Function Estimation and Time Series Prediction

James O'Reilly
james.oreilly@student.kuleuven.be

1 Function Estimation

Construct a dataset where a linear kernel is better than any other kernel (around 20 data points). What is the influence of e and of Bound. Where does the sparsity property come in?
The ϵ-insensitive loss function used to introduce sparsity of the support vectors, proposed by Vapnik, is given by:

$$|y - f(x)|_\epsilon = \begin{cases} 0 & , \quad \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & , \quad \text{otherwise} \end{cases} \tag{1}$$

Minimising a convex ϵ-insensitive loss function given above is equivalent to defining an ϵ-tube that contains the maximum possible number of training instances. The difference between the predicted values and the true values are minimised, and a penalty is applied to points that fall outside of the tube. The value of ϵ determines the width of the tube. A large ϵ value gives a higher tolerance for errors, a wider ϵ-tube, and fewer support vectors. The ϵ parameter controls sparsity by determining the number of support vectors. Conversely, a small ϵ value gives a lower tolerance for errors, a narrower ϵ-tube, and decreased sparsity, as training instances are outside the bounds of the ϵ-tube. Table 2 shows the results of a linear regression on the constructed dataset, varying both the bound parameter and ϵ. It is clear that increasing ϵ increases the width of the ϵ-tube. Table 1 demonstrates that sparsity increases with ϵ.

The *bound* parameter seems to have the same role as the gamma parameter in RBF kernels, where it is seen as the inverse of the radius of influence of instances selected by the model as support vectors. Similarly, the bound parameter here controls the range of influence of the instances. A lower bound parameter increases the range of influence for each instance, resulting in a flatter function with decreased flexibility (as seen in Table 2). Increasing the bound parameter decreases this range of influence, therefore increasing the flexibility of the function, and allowing for functions which are less flat. Naturally, reducing the flexibility of the function decreases the sparsity, as a greater number of instances fall outside the ϵ-tube, in this way, the bound parameter also controls the sparsity.

	$\epsilon = 0.1$				$\epsilon = 0.25$				$\epsilon = 0.5$			
bound	0.01	0.1	1	10	0.01	0.1	1	10	0.01	0.1	1	10
#SV	18	12	7	6	14	8	2	2	8	4	2	2

Table 1: Number of support vectors for each combination of ϵ and bound parameters.

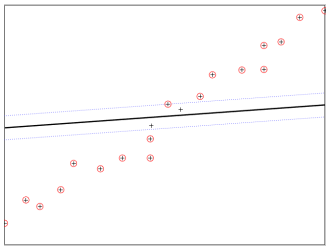
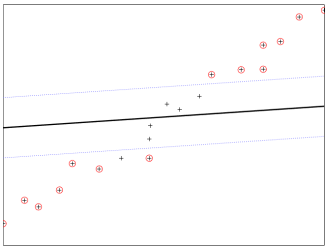
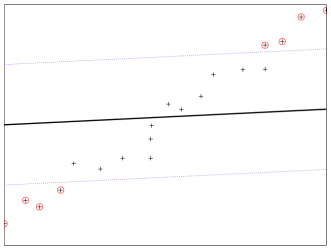
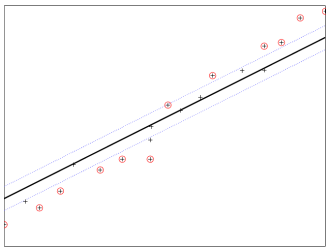
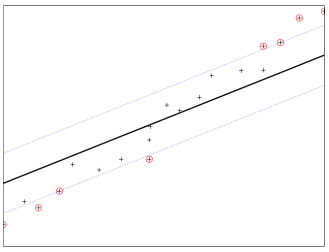
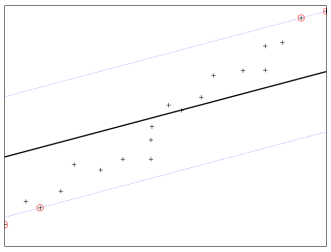
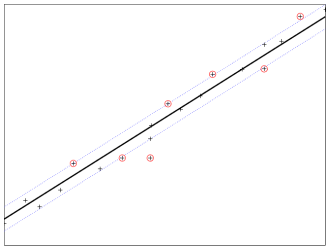
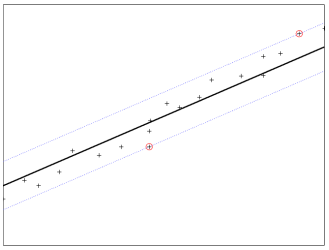
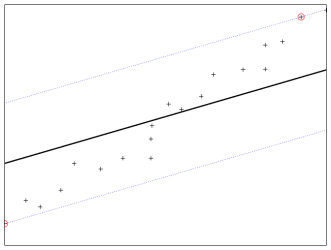
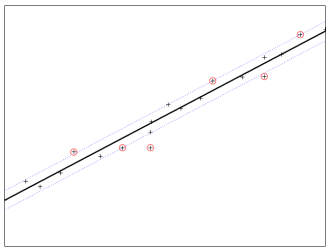
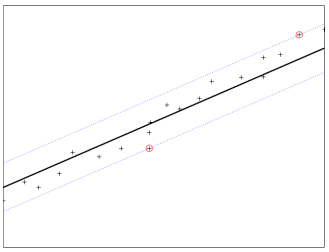
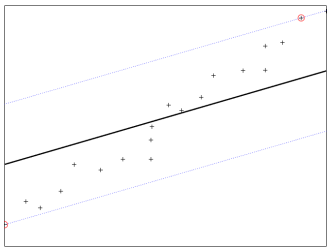
	$\epsilon = 0.1$	$\epsilon = 0.25$	$\epsilon = 0.5$
Bound = 0.01			
Bound = 0.1			
Bound = 1			
Bound = 10			

