Final Project

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.

This exam is to be solved using R. Python is not allowed for this exam.

Problem 1

One-Class Support vector machine (OC-SVM) is a machine learning technique used in anomaly detection for detecting outliers.
Let $\mathbf{x}_i, i = 1, 2, \ldots, N$ be a sequence of $p-$variate training (or target) observations. So, the OC-SVM tries to separate the data set from the origin with maximum margin. This separation is determined by solving the following problem:

$$\begin{aligned} \underset{\mathbf{w},\rho,\xi_j}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}'\mathbf{w} - \rho + C\sum_{j=1}^{N}\xi_j, \\ \text{subject to} \quad & \mathbf{w}'\varphi(\mathbf{x}_j)) \geq \rho - \xi_j, \quad j = 1, 2, \ldots, N \\ & \xi_j \geq 0, \quad j = 1, 2, \ldots, N. \end{aligned} \tag{1}$$

where ξ_j is the slack variable, the parameter $C > 0$ is introduced to control the influence of the slack variables, and φ is a function mapping data to a higher dimensional Hilbert space.
This problem can be solved through the Lagrange dual problem, which is usually easier to solve than the primal. the dual problem is:

$$\begin{aligned}\underset{\alpha}{\text{Minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} \quad & 0 \le \alpha_j \le C \quad \forall j, \\ & \sum_{j=1}^{N} \alpha_j = 1. \end{aligned} \tag{2}$$

The optimization problem is a quadratic problem (QP) and can be solved easily using any quadratic programming software to obtain the solution $\boldsymbol{\alpha}^*$. ρ can be found by using a non-zero support vector $\mathbf{x}_s$. Alternatively, this can also be achieved using the set of all support vectors, N_s, and finding the average over all support vectors as

$$\rho = \frac{1}{N_s} \sum_{s \in S} \left(\sum_{j \in S} \alpha_j^* K(\mathbf{x}_s, \mathbf{x}_j) \right). \tag{3}$$

The hyperplane is given by

$$f(\mathbf{x}) = \sum_j \alpha_j K(\mathbf{x}, \mathbf{x}_j) - \rho. \tag{4}$$

Each new observation $\mathbf{u}$ is classified by evaluating the decision function as:

$$g(\mathbf{u}) = sgn(f(\mathbf{u})) = sgn\left(\sum_j \alpha_j K(\mathbf{u}, \mathbf{x}_j) - \rho \right). \tag{5}$$

If $g(\mathbf{u}) = 1$, then $\mathbf{u}$ is classified as normal; otherwise, it is classified as outlier.

(a) Write R codes to solve the OC-SVM problem using the 'quadprog' package from R. We will use class 1 of 'pb2.txt' as our training set.

(a) Evaluate the performance of OC-SVM on the class 2 test set.

Problem 2

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

$$\boldsymbol{\alpha}_{n-1} = \frac{1}{2}\mathbf{H}_{n-1}^{-1} \left(\mathbf{k}_{n-1} + \frac{2 - \mathbf{e}_{n-1}' \mathbf{H}_{n-1}^{-1} \mathbf{k}_{n-1}}{\mathbf{e}_{n-1}' \mathbf{H}_{n-1}^{-1} \mathbf{e}_{n-1}} \mathbf{e}_{n-1} \right). \tag{6}$$

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

In the next iteration, n, a new observation $\mathbf{x}_n$ is received and we wish to update the solution (6) 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 $\boldsymbol{\delta}_n = (k(\mathbf{x}_n, \mathbf{x}_1), \ldots, k(\mathbf{x}_n, \mathbf{x}_{n-1}))'$, 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

$$\mathbf{H}_n^{-1} = \frac{1}{\gamma_n}\begin{bmatrix} \gamma_n\mathbf{H}_{n-1}^{-1} + \mathbf{a}_n\mathbf{a}_n' & -\mathbf{a}_n \\ -\mathbf{a}_n' & 1 \end{bmatrix}. \tag{7}$$

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

$$\boldsymbol{\alpha}_n = \frac{1}{2}\mathbf{H}_n^{-1}\left(\mathbf{k}_n + \frac{2 - \mathbf{e}_n'\mathbf{H}_n^{-1}\mathbf{k}_n}{\mathbf{e}_n'\mathbf{H}_n^{-1}\mathbf{e}_n}\mathbf{e}_n\right). \tag{8}$$

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

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

b. Compare the training times (in sec) between the recursive and direct algorithms.

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

d. 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 ($\boldsymbol{\alpha}_2$, $\boldsymbol{\alpha}_3$, $\boldsymbol{\alpha}_4$, $\boldsymbol{\alpha}_5$, $\boldsymbol{\alpha}_6$, $\boldsymbol{\alpha}_7$).

e. Use your results from (c) 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. If an observation is an outlier, remove it from the training set. If it is a target, add it to the training set.

