Project I

The project is open book. You may complete this project by any means you wish. Any help from the internet should be reported in the appendix. There are a total of 100 points.

Read each problem carefully. Show enough work to prove that you know what you are doing and to receive as much partial credit as possible. If computations are required, you must show supporting calculations or/and computer output. State all conclusions in the context of the problem.

Problem 1

We want to solve the online LS-SVDD problem in Python using the dataset “charlie.csv”. We will use the variables $X_1, \dots X_4$. Assume an online scenario in which $n-$ 1 data have been processed at the $n-$ 1-th iteration. The LS-SVDD solution reads

$$\boldsymbol{\alpha}_{n-1} = \frac{1}{2}\mathbf{H}_{n-1}^{-1}\left(\mathbf{k}_{n-1} + \frac{2 - \mathbf{e}_{n-1}'\mathbf{H}_{n-1}^{-1}\mathbf{k}_{n-1}}{\mathbf{e}_{n-1}'\mathbf{H}_{n-1}^{-1}\mathbf{e}_{n-1}}\mathbf{e}_{n-1}\right). \tag{1}$$

where $\mathbf{H}_{n-1} = \mathbf{K}_{n-1} + \frac{1}{2C}\mathbf{I}_{n-1}$, $\boldsymbol{\alpha}_{n-1} = (\alpha_1, \dots, \alpha_{n-1})'$ represents the column vector of Lagrange multipliers. The matrix $\mathbf{K}_{n-1}$ is the Gram matrix and has entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j), i, j = 1, \dots, n-1$, $\mathbf{e}_{n-1} = (1, \dots, 1)'$, $\mathbf{I}_{n-1}$ denotes the $n-1 \times n-1$ identity matrix, and $\mathbf{k}_{n-1}$ denotes a vector with entries $k_j = k(\mathbf{x}_j, \mathbf{x}_j), j = 1, \dots, n-1$.

In the next iteration, n, a new observation $\mathbf{x}_n$ is received and we wish to update the solution (1) recursively.
The updated kernel matrix is

$$\mathbf{H}_n = \begin{bmatrix} \mathbf{H}_{n-1} & \boldsymbol{\delta}_n \\ \boldsymbol{\delta}_n' & \delta_{nn} + \frac{1}{2C} \end{bmatrix}$$

where $\boldsymbol{\delta}_n = (k(\mathbf{x}_n, \mathbf{x}_1), \dots, k(\mathbf{x}_n, \mathbf{x}_{n-1}))'$, and $\delta_{nn} = k(\mathbf{x}_n, \mathbf{x}_n)$.
By introducing the variables

$$\mathbf{a}_n = \mathbf{H}_{n-1}^{-1}\boldsymbol{\delta}_n,$$

and

$$\gamma_n = \delta_{nn} + \frac{1}{2C} - \boldsymbol{\delta}_n'\mathbf{a}_n,$$

the new inverse kernel matrix is calculated as

$$\mathbf{H}_n^{-1} = \frac{1}{\gamma_n}\begin{bmatrix} \gamma_n\mathbf{H}_{n-1}^{-1} + \mathbf{a}_n\mathbf{a}_n' & -\mathbf{a}_n \\ \mathbf{a}_n & 1 \end{bmatrix}$$

Finally, the updated solution $\boldsymbol{\alpha}_n$ is

$$\boldsymbol{\alpha}_n = \frac{1}{2}\mathbf{H}_n^{-1}\left(\mathbf{k}_n + \frac{2 - \mathbf{e}_n'\mathbf{H}_n^{-1}\mathbf{k}_n}{\mathbf{e}_n'\mathbf{H}_n^{-1}\mathbf{e}_n}\mathbf{e}_n\right). \tag{2}$$

where $\boldsymbol{\alpha}_n = (\alpha_1, \dots, \alpha_n)'$ represents the column vector of Lagrange multipliers. $\mathbf{e}_n = (1, \dots, 1)'$ is a $n \times 1$ column vector, and $\mathbf{k}_n$ denotes a vector with entries $k_j = k(\mathbf{x}_j, \mathbf{x}_j), j = 1, \dots, n$.

a. Write a Python function to obtain the update inverse matrix $\mathbf{H}_n^{-1}$.

b. Write a Python function to obtain the updated solution $\boldsymbol{\alpha}_n$

c. Apply your function to the set obtained with the first row and add sequentially the next 6 rows. Check that your code work correctly by comparing the $\boldsymbol{\alpha}$ obtained using the recursive updates (i.e. $\boldsymbol{\alpha}_2$, $\boldsymbol{\alpha}_3$, $\boldsymbol{\alpha}_4$, $\boldsymbol{\alpha}_5$, $\boldsymbol{\alpha}_6$, $\boldsymbol{\alpha}_7$) to the actual $\boldsymbol{\alpha}$'s using the direct approach ($\boldsymbol{\alpha}_2$, $\boldsymbol{\alpha}_3$, $\boldsymbol{\alpha}_4$, $\boldsymbol{\alpha}_5$, $\boldsymbol{\alpha}_6$, $\boldsymbol{\alpha}_7$).

d. Use your results from (b) to sequentially update the radius R_n^2 and d_z. Check sequentially if the next 7 rows are targets or outliers by comparing $d_{\mathbf{x}_n}$ to R_{n-1}^2 .

e. Compare the training times (in sec) between the 2 algorithms.

