Exercise Session2: Function Estimation and Time Series Prediction

Support Vector Machines

Zirui Yan r0916941

1.1 Support vector machine for function estimation

(a) In primal space, we have

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

subject to

$$\begin{cases} y_k - w^T x_k - b \leq \epsilon + \xi_k, k = 1, ..., N \\ w^T x_k + b - y_k \leq \epsilon + {\xi_k}^*, k = 1, ..., N \\ \xi_k, {\xi_k}^* \leq 0, k = 1, ...N \end{cases}$$

In dual space, we have

$$\max_{\alpha,\alpha^*} Q(\alpha,\alpha^*) = -\frac{1}{2}\sum_{k,l=1}^{N}(a_k - {a_k}^*)(a_l - {a_l}^*)K(x_k, x_l) - \epsilon\sum_{k,l=1}^{N}(a_k + {a_k}^*) + \sum_{k,l=1}^{N} y_k(a_k + {a_k}^*) \tag{2}$$

subject to

$$\begin{cases} \sum_{k=1}^{N}(a_k + {a_k}^*) = 0 \\ a_k, a_k^* \in [0, c] \end{cases}$$

ϵ here is for Vapnik's ϵ-insensitive loss function, which creates a tube on the regression result. All the data points inside this tube do not give punishment for its deviating from the regression line. ξ_k and ${\xi_k}^*$ measure how far the data points deviating from the tube. And bound c decides the value of punishment.

For sparsity, we want $\alpha, \alpha^* = 0$, which means we need

$$\begin{cases} y_k - w^T x_k - b < \epsilon + \xi_k \\ w^T x_k + b - y_k < \epsilon + {\xi_k}^* \end{cases}$$

So larger ϵ, ξ_k and ${\xi_k}^*$ are needed for sparsity. ϵ cannot be too large, otherwise the loss the functionality of loss function. If we want ξ_k and ${\xi_k}^*$ be larger, we need c becomes smaller.

Let's look at the dataset shown in Figure 1. When ϵ is big, the tube is wide. And the tube try to cover most of data points and make the punishment caused by the uncovered data points as small as possible. Larger ϵ is, the fewer data points become support vector. When ϵ small enough, every data point becomes support vector. Specially, when $\epsilon = 0$, bound=inf, the primal problem becomes minimizing $c\sum_{k=1}^{N}(\xi_k + \xi^*)$, a linear regression model with MAE as loss function.

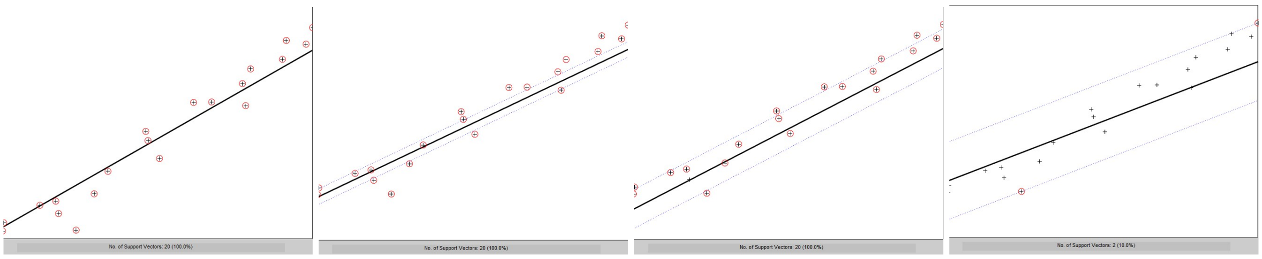


Figure 1: Influence of ϵ with bound=inf: (left to right) $\epsilon = 0$, $\epsilon = 0.1$, $\epsilon = 0.25$, $\epsilon = 5$

c decides the value of punishment. As discussed above, larger c raise the importance of reducing the value of ξ_k and ξ^*. When c is small, the optimization problem solver focuses on reducing the value of $\frac{1}{2}w^T w$, so the slope decreases as c is smaller, see Figure 2.