Table 2

Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.

Figure 1 shows the results of fitting linear, polynomial, linear-bspline, and RBF kernels to the data. For each kernel I varied the parameters to try to get the best estimate from each kernel. Based on the results, I would suggest that either Gaussian RBF or polynomial are best suited to this dataset. Both the linear and linear-bspline kernels underfit the data. The Gaussian and polynomial kernels are better able to capture the non-linearity present in the data. The Gaussian RBF kernel has better sparsity properties than the polynomial kernel, with significantly fewer support vectors, and for that reason it would be my top choice. I would rank the models as follows: Gaussian RBF > Polynomial > Linear-BSpline > Linear.

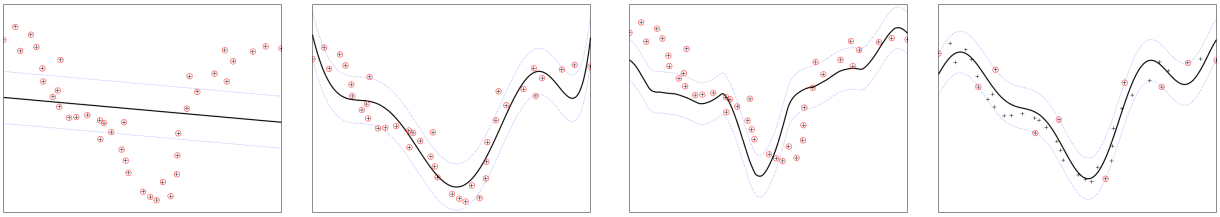


Figure 1: *(left to right)* Linear, polynomial, linear-bspline, and Gaussian RBF function estimation results on the more challenging dataset.

In what respect is SVM regression different from a classical least squares fit?
The primary difference between the two methods is clear in the cost function which they aim to minimise. Classical least squares regression uses the L2 squared loss function, which finds a line with minimum distances from the observations:

$$cost = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{2}$$

Support vector regression with a linear kernel uses an ϵ-insensitive L1 loss function (see Equation 1), in which only the observation outside of the ϵ-tube contribute to the loss function. This gives support vector regression the ability to regularise and avoid overfitting, which is an advantage over classical least squares fit.

2 A simple example: the sinc function

2.1 Regression of the sinc function

Function estimation was performed on noised data from the sinc function. The estimation was performed using a RBF kernel and for a number of different sigma and gamma parameters. The results of the estimation are given in Table 3.

