Submission by:

Tikhon Riazantsev

Agastya Heryudhanto

Exercise sheet 9

TASK 1: Support Vector Regression

Task 1: Support Vector Regression (SVR) (20 points)

The dual QP of the kernel SVR is given by:

$$\text{Maximize} \quad -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{m}\varepsilon(\alpha_i + \alpha_i') + \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i')$$

$$\text{subject to} \quad \sum_{i=1}^{m}(\alpha_i - \alpha_i') = 0$$

$$\text{with} \quad 0 \le \alpha, \alpha' \le C.$$

a) Before implementing, set up the QP in its standard form. Thus, derive the matrices Q, A and A$_{equ}$ as well as the vectors **c**, **b** and **b**$_{equ}$. Remember the upper and lower bounds. How does the variable vector of the QP look like in this scenario? **Derive** does not mean "just write them down". How does the vector **x** look like in this scenario? (6 points)

As we know from Matlab documentation, quadprog() function is specified as:

quadprog finds a minimum for a problem specified by

$$\min_x \frac{1}{2}x^T Hx + f^T x \text{ such that } \begin{cases} A \cdot x \le b, \\ Aeq \cdot x = beq, \\ lb \le x \le ub. \end{cases}$$

H, A, and Aeq are matrices, and f, b, beq, lb, ub, and x are vectors.

Before we derive all of the matrices, we should first settle on how to describe the variable vector x, as from its definition everything else can be calculated.

NON-CURSIVE ‘x’ IS THE VARIABLE VECTOR, ‘x’ IS THE DATA POINT OF DATA POINT MATRIX ‘X’

In the case of solving the dual SVR problem we have two unknown vectors: α, α'. Because the variable x in quadprog is always a vector (a one-dimensional matrix) let us define it as:

$$\mathbf{x} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \\ \alpha_1' \\ \alpha_2' \\ \vdots \\ \alpha_m' \end{pmatrix}$$

Firstly, because quadprog solves a minimization problem but SVR is a maximization problem, we can negate the target function to bring it to the standard form.

The dual QP of the kernel SVR is thus given by:

Minimize:

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=1}^{m}\varepsilon(\alpha_i + \alpha_i') - \sum_{i=1}^{m} y_i(\alpha_i - \alpha_i')$$

Subject to:

$$\sum_{i=1}^{m}(\alpha_i - \alpha_i') = 0$$

With:

$$0 \leq \boldsymbol{\alpha}, \boldsymbol{\alpha}' \leq C$$

Let us now define the matrix Q (or H as in Matlab documentation). We can more closely look at this part of the dual QP of the kernel SVR:

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

We can define a new vector **h** as:

$$\boldsymbol{h} = \boldsymbol{\alpha} - \boldsymbol{\alpha}'$$

And from that write:

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} h_i h_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}\boldsymbol{h}^T Q^* \boldsymbol{h}$$

where: $q_{ij}^* = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

(from Chapter 8: Kernel functions)

To derive ***h*** from our variable vector **x** we can use the following matrix A_h of the length 2m and height m:

$$A_h = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & -1 \end{pmatrix}$$

$$A_h\mathbf{x} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & -1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \\ \alpha'_1 \\ \alpha'_2 \\ \vdots \\ \alpha'_m \end{pmatrix} = \begin{pmatrix} \alpha_1 - \alpha'_1 \\ \alpha_2 - \alpha'_2 \\ \vdots \\ \alpha_m - \alpha'_m \end{pmatrix} = \boldsymbol{\alpha} - \boldsymbol{\alpha}' = \boldsymbol{h}$$

As we know, the transpose of the product of two matrices is equal to the product of their transposes taken in reverse order. So:

$$\boldsymbol{h}^T = (A_h\mathbf{x})^{\mathrm{T}} = \mathbf{x}^T A_h^T$$

We can now finally write:

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i - \alpha'_i)(\alpha_j - \alpha'_j)K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \frac{1}{2}\boldsymbol{h}^T Q^* \boldsymbol{h} = \frac{1}{2}\mathbf{x}^T A_h^T Q^* A_h \mathbf{x} = \frac{1}{2}\mathbf{x}^T Q \mathbf{x}$$

Where: $$Q = H = A_h^T Q^* A_h = \begin{pmatrix} Q^* & -Q^* \\ -Q^* & Q^* \end{pmatrix} = \begin{pmatrix} K(\boldsymbol{X},\boldsymbol{X}) & -K(\boldsymbol{X},\boldsymbol{X}) \\ -K(\boldsymbol{X},\boldsymbol{X}) & K(\boldsymbol{X},\boldsymbol{X}) \end{pmatrix}$$

We can now look at another part of the dual QP SVR target function to find the value of vector **f**:

$$\sum_{i=1}^{m}\varepsilon(\alpha_i + \alpha'_i) - \sum_{i=1}^{m} y_i(\alpha_i - \alpha'_i)$$