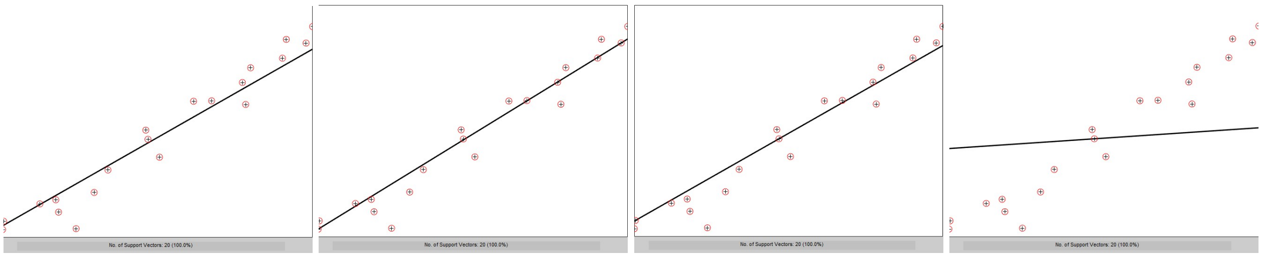
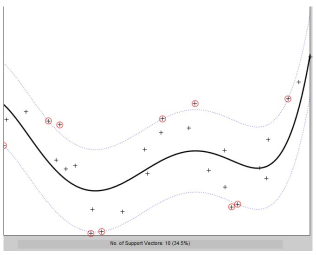


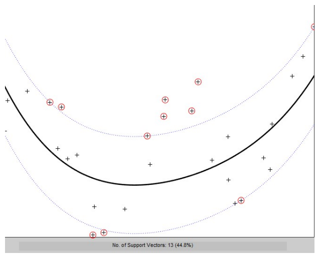
Figure 2: Influence of bound with $\epsilon = 0$: (left to right) bound=inf, bound=1, bound=0.1, bound=0.01

(b) In Figure 3, the target curve is much more complex than a line and RBF kernel has to find a good sig2 to make resulting curve smooth. So the polynomial kernel has the best performance since I can easily find degree=5 suitable for this problem.

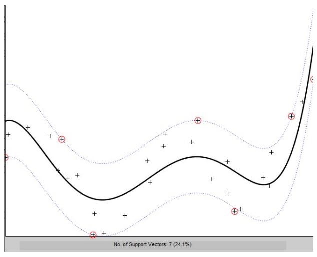
And I also tried polynomial kernel with different c. If c=0.1, the curve has much fewer support vectors and is smooth, but does not bent as target curve. If c=1000, the curve is overfitting (see left end of the third plot). So it is important to choose a good c.



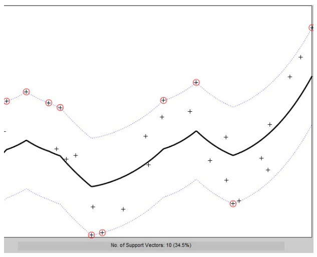
(a) polynomial kernel with degree=5 c=10 $\epsilon = 0.25$



(b) polynomial kernel with degree=5 c=0.1 $\epsilon = 0.25$



(c) polynomial kernel with degree=5 c=1000 $\epsilon = 0.25$



(d) RBF kernel with sig2=0.5 c=10 $\epsilon = 0.25$

Figure 3: different kernel implemented on the same dataset

(c) A classical least squares fit can be considered as a special case of SVM regression. c=inf and $\epsilon = 0$, then SVM regression becomes a classical least squares fit. $c \neq inf$ gives importance to $\frac{1}{2}w^T w$ which decides estimated function itself. And $\epsilon \neq 0$ adds tube to the regression result which allows little deviation without punichment.

1.2.1 Regression of the sinc function

(a) LS-SVM regression model in primal space:

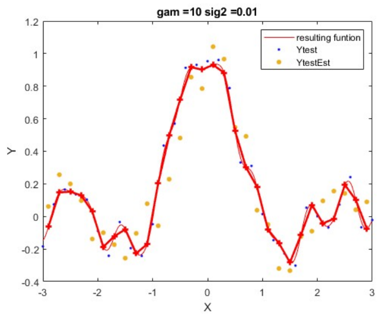
$$y(x) = w^T\phi(x) + b \tag{3}$$

$$\min_{w,b,e} J(w,b,e) = \frac{1}{2}w^T w + \gamma \sum_{k=1}^{N} {e_k}^2 \tag{4}$$

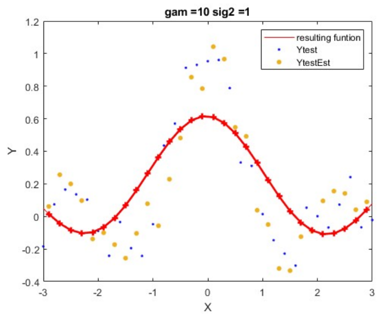
subject to

$$y_k = w^T\phi(x_k) + b + e_k, k = 1, ..., N$$

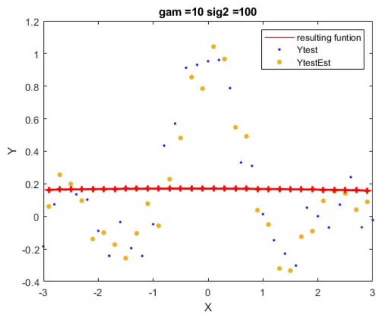
sig2 is the parameter of RBF kernel. Higher sig2 is, the kernel performs more like a linear kernel. sig2=1 has the best performance on our dataset. When sig2=0.01, RBF kernel tries to capture more non-linear features, so overfitting happens and the resulting curve no longer smooth. When sig2=100, it fails to capture nonlinear feature and performs like a low-degree polynomial kernel.