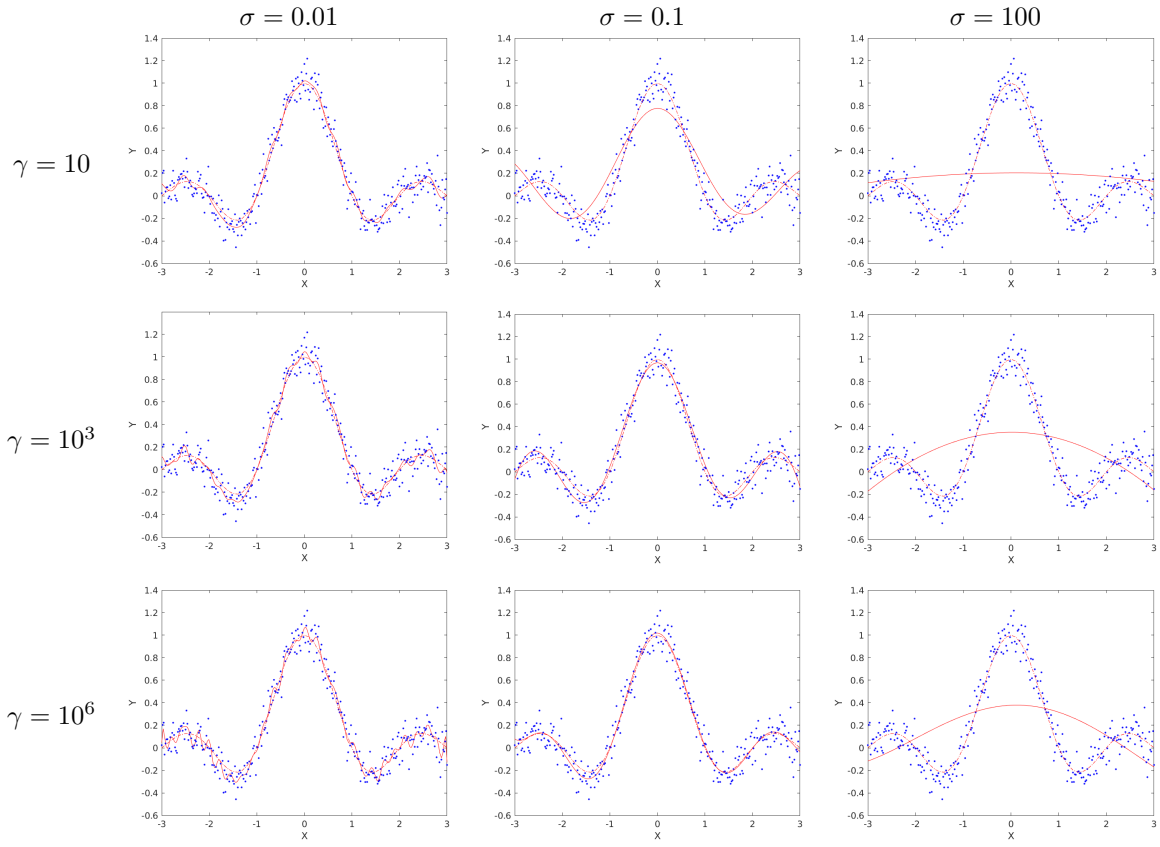


Table 3

From the resulting estimations, we can see the effects of gamma and sigma. Gamma is the regularisation parameter, determining the trade-off between the training error minimisation and the smoothness of the estimated function. This can be clearly seen in the leftmost column of the image grid. Increasing gamma prioritises training error minimisation, sacrificing function smoothness and possibly leading to some overfitting. Sigma is the kernel function parameter, and determines the radius of influence of each instance. Increasing sigma leads to a smoother function, with the function tending toward linear as $\sigma \longrightarrow \infty$. Both these parameters control the 'smoothness' of the estimated function, but at two different levels. The optimal combination of these parameters

finds the correct trade-off between training error minimisation and function smoothness (at both levels). The MSE for the estimated functions was calculated on the test set. The results are given in Table 4.

MSE	$\sigma = 0.01$	$\sigma = 0.1$	$\sigma = 100$
$\gamma = 10$	3.01	10.11	39.7
$\gamma = 10^3$	3.13	3.46	33.71
$\gamma = 10^6$	3.33	2.89	31.96

Table 4

Do you think there is one optimal pair of hyperparameters?
No. I think there is likely a number of optimal pairs which give very similar performances on the test set. To verify this, I performed a comprehensive grid search of the hyperparameter space, varying both gamma and sigma for the RBF kernel. The result is given in Figure 2, and shows that there are many combinations of hyperparameters which give very low MSE. I tried to search the parameter space for values of gamma greater than 10^{-12}, in the hopes of finding an increase in MSE once more, but the `trainlssvm` function started screaming at me so I stopped. The figure doesn't disprove the existence of a global minimum, although my opinion is that there is unlikely a single global optimum, at least to two significant digits.

