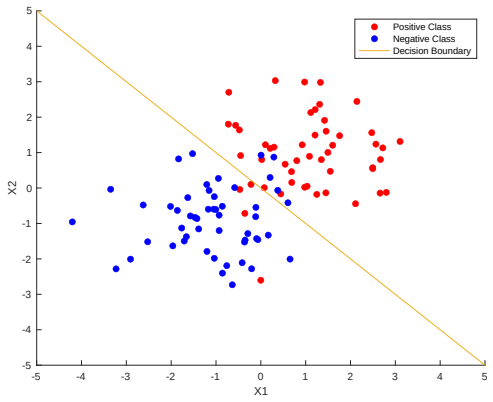


Figure 1: Decision boundary (in orange) for two Gaussian distributions with equal covariance matrices.

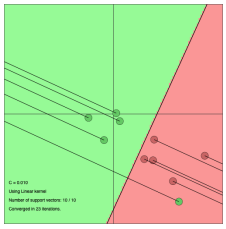
Support Vector Machine Classifier

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 :

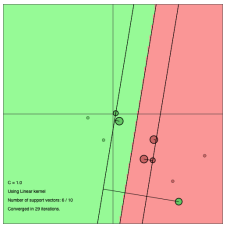
$$\min_{w,b,\xi} \frac{1}{2} \cdot w^T w + C \cdot \sum_{k=1}^{N} \xi_k \qquad \text{such that } y_k \cdot [w^T \cdot x_k + b] \geq 1 - \xi_k \text{ and } \xi_k \geq 0 \ (\forall k \in \{1, .., N\})$$

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[1].

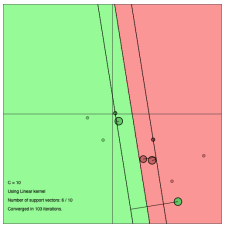
[1] In the dual problem formulation the α values are minimised for those data points that are 'similar' to data points that belong to the same class.



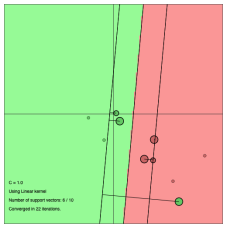
(a) $C = 0.01$ (large margin, lots of *'slacking'*.)



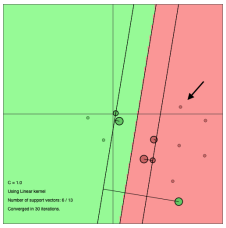
(b) $C = 1$



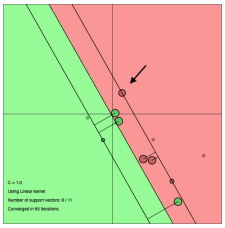
(c) $C = 10$ (small margin, little *'slacking'*.



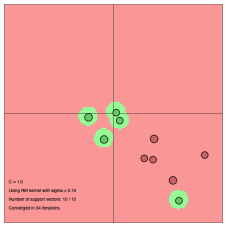
(d) Initial solution.



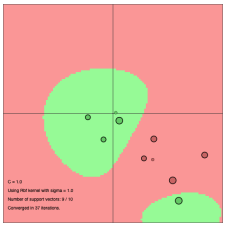
(e) Negative data points added at the right side of the margin.



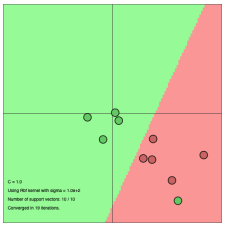
(f) Negative data point added at the other side of the margin.



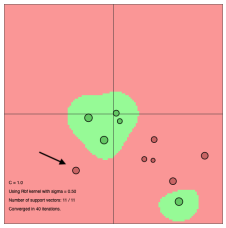
(g) $\sigma = 0.1$



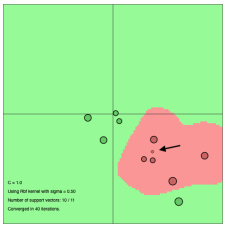
(h) $\sigma = 1.0$



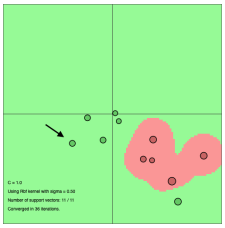
(i) $\sigma = 100$



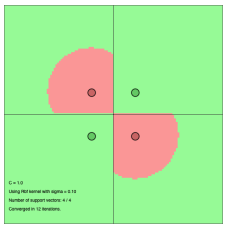
(j) Negative data point added far away from the boundary.



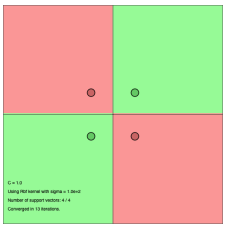
(k) Positive data point added inside the boundary.



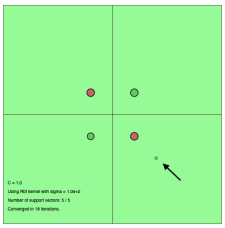
(l) Positive data point added inside the boundary.



(m) Modelling the `XOR` function ($\sigma^2 = 0.1$).



(n) Modelling the `XOR` function ($\sigma^2 = 100$).



(o) Same as the left two pictures with another data point belonging to the positive class.

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

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.

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 any two classes get separated in a very strict way - the Gaussian kernels are universal. In the limit the kernel matrix becomes an identity matrix. For large bandwidths the reach of each data point becomes large and the decision boundary can become near-linear as the density areas of each class end up competing[2]. In the limit the kernel matrix is filled with ones and the classification fails (in the demo this can happen at $\sigma^2 = 10^5$, in the LS-SVM toolbox a much larger value is needed). 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.

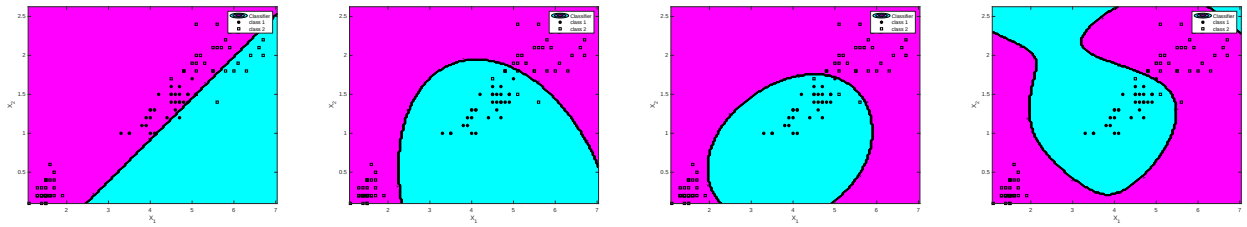
Least-squares support vector machine classifier

Figure 3 depicts results on the iris dataset using various polynomial kernels with $t = 1$. 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, t is 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. The experiment is of limited value as only few parameter combinations were tried.

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

[2]When the demo is run on a 4-point dataset representing the `XOR` function a checkerboard pattern is obtained when σ^2 is large (as seen in figure 2, last row). Even when σ^2 is low the separation is linear in between the data points as the problem reduces to that seen in the introductory example.

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 6, 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 (which admittedly can be addressed).

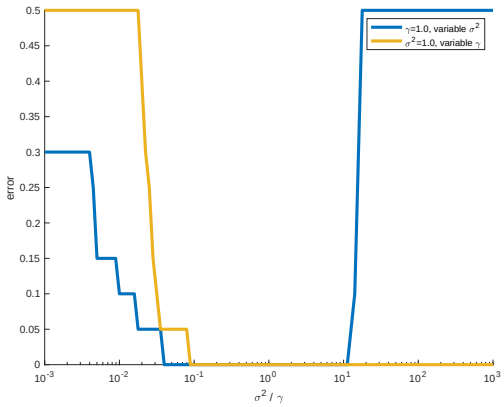


Figure 4: Performance of various RBF kernels on the iris dataset. The orange curve denotes the performance of the model for varying γ and fixed $\sigma^2 = 1.0$, the blue curve does the opposite ($\gamma = 1.0$, varying σ^2).

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 make the average γ and σ^2 large :

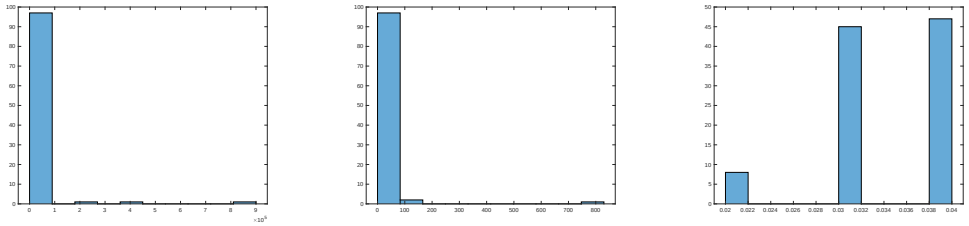
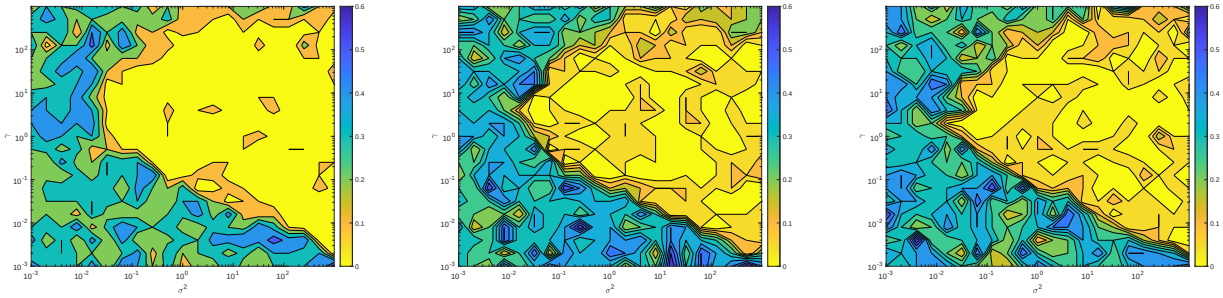


Figure 5: Histograms for 100 γ, σ^2 and costs returned by the automated tuning process.

