Master of bioinformatics

Support Vector Machines

Assignment 2: Function estimation and Time-series prediction

Spring 2016

Author:
Cedric Lood

Supervisors:
Dr. Carlos Alaiz
Dr. Emanuele Frandi
Prof. Johan Suykens

May 27, 2016

Contents

Context

The analysis presented in this report was produced for the class of "Support Vector Machines: methods and applications" (Spring 2016) at KU Leuven. The goal is to display understanding of the techniques and of their practical use. This second report focuses on function estimation and time series prediction using SVM, and particularly Least-Squares SVM (LS-SVM). The implementation was done using the MatLab software (v2015a) and the libraries for LS-SVM developed at KU Leuven [1].

1 Support Vector Machine for regression

1.1 Datasets

I experimented with 3 noiseless datasets of 20 observations for this section, based on the true underlying function shown in figure 1. I used the "load dataset" functionality of the *uiregress* function to load them and perform the regress:

```
X = (linspace(-5, 5, 20))'; Y = (2.*X + 1)';
save('lin.mat');

X = (linspace(-5,5,20))'; Y = (X.^3 + X.^2 - 1)';
save('poly.mat');

X = (linspace(-pi,pi,20))'; Y = (exp(-X.^2).*sin(10.*X))';
save('sinc.mat');
```

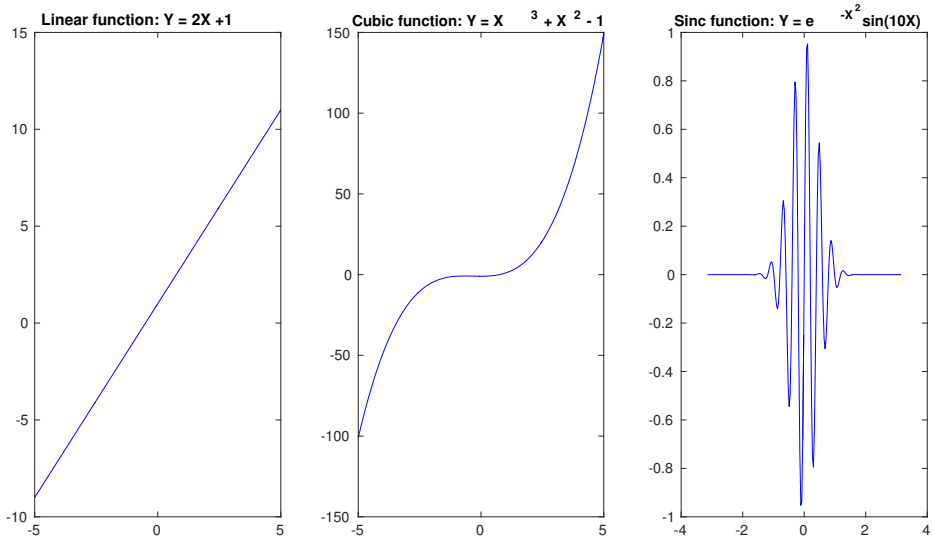

Figure 1: Functions to regress

1.2 Analysis

Here we investigate the effect of fixing the amount of different settings on the function estimation:

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

- ϵ or e controls the loss function
- σ controls the bandwidth of the RBF kernel
- the polynomial degree (1 means linear kernel)
- γ or *bound* controls the amount of regularization (hence the smoothness of the decision boundary)

For the linear and cubic generated dataset, I got good results with polynomial kernel of degree 1 and 3 respectively. In general, the RBF kernel worked well on all the dataset.

I did an extensive survey of the parameters for the first linear dataset. On that particular set, the linear kernel seem to have performed the best. Even for extremely low ϵ (eg $\epsilon = 0.005$), the number of support vectors needed to describe the classifier is equal to two. Another obvious advantage of that kernel was that no wiggle artefact appeared, in contrast with the other more flexible kernels. Note that this performance was achievable using the RBF kernel too, using a very large σ (as large values of that parameter tend to make decision boundaries linear).

	e-value (Bound set to Inf)					Bound (e-value set to 0.1)				
	0.005	0.01	0.05	0.1	0.5	0.01	0.1	1	10	100
Poly 1	2	2	2	2	2	20 **	20 **	13	2	100
Poly 2	3	6	6	6	3	20 **	20 **	8	3	3
Poly 3	7	7	4	4	2 *	20 **	20 **	15 *	4 *	4 *
Poly 4	5	5	5	5	4 *	20 **	20 **	14 *	5 *	5 *
RBF $\sigma^2 = 1$	16	7	7	4	4 *	20 **	20 **	20 **	15 **	4
RBF $\sigma^2 = 0.5$	10	9	9	7	5 *	20 **	20 **	20 **	10 **	7 *

Table 1: Number of support vectors required for different kernels and parameters (: wiggle artefact visible, **: function estimation completely off)*

A general remark concerning the regularization applied via the bound parameter, is that if too little of it is applied, the regression performs poorly on my artificial datasets. Given that it controls how error is allowed in the model, this makes sense. The same remark goes for the e-value, which controls the loss function. If this value is small, the model doesn't allow for a large margin.