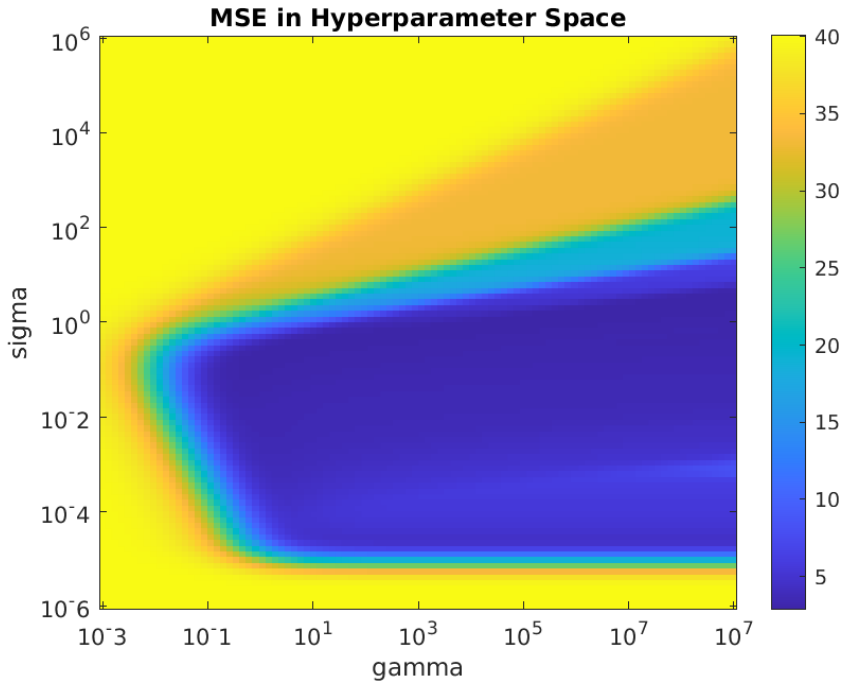


Figure 2: Colour plot showing MSE for a grid search of hyperparameter space.

The parameters were then tuned using the `tunelssvm` procedure, with both the NM-simplex and grid search algorithms. Both of the optimisation algorithms gave an MSE of 0.01 consistently over a number of runs. The gamma and sigma parameters varied between each run, validating that there is no one global optimum, but many local optima.

2.2 Application of the Bayesian framework

Discuss in a schematic way how parameter tuning works using the Bayesian framework. Illustrate this scheme by interpreting the function calls denoted above.
A schema for the Bayesian framework is given in Figure 3. Parameter tuning within the Bayesian framework is done by estimating the posterior probabilities on three different inference levels, as illustrated in the schema. The likelihood at a given level is equivalent to the evidence at the previous level.

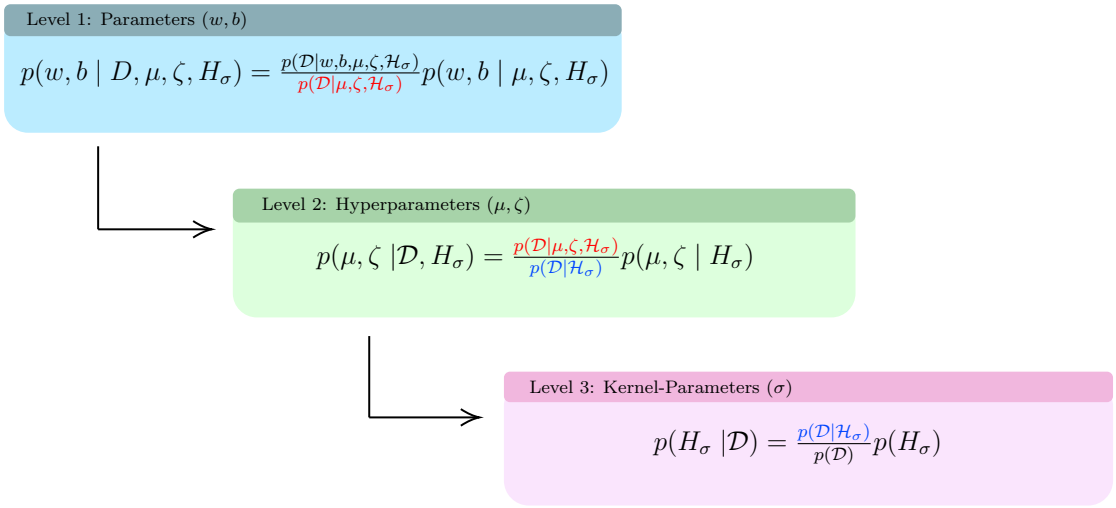


Figure 3: Schema of the multi-level Bayesian framework.