Results of the automated tuning process correspond to the contour plots given in figure 6. As for the runtimes, the grid search ended up being twice as slow as the simplex method.



(a) Random split validation. (b) k-fold validation ($k = 10$). (c) Leave-one-out validation.

Figure 6: 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.

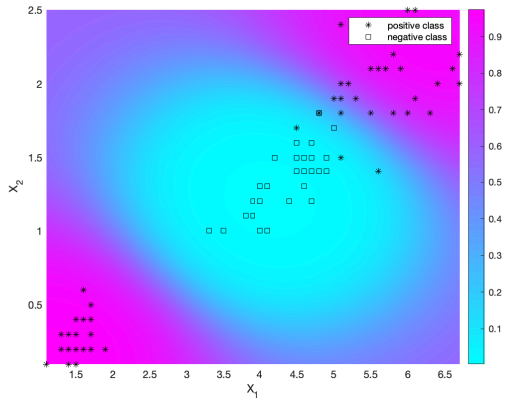
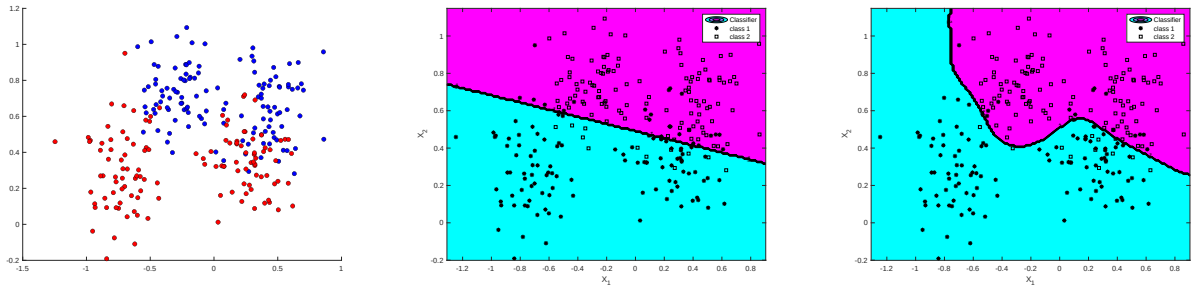


Figure 7: Posterior probability estimates using a Bayesian approach ($\gamma \approx 0.038, \sigma^2 \approx 0.56$). The hue indicates the probability that the data point at a location in the 2-dimensional plane belongs to one or the other class. A smaller σ^2 can lead to overfitting and makes for a 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 $\gamma = 0.037622, \sigma^2 = 0.559597$ it happens to equal 1 indicating a perfect classifier. 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 (considered in greater detail in the second part of this report) 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 7. The prior is taken to be 0.67 (the classes are unbalanced).

Ripley dataset



(a) Ripley dataset. (b) Results with a linear kernel (tuned parameter $\gamma \approx 0.0027$, misclassification rate of 10.2%, AUC ≈ 0.956). (c) Results with an RBF kernel (tuned parameters $\gamma \approx 0.537, \sigma^2 \approx 0.338$, misclassification rate of 9.1%, AUC ≈ 0.968).

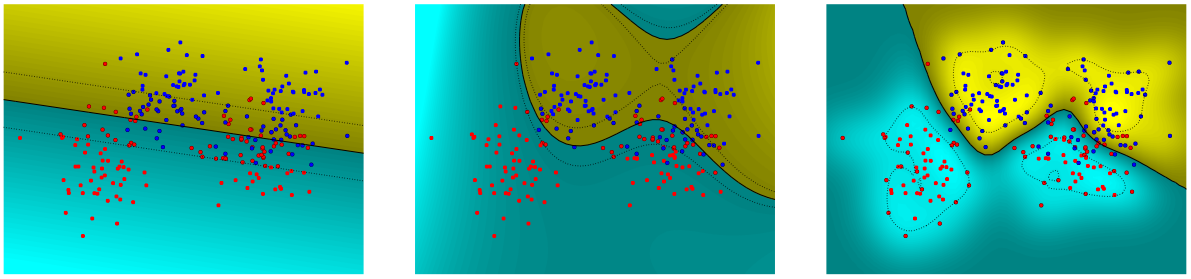
Figure 8: The Ripley dataset and results with a linear and an RBF kernel. LSSVMs are used. The number of support vectors equals 250 (100%) in all cases.

The Ripley dataset is a 2-dimensional so it can easily be visualised (figure 8). The data for each class is generated by a mixture of two Gaussian distributions and it is not linearly separable in the input space. The distributions were chosen as to *'allow a best-possible error rate of about 8%'*, each component has the same covariance matrix

(basically a 2-component version of the introductory example). The data is a *'realistic'* toy problem and it is rather noisy.

An LS-SVM with a linear kernel makes for a classification rate of around 10% while an RBF kernel performs a tad better at a rate of about 9% (and so does a polynomial one[3]). The ROC curves give a more complete picture and aren't shown here, but the AUC was the highest for the RBF kernel (about 0.97) with accuracy being the highest at a threshold a bit below zero. In practice (when met with real-life datasets) operating points on the ROC curve calculated on the validation set may be chosen depending on the cost of each type of misclassification.

Steve Gunn's `MATLAB` toolbox which is also used for regression in the second part of the report provides functions for generating classical SVMs and some more kernels that can be experimented with though it provides no automated tuning method for these. The results for linear -, RBF and polynomial kernels are about the same though there are less support vectors. Results after manual tuning (through the `uiclass` interface) are shown in figure 9.



(a) Linear kernel ($\gamma = 1.0, \rho = 11.5\%$, 125 support vectors)).

(b) Polynomial kernel ($\delta = 5, \gamma = 1.0, \rho = 10.0\%$, 77 support vectors)).

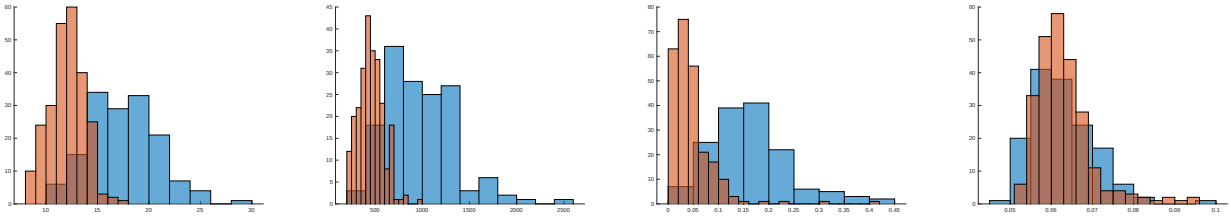
(c) RBF kernel ($\gamma = 0.5, \sigma^2 = 0.15, \rho = 9.2\%$, 124 support vectors).

Figure 9: The Ripley dataset and results with a linear and an RBF kernel. LSSVMs are used.

Since the dataset is Gaussian an RBF kernel should be an appropriate pick. As it's also a noisy dataset a smoother decision surface (higher σ^2 and/or lower γ) should be opted for. Of course in this case the optimal classifier (in terms of maximisation of the posterior probabilities) can be defined based on Bayes' theorem.

Wisconsin breast cancer dataset

With 30 features for each of the observations the Wisconsin breast cancer diagnosis dataset isn't trivial to visualise. `MATLAB` has an implementation of t-SNE or two principal components determined through PCA could be plotted (see figure 11). Distributions of some of the features are shown below. The two classes denote benign - or malignant cancer tissue. In contrast with the previous dataset this one's a bit unbalanced (62.5% belongs to the negative class) which can be accounted for through bias correction[4] or by using weights in the SVM formulation to give more emphasis to the terms associated to the minority class.



(a) Mean radius.

(b) Mean texture.

(c) Mean perimeter.

(d) Mean area.

Figure 10: Distributions of the first 4 features of the Wisconsin breast cancer dataset (red denotes malignant -, blue denotes benign tumors). There are 10 features in total and for each of these the mean, standard error and maximum value is given. For some of the features none of those three appear to be particularly informative.

A simple automated tuning experiment of an LS-SVM with linear -, polynomial - and RBF kernels yielded average AUCs of 0.996 ± 0.001, 0.937 ± 0.058 and 0.995 ± 0.002 respectively. It may be surprising that a linear

[3] Automated tuning of the polynomial usually resulted in a degree δ between 4 and 7.

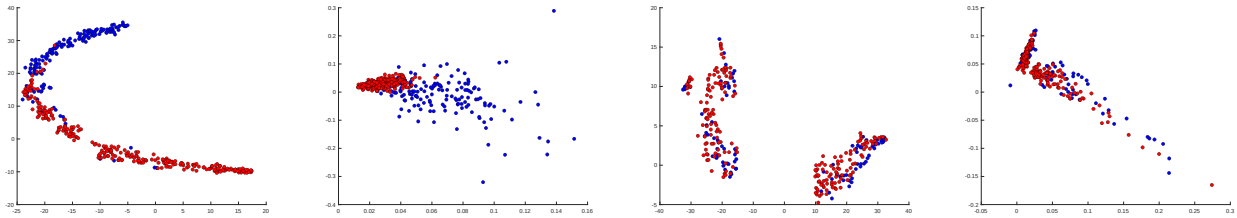
[4] Whilst doing so it came to my attention that the `roc` function in the LS-SVM toolbox *seems* to return the thresholds in the wrong (reverse) order.

	ARD (RBF kernel)	PCA ($n_h = 2$)	PCA ($n_h = 3$)	PCA ($n_h = 10$)
Error	$2.075\% \pm 0.206$	$6.125\% \pm 0.339$	$3.825\% \pm 0.237$	$3.750\% \pm 0.167$
AUC	0.99723 ± 0.00040	0.98402 ± 0.00335	0.99077 ± 0.00238	0.98729 ± 0.00520

Table 2: Mean and standard deviation of accuracy and AUC for linear kernel models after application of two types of dimensionality reduction ; PCA or automatic relevance determination (which selected 21 features). PCA was applied by subtracting the mean, applying PCA on the training set and using the resulting projection on both the training - and test set after which an LS-SVM with an RBF kernel was tuned and trained. A linear kernel wasn't as performant as the number of dimensions decreased.

kernel outperformed a non-linear one - the number of features is larger and there's not that many data points so a simpler decision boundary may help to avoid overfitting since Cover's theorem implies that it is more likely that the data is linearly separable when the number of dimensions is high (and the number of data points is low).