Let us rewrite this sum in a more comprehensible manner:

$$\sum_{i=1}^{m}\varepsilon(\alpha_i + \alpha'_i) - \sum_{i=1}^{m} y_i(\alpha_i - \alpha'_i) =$$

$$= \varepsilon(\alpha_1 + \alpha'_1) + \varepsilon(\alpha_2 + \alpha'_2) + \cdots + \varepsilon(\alpha_m + \alpha'_m) - y_1(\alpha_1 - \alpha'_1) - y_2(\alpha_2 - \alpha'_2) - \cdots - y_m(\alpha_m + \alpha'_m) =$$

we can now take out alphas with the same indices

$$= \alpha_1(\varepsilon - y_1) + \alpha_2(\varepsilon - y_2) + \cdots + \alpha_m(\varepsilon - y_m) + \alpha'_1(\varepsilon + y_1) + \alpha'_2(\varepsilon + y_2) + \cdots + \alpha'_m(\varepsilon + y_m)$$

It is fairly obvious to see that the vector **f** can be defined as:

$$\boldsymbol{f} = \begin{pmatrix} (\varepsilon - y_1) \\ (\varepsilon - y_2) \\ \vdots \\ (\varepsilon - y_m) \\ (\varepsilon + y_1) \\ (\varepsilon + y_2) \\ \vdots \\ (\varepsilon + y_m) \end{pmatrix}$$

As there are no inequality constraints in this problem, the matrix A and vector **b** are null:

$$A = [\,]; \quad \boldsymbol{b} = [\,]$$

To find out the vector **b**$_{\textbf{eq}}$ and matrix A$_{\text{eq}}$ we need to look at the equality constraints

$$\sum_{i=1}^{m} (\alpha_i - \alpha_i') = 0$$

Obviously, **b**$_{\text{eq}}$ is a scalar which just equals 0.

$$b_{eq} = 0$$

To find out A$_{\text{eq}}$ let us break down the constraint:

$$\sum_{i=1}^{m} (\alpha_i - \alpha_i') = a_1 - a_1' + a_2 - a_2' + \cdots + a_m - a_m' = a_1 + a_2 + \cdots + a_m - a_1' - a_2' - \cdots - a_m'$$

From this we can now find the value of A$_{\text{eq}}$:

$$A_{eq} = (\underbrace{1 \quad 1 \quad \cdots \quad 1}_{\text{m}} \quad \underbrace{-1 \quad -1 \quad \cdots \quad -1}_{\text{m}})$$

To find the values of vectors lb and ub we would not need to do any calculations as they are vectors where all elements have the same value, 0 for lb and C for ub of length 2m. That is possible because **x** contains all of the alphas so that they all can be constrained in this way.

b) Implement the missing functionality in `svrTrain.m`. Your implementation should be able to handle data of any finite dimension. (5 points)

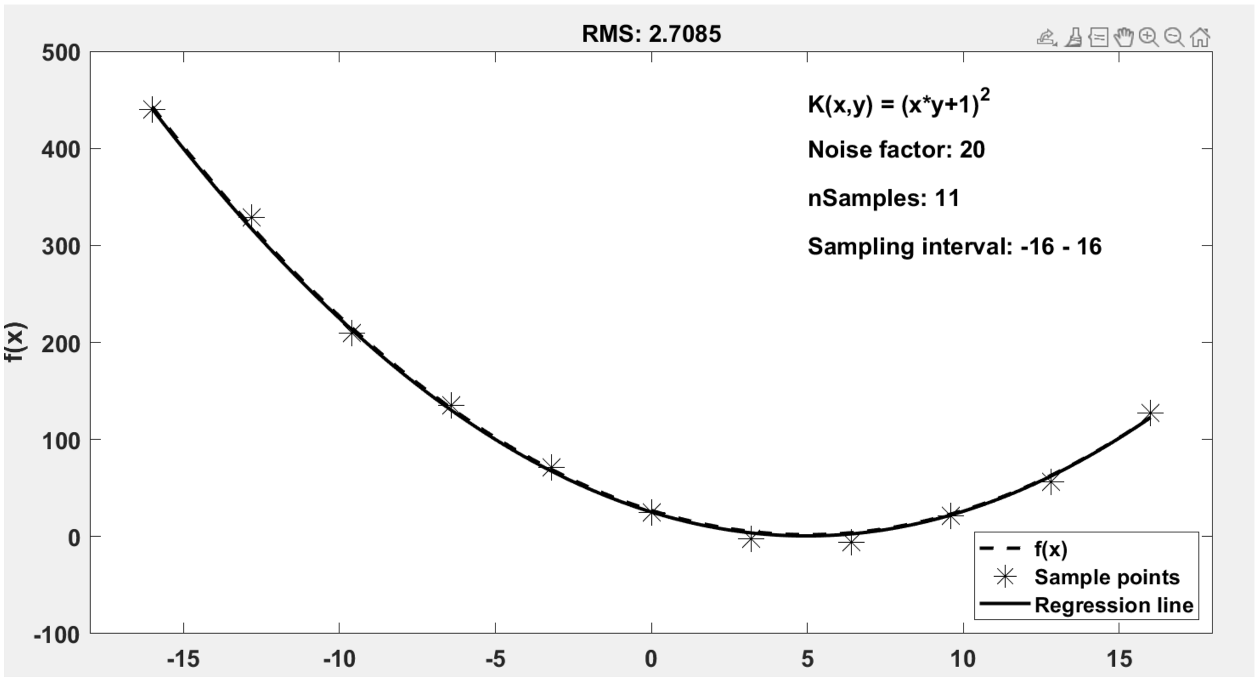
c) Implement the missing functionality in `svrProduce.m`. Your implementation should be able to handle one-dimensional and two-dimensional data. (3 points)