- **Level 1:** Inference of model parameters w and b occurs at the first level. This is done by finding w_{MP} and b_{MP} which maximise the logarithm of the posterior at level 1 (see schema).
- **Level 2:** Inference of hyperparameters μ and ζ which then form the hyperparameter $\gamma = \frac{\zeta}{\mu}$. The likelihood at this level $p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\alpha)$ is equal to the evidence from level 1. Optimal μ and ζ are calculated by maximising the log posterior. Solving this optimisation problem requires computing the effective number of parameters d_{eff} from the eigenvalue decomposition of the Gram matrix G.
- **Level 3:** Inference of kernel parameters (α in the case of RBF kernel) and model comparison. The likelihood at this level $p(\mathcal{D}|\mathcal{H}_\alpha)$ is equal to the evidence from level 2.

The posterior probabilities of model parameters at the different inference levels are estimated using the `bay_lssvm` function. The cost at each level is computed by taking the negative logarithm of the posterior.

```
crit_L1 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 1);
crit_L2 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 2);
crit_L2 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

Listing 1

The posterior probabilities of the model parameters and hyperparameters at different levels of inference are then optimised using `bay_optimize`.

```
[~, alpha, b] = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 1);
[~, gam]      = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 2);
[~, sig2]     = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

Listing 2

3 Automatic Relevance Determination

Input selection can be done by Automatic Relevance Determination (ARD). This allows one to determine the most relevant inputs of an LS-LSVM within the Bayesian evidence framework. A different weighting parameter is assigned to each dimension in the kernel and this is then optimised using the third level of inference. Backward selection of inputs is performed by iteratively removing the input with the largest optimal sigma.

Automatic relevance determination is performed using the `bay_lssvmARD` function. This determines the relevant subset of variables, returns a ranked list of variable importance, and the costs associated with third level of inference in every selection step, which is an approximation of generalisation performance. This process can be visualised for a simple toy example with few dimensions (see Figure 4). Starting with a three input variables x_1, x_2, x_3, ARD selected x_1 as the most relevant subset. The ranked list indicated the ordering of importance: $x_1 > x_2 > x_3$, with the costs at each selection step given by -19, -63, and -75.

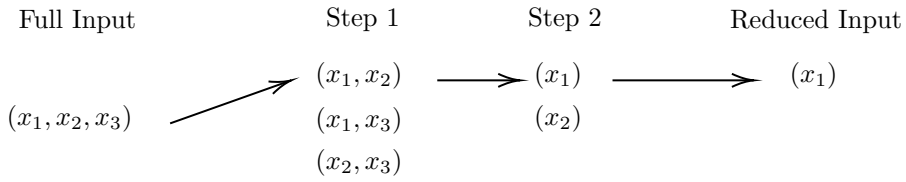


Figure 4: Illustration of ARD on the toy example.

How can you do input selection in a similar way using the `crossvalidate` *function instead of the Bayesian framework?*
Instead of determining the most relevant input variables by minimising the cost function in the Bayesian framework. One can perform forward or backward selection and evaluate the performance of the model at each step using cross validation. This will again give a ranking of variable importance, but it should be noted that these methods have their drawbacks and are not exhaustive (do not check every possible subset).

4 Robust Regression

Compare the non-robust version with the robust version. Do you spot any differences?
Figure 5 shows the function estimation of the non-robust and robust LS-SVM regression models on a dataset with outliers. The non-robust version is affected by the outliers while the robust version is not.

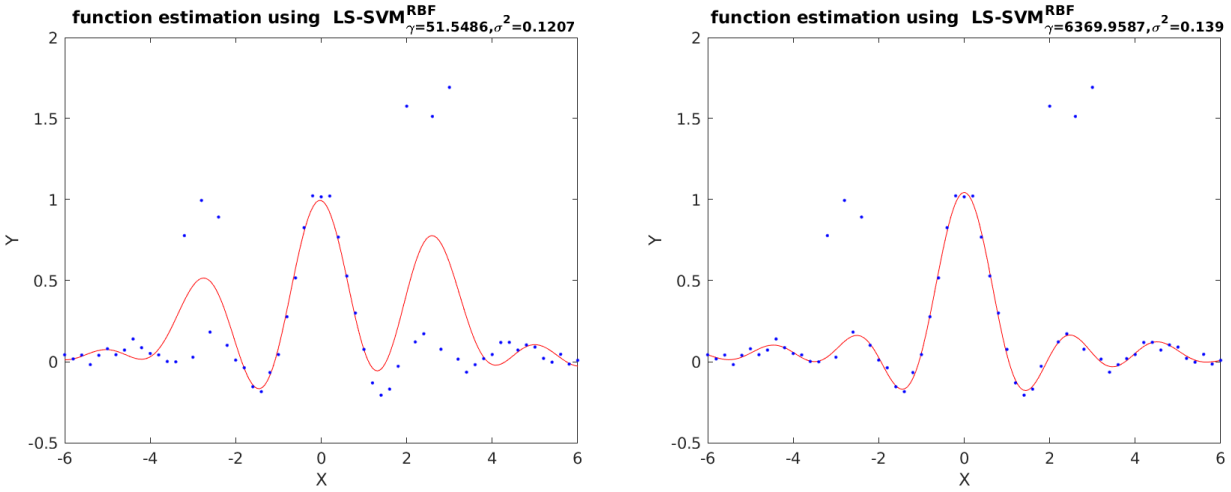


Figure 5: *(left)* Non-robust LS-SVM regression. The effect of the outliers is clearly seen skewing the function. *(right)* Robust regression, mitigating the effect of the outliers.