As some of the features seem less informative than others it seemed worthwhile to remove some of them either through PCA or a more sophisticated approach like automatic relevance determination. A basic experiment ended up with the latter selecting 21 features. Training the model on these resulted in an AUC of 0.9973. The results with a larger number of tuning and training runs as well as a comparison with a basic PCA approach are shown in table 2.



(a) Breast cancer diagnosis dataset (t-SNE).

(b) Breast cancer diagnosis dataset (PCA).

(c) Diabetes dataset (t-SNE).

(d) Diabetes dataset (PCA).

Figure 11: Visualisation of the Wisoncin breast cancer - and the Pima indians diabetes datasets give an idea of how separable they are. The diabetes dataset appears to be significantly harder and this is reflected in the lower accuracy obtained with various models.

Diabetes dataset

A 2-dimensional visualisation of the 8-dimensional Pima indians dataset (figure 11) reveals it to be more difficult than the previous one (distributions of the input variables tell the same story). Mean error rates of $21.7\% \pm 1.4$, $27.6\% \pm 5.3$ and $22.2\% \pm 1.8$ were obtained with a linear, polynomial and RBF kernel (averaged over 10 runs using tuned parameters)[5]. The average AUCs were 0.844 ± 0.001, 0.721 ± 0.112 and 0.848 ± 0.004. An RBF kernel appeared most performant. Automatic relevance determination did not prune any of the input variables which made subsequent poor results with PCA not surprising.

The original paper dealing with this dataset used a custom algorithm related to neural networks. They used all training examples and reached an accuracy of 76%. The performance of an LS-SVM with an RBF kernel trained on the full training set (named `total`) was typically comparable.

A quick comparison was done with k-Nearest Neighbour (kNN) and a random forest classifier. For $k \in [1, 10]$ the nearest neighbour algorithm had an accuracy below 70% which is significantly worse while the random forest classifier (`TreeBagger` in `MATLAB`) did a bit better with an accuracy of about 73%.

It looks like the accuracy is amongst the better ones. Fisher Discriminant Analysis (FDA) appears to perform alright as well but isn't experimented with here since it seemed redundant due to the close relation with LS-SVMs. Maybe more advanced neural networks might do better. A simple neural network constructed in `MATLAB` also got into the 70% accuracy[6].

[5]The AUC for the training set turned out to be quite high for the polynomial kernel (> 0.9) which seemed to imply some degree of overfitting. Changing the cross-validation method to leave-one-out made the AUC for the test set comparable to that of the RBF kernel.

[6]Running a Python notebook found online which claimed >90% accuracy using a fairly simple neural network turned out to be overfitting and evaluation was done on the training set, the real accuracy on the test set was 64%.

Function Estimation and Time Series Prediction

`/src/session_2.m`, `/src/homework_2.m`

Support vector machine for function estimation

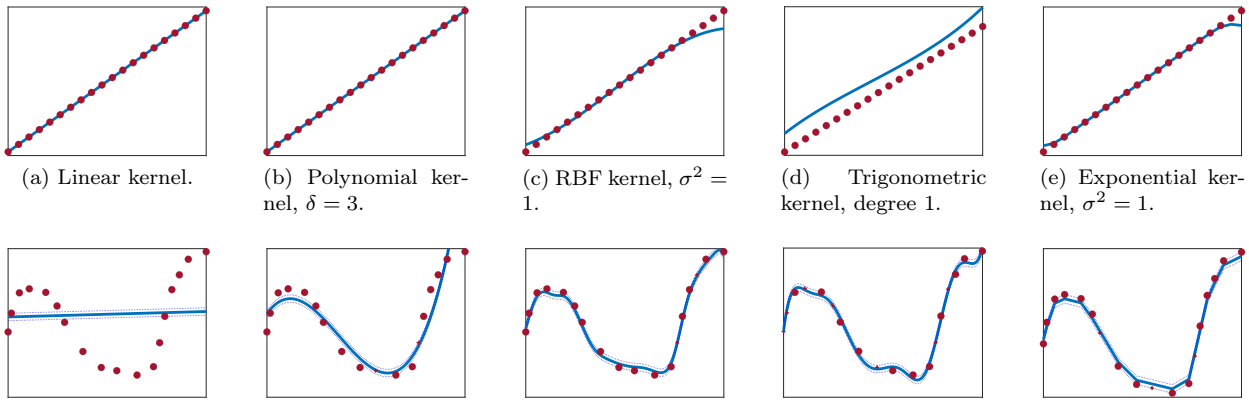
Two datasets are experimented with in figure 12. The first exists of 20 points lying on a slope. Therefore one can expect the linear kernel (which makes for a linear model) to outperform the others. The second (non-linear) dataset is more challenging such that other kernels which enable the SVM to model the non-linearities have to be used instead.

From the problem statement :

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

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 constrained 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 12: Visualisation of the results of experiments with various kernels and parameters for two different artificial 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

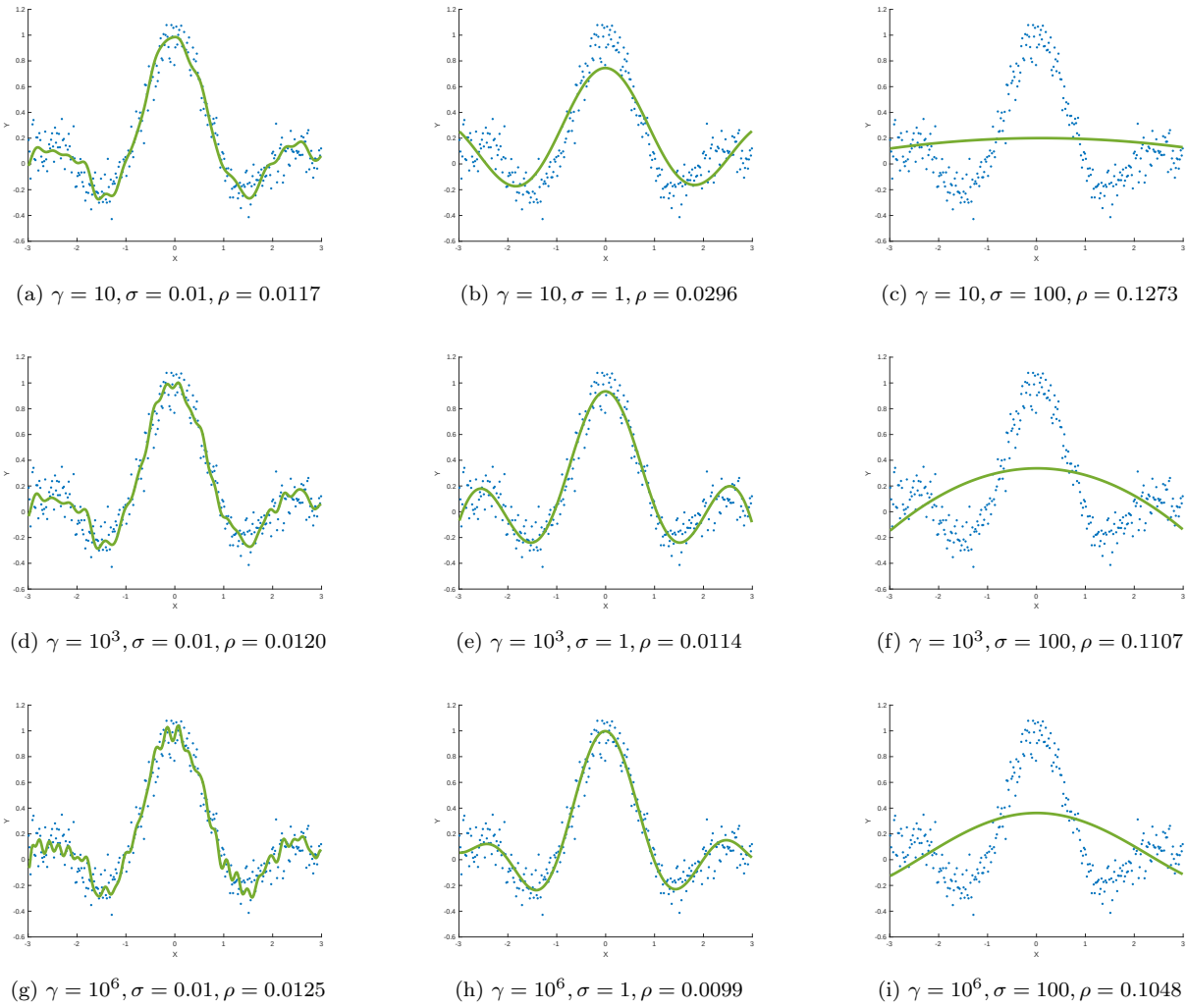
Figure 13 shows the results of LS-SVM regression applied on the `sinc` function. The mean squared error is lowest for $\sigma^2 = 1.0, \gamma = 10^6$. The smaller σ^2 value (0.01) leads to overfitting (the noise is being modelled). In this case the underlying function is known and it seems reasonable to take $\sigma \in [0.1, 1.0]$ and, say, $\gamma = 10^3$. However, an optimal parameter cannot be found as it depends on the training set that is being dealt with 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 3: 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 13: 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.

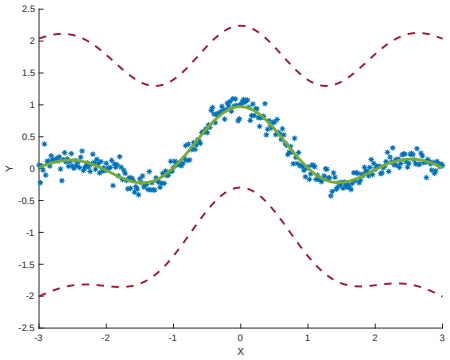


Figure 14: Results of Bayesian inference applied on training data based on the `sinc` function with some added white noise (blue). The function estimate is coloured in green, 95% error bars are indicated in red.

