# Advanced Machine Learning
# Practical 4a Solution: Regression
# ($\epsilon$-SVR, $\nu$-SVR, RVR)

Professor: Aude Billard
Assistants: Nadia Figueroa, Ilaria Lauzana and Brice Platerrier
E-mails: aude.billard@epfl.ch, nadia.figueroafernandez@epfl.ch
ilaria.lauzana@epfl.ch, brice.platerrier@epfl.ch

Spring Semester 2017

## 1 Introduction

During this week's practical we will focus on understanding and comparing the performance of the different regression methods seen in class, namely SVR and its variants ($\epsilon$-SVR, $\nu$-SVR and its Bayesian counterpart RVR).

Similar to non-linear classification methods, the non-linear regression methods we will see in this tutorial (SVR/RVR) rely on a set of hyper-parameters ($\epsilon$: tube sensitivity, $\nu$: bounds, $\sigma$: kernel width for RBF) that are part of the optimization of an objective function (and consequently influence the estimation of the parameters of the regressive function). Choosing the best hyper-parameters for your dataset is not a trivial task, we will analyze their effect on the regression accuracy and how to choose an admissible range of parameters. A standard way of finding these optimal hyper-parameters is by doing a grid search, i.e. systematically evaluating each possible combination of parameters within a given range. We will do this for different datasets and discuss the difference in performance, model complexity and sensitivity to hyper-parameters.

## 2 ML_toolbox

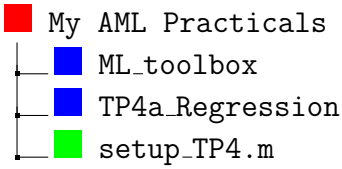
`ML_toolbox` contains a set of methods and examples for easily learning and testing machine learning methods on your data in MATLAB. It is available in the following link:

`https://github.com/epfl-lasa/ML_toolbox`

From the course Moodle webpage (or the website), the student should download and extract the .zip file named `TP4a-Regression.zip` which contains the following files:

| Code all Tasks | |
|---|---|
| TP4_svr.m | setup_TP4.m |

Before proceeding make sure that `./ML_toolbox` is at the same directory level as the TP directory `./TP4a-Regression` and `setup_TP4.m` as follows:




Now, to add these directories to your search path, type the following in the MATLAB command window:

```
>> setup_TP4.m
```

NOTE: To test that all is working properly with `ML_Toolbox` you can try out some examples of the toolbox; look in the **examples** sub-directory.

# 3 Regression Metrics

The metric you will use during this practical to compare the performance of each regressor will be the *Normalized Mean Square Error* ($NMSE$). If you have a vector $\hat{Y}$ consisting in $n$ predictions and a vector $Y$ of the observed values corresponding to these predictions, you can compute the *Mean Square Error* ($MSE$) of the predictor :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2 \tag{1}$$

The *Normalized Mean Square Error* ($NMSE$) is simply the $MSE$ normalized by the variance of the observed values :

$$NMSE = \frac{MSE}{VAR(Y)} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2}{\frac{1}{n-1}\sum_{i=1}^{n}\left(Y_i - \mu\right)^2}, \tag{2}$$

where $\mu$ is the mean of the observed values : $\mu = \frac{1}{n}\sum_{i=1}^{n} Y_i$.

# 4 SVR

Support Vector Machines can be applied not only to classification problems but also to regression problems. In Support Vector Regression (SVR), given a training data set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_M, y_M)\}$ of $M$ samples, where $\mathbf{x}^i \in \mathbb{R}^N$ are multi-dimensional inputs and $y_i \in \mathbb{R}$ are continuous uni-dimensional outputs, we seek to find a continuous mapping function $f : \mathbb{R}^N \to \mathbb{R}$ that best predicts the set of training points with the function $y = f(\mathbf{x})$ (Figure 1).

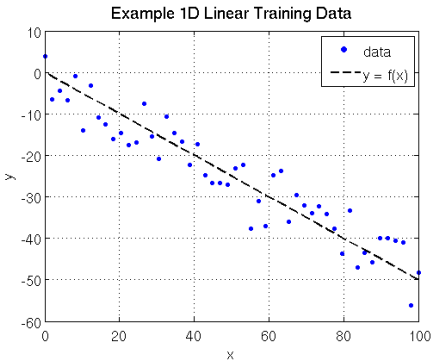



Figure 1: Regression example on 1D linear data.