Why in this case is the mean absolute error preferred over the classical mean squared error?
Mean absolute error is less sensitive to outliers than mean squared error, which exaggerates the impact of outliers. As there are outliers present, MSE will give a poor performance for the robust model, which is an inaccurate reflection of its performance. MAE is more appropriate for comparing models in this case.

Try alternatives to the weighting function `wFun`*. Report on differences.*
The choice of weighting function in the cost function is very important for robust kernel regression. Brabanter et al conducted a comparison of different weighting schemes in the context of robust kernel regression [1]. They compared four different types of weight functions (Huber, Hampel, Logistic, and Myriad) and their use in iterative re-weighted LS-SVM. They discovered a trade-off between speed of convergence and the degree of robustness between weighting schemes. They concluded that the Myriad weight function is highly robust against (extreme) outliers but has a slow speed of convergence. A good compromise between speed of convergence and robustness can be achieved by using Logistic weights. The mean absolute error of each weighting function on the modified sinc dataset is given in Table 5.

	Huber	Myriad	Hampel	Logistic
MAE	0.154	0.141	0.146	0.143

Table 5: Mean absolute error for the different weighting functions.

5 Homework Problems

5.1 Logmap dataset

As indicated numerous times before, the parameters gam and sig2 can be optimised using cross-validation. In the same way, one can optimise order as a parameter. Define a strategy to tune these 3 parameters.
The sigma and gamma parameters were optimised using the `tunelssvm` function with 10-fold cross validation, simplex and an MAE cost function. The order was tuned simply by looping through order values from 1-100 and calculating the MAE on the prediction set using the tuned parameters. The results for the untuned and tuned parameters are given in Figure 6, along with a plot of MAE against order in Figure 7.

Do time series prediction using the optimised parameter settings. Visualise your results. Discuss.
The results show that the untuned model with an order of 10 does not have good prediction on the data, and only predicts well for a small portion. The tuned model has somewhat better performance, and accurately predicts in areas of low volatility. However, it fails to correctly predict the volatile movements. This is likely because these volatile movements are the result of noise, and the tuned model has learned the underlying non-noisy function. It is also interesting to note that prediction accuracy does not increase monotonically with order (see Figure 7). There is a 'sweet-spot' for order, with low prediction accuracies. Very low and very high orders give more volatile and worse prediction accuracies.

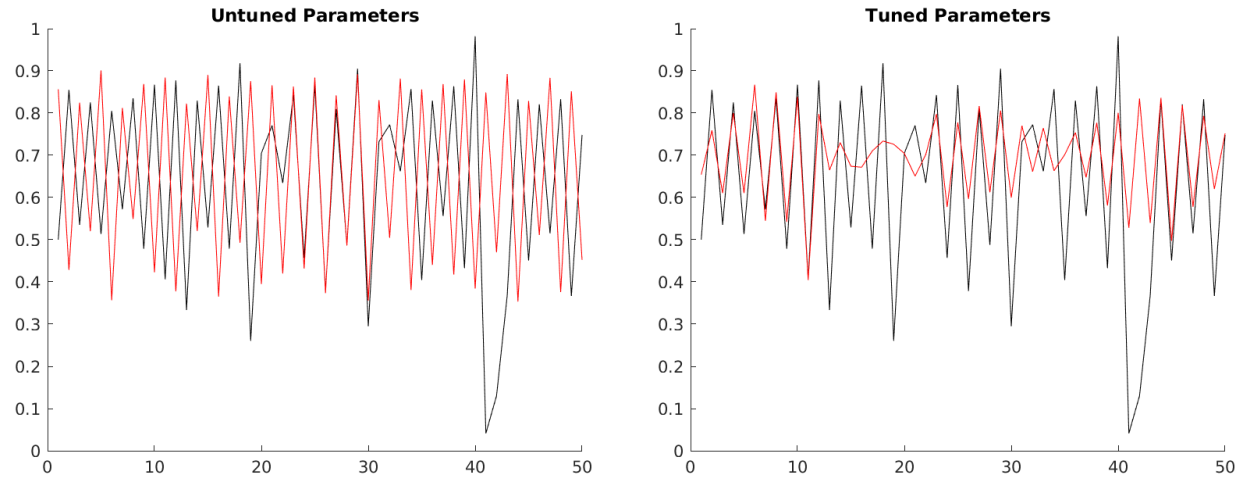


Figure 6: *(left)* Untuned logmap prediction results. Actual data is in black. Predicted data is in red. *(right)* Prediction results on logmap data using tuned model parameters and order = 23.