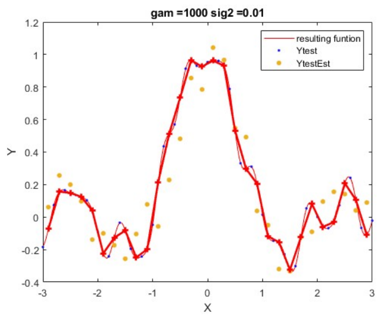
(a) gam=10, sig2=0.01, mse=0.0200576



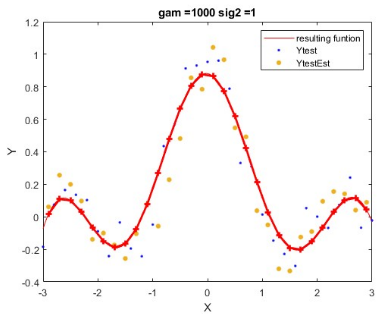
(b) gam=10, sig2=1, mse=0.0596549



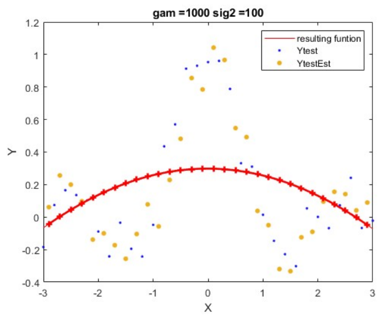
(c) gam=10, sig2=100, mse=0.131231



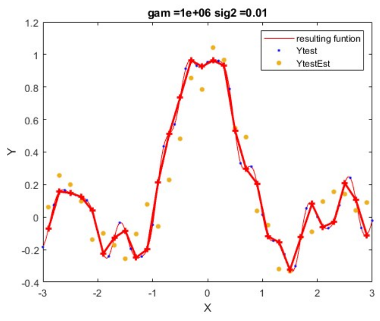
(d) gam=1000, sig2=0.01, mse=0.0233507



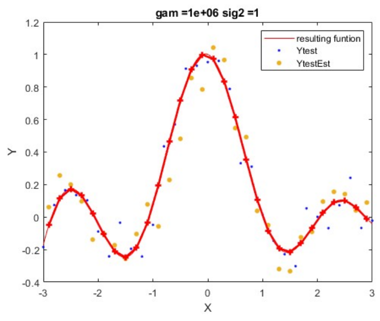
(e) gam=1000, sig2=1, mse=0.0169908



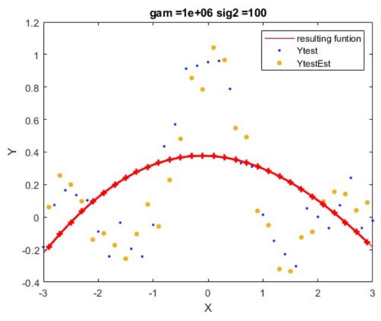
(f) gam=1000, sig2=100, mse=0.11462



(g) gam=10^6, sig2=0.01, mse=0.0234018



(h) gam=10^6, sig2=1, mse=0.0136902



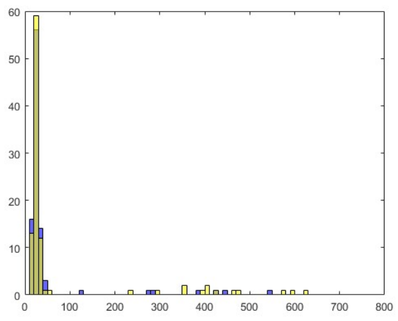
(i) gam=10^6, sig2=100, mse=0.115484

Figure 4: Influence of ϵ with bound=inf: (left to right) $\epsilon = 0$, $\epsilon = 0.1$, $\epsilon = 0.25$, $\epsilon = 5$

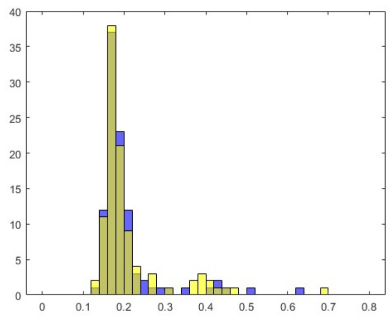
gam decides the punishment coming from $\sum_{k=1}^{N} {e_k}^2$. When gam is small, underfitting often happens. When gam is big, it is easy to happen overfitting.