Such regression problems are found in a wide spectrum of research, financial, commercial and industrial areas. For example, one might want to predict the cost of a car given its attributes, the performance of a CPU given its characteristics or the exchange rate of a currency given some relevant economic indicators. Many linear/non-linear methods exist to obtain this regressive function $f(\mathbf{x})$. What makes SVR a powerful method is that its goal is to obtain a function $f(\mathbf{x})$ that has at most an $\epsilon$-deviation from the training outputs $\{y_1, \ldots, y_M\}$ and is as *flat*[1] as possible. Intuitively, this means that we don't mind having some errors in our regression, as long as they're within an $\epsilon$-deviation of $f(\mathbf{x})$. This type of function is often called $\epsilon$-insensitive loss function and the allowed deviation is called $\epsilon$-insensitive tube. As in SVM, the variants of SVR differ in the way the objective function is formulated and their hyper-parameters, we have $\epsilon$-SVR and $\nu$-SVR, as well as Relevant Vector Regression (RVR), which is analogous to RVM in classification and can be considered as the Bayesian counterpart of SVR.

## 4.1 $\epsilon$-SVR

Following the problem formulation of SVR in the introduction of this section, in $\epsilon$-SVR we seek to find an estimate of the true function:

$$f(\mathbf{x}) = \langle \mathbf{w}^T, \mathbf{x} \rangle + b \quad \text{with} \quad \mathbf{w} \in \mathbb{R}^N, n \in \mathbb{R} \tag{3}$$

[1]Flatness in a regressive function can mean less sensitive to errors in measurement/random shocks/non-stationarity of the input variables. In SVR, this is encoded as maximizing the prediction error within the $\epsilon$-tube

such that, points which are contained withing the $\epsilon$-tube are not penalized, $|f(\mathbf{x}) - y| \leq \epsilon$. This function can be learned by minimizing the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},\xi,\xi^*} & \left( \frac{1}{2}||\mathbf{w}||^2 + \boxed{C} \sum_{i=1}^{M} (\xi_i + \xi_i^*) \right) \\ \text{s.t.} \quad & y^i - \langle \mathbf{w}^T, \mathbf{x}^i \rangle - b \leq \boxed{\epsilon} + \xi_i^* \\ & \langle \mathbf{w}^T, \mathbf{x}^i \rangle + b - y^i \leq \boxed{\epsilon} + \xi_i \\ & \xi_i, \xi_i^* \geq 0, \forall i = 1 \dots M, \epsilon \geq 0 \end{aligned} \tag{4}$$

where $\mathbf{w}$ is the separating hyper-plane, $\xi_i, \xi_i^*$ are the slack variables, $b$ is the bias, $\epsilon$ the allowable error and $C$ is the penalty factor associated to errors larger than $\epsilon$. By solving the dual of this objective function, we achieve a regressive function of the following form:

$$y = f(\mathbf{x}) = \sum_{i=1}^{M} (\alpha_i - \alpha_i^*) \langle \mathbf{x}, \mathbf{x}^i \rangle + b \tag{5}$$

where $\alpha_i$ are the Lagrangian multipliers which define the support vectors ($\alpha_i, \alpha_i^* > 0$ are support vectors). As seen in class, for non-linear datasets, instead of transforming the data to a high-dimensional feature space, we use the inner product of the feature space, this is the so-called *kernel trick*, the regressive function then becomes:

$$y = f(\mathbf{x}) = \boxed{\sum_{i=1}^{M} (\alpha_i - \alpha_i^*) k(\mathbf{x}, \mathbf{x}^i) + b} \tag{6}$$

where $k(\mathbf{x}, \mathbf{x}^i)$ is the kernel function. The kernel function can be any type of function that corresponds to a dot-product of the features transformed to a high-dimensional space, choices of kernel functions for SVR in our **ML_toolbox** are the following:

**Kernels**

- Homogeneous Polynomial: $k(\mathbf{x}, \mathbf{x}^i) = (\langle \mathbf{x}, \mathbf{x}^i \rangle)^p$, where $p > 0$ is the hyper-parameter and corresponds to the polynomial degree.
- Inhomogeneous Polynomial: $k(\mathbf{x}, \mathbf{x}^i) = (\langle \mathbf{x}, \mathbf{x}^i \rangle + d)^p$, where $p > 0$ and $d \geq 0$, generally $d = 1$ are the hyper-parameters that correspond to the polynomial degree and the offset.
- Radial Basis Function (Gaussian): $k(\mathbf{x}, \mathbf{x}^i) = \exp\left\{ -\frac{1}{2\sigma^2} ||\mathbf{x} - \mathbf{x}^i||^2 \right\}$, where $\sigma$ is the hyper-parameter and corresponds to the width or scale of the Gaussian kernel centered at $\mathbf{x}^i$

**Hyper-parameters** Apart from the kernel hyper-parameters, $\epsilon$-SVR has two open-parameters:

- $C$: Cost $[0 \to \infty]$ represents the penalty associated with errors larger than epsilon. Increasing cost value causes closer fitting to the calibration/training data.
- $\epsilon$: epsilon represents the minimal required precision.

**Q: What is the effect of $\epsilon$ on the regressive function?**
When training $f(\mathbf{x})$, there is no penalty associated with points which are predicted within distance $\epsilon$ from the $y^i$. Small values of $\epsilon$ force closer fitting to the training data. Since $\epsilon$ controls the width of the insensitive error zone, it can directly affect the number of support vectors. By increasing $\epsilon$, we might get fewer support vectors, but could yield more flat estimates (see Figure 2).

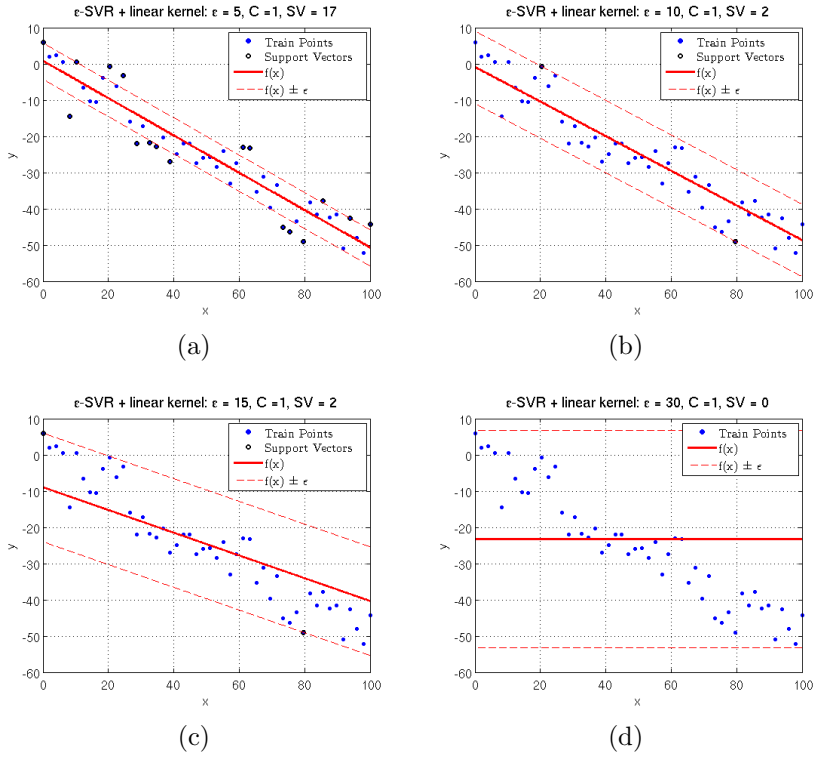
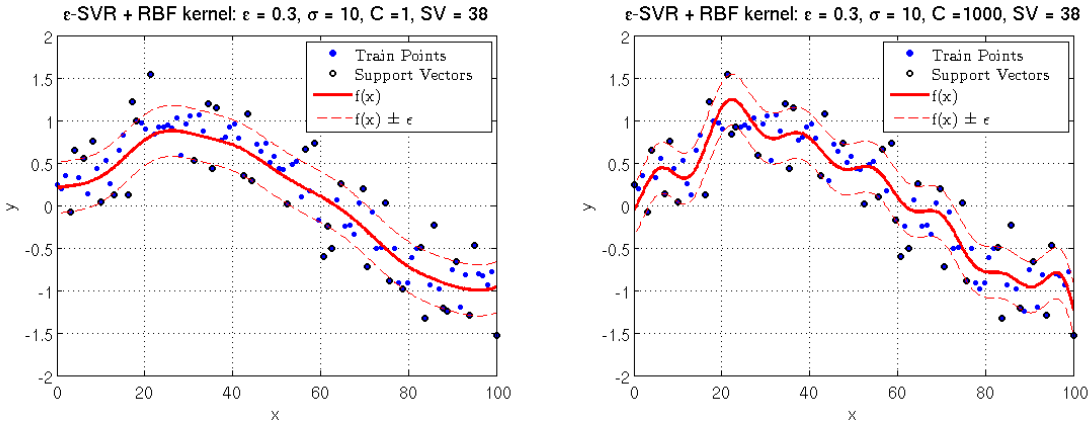