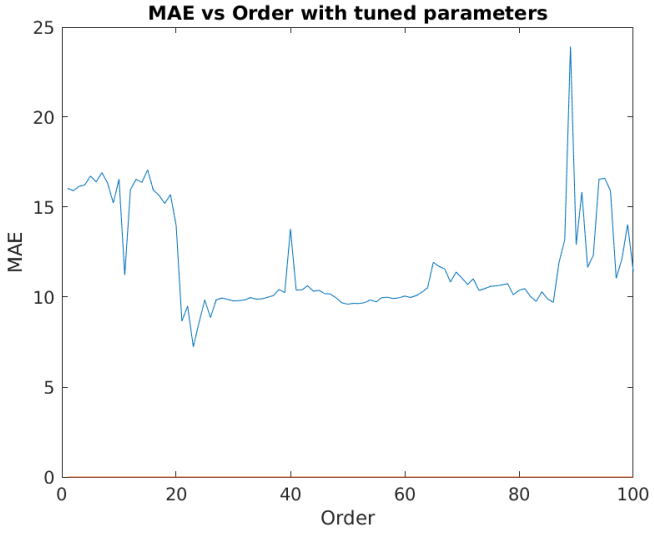


Figure 7: Mean absolute error plotted against order, using tuned hyperparameters for each order.

5.2 Santa Fe dataset

In this section I apply time series prediction on the Santa Fe laser dataset. The concatenated training and test data is given in Figure 8.

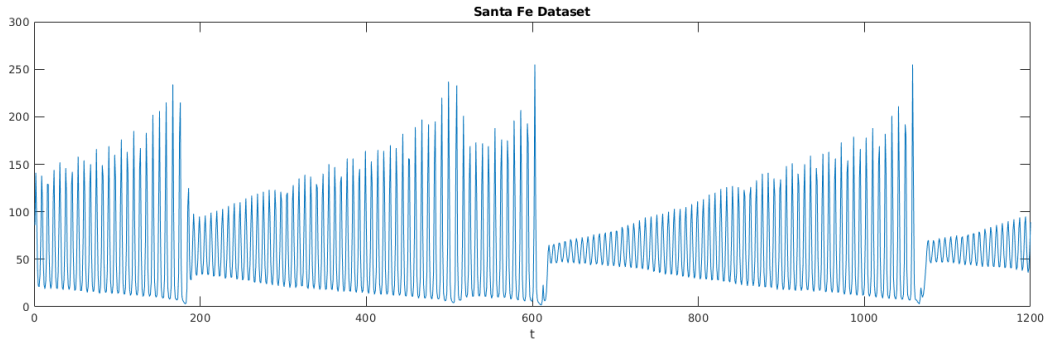


Figure 8: Visualising the Santa Fe dataset.

Does order = 50 for the utilised auto-regressive model seem like a good choice?
Looking at the plot given above, I would intuit that an order of 50 would be a good choice. It is sufficiently large to span the spike-crash-recover behaviour which is present. The model should therefore be able to learn this pattern and determine at what point it is going to occur.

Would it be sensible to use the performance of this recurrent prediction on the validation set to optimise hyperparameters and the model order?
There is no distinct validation set present. If one were to do a training-validation-test split of the data, I would think it sensible to use the validation set to optimise model hyperparameters and model order.

Tune the parameters (order, gam and sig2) and do time series prediction. Visualise your results. Discuss.
The parameters were tuned using the same strategy employed for the logmap dataset, tuning the hyperparameters for a given range of orders (0-100), using cross-validation, simplex and a mean absolute error cost function. The mean absolute error and of prediction was then calculated on the test set, and the order with the smallest MAE was determined to be the best. At orders less than 20 the models performed very poorly, with very high MAEs. It is interesting again that prediction accuracy doesn't increase with order. Being honest, I cannot figure out why exactly this is the case. The prediction is clearly not perfect, but it captures the qualitative behaviour of the data (spike-drop-recover) nonetheless.