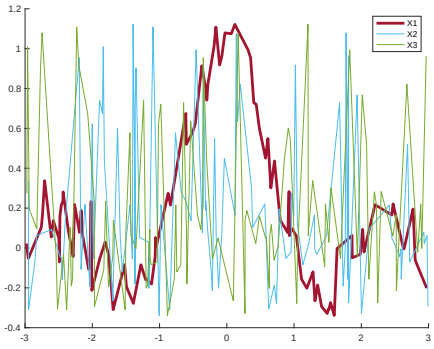


Figure 15: Plot of 3-dimensional input data of which only the first dimension models the `sinc` function, the other 2 being white noise.

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 14 where the method is applied on the `sinc` function.

Automatic relevance determination (ARD)

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 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 x_1 to be the sole relevant one which is also illustrated in figure 15. An other, crude way of doing relevance determination is to consider all possible non-empty subsets of the input features and applying cross-validation on the resulting data to figure out which subset generalises 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 ($mse \approx 0.012$). This method wouldn't be satisfactory in practice as the power set grows sharply as the number of input dimensions increases. Treating the problem as a global optimisation is more appropriate for example. Or even simply removing random features one by one as long as the results keep on improving.

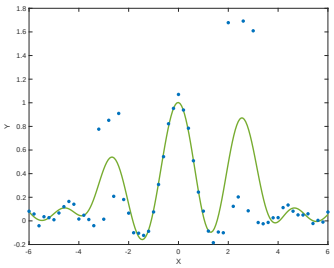
Robust regression

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 due to its gradient (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. Specifically, a weight is assigned to each data point based on its error. Some possible weight functions are shown below.

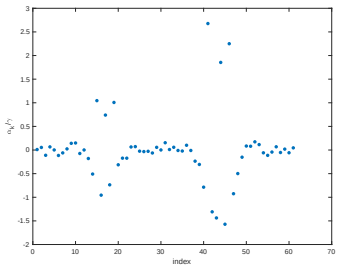
After assigning the weights a weighted LS-SVM is trained and the whole process may be repeated as desired. Results with 4 weight functions are compared with a non-robust LS-SVM approach in figure 16.

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

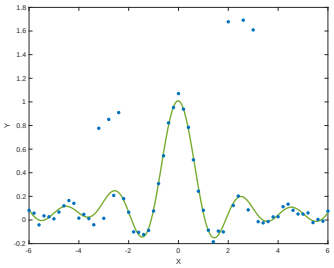
Table 4: Possible weight functions for use in robust regression with LS-SVMs.



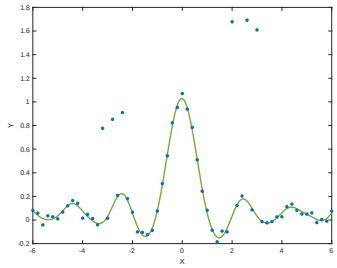
(a) Non-robust regression.



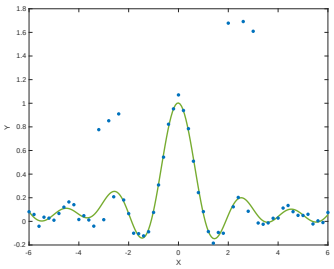
(b) Error values for the non-robust regression ($\frac{\alpha_k}{\gamma}$). After robust regression the errors decrease for the non-outliers which means that aside from those 6 outliers at the top the remaining values are centered around the x-axis.



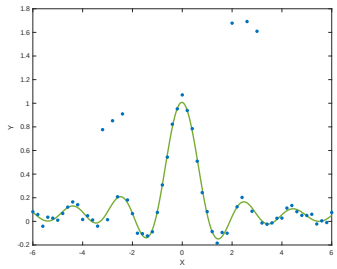
(c) *Huber* weight function.



(d) *Hampel* weight function.



(e) *Logistic* weight function.



(f) *Myriad* weight function.

Figure 16: 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

The provided model for predicting measurements for the logmap dataset performs quite badly as can be seen below. The results call for the tuning of the three parameters.

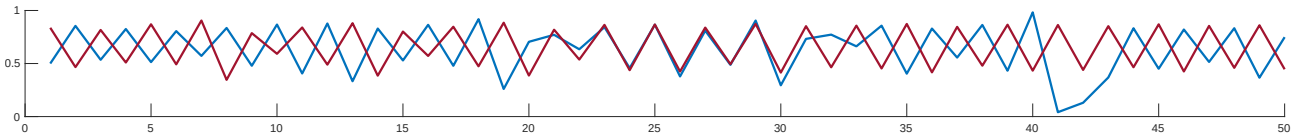


Figure 17: Initial attempt at modelling the logmap dataset (ground truth in blue, prediction in red).

A straightforward way of picking a so-called order of a time series model (i.e. the number of past measurements taken into account for prediction of the next measurement) is to take advantage of the fact that the given dataset is rather small and the order is a discrete variable. This makes it sensible to go over all possible orders and tune regularisation parameter γ and any of the kernel parameters for every possible order separately.

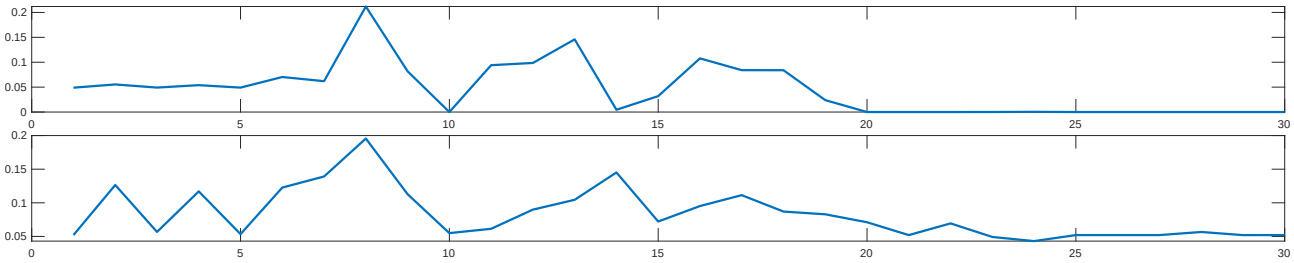


Figure 18: Mean squared errors for just one tuned model per possible order. The top row gives the error for the training set, the bottom row for the test set. A minimum is obtained starting at `order` = 21. The mean absolute errors give a slightly more meaningful picture here because a model predicting the mean value for every time step could have a smaller mean squared error than a more meaningful model (due to the fact that the data lies between 0 and 1).

To find good values for the parameters either cross-validation or the Bayesian framework discussed previously can be used. Both methods were tried and the results with the Bayesian framework are shown in figure 18. In that figure the mean squared errors are plotted in function of the order for LS-SVMs using RBF kernels with tuned parameters. This was measured for the training and the test set. A useful order appeared to lie somewhere in the early 20s and picking anything higher may cause poor generalisation due to overfitting. Results with a model with its order equal to 20 are shown below. The performance has clearly improved.

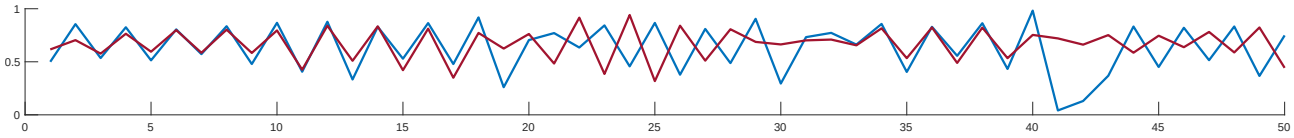


Figure 19: Results for a model using an RBF kernel with tuned parameters. The order is 21, $\gamma \approx 114481.964$ and $\sigma^2 \approx 985.571$.

Santa Fe dataset

The per-order mean squared errors for an analogous tuning approach applied to the Santa Fe dataset (this time by using cross-validation rather than making use of the Bayesian network) are shown in figure 20.

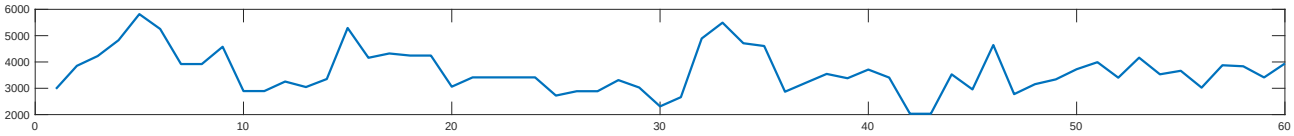


Figure 20: Mean squared errors for various models in function of the order. Parameters were tuned automatically using a grid search. Outliers (large errors) are 'ignored' (i.e. set equal to the error for the previous order) to improve the visualisation.

Picking the model with lowest error (prioritising a small order) gave the following result :

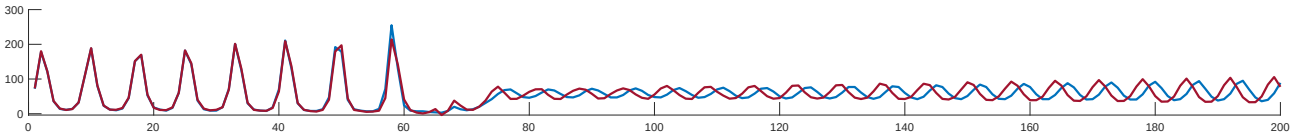


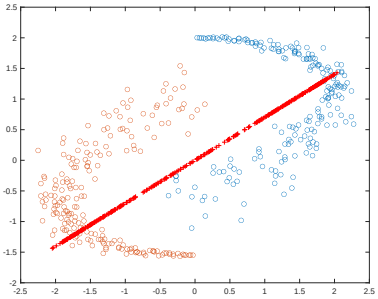
Figure 21: Results for a model using an RBF kernel with tuned parameters. The order is 42, $\gamma \approx 8423.289$ and $\sigma^2 \approx 14.017$.