(b) Plot (h) of Figure 4 (gam=10^6, sig2=1) has the best performance since its lowest mse. I think gam=10^6, sig2=1 is close to optimal hyperparameter. I tried **tunelssvm** multiple runs, but there's no pair of hyperparameter whose mse is much lower than the pair of (h).

(c) I use simplex and gridsearch respectively 100 times. Figure 5 shows all the results, except some gam's computed by gridsearch go far beyond 1000, even reach 48000. And both simplex and gridsearch give are likely to give gam$\approx$25 and sig2$\approx$0.17, but the variances of the distributions of gam and sig2 computed by simplex are smaller.



(a) histogram for gam: (yellow) simplex; (blue) gridsearch



(b) histogram for sig2: (yellow) simplex; (blue) gridsearch

Figure 5: histograms for gam and sig2

1.2.2 Application of the Bayesian framework

(a) Using Bayes' theorem, we have the three-level equations:

Level 1 (inference of parameters w, b)

$$p(w, b|D, \mu, \zeta, H_\sigma) = \frac{p(D|w, b, \mu, \zeta, H_\sigma)}{p(D|\mu, \zeta, H_\sigma}p(w, b|\mu, \zeta, H_\sigma) \tag{5}$$

Level 2(inference of hyperparameters μ, ζ)

$$p(\mu, \zeta|D, \mu, H_\sigma) = \frac{p(D|\mu, \zeta, H_\sigma)}{p(D|H_\sigma)}p(\mu, \zeta|H_\sigma) \tag{6}$$

Level 3([inference of kernel parameter σ and model comparison)

$$p(H_\sigma|D) = \frac{p(D|H_\sigma)}{p(D)}p(H_\sigma) \tag{7}$$

Each equation is of the form Posterior = Likelihood / Evidence × Prior. And the likelihood appearing in the equation of each level is the evidence term in the previous level, which allows us to solve these problems from level 3 to level 1. Then, based on maximum posterior, one can solve the parameters tuning task as an optimization problem over all these parameters. When w, b statistically independent and $p\left(\mathcal{H}_\sigma\right)$ prior uniform, we have

$$p\left(w, b \mid \mathcal{D}, \mu, \zeta, \mathcal{H}_\sigma\right) \propto \exp\left(-\mu\frac{1}{2}w^T w - \zeta\frac{1}{2}\sum_{k=1}^{N} e_k^2\right) \tag{8}$$

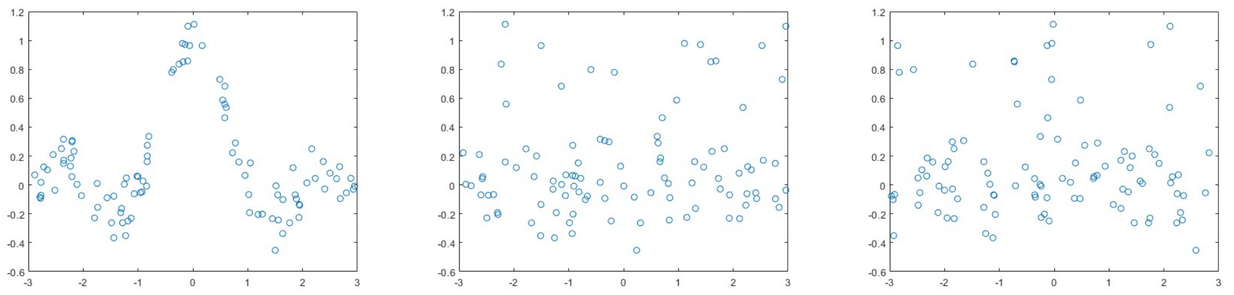
$$p\left(\mu, \zeta \mid \mathcal{D}, \mathcal{H}_\sigma\right) \propto \exp\left(-J_{\mathrm{P}}\left(w_{\mathrm{MP}}, b_{\mathrm{MP}}\right)\right) \times \sqrt{\mu^{n_h}\zeta^N}\sqrt{\det H^{-1}} \tag{9}$$

$$p\left(\mathcal{H}_\sigma \mid \mathcal{D}\right) \propto p\left(\mathcal{D} \mid \mathcal{H}_\sigma\right) \simeq \exp\left(-h\left(\theta_{\mathrm{MP}}\right)\right) \sqrt{(2\pi)^{n_p}} \sqrt{\det H^{-1}} \tag{10}$$

So we can use the negative logarithm of the posteriors as the cost which can be computed by **bay_lssvm**. And **bay_optimize** function is used to optimize the posterior probabilities of either the primal parameters (first level), the hyperparameters (second level) or the kernel parameters (third level).