2 Sum of Cosines

I started by investigating the regression of the function with a given σ^2 value of 0.1, as illustrated by the following code:

```
gam_list = [1000, 100, 10, 1];
sig2_list = [0.1, 0.1, 0.1, 0.1];
figure('Color',[1 1 1]);

for i = 1:4
    gam = gam_list(i); sig2 = sig2_list(i);
    [alpha, b] = trainlssvm({Xtrain, Ytrain, 'f', gam, sig2, 'RBF_kernel'});
    YtestEst = simlssvm({Xtrain, Ytrain, 'f', gam, sig2, ...
        'RBF_kernel'},{alpha, b}, Xtest);
```

```
    subplot(2,2,i);
    plot(XX, YY, 'k-'); hold on;
    plot(Xtest, Ytest, '.'); hold on;
    plot(Xtest, YtestEst, 'r+');
    legend('true', 'Ytest', 'YtestEst');
end
```

The results are visually reported on figure 2

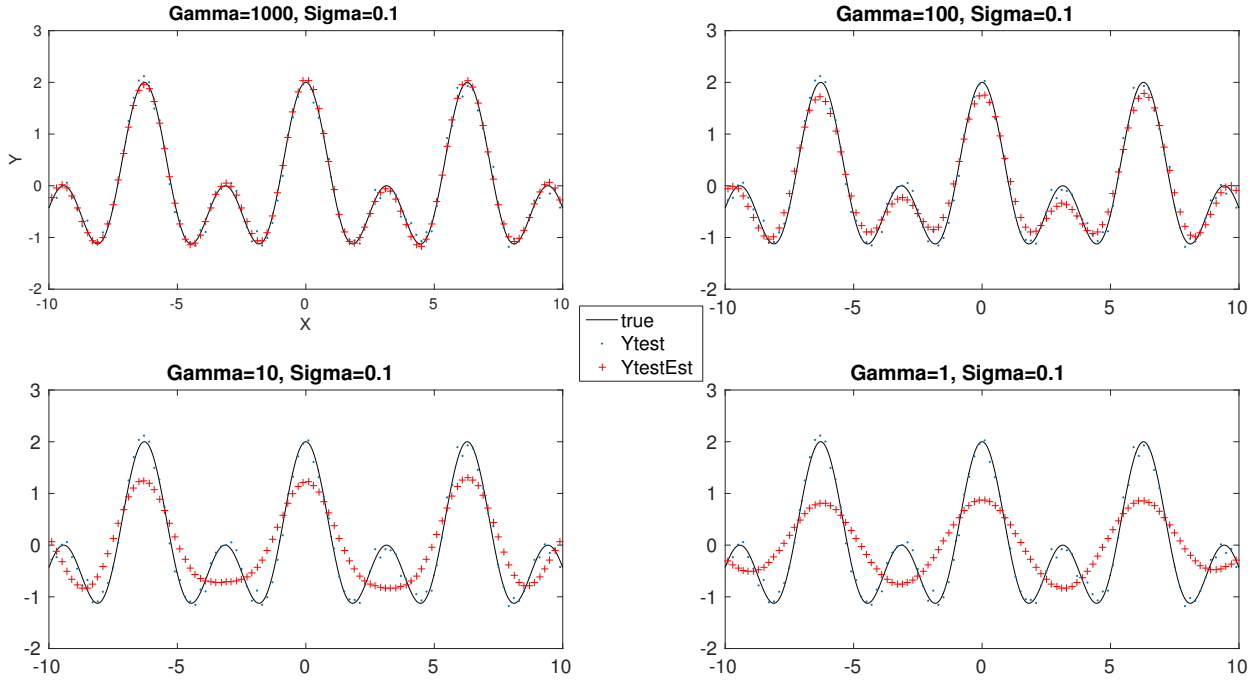

Figure 2: Function estimation for various values of γ (fixed σ^2)

I then investigated the function estimation for a fixed value of γ using similar code. The visual results can be seen on figure 3