Due to the limited number of runs the whole tuning procedure by no means proves that this model is best or that this order is optimal. By doing more experiments confidence can grow that certain orders are optimal for this particular dataset.

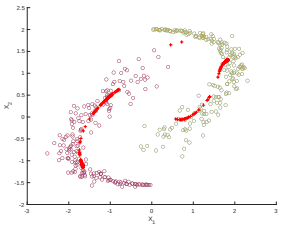
Up til now only LS-SVMs were toyed with. It's also possible to work with classical SVMs which - due to the choice of loss function - provide sparsity without necessitating a pruning post-processing step or some other approach to enforce it. Using `scikit-learn` a script was built to do classical SVM regression using the `SVR` class. Grid search was done to tune the parameters of an RBF kernel first, which proved to be equivalent to the LS-SVM result while having less support vectors. An attempt was made at using an other kernel called the ANOVA kernel which is said to perform well in the case of multivariate regression problems, but despite some limited tuning (with a grid search) the results weren't close to those of the RBF kernel. The sparsity of the classical SVMs with RBF kernels wasn't impressive ; about 90% of the training set was kept. Other strategies such as LSTMs are always possible as well though the obtained results appear quite satisfactory.

Kernel principal component analysis

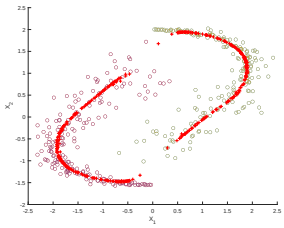
The point of *linear principal component analysis (PCA)* is to find a linear transformation (a projection onto an orthogonal basis) such that the variance of the projected data points is maximised. It's an old method introduced by Pearson. The transformed variables are called principal components. By 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.



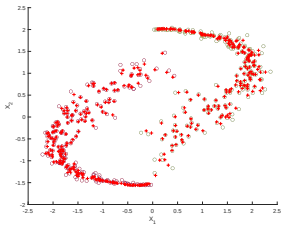
(a) De-noising with linear PCA.



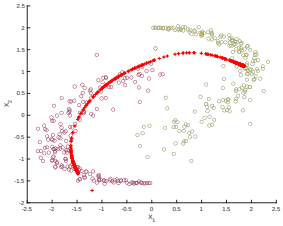
(b) $\sigma^2 = 0.4, n_h = 2$.



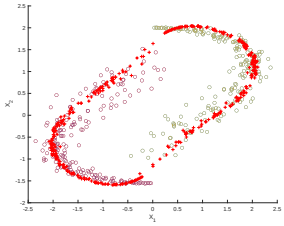
(c) $\sigma^2 = 0.4, n_h = 6$.



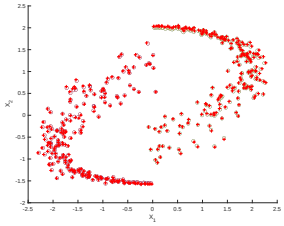
(d) $\sigma^2 = 0.4, n_h = 20$.



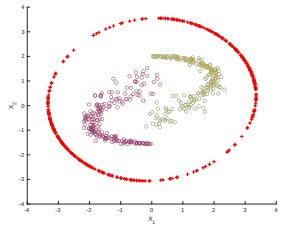
(e) $\sigma^2 = 1.0, n_h = 2$.



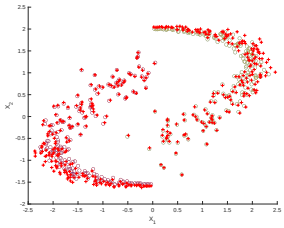
(f) $\sigma^2 = 1.0, n_h = 6$.



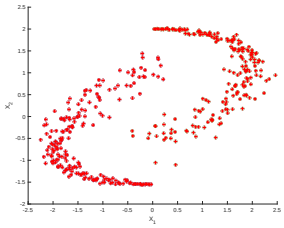
(g) $\sigma^2 = 1.0, n_h = 20$.



(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 22: 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. All 9 images below the top one use an RBF kernel. De-noising is done by minimising the distance between the feature mapping of an input and its projection in the feature space onto principal axes. 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 :

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

The errors represent the variance after projection which is to be maximised. 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 and where centring is not a straightforward thing to do.

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 apply cross-validation techniques 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). They can be determined for any input vector x by finding the input vector z which minimises $\|\phi(z) - P_{n_c} \cdot \phi(x)\|^2$ (where $\phi(x)$ is the feature mapping of input z and $Pn_c \cdot \phi(x)$ is the projection of x onto a subspace in the feature space spanned by the n_c first eigenvectors found through kPCA.

More appropriate (less costly) approaches to tune the parameters include kernel Parallel Analysis (kPA) which is an extension of classical Parallel Analysis (PA). PA is used to find an appropriate number of components for PCA instead of just going by manual inspection of the scree plot or applying a rule like Kaiser's (taking all eigenvectors for which the eigenvalue is larger than 1). Also possible is Model selection based on Distance Distributions (MDD) which improves upon kPA by improving noise estimation when the number of dimensions is low.

Consider the de-noising of a toy dataset for which the resulting pre-images after kernel PCA using various parameter values are shown in figure 22. 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 a loss of non-noisy information when set too low (or a lack of de-noising if set too high, as seen in figure 22).

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.

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-complete a relaxed version (where the indicators can be real) is considered instead. The solution of the relaxed version is based on the solution of a generalised eigenvalue problem after which the data points can be embedded in an eigenspace on which a clustering algorithm like *k-means* can be applied. In the case of the binary toy dataset dealt with in this section the Fiedler vector is used in conjunction with the `sign` function.

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. The bias term plays the role of a centering operation which is not trivial like in the case of kernel PCA. It is not used in the provided code snippets which causes some issues with thresholding (k-means could be used or manual inspection in simple cases).

Experiments with an RBF kernel are shown in figure 30. 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 too small or too large 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 an approach that can be considered in the case of fixed-size LS-SVMs is related to the Nyström method which is traditionally used for integral approximation. In the current setting it means that one takes a set of support vectors of the input data (of size $M \ll N$) and then uses it 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. This makes it fast to compute which is important in the context of big data (besides accuracy).

Examples of subsamples obtained through optimisation of the Renyi entropy are shown in figure 23. 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. This is normal ; maximising the entropy means that one maximises the uncertainty, diversity or un-informativeness of the system. As the bandwidth increases the reach of each data point gets larger, the similarity with distant points increases and maximising entropy - which involves prioritisation of dissimilar points - causes points maximally distantiated from each other to be picked as support vectors.

In figure 24 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. The goal is to increase sparsity as determining M isn't trivial. The 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 $\mathcal{O}(NM^2)$. While the error is about the same 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. It may not be the most sparse representation possible ; as obtaining ℓ_0 is NP-hard only a local minimum is found through an iterative procedure.

Kernel principal component analysis

De-noising a quite noisy USPS dataset with classical PCA gives poor results at best (see figure 25). Kernel PCA with an RBF kernel manages to exploit the data much more and the results are a big improvement (see figure 26).

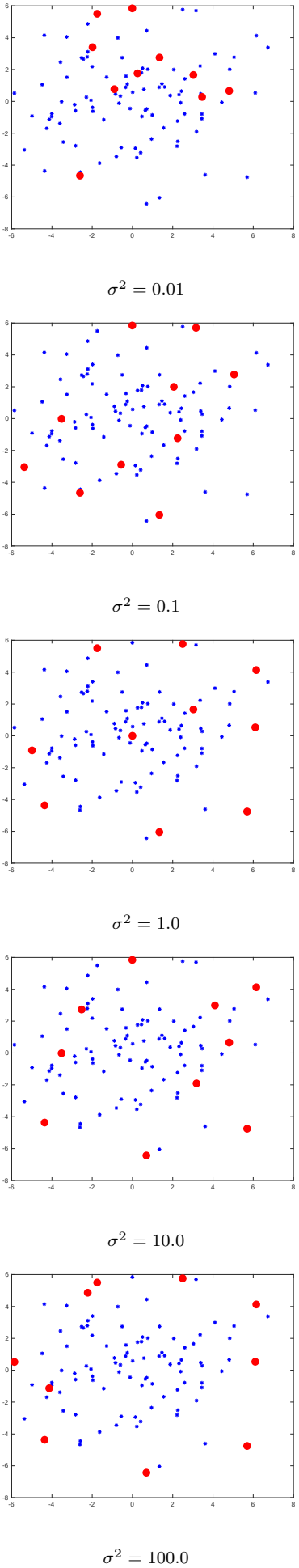
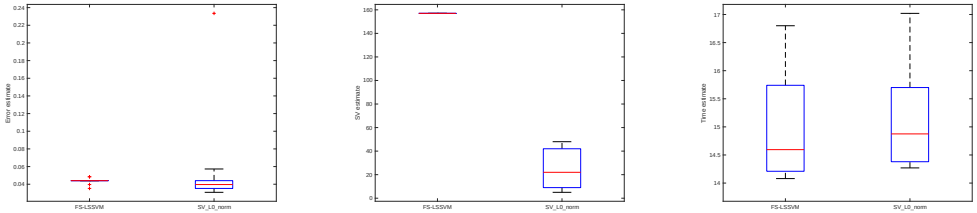


Figure 23: Subsample of input data points obtained for a fixed-sized LS-SVM using an RBF kernel of varying σ^2.

What was noted in the yin-yang toy dataset shown at the beginning of this section can be observed again in figure 27 ; small bandwidths remove important information (such that some digits get reconstructed without noise but as the wrong number) while larger ones fail to de-noise and noise is re-introduced slowly but surely in the reconstructions. Eventually the results resemble that of classical, linear PCA.

The small bandwidths cause the data points to be considered highly dissimilar and directions of largest variation do not correspond to clusters with the same digits - rather, every data point is its own little cluster and variation is large in many meaningless directions. Conversely, very high bandwidths cause the data points to be considered very similar to each other and it once again gets hard to find meaningful directions of variation such that noise shared by the data points may be considered informative again.