Figure 2: Effect of $\epsilon$ on regressive function

**Q: What is the effect of C on the regressive function?**
Parameter C determines the trade off between the model complexity (flatness) and the degree to which deviations larger than $\epsilon$ are tolerated in the optimization formulation. For example, if C is too large (infinity), then the objective is to minimize the error at all costs (see Figure 3).

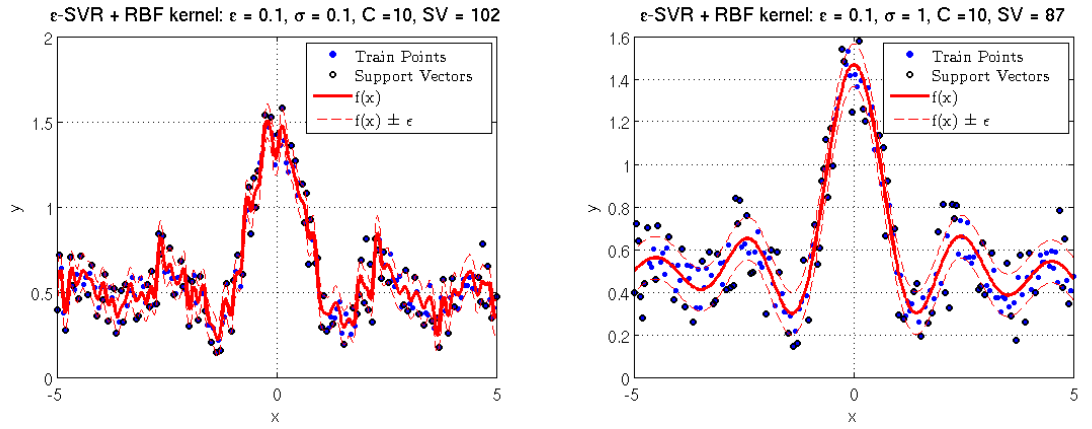
**Q: What is the effect of the kernel hyper-parameters on the regressive function?**
When using the RBF kernel, we would need to find an optimal value for $\sigma$, which is the width in the squared-exponential function. Intuitively, $\sigma$ is the radius of influence of the selected support vectors, if $\sigma$ is very low, it will be able to encapsulate only those points in the feature space which are very close, on the other hand if $\sigma$ is very big, the support vector will influence points further away from them. It can be thought of as a parameters that controls the shape of the separating hyperplane. Thus, the smaller $\sigma$ the more likely to get more support vectors (for some values of $C$ and $\epsilon$). (See Figure 4).

You can generate several datasets and test different hyper-parameter combinations in the matlab script **TP4_SVR.m**.