1.3 Automatic Relevance Determination

(a) Automatic relevance determination (ARD) is a method used for measuring the relevance of input space and output. Since Y = sinc (X (: ,1)) + 0.1.* randn (100 ,1), Y only relates to the first column of X. The output of **bay_lssvmARD** contains the dimension which relates to Y and the ranking of all inputs. In our case, **bay_lssvmARD** only selects the first dimension and the visualization shows in Figure 6.



(a) X1-Y (b) X2-Y (c) X3-Y

Figure 6

(b) RBF kernel can be used to determine relevance of inputs:

$$K(x, z) = \exp\left(-(x-z)^T S^{-1}(x-z)\right) \tag{11}$$

where $S = \operatorname{diag}\left(\left[s_1^2; \ldots; s_n^2\right]\right)$. Small values of $1/s_i^2$ indicate that the corresponding input is not very relevant in the chosen model.

The process of automatic relevance determination by using crossvalidate function:

Step 1: Normalize the inputs to zero mean and unit variance

Step 2: Start with equal elements $s_1 = s_2 = \ldots = s_n$ or $S = s^2 I$. But on longer use Bayesian framework. Use crossvalidation to compute the cost. Optimize the cost, then get the new values of s_1, $s_2 \ldots s_n$.

Step 3: Remove the least relevant input (with the least s_i). Reduce the dimensionality of the problem and set $n := n-1$.

Step 4: Back to step 1 and remove inputs as long as one improves the cost.

1.4 Robust regression

The optimal problem for LS-SVM regression model:

$$\min_{w,b,e} J_{\mathrm{P}}(w,e) = \frac{1}{2}w^T w + \gamma \sum_{k=1}^{N} L\left(e_k\right) \tag{12}$$

subject to

$$y_k = w^T \varphi\left(x_k\right) + b + e_k, \quad k = 1, \ldots, N$$

where L(·) is a general and differentiable cost function.

We use L_2 norm as cost function in this section, which makes the model non-robust. If the error e_k of an outlier point is large then the least squares term $\sum_{k=1}^{N} e_k^2$ is going to make this error even larger. This leads to the regression fit being attracted to this outlier point.

The idea of robust version is to use weighted LS-SVM:

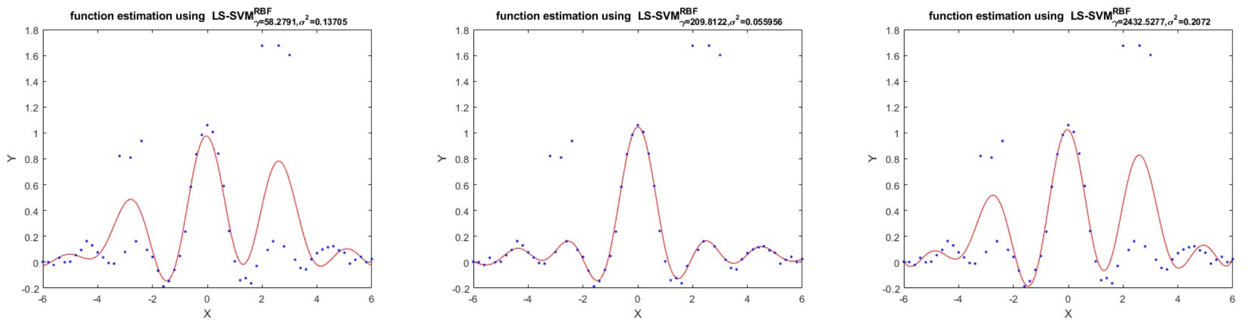
$$\min_{w^\star,b^\star,e^\star} J_{\mathrm{P}}\left(w^\star, e^\star\right) = \frac{1}{2}w^{\star T} w^\star + \gamma \frac{1}{2} \sum_{k=1}^{N} v_k e_k^{\star 2} \tag{13}$$

subject to

$$y_k = w^{\star T} \varphi\left(x_k\right) + b^\star + e_k^\star, k = 1, \ldots, N$$

The only difference between non-robust version and robust version is the cost function L(·). Since we use small weight v_k for large error e_k, the model becomes robust.