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

Figure 24: Comparison of the standard fixed-size LS-SVM approach with a modified approach applying an ℓ_0-penalty to obtain a sparser representation.

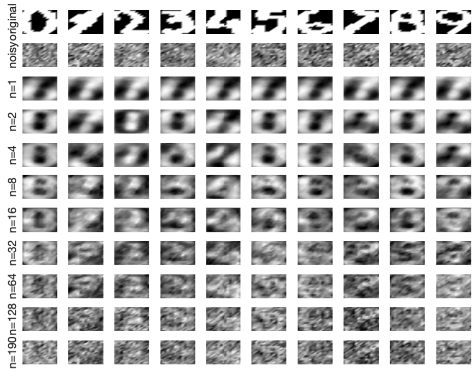


Figure 25: Results of linear PCA applied to the USPS dataset (top rows are original and noisy data, next 9 rows are results for $n_c \in [1, 2, 4, 8, 16, 32, 64, 128, 190]$).

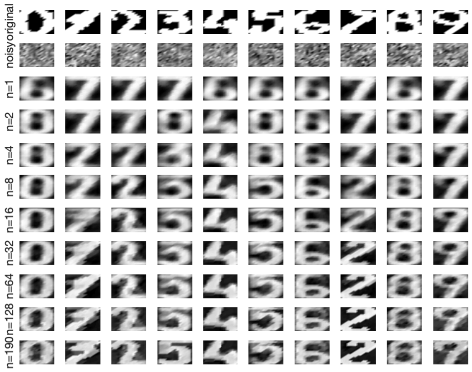
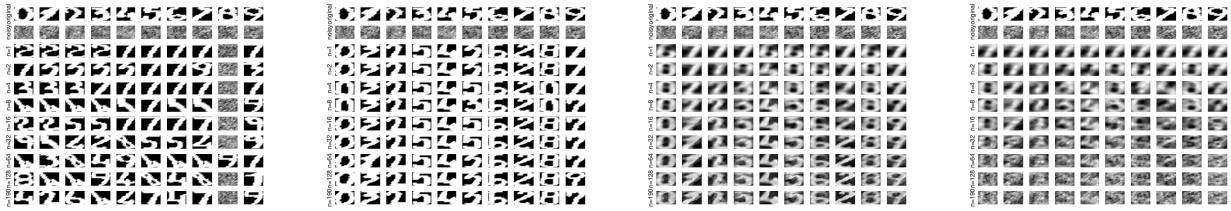


Figure 26: Results of kernel PCA applied to the USPS dataset (top rows are original and noisy data, next 9 rows are results for $n_c \in [1, 2, 4, 8, 16, 32, 64, 128, 190]$).

Of note ; for slightly too small bandwidths, say $f = 0.01$ or $f = 0.1$ it looks like reconstructions are simply training data samples. This appears to be caused by the algorithm that calculates pre-images. It's a fixed-point iteration algorithm that takes a linear combination of the training data at each step.

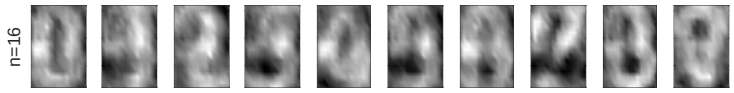


(a) $f = 0.01$. (b) $f = 0.1$. (c) $f = 1.0$. (d) $f = 10.0$.

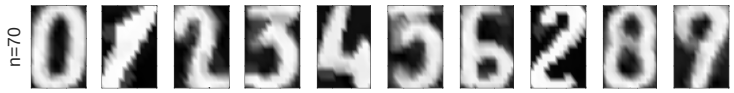
Figure 27: De-noising the USPS hand-written digit dataset. f is a factor with which is multiplied by about 51 to get the bandwidth. Very low and very high factors f (e.g. 0.001 and 1000) lead to noisy reconstructions (resembling the second rows) and aren't pictured here.

It was stated previously that one way to tune kPCA is to measure reconstruction errors for the validation set to gauge the effectiveness of certain parameter combinations. A plot of the reconstruction errors measured for the provided training and validation sets is shown in figure 29. The errors tell a meaningful story which also corresponds to the visualisation of the performance of various parameter combinations in figure 27.

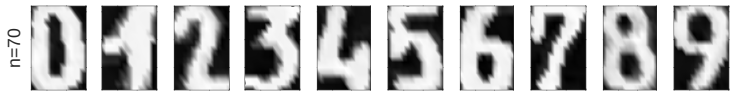
Using the tuned parameters based on the contour plots better results are obtained. This can be seen in figure 28. The noise has been removed for the most part though one (and a half) de-noised digit is wrong.



(a) Results with linear PCA ($n_c = 16$, first validation set) for comparison's sake.

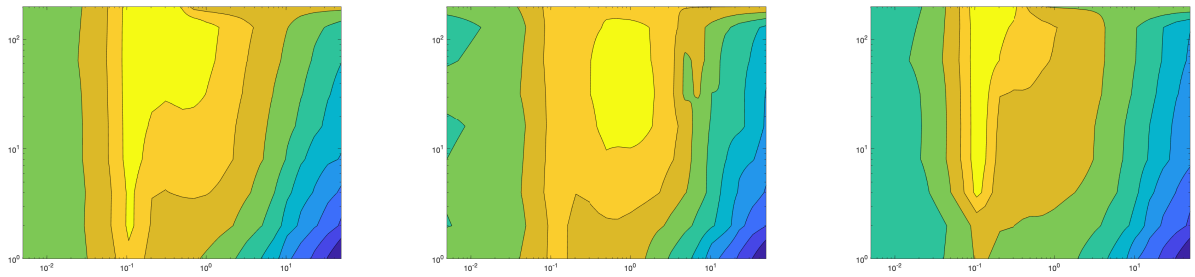


(b) Results for validation set 1.



(c) Results for validation set 2.

Figure 28: Denoising of the USPS dataset using tuned parameters ($n_c = 70, f = 0.4$).



(a) Errors (root mean squared errors) for the training set.

(b) Errors (root mean squared errors) for the first test set.

(c) Errors (root mean squared errors) for the second test set.

Figure 29: Reconstruction error for the training and validation sets in function of the bandwidth and the number of components. This is a crude grid search (x-axis represents the factor f, y-axis is the number of components) in combination with interpolation to construct a contour plot. The axes are scaled logarithmically. More appropriate tuning methods include kernel parallel analysis or Model selection based on Distance Distributions (MDD). As the number of components rises, the bandwidth can be higher before de-noising fails.

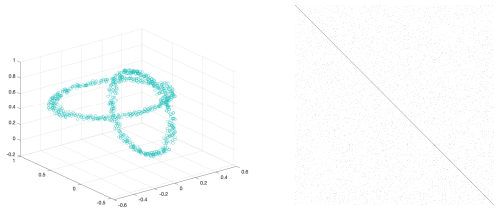
Fixed-size LS-SVM

Next, two datasets are experimented with. LS-SVM with or without sparisification is applied to the Shuttle - and then the California housing dataset. The first is a classification problem, the second a regression problem.

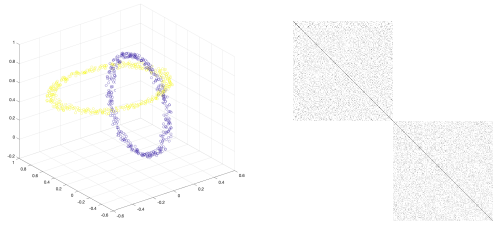
Shuttle dataset

The Shuttle dataset has 58.000 train or test samples. It's a multi-class classification problem, having a total of 7 classes which are not equally represented. The first class makes for 45.586 out of 58.000 samples or about 78.5% of the dataset so generally 80% is seen as the so-called majority rule and the aim is to reach an accuracy of at least 99%. The class imbalance can be observed in the distributions (figure 31).

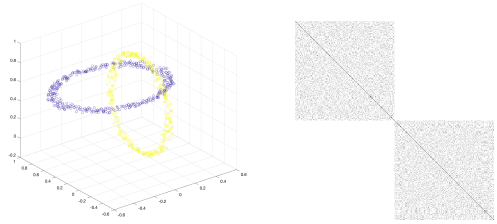
The provided code reduces the dataset's size to 700 samples in which only 5 of the classes are present (more than 81% being the majority class), presumably because of the long runtimes. It does not seem to handle multi-class problems per se but still makes use of the sign function for predictions effectively treating the problem as a sort one-versus-all binary classification problem (though because the class labels aren't set to minus one it's not even exactly that). Therefore initial results see a fairly large error rate which tend to get worse than the majority rule.



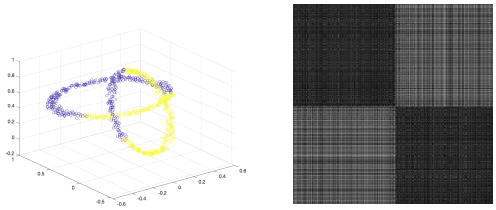
(a) $\sigma^2 = 0.001$.



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



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

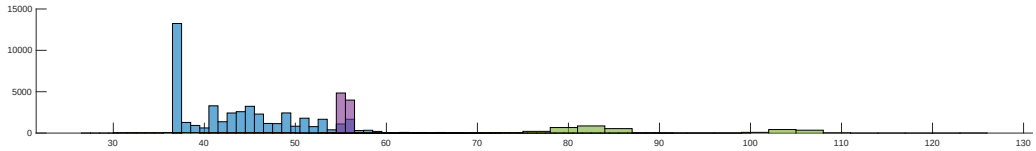


(d) $\sigma^2 = 1.0$.