(a) (b)

Figure 3: Effect of $C$ on regressive function




(a) (b)

Figure 4: Effect of $\sigma$ on regressive function

## 4.2 $\nu$-SVR

As it is difficult to select an appropriate $\epsilon$, $\nu$-SVR was introduced. Rather than controlling for the maximum allowable $\epsilon$-deviation, this new $\nu$ parameter lets us control the number of support vectors and training errors and gives an estimate of the $\epsilon$ in the data. The original $\epsilon$-SVR objective function now is of the following form:

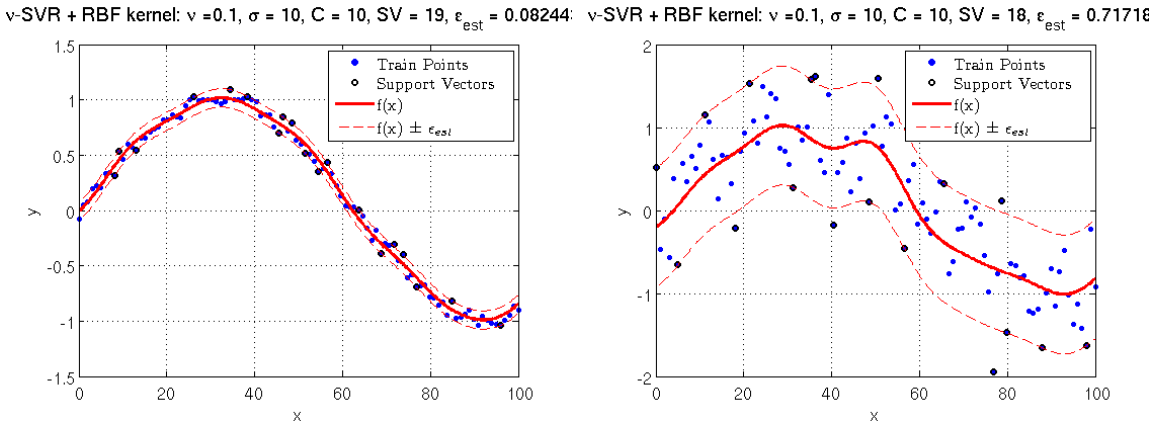
$$\begin{aligned} \min_{\mathbf{w},\xi,\xi^*} & \frac{1}{2}||\mathbf{w}||^2 + \boxed{C}\left(\boxed{\nu\epsilon} + \frac{1}{M}\sum_{i=1}^{M}(\xi_i + \xi_i^*)\right) \\ \text{s.t.} \quad & y^i - \langle \mathbf{w}^T, \mathbf{x}^i \rangle - b \leq \epsilon + \xi_i^* \\ & \langle \mathbf{w}^T, \mathbf{x}^i \rangle + b - y^i \leq \epsilon + \xi_i \\ & \xi_i, \xi_i^* \geq 0, \forall i = 1 \ldots M, \epsilon \geq 0 \end{aligned} \tag{7}$$

$\nu$ represents an *upper bound* on the fraction of margin errors and a *lower bound* for the number of support vectors, which is proportional to the ratio $\nu \leq \frac{\#SV}{M}$. Adding these new variables only affects the objective function (Equation 7), the regressive function stays the same as for $\epsilon$-SVR (Equation 6).

**Hyper-parameters** In $\nu$-SVR rather then setting $\epsilon$ we set this new parameter $\nu$, alongside $C$ and the hyper-parameter for the chosen kernel function.

- $\nu$: $\nu$ $(0 \rightarrow 1]$ indicates a lower bound on the number of support vectors to use, given as a fraction of total calibration samples, and an upper bound on the fraction of training samples which are errors (poorly predicted).

**Q: What is the effect of $\nu$ on the regressive function?**
The parameter $\nu$ is used to determine the proportion of the number of support vectors you desire to keep in your solution with respect to the total number of samples in the dataset. $\epsilon$ is introduced into the optimization problem formulation and it is estimated automatically for you depending on the data at hand. Hence, given the same parameters, the regressive function will automatically adapt to the noise of the data (see Figure 5).