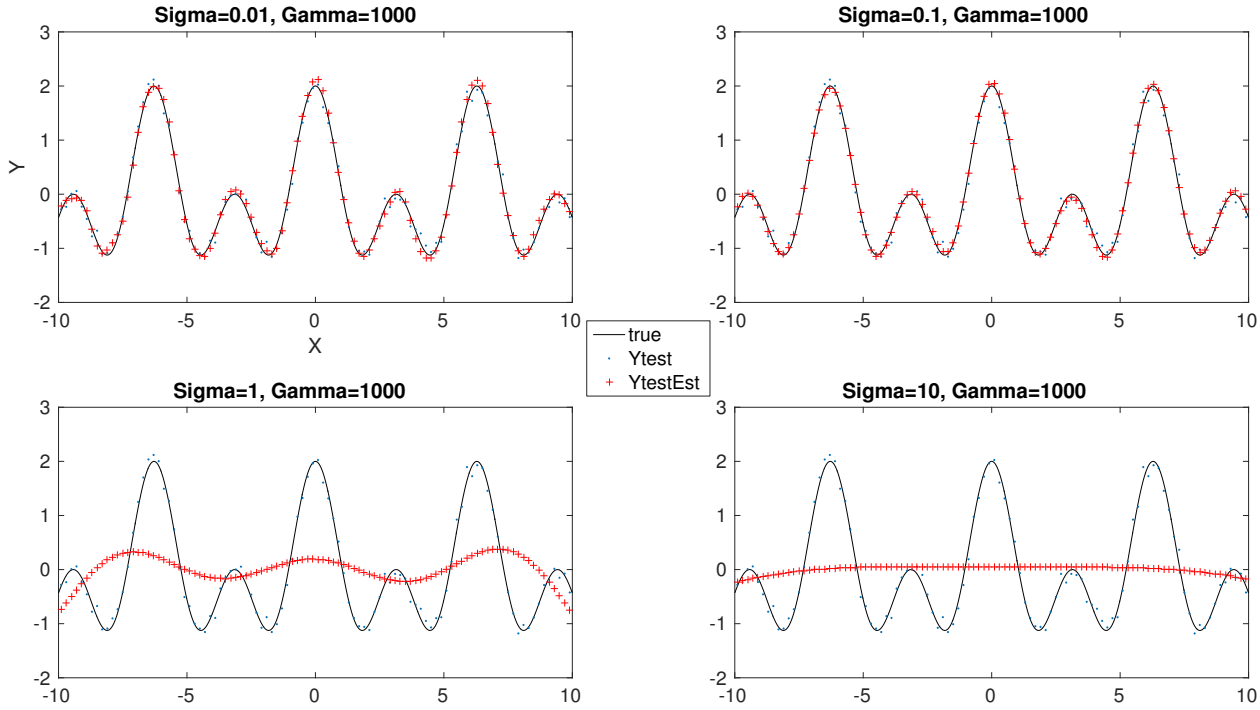

Figure 3: Function estimation for various values of σ^2 (fixed γ)

One of the question from this exercise is to determine whether there is an optimal pair of parameters that would reach the best estimation. Although the SVM formulation is convex, and thus has a single solution, when we start exploring hyperparameter space, the problem is no more convex.

I thought that there would not be a single pair, and decided to verify this by computing and visualizing the error surface consisting of systematic evaluation between the estimated Y for a given pair of σ^2 and γ and the true Y (known in our case given that we are working with a synthetic dataset.) Here is the code I used for this part:

```
gam_list = logspace(-2,2,100); sig2_list = logspace(-2,1,100);
err_matrix = zeros(100,100); i = 1; j = 1;
true_ys = cos(Xtest) + cos(2.*Xtest);

for sig2=sig2_list,
    j = 1;
    for gam=gam_list,
        [alpha, b] = trainlssvm({Xtrain, Ytrain, 'f', gam, sig2, 'RBF_kernel'});
        YtestEst = simlssvm({Xtrain, Ytrain, 'f', gam, sig2, ...
            'RBF_kernel'},{alpha, b}, Xtest);
        err_matrix(i, j) = sum((YtestEst - true_ys).^2);
        j = j + 1;
    end
    i = i + 1;
end

figure('Color',[1,1,1]);
h = surf(gam_list, sig2_list, err_matrix);
set(get(h,'Parent'),'XScale','log');
set(get(h,'Parent'),'YScale','log');
```

The result of the parameter space exploration can be seen on figure 4. One can recognize that there is indeed no single best pair of parameters to obtain. In general a large gamma will reap better results in lowering the error for this synthetic dataset. For larger γ, the value of σ^2 can be larger.