Problem 2

Considering that LS-SVDD model based on the first N data has been constructed, and the new data $\mathbf{x}_{N+1}, \dots \mathbf{x}_{N+K}$ is fed. The subscript N means that the current model is based on the first N observations. For $N+K$ new observations, one has

$$\boldsymbol{\alpha}_{N+K} = \frac{1}{2}\mathbf{H}_{N+K}^{-1}\left(\mathbf{k}_{N+K} + \frac{2 - \mathbf{e}'_{N+K}\mathbf{H}_{N+K}^{-1}\mathbf{k}_{N+K}}{\mathbf{e}'_{N+K}\mathbf{H}_{N+K}^{-1}\mathbf{e}_{N+K}}\mathbf{e}_{N+K}\right). \tag{3}$$

where

$$\mathbf{H}_{N+K} = \begin{bmatrix} \mathbf{H}_N & \mathbf{B}' \\ \mathbf{B} & \mathbf{D} \end{bmatrix}$$

and

$$\mathbf{B} = \begin{pmatrix} k(\mathbf{x}_{N+1}, \mathbf{x}_1) & k(\mathbf{x}_{N+1}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{N+1}, \mathbf{x}_N) \\ k(\mathbf{x}_{N+2}, \mathbf{x}_1) & k(\mathbf{x}_{N+2}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{N+2}, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_{N+K}, \mathbf{x}_1) & k(\mathbf{x}_{N+K}, \mathbf{x}_2) & \cdots & k(\mathbf{x}_{N+K}, \mathbf{x}_N) \end{pmatrix}$$

and

$$\mathbf{D} = \begin{pmatrix} k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) & k(\mathbf{x}_{N+1}, \mathbf{x}_{N+2}) & \cdots & k(\mathbf{x}_{N+1}, \mathbf{x}_{N+K}) \\ k(\mathbf{x}_{N+2}, \mathbf{x}_{N+1}) & k(\mathbf{x}_{N+2}, \mathbf{x}_{N+2}) & \cdots & k(\mathbf{x}_{N+2}, \mathbf{x}_{N+K}) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_{N+K}, \mathbf{x}_{N+1}) & k(\mathbf{x}_{N+K}, \mathbf{x}_{N+2}) & \cdots & k(\mathbf{x}_{N+K}, \mathbf{x}_{N+K}) \end{pmatrix} + \frac{1}{2C}\mathbf{I}.$$

Now, The matrix $\mathbf{H}_{N+K}$ can be obtained from $\mathbf{H}_N$ without computing the matrix inverse.

$$\mathbf{H}_{N+K}^{-1} = \begin{bmatrix} \mathbf{H}_N & \mathbf{B}' \\ \mathbf{B} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \left[\mathbf{H}_N - \mathbf{D}^{-1}\mathbf{B}'\mathbf{B}\right]^{-1} & \mathbf{H}_N^{-1}\mathbf{B}'\left[\mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}' - \mathbf{D}\right]^{-1} \\ \left[\mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}' - \mathbf{D}\right]^{-1}\mathbf{B}\mathbf{H}_N^{-1} & \left[\mathbf{D} - \mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}'\right]^{-1} \end{bmatrix}$$

Also,

$$\left[\mathbf{H}_N - \mathbf{D}^{-1}\mathbf{B}'\mathbf{B}\right]^{-1} = \mathbf{H}_N^{-1} - \mathbf{H}_N^{-1}\mathbf{B}'\left[-\mathbf{D} + \mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}'\right]^{-1}\mathbf{B}\mathbf{H}_N^{-1}.$$

a. Write a Python function to obtain the block update inverse matrix $\mathbf{H}_{N+K}^{-1}$.

b. Check that the results match the ones using the direct approach

c. We will test the speed and effectiveness of the presented algorithm by applying LS-SVDD, LS-SVDD update (K = 1) from problem 1, and LS-SVDD block update (K = 15) to the following data sets available from the UCI Machine Learning Repository. We will use a Gaussian kernel with $\sigma = 1$. Also, we will set C to 5 and N to 1. Your results should include "Training accuracy (%)", "Testing accuracy (%)" and "Training Time (s)".

   The data sets are: Tic-Tac-Toe Endgame data set 'https://archive.ics.uci.edu/ml/data sets/Tic-Tac-Toe+Endgame' for Group I, Splice-Juncion Gene Sequences data set 'https://archive.ics.uci.edu/ml/data sets/Molecular+Biology+(Splice-junction+Gene+Sequences)' for Group II,
   and Sat Image data set 'https://archive.ics.uci.edu/ml/data sets/Statlog+(Landsat+Satellite)' for Group III.

   For Group I, we will use 90% of 'positive' class as training and the remaining 10% with 'negative' class as testing set. For Group II, we use 90% of 'ei' class as training and the remaining 10% with 'ie' class as testing. Group III will use 90% of 'red soil' as training and the remaining 10% with the other classes as testing.