(a) True noise level $\epsilon = 0.05$ (b) True noise level $\epsilon = 0.5$

Figure 5: Adaptation of $\epsilon$ given different noise levels from the same signal using $\nu$-SVR

## 4.3 RVR

The Relevance Vector Machine (RVM) for regression follows essentially the same formulation as for the classification case, we only modify the form of the likelihood function (output conditional distribution) to account for continuous outputs with an estimated noise variance. To recall, the predictive linear model for RVM has the following form:

$$y(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i \, k(\mathbf{x}, \mathbf{x}^i) = \alpha^T \Psi(\mathbf{x}) \tag{8}$$

where $\Psi(\mathbf{x}) = [\Psi_0(\mathbf{x}), \ldots, \Psi_M(\mathbf{x}),]^T$ is a linear combination of basis functions $\Psi(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}^i)$, $\alpha$ is a sparse vector of weights and $y(\mathbf{x})$ is the real-valued mapping function $y(\mathbf{x}) \to \mathbb{R}$. To calculate $\alpha$ we take a Bayesian approach and assume that all outputs $y^i$ are i.i.d samples from a true model $y(\mathbf{x^i})$ (Equation 8) subject to some Gaussian noise with zero-mean:

$$\begin{aligned} y^i(\mathbf{x}) &= y(\mathbf{x}^i) + \epsilon \\ &= \alpha^T \Psi(\mathbf{x}^i) + \epsilon \end{aligned} \tag{9}$$

where $\epsilon \propto \mathcal{N}(0, \sigma_\epsilon^2)$. We can then estimate the probability of an output $y^i$ given $x$ with a Gaussian distribution with mean on $y(\mathbf{x^i})$ and variance $\sigma_\epsilon^2$, this is equivalent to:

$$\begin{aligned} p(y^i|\mathbf{x}) &= \mathcal{N}(y^i|y(\mathbf{x^i}), \sigma_\epsilon^2) \\ &= (2\pi\sigma_\epsilon^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2\sigma_\epsilon^2}(y^i - \alpha^T\Psi(\mathbf{x}^i))^2\right\} \end{aligned} \tag{10}$$

Thanks to the independence assumption we can then estimate the likelihood of the complete dataset with the following form:

$$p(\mathbf{y}|\mathbf{x}, \sigma_\epsilon^2) = (2\pi\sigma_\epsilon^2)^{-\frac{M}{2}} \exp\left\{-\frac{1}{2\sigma_\epsilon^2}||\mathbf{y} - \alpha^T\Psi(\mathbf{x})||^2\right\} \tag{11}$$

To impose sparsity on the solution, we define an explicit prior probability distribution over $\alpha$, namely a zero-mean Gaussian:

$$p(\alpha|\mathbf{a}) = \prod_{i=0}^{M} \mathcal{N}(\alpha^i|0, a_i^{-1}) \tag{12}$$

where $\mathbf{a} = [a^0, \ldots, a^M]$ is a vector of parameters $a^i$, each corresponding to an independent weight $\alpha^i$ indicating the strength or *relevance* of a sample. Now, for the *regression problem*, following Bayesian inference, to learn the unknown parameters $\alpha, \mathbf{a}, \sigma_\epsilon^2$ we start with the decomposed posterior[2] over unknowns given the data:

$$p(\alpha, \mathbf{a}, \sigma_\epsilon^2|\mathbf{y}) = p(\alpha|\mathbf{y}, \mathbf{a}, \sigma_\epsilon^2)p(\mathbf{a}, \sigma_\epsilon^2|\mathbf{y}) \tag{13}$$

where the posterior distribution over the weights $p(\alpha|\mathbf{y}, \mathbf{a}, \sigma_\epsilon^2)$ can be computed analytically[3]. The parameters posterior $p(\alpha, \sigma_\epsilon^2|\mathbf{y})$ is unfortunately not analytically solvable.

[2]Refer to the Tipping paper on RVM to understand how this and subsequent terms were derived.

[3]This comes from the application of the Bayes rule $p(\alpha|\mathbf{y}, \mathbf{a}, \sigma_\epsilon^2) = \frac{p(\mathbf{y}|\alpha, \sigma_\epsilon^2)p(\alpha|\mathbf{a})}{p(\mathbf{y}|\mathbf{a}, \sigma_\epsilon^2)}$.