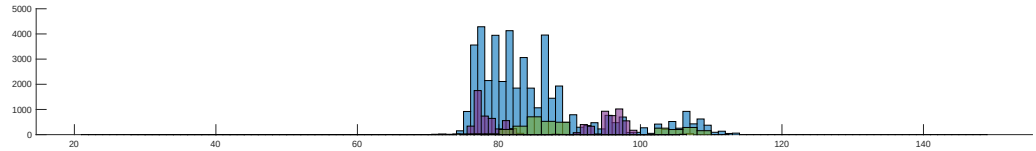
Figure 30: Clustering of a toy dataset consisting of two rings. KSC was applied using a kernel with varying σ^2 values. When this bandwidth is too large the reach of influence of each data point becomes too large and the clustering becomes subpar. The block-diagonals that are shown are similarity (or affinity) matrices after they've been sorted based on cluster membership. The blue rings for σ^2 are due to poor clustering because of using the `sign` function to threshold. With a better threshold the clustering works fine. For even smaller bandwidths the clustering deteriorates again. Projections onto the principal subspace are supposed to be collinear in the case of ideal clustering and are not shown here.

However, from these results which are shown in figure 36 it can be seen that the ℓ_0 post-processing step does not increase the required time (nor does windowing), which mirrors the results in the study proposing these methods. The required time across values for k (which specifies the number of prototype vectors indirectly through the relation $M = k \cdot \sqrt{N}$) matches the theoretical complexity $\mathcal{O}(k^2 \cdot N^2)$. The post-processing also makes the number of support vectors drop, and the error tends to increase a bit.

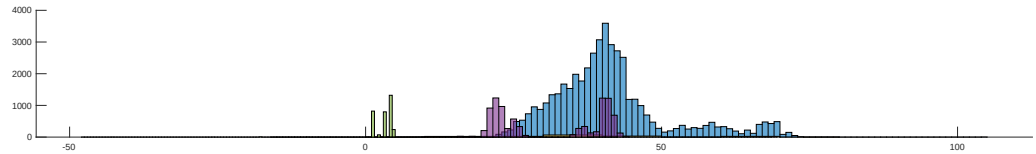
In further experiments instead of using the provided set, at least 3000 samples were picked (such that class 3 was represented more than once and stratification could be applied). For parameter tuning 5-fold cross-validation was applied (minimising misclassification), whereas the evaluation was done by creating a 75/25% training & test set split through stratified sampling.



(a) First feature (time).



(b) Third feature (time).



(c) Seventh feature (time).

Figure 31: 3 out of 9 features of the Shuttle dataset. 48.586, 50, 171, 8.903, 3.267, 10 and 13 samples belong to class 1, 2, 3, 4, 5, 6 and 7 respectively. The class names are Rad Flow, Fpv Close, Fpv Open, High, Bypass, Bpv Close, Bpv Open. The data appears reasonably separable, especially when using non-linear kernels.

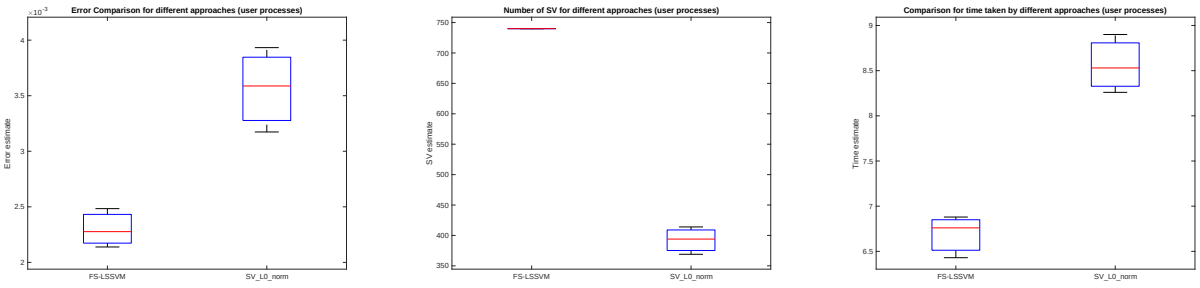
To get more meaningful error estimates and an overall comparison between a fixed-size LS-SVM with or without sparsification a multi-class model was built which starts with distinguishing the majority class, then proceeds to distinguish each of the next classes taking the largest classes first. In other words ; class 1 was compared with the others, a model was built on that basis, then for all samples not belonging to class 1 this same procedure was applied, this time for the second largest class (class 4). And so on until only two classes remain. The whole dataset was used except for classes 6 & 7 as misclassifying them would barely affect the accuracy anyways. In reality the cost of these misclassifications could be considered more serious which would call for an other approach (the ROC curve can always be used during validation to pick useful operating points). Average errors, number of support vectors and time requirements are shown in figure 32 and a short comparison of accuracy with some other approaches is shown below.

Method	*Accuracy*	*Computational Time*	*Support Vectors*
kNN ($k = 1$)	99.86%	*negligible*	*NA*
kNN ($k = 2$)	99.82%	*negligible*	*NA*
kNN ($k = 3$)	99.81%	*negligible*	*NA*
FS LS-SVM multiclass ($k = 2$)	99.77%	7.69s	740.00
FS LS-SVM multiclass ($k = 4$)	99.82%	18.03s	1466.67
FS LS-SVM multiclass ($k = 6$)†	99.83%	31.37s	2176.00
FS LS-SVM multiclass (ℓ_0 reduced, $k = 2$)	99.61%	9.15s	451.67
FS LS-SVM multiclass (ℓ_0 reduced, $k = 4$)	99.74%	33.09s	815.67
FS LS-SVM multiclass (ℓ_0 reduced, $k = 6$)	99.70%	87.72s	1804.00
FS LS-SVM multiclass (ℓ_0 reduced, $k = 8$)	99.67%	158.27s	2791.67

Table 5: Comparison of some methods for classification of the Shuttle dataset. Averaged over 3 runs. Tuning the models was done through 5-fold cross-validation and evaluation was done on 25% of the data (14.494 samples). RBF kernels were used (linear models reached an accuracy of a little above 90%). Classes 6 & 7 were disregarded. For kNN the algorithm keeps on degrading as k (the number of neighbours) increases. At $k = 4$ it tends to start performing worse than the fixed-sized LS-SVMs. Note that the number of support vectors is for all SVMs together and the time for tuning wasn't measured. †Only 1 run was used for this setting due to long runtimes.

A general trend that was seen in all experiments is that increasing k tends to decrease the error rate up until a certain point (usually until $k = 6$). The results speak for themselves and are proportional to the one-versus-the-rest approach discussed previously when it comes to computational time and the number of support vectors.

Ironically a simple k-nearest-neighbor (kNN) search using the `MATLAB` function `knnsearch` performed better for low values of k (this time denoting the number of neighbours that are considered). A bit surprising maybe, though the dataset is not particularly hard to classify and kNN can be quite powerful due to its complex decision boundaries. On the flipside it takes in more memory as the training set is quite large in comparison with any of the measured number of support vectors. And it cannot be tuned as much ; only k and the distance function can be.



(a) Error estimation.

(b) Estimation of the number of support vectors.

(c) Time estimation.

Figure 32: Estimates of the error, number of support vectors and computational time for a multiclass SVM approach. The estimates are based on a 75/25% train & test split obtained through stratified sampling. The number of prototype vectors is $M = k \cdot \sqrt{(N}$ with $k = 2$.

In the case of fixed-sized LS-SVMs γ, the type of kernel and its parameters, k and even some other things such as the tuning algorithm can be changed. However, because of costly tuning processes no more experiments were done.

California housing dataset

The California housing dataset contains information about block groups in California, spanning a population of about 30 million people in total. The 8 features include such things as median income, mean number of rooms in houses, ... and the dependent variable is the median house value. In other words, a 9-dimensional regression problem. The linear dependences (correlation coefficients) of each feature with the median house value are shown below in table 6. Median income correlates most strongly. It looks like the 'ocean proximity' attribute has been removed as it is a text attribute.

At first it looks like there are outliers in the mean house value distribution. These could be removed or a reweighing scheme could be used. It's unclear why they're there. Maybe a cap was set at 500k and anything above that was rounded down. Or maybe they're genuine values due to the the big cities in California (which seems fairly unlikely). The regions where houses are pricey are shown in figure 34 - they coincide with Los Angeles and San Fransisco. Regions with a higher median income that are located close to the ocean.

Feature	*Correlation*
Longitude	-0.04597
Latitude	-0.14416
Housing Median Age	0.10562
Total Rooms	0.13415
Total Bedrooms	0.05059
Population	-0.02465
Households	0.06584
Median Income	0.68808
Median House Value	1.00000

Table 6: Correlation coefficients between any feature and the dependent variable (median house value).

For testing purposes a test set was generated by doing a random split. This turned out to be fairly representative (based on the means of the variables for the train and the test set as well as visual inspection of the resulting histograms). There are more rigorous ways to do stratified sampling for multivariate regression problems which are not dealt with here.

Initial experiments obtained by running the provided code resulted in an `mse` of 0.31 for RBF kernels and bad results for linear ones. The results are not visualised here as they're not very interpretable and the influence of k on time - and memory complexity was discussed before. Instead of using the mean squared error the mean absolute error was determined through inversion of the standardisation procedure (in `initial.m`) to facilitate interpretability. The results showed the average error to be about 42k. Without the median house values of at least 500k this dropped by about 3-4k. For polynomial kernels the average mean absolute error turned out to be comparable. At all times, tuned parameters were used.

There's an other way to tackle large-scaled problems that is a little more straightforward, and that is to split the dependent variable into intervals and create a so-called committee network. After these are tuned and trained they can be combined in a linear or even a non-linear way. The latter is generally applied through the use of a multi-layer perceptron (MLP).

This approach was also toyed with using the linear approach outlined in *Least-Squares Support Vector Machines* (Suykens, 2003). Where the following covariance matrix is calculated :

$$C_{ij} = \frac{1}{N}\sum_{k=1}^{N}[f_i(x_k) - y_k] \cdot [f_j(x_k) - y_k]$$

Then the weights for each model were found through the following formula :

$$\beta = \frac{C^{-1} \cdot 1_\nu}{1_\nu^T \cdot C^{-1} \cdot 1_\nu} \quad \text{where} \quad 1_\nu = [1\ 1\ ...1]^T$$

The validity of the implementation was tested by re-doing the regression on a noisy `sinc` problem the results of which are shown in figure 35.