Problem 3

In this problem, we want to update LS-SVDD when we remove an observation. We will be using again "charlie.csv" data set. Assume that we are discarding the oldest observation. Let

$$\mathbf{H}_n = \begin{bmatrix} a & \mathbf{b}' \\ \mathbf{b} & \mathbf{U} \end{bmatrix},$$

and its inverse be defined as

$$\mathbf{H}_n^{-1} = \begin{bmatrix} e & \mathbf{f}' \\ \mathbf{f} & \mathbf{D} \end{bmatrix}$$

is known. Then the inverse of the reduced matrix is obtained as

$$\mathbf{U}^{-1} = \mathbf{D} - \frac{1}{e}\mathbf{f}\mathbf{f}'. \tag{9}$$

a. Starting with the first row, add sequentially the next 9 rows and use your updating codes from problem 2 to obtain $\mathbf{H}_{10}^{-1}$.

b. Next, at each future step, we add a new observation and discard the oldest observation. This is sometimes called the "Add and Forget Algorithm". Write an R function to update $\mathbf{H}^{-1}$ for the "Add and Forget Algorithm", i.e. after adding a new observation, using eq(7), and removing an old observation, using eq(9).

c. Compare the "Training accuracy (%)", "Testing accuracy (%)" and "Training Time (s)' between the recursive and direct algorithms. Note that the direct approach will be containing 11, 12, 13, ..., 20 observations at each iteration while the updating algorithm will only have 10 obervations at each iteration.

Problem 4

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

$$\boldsymbol{\alpha}_{N+K} = \frac{1}{2}\mathbf{H}_{N+K}^{-1}\left(\mathbf{k}_{N+K} + \frac{2 - \mathbf{e}'_{N+K}\mathbf{H}_{N+K}^{-1}\mathbf{k}_{N+K}}{\mathbf{e}'_{N+K}\mathbf{H}_{N+K}^{-1}\mathbf{e}_{N+K}}\mathbf{e}_{N+K}\right). \tag{10}$$

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.

$$\mathbf{H}_{N+K}^{-1} = \begin{bmatrix} \mathbf{H}_N & \mathbf{B}' \\ \mathbf{B} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} [\mathbf{H}_N - \mathbf{B}'\mathbf{D}^{-1}\mathbf{B}]^{-1} & \mathbf{H}_N^{-1}\mathbf{B}'\left[\mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}' - \mathbf{D}\right]^{-1} \\ \left[\mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}' - \mathbf{D}\right]^{-1}\mathbf{B}\mathbf{H}_N^{-1} & \left[\mathbf{D} - \mathbf{B}\mathbf{H}_N^{-1}\mathbf{B}'\right]^{-1}. \end{bmatrix} \tag{11}$$

Also,

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

and

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

a. Write an R 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 and LS-SVDD update (K = 1), and LS-SVDD and LS-SVDD block update (K = 15) to the following data sets available from the UCI Machine Learning Repository. All features for all experiments are normalized to the range $[-1, +1]$. 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.

Problem 5

Assume that there are N samples in the original training set, and there are h samples discarded in each decrement. Let

$$\mathbf{H}_N = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}' & \mathbf{D} \end{bmatrix},$$

and its inverse defined as

$$\mathbf{H}_N^{-1} = \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{U}_{12}' & \mathbf{U}_{N-h} \end{bmatrix}$$

is known. Then the inverse of the reduced matrix is obtained as

$$\mathbf{D}^{-1} = \mathbf{U}_{N-h} - \mathbf{U}_{12}'\mathbf{U}_{11}^{-1}\mathbf{U}_{12}. \tag{12}$$

a. Using the 'charlie.csv' data set, starting with the first row, add a block of $k = 4$ samples rows and use your updating codes from problem 4 to obtain $\mathbf{H}_5^{-1}$.

b. Next, at each future step, we add a chunk of $k = 5$ observations and discard the oldest $h = 3$ observations. This is sometimes called the "Block Add and Forget Algorithm". Write an R function to update $\mathbf{H}^{-1}$ for the "Block Add and Forget Algorithm", i.e. after adding k observations, using eq(11), and removing the oldest h observations, using eq(12).

c. Compare the "Training accuracy (%)", "Testing accuracy (%)" and "Training Time (s)' between the 'Block Add and Forget Algorithm' and direct algorithms. Note that the direct approach will be containing 5, 10, 15, 20 observations at each iteration while the updating algorithm will have 5, 7, 9, 11, ... obervations at each iteration.

d. Repeat question (c) but now by using the data set corresponding to your group. For this case, use $k = 15$ and $h = 10$.