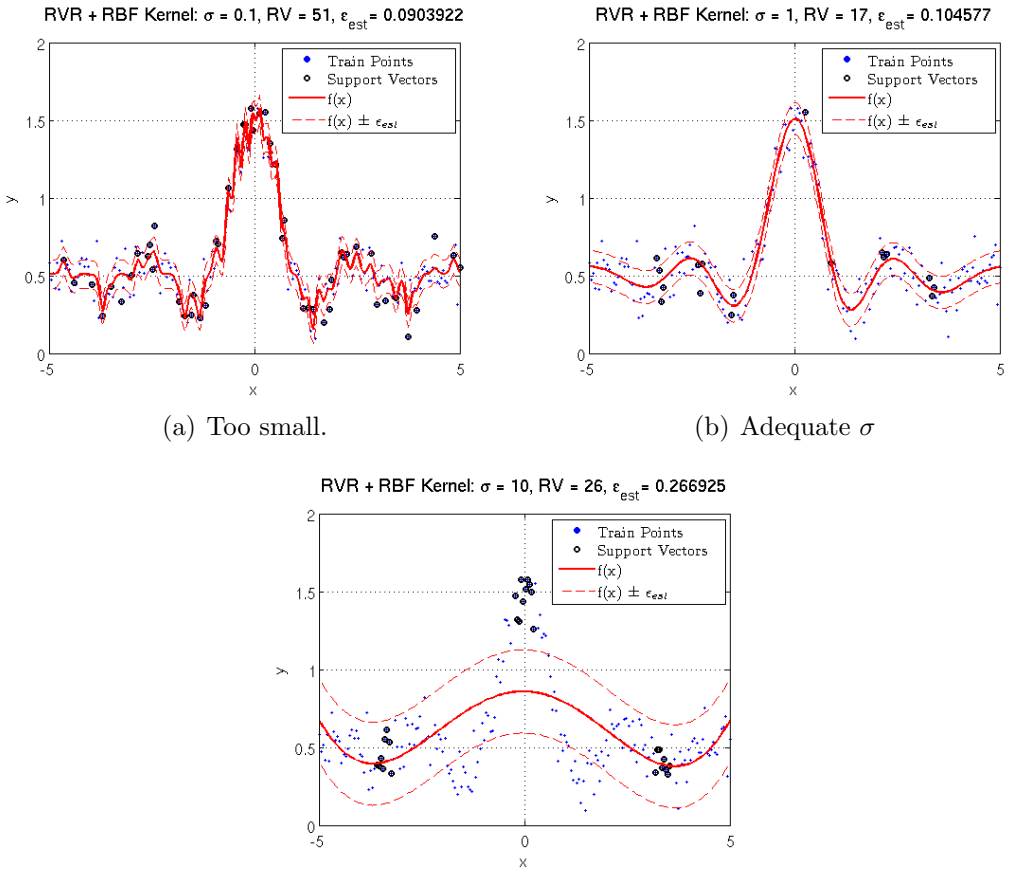
The problem then becomes the maximization of $p(\mathbf{a}, \sigma_\epsilon^2|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{a}, \sigma_\epsilon^2)p(\mathbf{a})p(\sigma_\epsilon^2)$. Assuming uniform priors $p(a), p(\sigma_\epsilon^2)$, the problem reduces to maximizing the term $p(\mathbf{y}|\mathbf{a}, \sigma_\epsilon^2)$, given by:

$$\begin{aligned} p(\mathbf{y}|\mathbf{a}, \sigma_\epsilon^2) &= \int p(\mathbf{y}|\mathbf{w}, \sigma_\epsilon^2)p(\mathbf{w}|\alpha)d\mathbf{w} \\ &= (2\pi)^{-\frac{M}{2}}|\sigma_\epsilon^2\mathbf{I} + \mathbf{\Psi}\mathbf{A}^{-1}\mathbf{\Psi}^T|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\mathbf{y}^T(\sigma_\epsilon^2\mathbf{I} + \mathbf{\Psi}\mathbf{A}^{-1}\mathbf{\Psi}^T)^{-1}\mathbf{y}\right\}. \end{aligned} \tag{14}$$

where $\mathbf{A} = diag(\alpha_0, \ldots, \alpha_M)$. By solving this optimization problem[4] we find our weight vector $\alpha$, consequently the optimal 'relevant vectors', and the estimate of the noise variance $\sigma_\epsilon^2$.

