Northeastern University
College of Engineering
Group member name(s): Yanming Liu
Group member UID(s): 002199402

June 30, 2022
Homewrok 3

IE 7374: Machine Learning

Summarize the Sequential Minimal Optimisation(SMO) Algorithm.

Answer to Question

This paper address mathematical details of Sequential Minimal Optimisation(SMO) algorithm and its implementation as a form of pseudo code. I will briefly introduce the optimization objective, explain the intuition behind the solution, and rephrase the pseudo code manually in the following context.

SMO targets at getting coefficients of the regularized support vector machine(SVM) optimization problem, which can be seen as maximizing quadratic function with linear constraints. The coefficients are α_i, w, b. Among them, w can be obtained once we have a knowledge of α_i. We accelerated the coefficient acquisition process with a application of simplified SMO method especially with a large data set, for only subsets of parameters combination are conducted for a local optimum.

The idea is that we select 2 optimized parameters, α_i, α_j, at the very beginning, then we iterate the coefficient acquisition process for all α, until α_i meets KKT conditions and is in tolerance range. Furthermore, he process will terminates if it does not update any parameters for a certain threshold times. Before reaching this point, we randomly pick the α_j, where j not equals to i, and calculate the following parameter b for each parameter updating optimization. Finally, the optimal $(m + 1)$ parameters for Lagrange multiplier α_i and intercept b are ready to predict the classification result for new data records.

Below is the memorial note for the Simplified SMO algorithm,

Algorithm 1: Simplified SMO

Data:

- $(x^(i), y^(i))$, $i \leftarrow 1 : m$: trainning data
- $tol \in \mathbb{R}$: tolerance:
- $stop\_number$: max of times to iterate without parameter updating
- C: regularization parameters

Result:

- $\alpha \in \mathbb{R}^m : Lagrangemultiplier$
- $b \in \mathbb{R} : intercept$

```
Initialize a_i ← 0, i ← 1 : m and b.
Initialize stop ← 0.
while stop < stop_number do
    flag_alpha ← 0 ;                /* alpha change status, if update 1, otherwise 0 */
    for i ← 1 : m do
        Calculate E_i ← f(x^(i) − y^(i))
        if ((y^(i)E_i < −tol && a_i < C)||(y^(i)E_i > tol && a_i > 0)) then
            Select j ≠ i randomly
            Cal. E_j = f(x^(j) − y^(j))
            Record a_j^old = a_j, a_i^old = a_i Cal L, H if L == H then
                continue
            end
            if η > 0 then
                continue
            end
            Ensure a_j is in the range [L, H]
            if |a_j − a_j^old| < tol then
                continue
            end
            Cal. a_i based on a_j
            Cal. b
            Set the change status of α as 1, flag_alpha ← 1
        end
    end
    if flag_alpha then
        stop ← 0
    else
        stop += 1
    end
end
```