Completed in the .m files

d) Have a look at the two test instances. Find a proper combination of the SVR parameters ε and C. To help you with this task, in the title of each plot, the root-mean-square (RMS) error is shown. For each test scripts, find **one** combination of parameters which returns the best results from your point of view. Briefly comment on your results. Zip your implementations and the generated PNG files from the test scripts and upload your archive to the Moodle course. (6 points)

For the 1d calculation we have ended up using these values of epsilon and C:

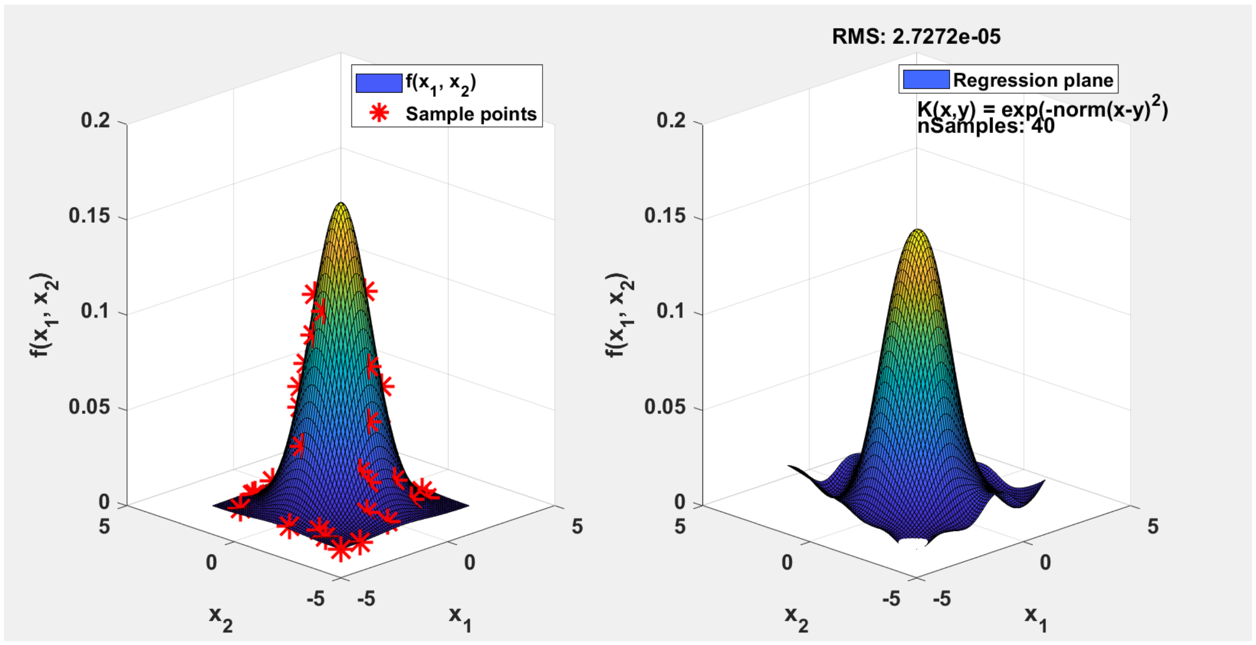
$$\mathrm{C} = 1000000; \quad \varepsilon = 0.0001$$



Using these values, it is possible to get the RMS value under 15 with curve very similar to the original. Smaller epsilon would make the graph too ‘narrow’ in the x direction, and larger ones make it too ‘wide’. C smaller than 0.1 make the calculation results almost unusable with the graph barely resembling the original. C larger than 10000000 do not change the resulting graph much but influence the number of alphas larger than 0. We believe that the values that we have decided on prove to be the middle ground in regards to those effects.

For the 2d calculations we have decided on these values for epsilon and C:

$$C = 0.2; \quad \varepsilon = 0.0001$$



Using these values, it is possible to get the RMS value under 8e-5 with a surface very similar to the original (although not on the bottom part, that would require more data points and/or perhaps a different kernel function). Any value of C more than 0.2 would work with this data as the upper limit is not reached by alphas. The smaller C heavily impact alphas calculation and in result the algorithm’s ability to capture the spike in the middle of the surface and make the surface ‘flatter’. Epsilon also chosen quite carefully: epsilons larger than 0.01 produce an almost completely flat surface and all alphas close to 0. Smaller epsilon make the surface more closely follow the data points which usually does not benefit the final surface as it usually also ‘smoothens’ out that spike on the top.