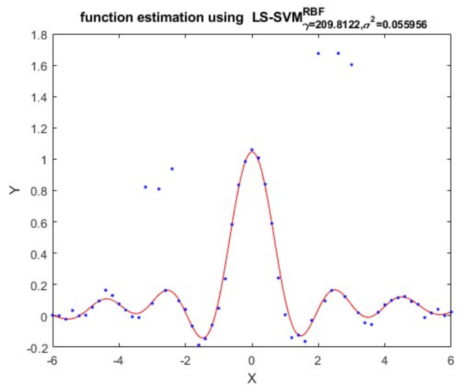
(a)(b) Compare (a) and (b) in Figure 7. The figure for non-robust version is like being lifted by the outliers. The peaks on the two side are far away from data points. But robust version performs perfectly. One reason for that is that the weight v_k reduces the effect of large error e_k. Another reason is that the robust version in (b) of Figure 7 uses L_1 norm as cost function. Then, the effect of outlier will not be squared by L_2 norm. That's also the reason why 'mae' performs better than 'mse' in this case.



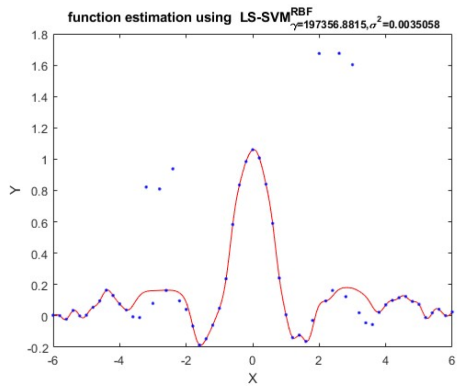
(a) non-robust version (b) robust version with 'mae' as cost function (c) robust version with 'mse' as cost function

Figure 7: Visualization for robust and non-robust LS-SVM

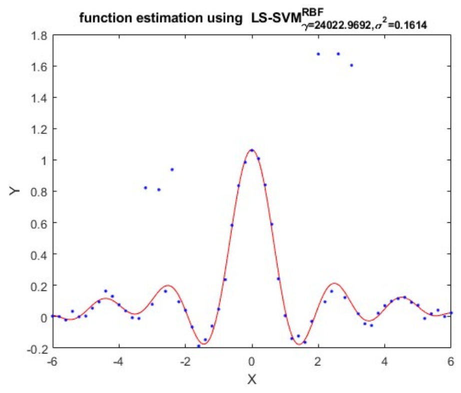
(c) Figure 8 is a table in [1], which shows the weight functions, corresponding loss and score function of four weight function. Huber and Logistic perform better than the other two weight function. The corresponding loss of Huber and Logistic are very similar. Linear or almost linear around 0 and horizontal when the error is very large. But Hampel and Myriad have the process of going up first and going down later as error grows. So the bad performances of these two weight function should come from bad parameter of weight function.



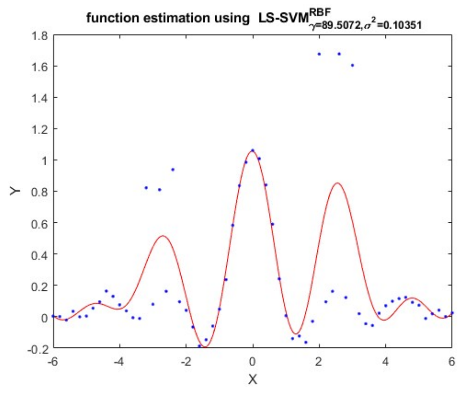
(a) wFun = ’whuber’



(b) wFun = ’whampel’



(c) wFun = ’wlogistic’



(d) wFun = ’wmyriad’

Figure 8: Visualization for robust LS-SVM with different weighting function wFun

2.2 Logmap dataset

(a) Logmap dataset consists of 200 points and our goal is to use the first 150 points to predict the next 50. crossvalidation (10-fold) allows us to do automated tuning for a certain **order**. Since this problem is simple, we can calculate the optimal pair of **sig2** and **gam** for each **order**. Then use these three parameter to calculate **mse** of the training set. The **order** with the least **mse** is the order we want.

(b) In my implementation, I make the raw data to be mean-zero and smoother before being used for finding **order.**. Use the algorithm explained in (a), we get optimal **order** =24. I also tried to divide our training set into training and validation in different ways. But the result is not as good as not divide them or use leave-one-out (need much more time). One of the reason for this is that noise highly affect the training and prediction. Only when using almost all the points for training can avoid this.

2.3 Santa Fe dataset

(a) The training set of Santa Fe dataset contains 1000 points, see Figure 11. And our goal is to predict the next 200 points. The data is oscillating signal with increasing amplitude, but the amplitude of this signal will decrease a lot immdiately in some certain points. As shown in Figure 12, **order** =50 can predict well when the signal is still growing, but will have phrase lag when decreasing occur.