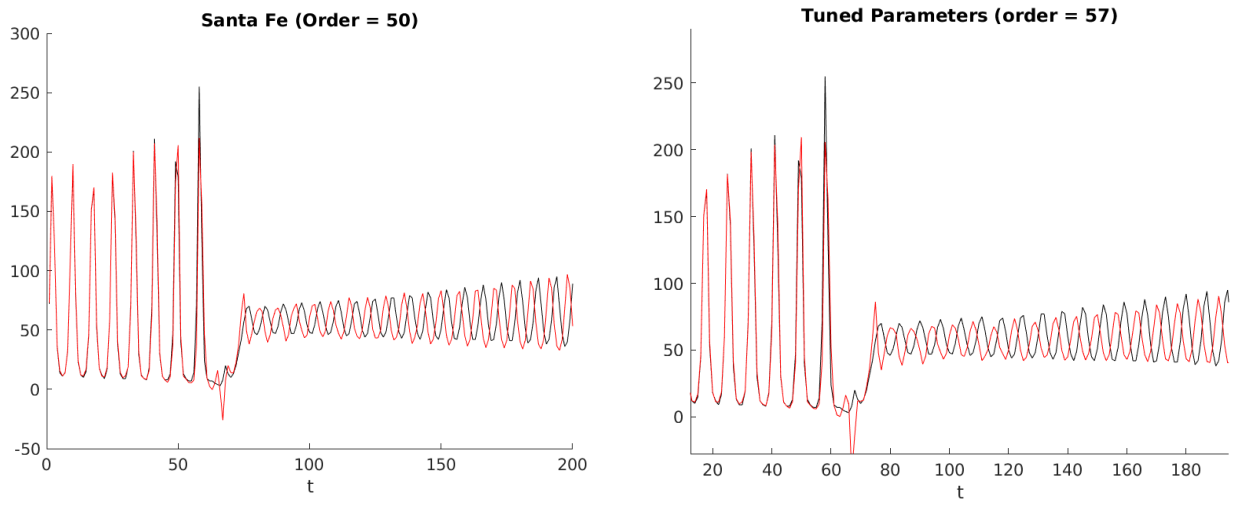


Figure 9: *(left)* Tuned Santa Fe prediction results with order = 50. Actual data is in black. Predicted data is in red. *(right)* Prediction results on logmap data using tuned model parameters and order (determined by MAE). The order of the best performing model was 57 (see Figure 10).

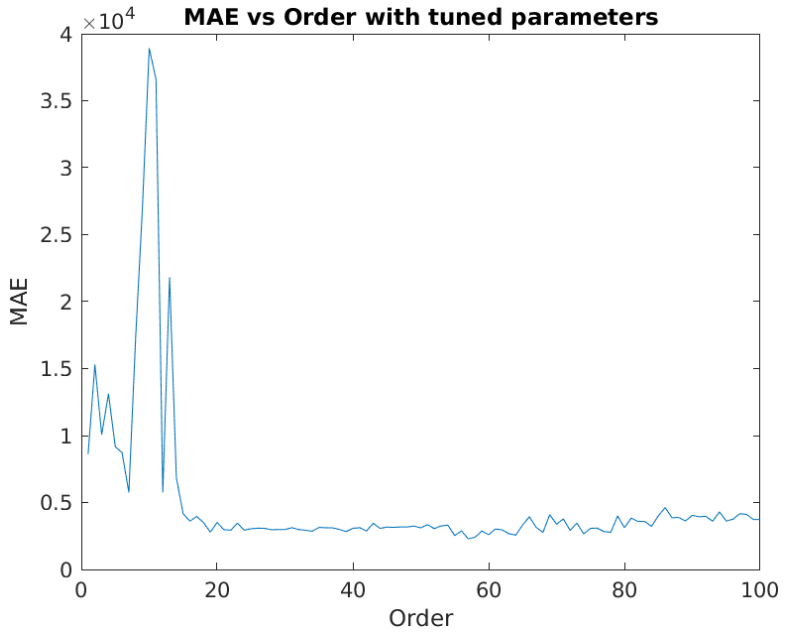


Figure 10: MAE of tuned Santa Fe prediction results plotted against order. Note the drop in MAE at around order = 20.

References

[1] Kris De Brabanter, Kristiaan Pelckmans, Jos De Brabanter, Michiel Debruyne, Johan AK Suykens, Mia Hubert, and Bart De Moor. Robustness of kernel based regression: a comparison of iterative weighting schemes. In *International conference on artificial neural networks*, pages 100–110. Springer, 2009.