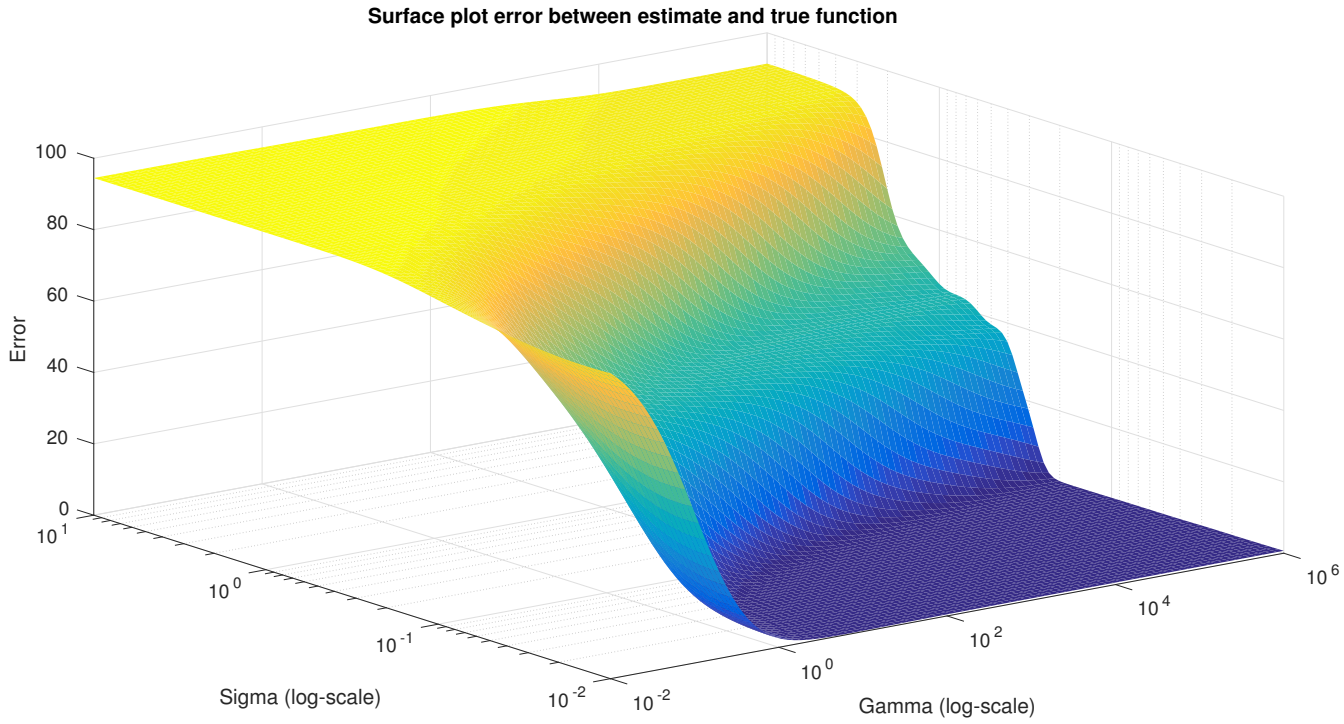

Figure 4: Exploration of parameter space effect on the function estimation

3 Hyper-parameter Tuning

In order to tune the parameters, I used a combination of the 2 global optimisation techniques available (coupled-simulated annealing and randomized directional search) with the 2 techniques to select the values of the hyperparameters (simplex and gridsearch). I could not see any difference in the result of the function estimation. All the values returned were systematically in the area of low test error discovered in the previous section.

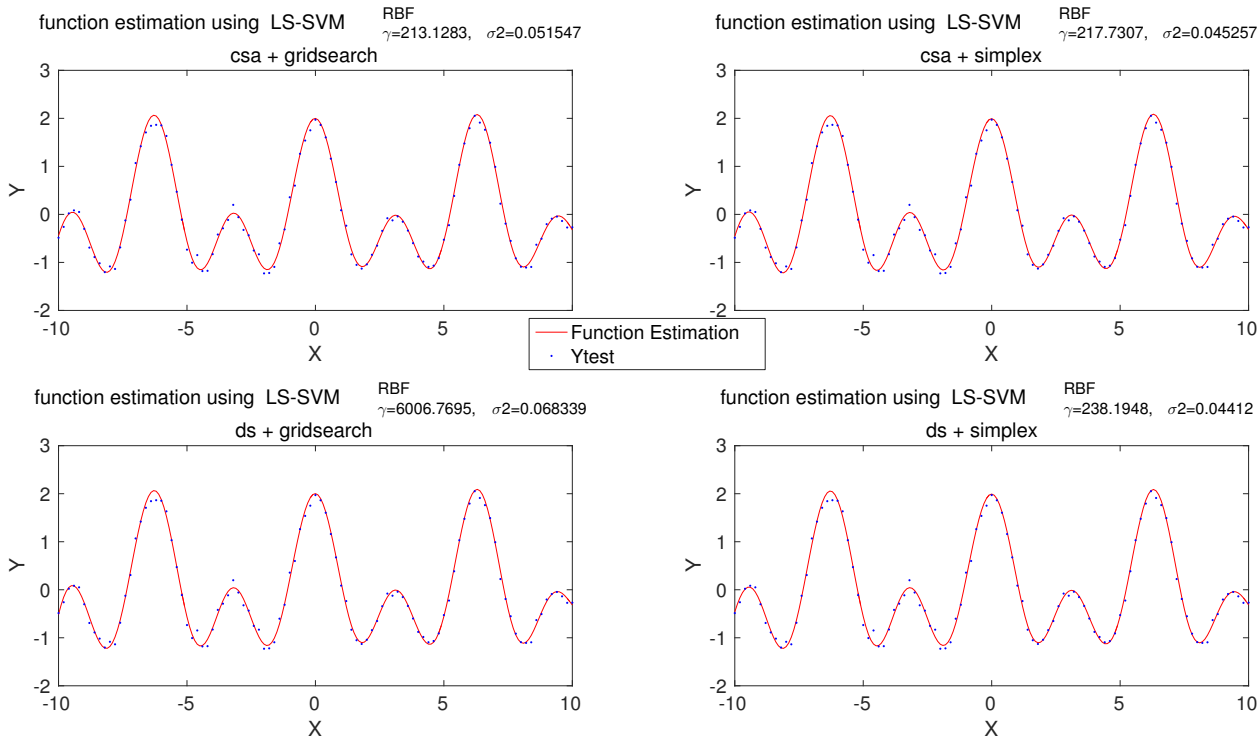

Figure 5: Parameter tuning with heuristics

The speed of execution can be estimated using the tic/toc commands in matlab. The results in execution time in seconds are reported in table 2. It would seem on this small experiment, that the simplex is faster at moving in the search space of γ and σ^2 than gridsearch. This makes sense in light of the functioning of these optimisation functions inner-working. Whereas *gridsearch* locates a minimum by creating a grid of parameter values, *simplex* finds a minimum from a given starting value. Hence the search is faster (though less exhaustive) using *simplex*

The global optimisation algorithm coupled simulated annealing (csa) converges to a minimum faster than the alternative randomized directed search (ds) algorithm.

	run 1	run 2	run 3	run 4	run 5	avg
csa + simplex	0.59	0.68	0.66	0.66	0.61	0.64
csa + gridsearch	1.01	0.92	0.95	0.92	0.91	0.94
ds + simplex	0.96	0.72	0.64	0.79	0.60	0.74
ds + gridsearch	1.06	1.11	1.29	1.07	1.08	1.12

Table 2: Computing time (seconds)

4 Application of the Bayesian framework

In this section, we investigate the use of the bayes rule for the tuning and analysis of function approximation (with a quick incursion in the classification task). Originally introduced within the context of VC-theory, support vector machines have little to do with a probabilistic point of view. The introduction of kernel functions such as RBF however create a link with that world.