(b) To use the performance of this recurrent prediction on the validation set to optimize hyperparameters and the

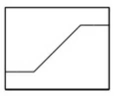
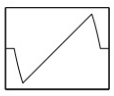
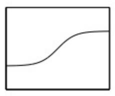
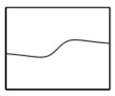
	Huber	Hampel	Logistic	Myriad
$V(r)$	$\begin{cases} 1, & \text{if } \lvert r\rvert < \beta; \\ \frac{\beta}{\lvert r\rvert}, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} 1, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2-\lvert r\rvert}{b_2-b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$\frac{\tanh(r)}{r}$	$\frac{\delta^2}{\delta^2 + r^2}$
$\psi(r)$				
$L(r)$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < \beta; \\ \beta\lvert r\rvert - \frac{1}{2}\beta^2, & \text{if } \lvert r\rvert \geq \beta. \end{cases}$	$\begin{cases} r^2, & \text{if } \lvert r\rvert < b_1; \\ \frac{b_2 r^2 - \lvert r^3\rvert}{b_2-b_1}, & \text{if } b_1 \leq \lvert r\rvert \leq b_2; \\ 0, & \text{if } \lvert r\rvert > b_2. \end{cases}$	$r\tanh(r)$	$\log(\delta^2 + r^2)$

Figure 9: Definitions for the Huber, Hampel, Logistic and Myriad (with parameter $\delta \in \mathbb{R}_0^+$) weight functions $V(\cdot)$. The corresponding loss $L(\cdot)$ and score function $\psi(\cdot)$ are also given.

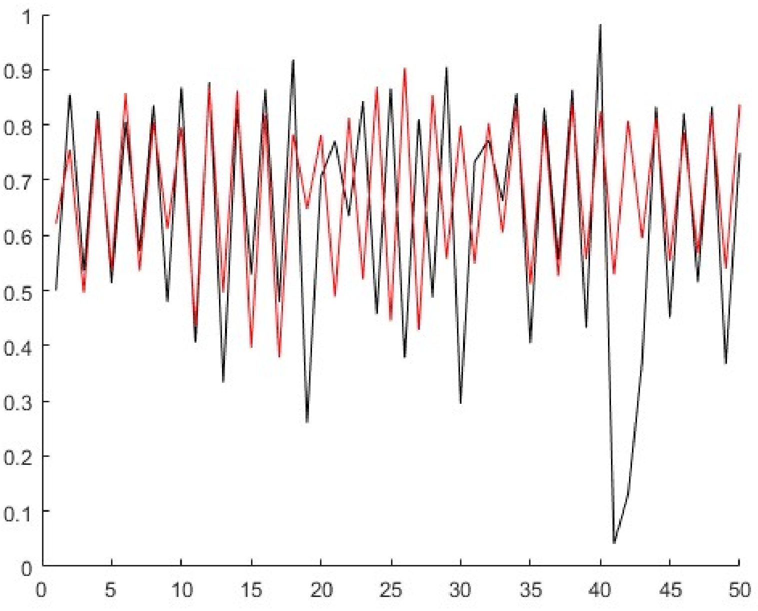


Figure 10: Time series prediction on the logmap dataset with **order** = 24

model order is not sensible in this case. Because the possible **order**'s of this model is quite large. If we still use the method in the last section with crossvalidation, it will spend much more time on optimization. On the other hand, it is important to make sure both training set and validation set have a point for increasing amplitude. So I use the first 500 data points as training and the last 500 dat points as validation set. And it works well and gives **order** =32.

(c) Figure 12(b) shows the predition with tuned **order**,**sig** and **sig2**. And the prediction does not have big phrase lag, but the amplitude after the decreasing point is higher than that in test set. Compare the mse of the two prediction with different **order**. The performance of **order** is much better and smaller **order** reduces the computation.

References

[1] De Brabanter, K., Pelckmans, K., De Brabanter, J., Debruyne, M., Suykens, J. A., Hubert, M., and De Moor, B. Robustness of kernel based regression: a comparison of iterative weighting schemes. In *Artificial Neural Networks–ICANN 2009: 19th International Conference, Limassol, Cyprus, September 14-17, 2009, Proceedings, Part I 19* (2009), Springer, pp. 100–110.