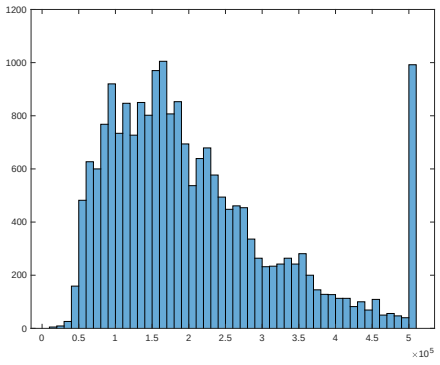


Figure 33: Histogram for the median house value.

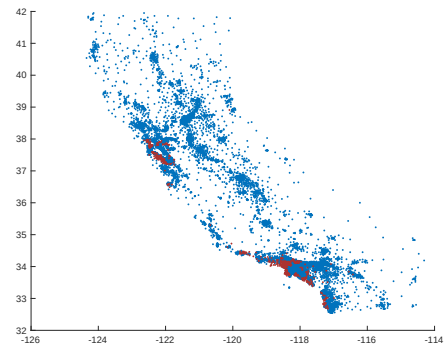


Figure 34: Regions with pricey houses.

The committee network performed in a comparable way to the FS LS-SVM approach when 4 models were used, having a mean absolute error of 38.460 on the test set. However, due to the long runtime (a couple of minutes) no check for statistical significance (or even a simple average of the results) was determined. In practice to compare models it's best to have many runs and do a t-test or something equivalent (possibly a non-parametric test depending on the assumptions that are deemed appropriate). More tests were done on smaller subsets of the training data which showed that the results depended greatly on the parameters. The kernel parameters are always important but in this case also the number of models (which was taken to be a power of 2) greatly affected performance. Toy experiments on the `sinc` function shown here on the right visualise this. In general splitting the dataset into 4 partitions appeared to perform best. At all times these partitions were generated randomly, not by taking fixed intervals (as per the recommendations).

What became apparent is that the standardisation procedure (in `initial.m`) in the case of the committee network sometimes reduced predictive power. This could be due to features with smaller ranges being less important or due to the normalisation (which brings the data to zero mean and unit variance, component-wise) not being sophisticated enough as it assumes independence of the features.

A limited experiment using an MLP with as inputs the outputs of the SVM models was tried as well (using the Deep Learning toolbox in `MATLAB` which is rather easy to use). This to obtain a non-linear combination. The model was trained with the Adam optimiser using a 75/25% train & validation split and the results turned out to be comparable to the linear approach as the mean absolute error turned out to be about 42k. This may not be representative as once again, only one run was done.

Finally, ARD was quickly tried but no features were removed. Of course none of the obtained models are anywhere near close to perfect so many improvements could be obtained through further experimentation or even by using entirely different frameworks altogether.

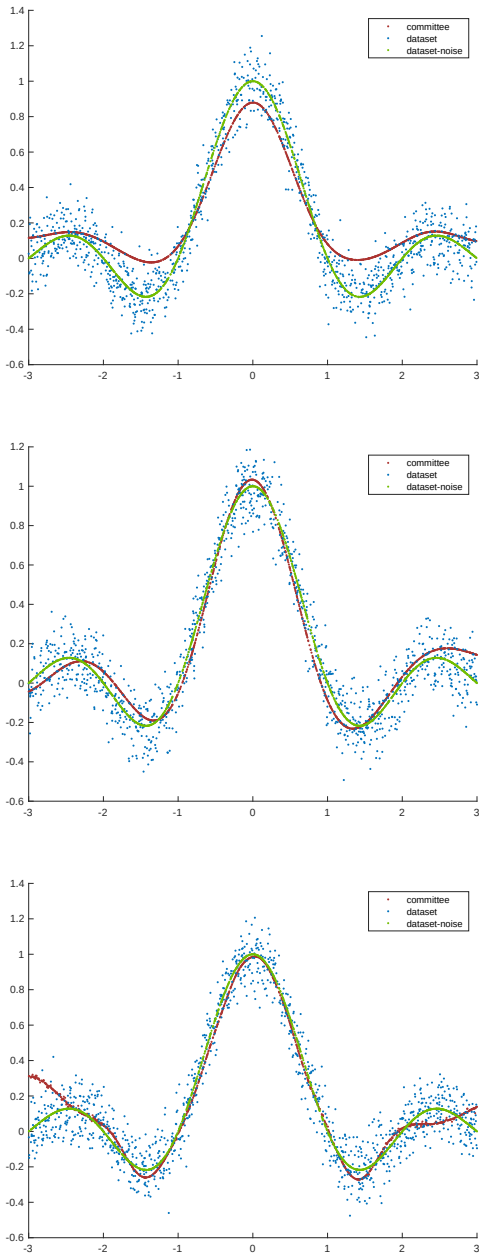
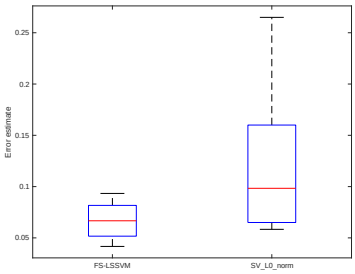
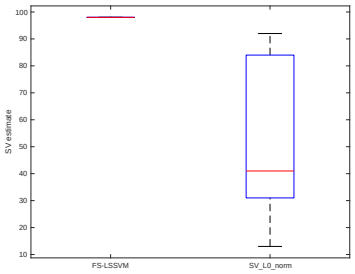


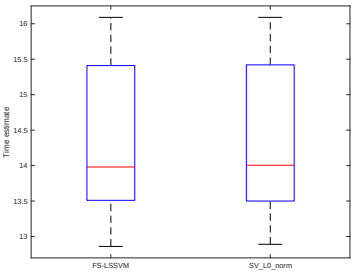
Figure 35: Regression on the `sinc` function using committee networks with a varying number of models (2, 4 and 8). Prediction in red, `sinc` function in green, noisy `sinc` in blue.



(a) Error estimation.

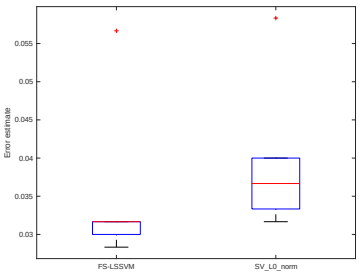


(b) Estimation of the number of support vectors.

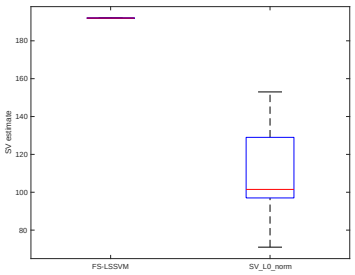


(c) Time estimation.

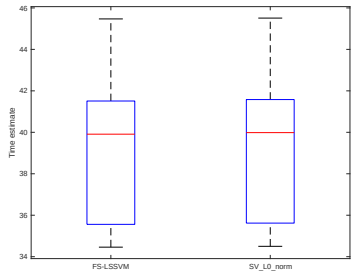
$k = 2$



(d) Error estimation.

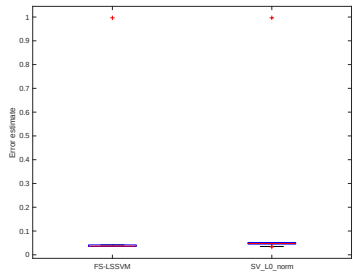


(e) Estimation of the number of support vectors.

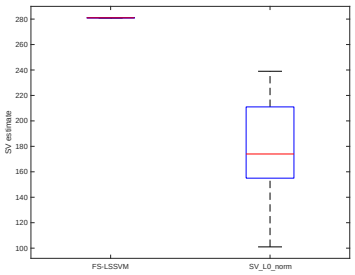


(f) Time estimation.

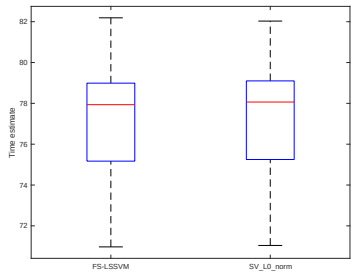
$k = 4$



(g) Error estimation.

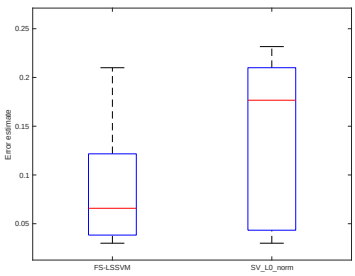


(h) Estimation of the number of support vectors.

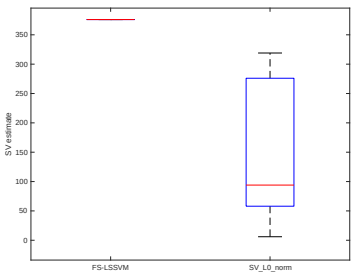


(i) Time estimation.

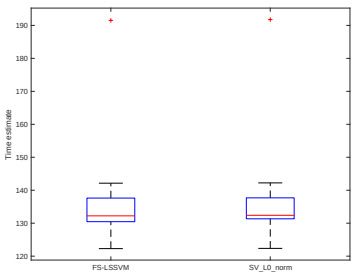
$k = 6$



(j) Error estimation.



(k) Estimation of the number of support vectors.



(l) Time estimation.

$k = 8$

Figure 36: Boxplots for estimates of the error, number of support vectors and time needed to generate FS-SVMs with or without ℓ_0 norm post-processing procedure. Estimates are based on results on a test set (20% of 3000 samples, obtained through stratified sampling). Estimates based on 10-fold cross-validation are about equivalent. The dataset is restricted to 3000 samples to avoid long runtimes. These belong to the first 5 classes. At all times an RBF kernel is used which has automatically tuned parameters. The increase in time requirements for increasing value of k corresponds to the theoretical complexity $\mathcal{O}(k^2N^2)$.