An interesting contribution of the introduction of such a probabilistic setting is related to the inference of the parameters, critical to building a coherent SVM model. As seen before, one way to deal with their estimation is via the use of a validation set. Here, we will see that the parameters can be deduced through the application of the Bayes rule within a hierarchical model. Here we will work with a RBF kernel function, hence we will have 3 levels of inference.

Using the same dataset as previously (sum of cosines), we can build our model using the *bay_optimize* function. Note that the examples in the assignment work on the training set, but there is no real need to limit them. Within this bayesian framework, we can estimate all the parameters without the need to check them through validation sets.

```
[~,alpha,b] = bay_optimize({X,Y,'f',gam,sig2}, 1);
[~,gam] = bay_optimize({X,Y,'f',gam,sig2},2);
[~,sig2] = bay_optimize({X,Y,'f',gam,sig2},3);
sig2e = bay_errorbar({X,Y,'f',gam,sig2},'figure');
```

The last line of code produces the figure 6 on which it is possible to visualize at least 2 key things. Firstly, that the modelling performed very well, and secondly, that it is possible to estimate the confidence interval of our regression in a straightforward manner.

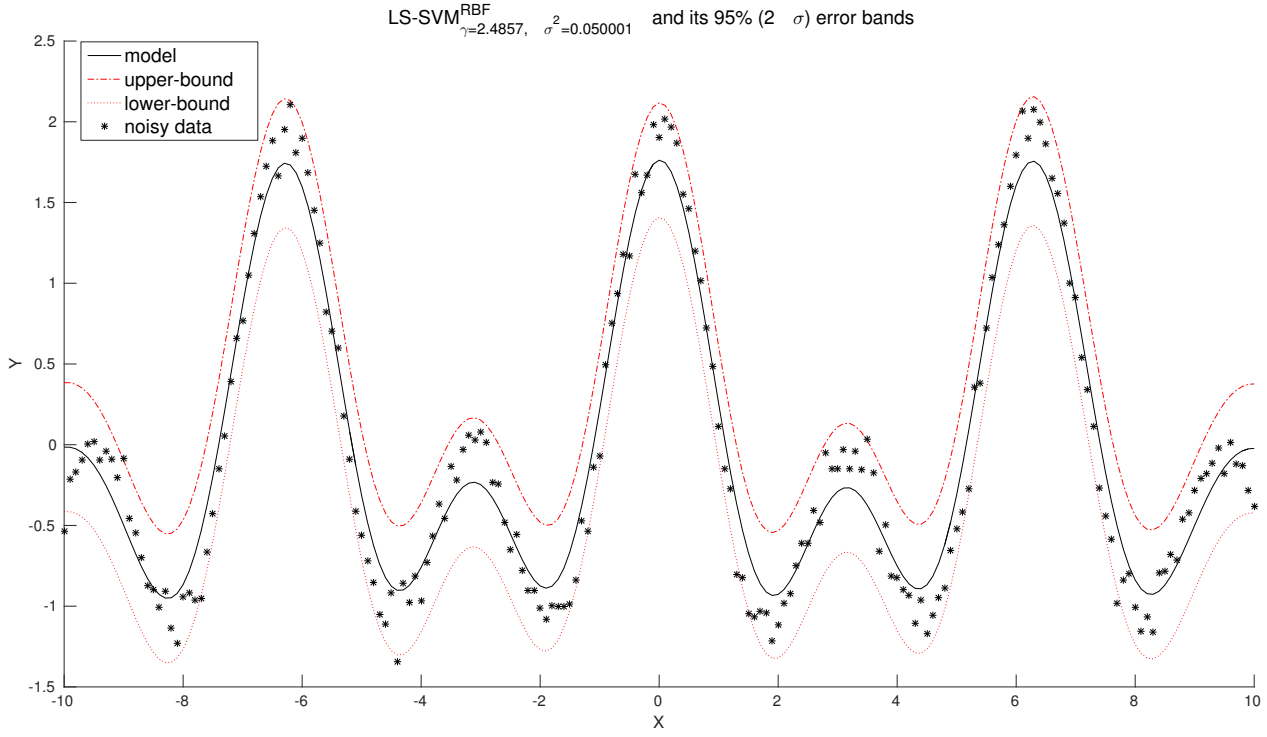

Figure 6: Bayesian parameter estimation and resulting model with error bars

A good way to visualize the dependence between the three levels of inference of our model is to realize the connection that each level has with the next one. Indeed, the so-called evidence bit of the bayes rule of level 1 corresponds to the likelihood bit of level 2, whose evidence is itself the likelihood of level 3.

The next figure 7 reports the results from the classification of the iris dataset. For this, the function *bay_modoutClass* was used with an array of different $\gamma \in \{1, 10, 100\}$ and $\sigma^2 \in \{0.01, 0.1, 1\}$