**Q: What is the effect of the kernel hyper-parameters on the regressive function?**

Even though the RVR formulation is set to optimize the number of relevant vectors as well as the $\epsilon$-deviation, if we set a kernel width (for example for RBF) too low/too high it will over-fit/under-fit the regressive function, and give a high number of relevant vector regardless of sparsity constraints (see Figure 6).




(a) Too small.

(b) Adequate $\sigma$

(c) Too big.

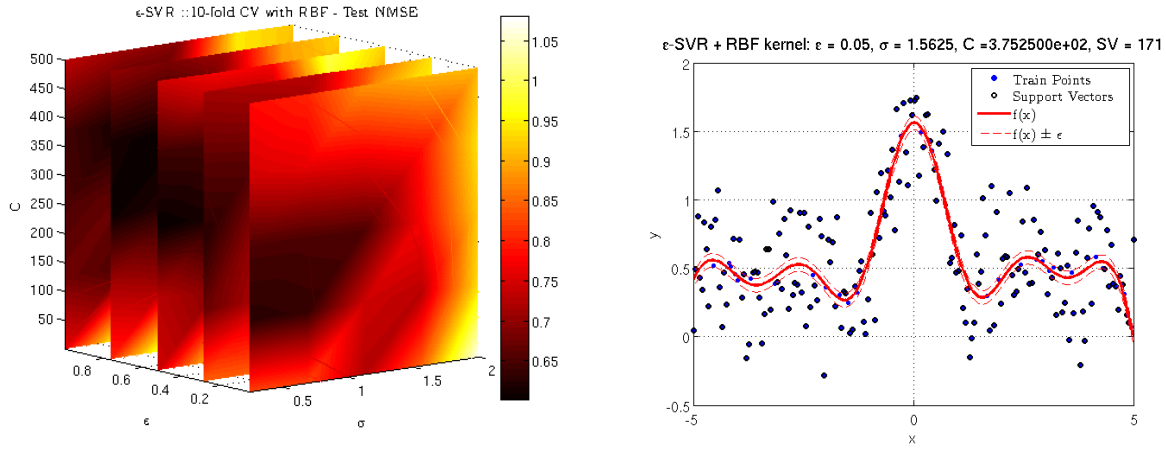
Figure 6: Effect of $\sigma$ for RBF kernel in RVR

[4]Refer to Tipping paper for parameter optimization methods.

## 4.4 SVR-RVR Evaluation

**Q: How to find the optimal parameter choices?**
As in classification, we can use grid search with cross-validation on the open parameters. We must choose admissible ranges for these, in the case of $\epsilon$-SVR with RBF kernel, it would be for $C$, $\epsilon$ and $\sigma$. Additionally, in order to get statistically relevant results one has to cross-validate with different test/train ratio. A standard way of doing this is applying 10-fold Cross Validation for each of the parameter combinations and keeping some statistics such as mean and std. deviation of the regression metrics for the 10 runs of that specific combination of parameters.

**Grid Search with 10-fold Cross Validation**
For the sinc dataset we can run $\epsilon$-SVR (with RBF kernel) with 10-fold CV over the following range of parameters $C\_range = [1 : 500]$, $\epsilon\_range = [0.05 : 1]$,and $\sigma\_range = [0.25 : 2]$, in the matlab script **TP4_SVR.m** you can specify the steps to partition the range within these limits. By selecting the combination of hyper-parameter values that yields the highest performance, according to some metric, on the test dataset, we can achieve an approximate regressive function as the one seen in Figure 7.




(a) Heatmaps for 3-parameter grid-search

(b) Best fitted function with $\epsilon$-SVR.

Figure 7: Grid Search result for 1D non-linear on $\epsilon$-SVR.

**Q: How did we choose these ranges? Any ideas?**
By running this part of the script you should get the plot in Figure 7. This shows a heatmap for the mean Train (left) and Test (right) NMSE (i.e. normalized mean squred error).

**Q: Which method should we use?**
The decision regarding which method to choose depends completely on the type of model you want. If you prefer an efficient solution in terms of model complexity (i.e. less support vectors), without having such a precise estimate, either $\nu$-SVR or RVR would be the way to go. However, if you wish to control the amount of error in the model and have the best performance possible, regardless of model complexity, then $\epsilon$-SVR would be the best choice.