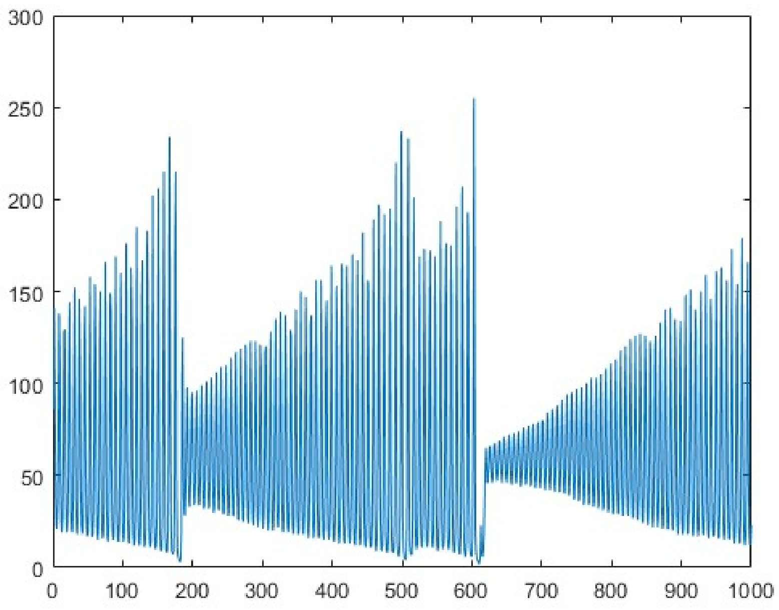
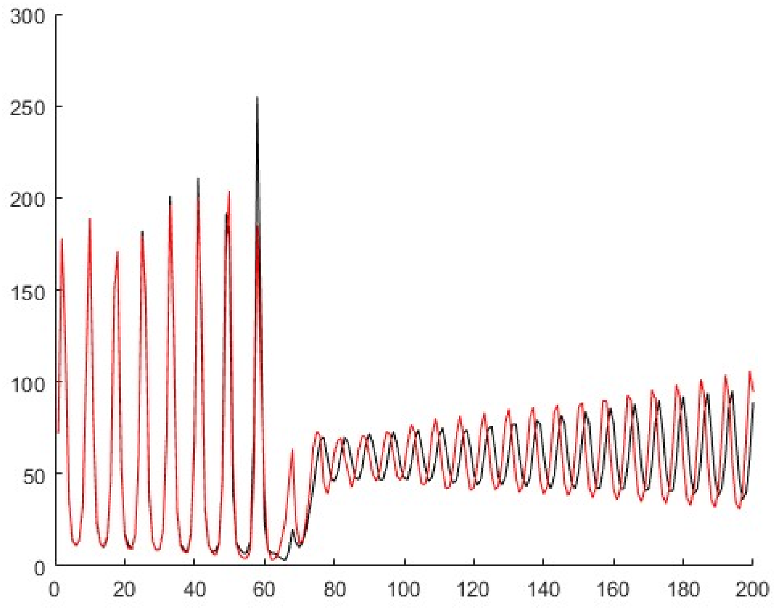
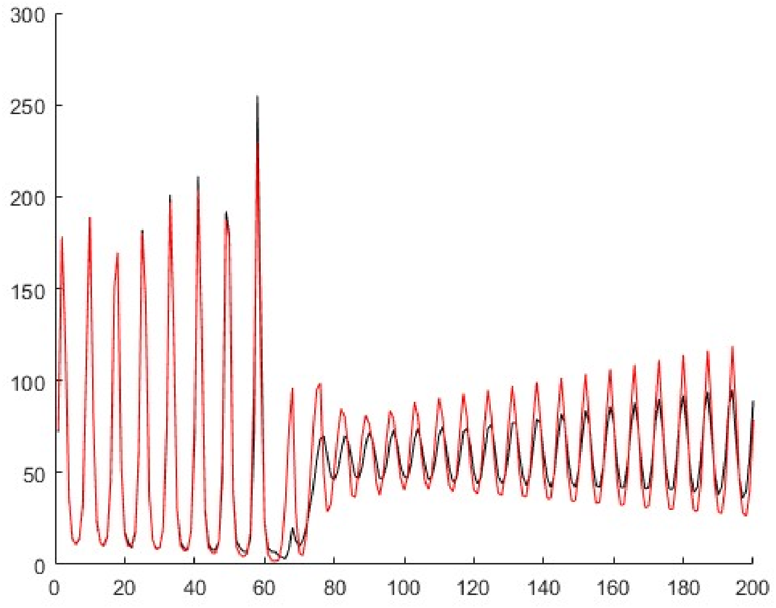


Figure 11: The training set of Santa Fe dataset



(a) with **order** =50, mse=374



(b) with **order** =32, mse=203

Figure 12: Time series prediction on the Santa Fe dataset