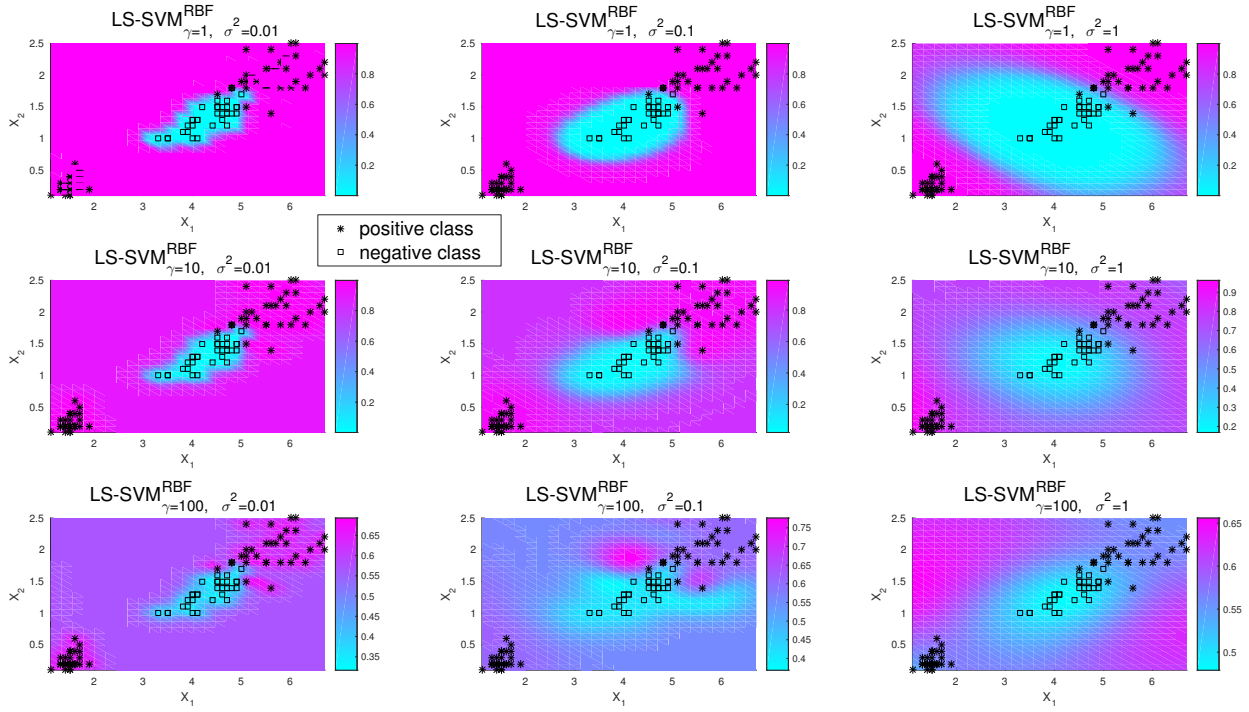

Figure 7: Bayesian parameter estimation and resulting model with error bars

5 Robust regression

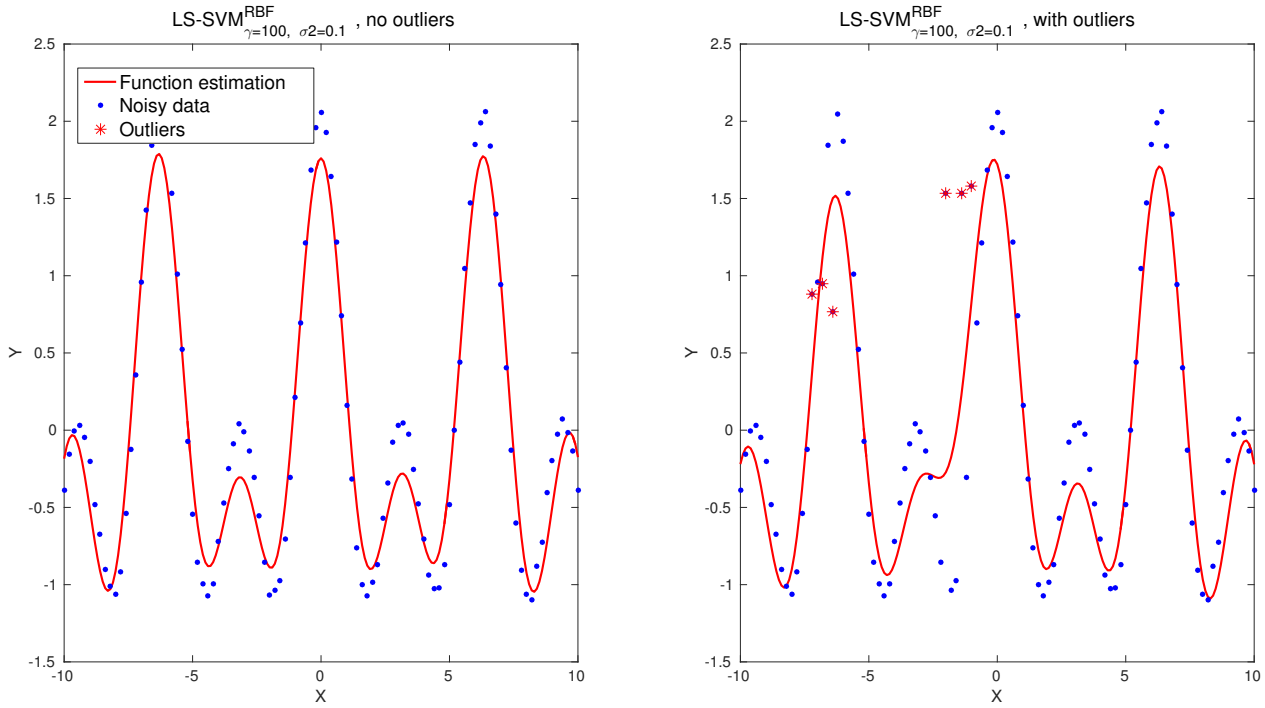

Figure 8: Robust vs. non-robust model

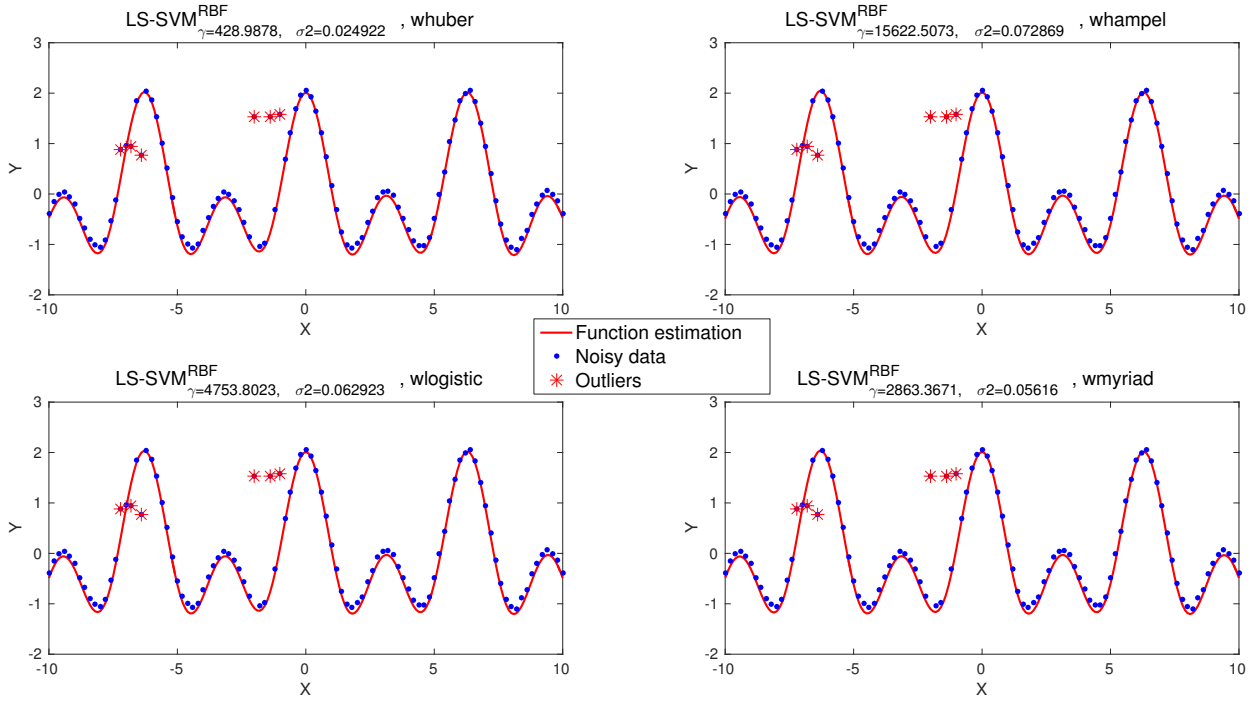

Figure 9: Bayesian robust function estimation

6 Applications

6.1 Time-series Prediction

The dataset we work with in this last section represents a time serie collected from an experiment with a laser [2]). It has been split into a training set and a test set, as illustrated on figure 10. The blue part, will be used to train our model with the goal of predicting the red part.

[2]http://www-psych.stanford.edu/ andreas/Time-Series/SantaFe.html

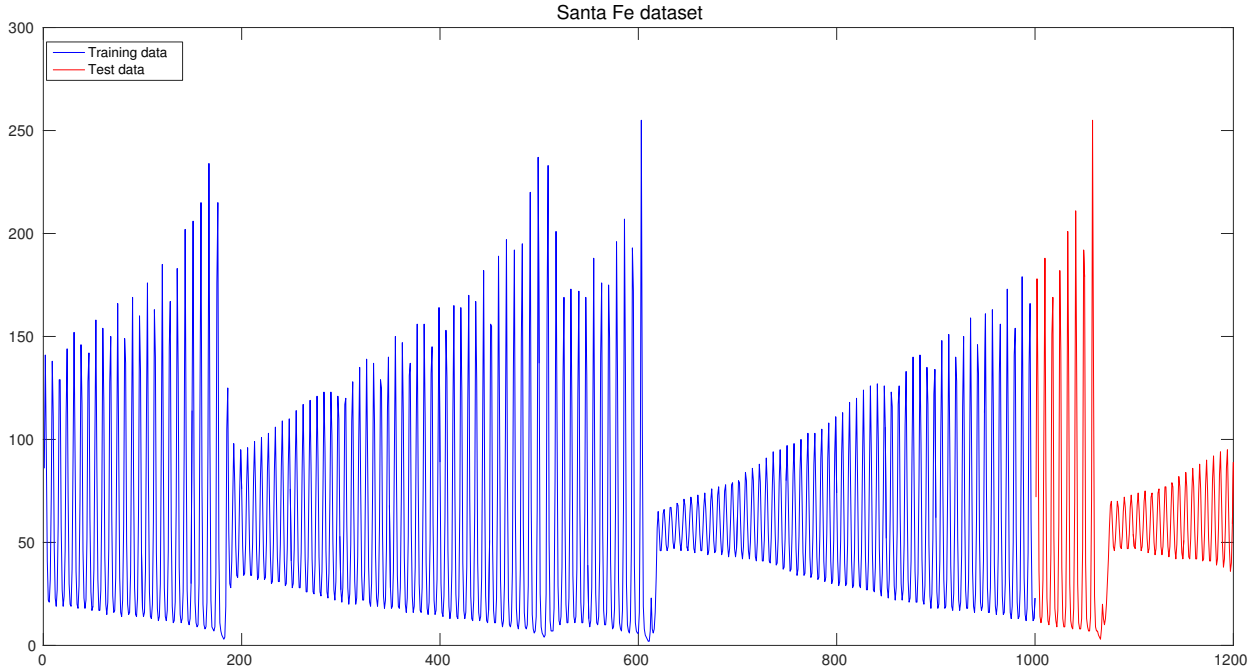

Figure 10: Time serie (santa fe dataset)

As suggested in the exercise, I optimized the kernel parameter σ^2 and the hyperparameter γ via the use of *tunelssvm* with crossvalidation on the training set. Then I proceeded to determine what would be the best *order* to use by looking at the mean squared error between the predictions and the actual value of the test set via a systematic model building over a range of possible values (see figure 11 for results):

```
mse_error = zeros(20,1); i = 1;
for order=5:5:100
    X=windowize(Z,1:(order+1));
    Y=X(:,end); X=X(:,1:order);

    [gam,sig2]=tunelssvm({X,Y,'f',[],[],'RBF_kernel'},'simplex','crossvalidatelssvm',{10,'mse'});
    [alpha,b]=trainlssvm({X,Y,'f',gam,sig2});

    horizon = length(Ztest)-order;
    Zpt = predict({X, Y, 'f', gam, sig2},Ztest(1:order), horizon);
    mse_error(i) = sqrt(mse(Zpt-Ztest(order+1:end)));
    i = i + 1;
end

% building best model based on order selection
order=60;
X=windowize(Z,1:(order+1));
Y=X(:,end); X=X(:,1:order);
[gam,sig2]=tunelssvm({X,Y,'f',[],[],'RBF_kernel'},'simplex','crossvalidatelssvm',{10,'mse'});
[alpha,b]=trainlssvm({X,Y,'f',gam,sig2});
horizon = length(Ztest)-order;
Zpt = predict({X, Y, 'f', gam, sig2},Ztest(1:order), horizon);
```

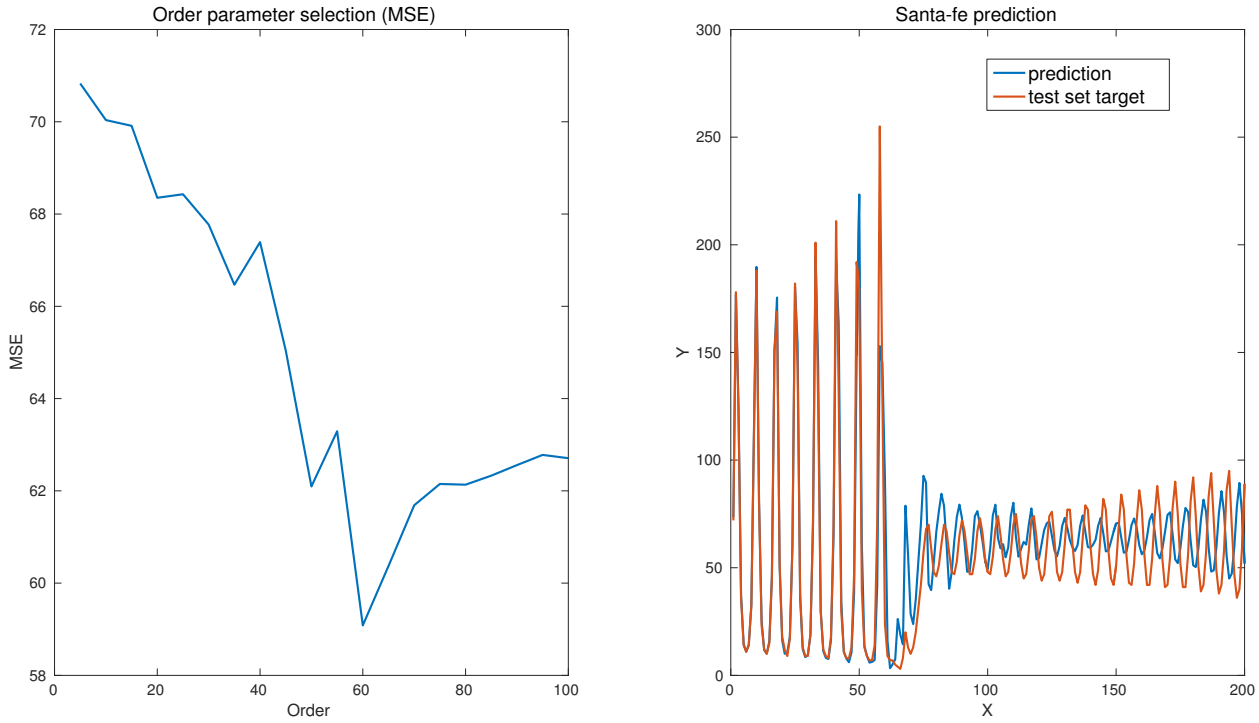

Figure 11: Left: MSE for different values of order, right: prediction

6.2 Santa Fe Laser Dataset

In light of the previous results, one can wonder whether it is possible to obtain even better prediction using approaches seen previously. In this section, inspired by the ESAT tutorial, I show that indeed it is possible to introduce a refinement through the automatic relevance determination.

Note that there is a significant computational cost associated to the method (as implemented through the function *bay_lssvmARD*). The run took over half an hour to complete on my laptop. The previous method returned almost immediately.

References