1 Knapsack Problem

After the previous formalization the optimization problem is the following

$$\begin{aligned} \max_{\beta_i} & -\frac{1}{2}\sum_i\sum_j \beta_i\beta_j K(x_i,x_j) \\ & -\varepsilon\sum_i|\beta_i| \\ & +\sum_i y_i\beta_i \\ & \textit{With the constraints} \qquad \begin{cases}\sum_i \beta_i = 0 \\ \beta_i \in [-C,\ C]\end{cases} \end{aligned} \tag{1}$$

The problem of maximization can be substituted by a minimization obtaining a formulation that recalls the one of a Convex Separable Knapsack problem.

$$\begin{aligned} \min_{\beta_i} \quad & \frac{1}{2}\sum_i\sum_j \beta_i\beta_j K(x_i,x_j) \\ & +\varepsilon\sum_i|\beta_i| \\ & -\sum_i y_i\beta_i \\ & \textit{With the constraints} \qquad \begin{cases}\sum_i \beta_i = 0 \\ \beta_i \in [-C,\ C]\end{cases} \end{aligned} \tag{2}$$

We define the Lagrangian Relaxation:

$$\begin{aligned} \mathcal{L}(\mu) = \quad & \frac{1}{2}\sum_i\sum_j \beta_i\beta_j K(x_i,x_j) \\ & +\varepsilon\sum_i|\beta_i| \\ & -\sum_i y_i\beta_i \\ & +\mu(\sum_i \beta_i - 0) \\ & \textit{With the constraints} \qquad \beta_i \in [-C,\ C] \end{aligned} \tag{3}$$

The objective is to minimize β and to do so we calculate $\frac{\partial \mathcal{L}}{\partial \beta_i} = 0$

$$\frac{\partial \mathcal{L}}{\partial \beta_i} \longrightarrow \beta_i K(x_i, x_i) + \varepsilon sgn(\beta_i) - y_i + \mu \tag{4}$$

At this point we can define the two breakpoints for each β_i, which represent the upper and lower bound of the Lagrangian multiplier μ.
Remark: upper and lower bound for each β_i is the same (-C; C).

$$\begin{aligned} &\mu_i^l = -(-C * K(x_i, x_i)) - \varepsilon sgn(-C) + y_i \\ &\mu_i^u = -(C * K(x_i, x_i)) - \varepsilon sgn(C) + y_i \\ &where \quad \forall_i \quad \mu_i^u < \mu_i^l \end{aligned} \tag{5}$$

Each β_i can be defined wrt to μ

$$\beta_i(\mu) = \begin{cases} C & if\ \mu < \mu_i^u \\ \frac{y_i - \varepsilon sgn(\beta_i) - \mu}{K(x_i, x_i)} & if\ \mu_i^u \leq \mu \leq \mu_i^l \\ -C & if\ \mu > \mu_i^l \end{cases} \tag{6}$$

At this point using for example *Algorithm 3.1* of [1], we can compute the value of μ^* that implies how to find the correct values of β_i that satisfy the constraints.

2 How to use it

Now that we have given a formulation for the resolution of the Knapsack problem, the focus shifts towards how we can use it in our minimization problem that needs to be solved using a deflected subgradient method.
At each step we compute the values of β and we want to make sure that their values satify the linear and box constraints imposed by the problem.
Here a pseudocode that summarize the steps.

Algorithm 1: Compute β

while *optima criteria are not satisfied* **do**
  $v \longleftarrow f(\beta)$
  $g \longleftarrow \nabla f(\beta)$
  `check if a better solution was found`
  $d \longleftarrow \gamma g + (1-\gamma)dprev$
  $stepsize \longleftarrow \frac{\psi(v - f_{ref} + \delta)}{\|d\|^2}$
  $\beta \longleftarrow \beta - stepsize \cdot d$
  `project` β `by solving the KP`
end

A probably trivial point, related to the computation of the direction, makes our approach incorrect. As we can see the current direction depends on the previous direction, but if we take a closer look at the execution flow, it does not.
What algorithm 1 does at every iteration is a lot of calculations to find a direction, a stepsize and therefore a new β which then gets projected in the feasible region to satisfy the constraints. The projection, as shown in *algorithm 3.1* of [1], is based uniquely on the use of breakpoints, created using the upper and lower bound of the β_i variable. Therefore the actual computed values of β_i before the projection are not taken into consideration. This leads to a deterministic computation of β_i values which depends only on the solution of the knapsack problem, giving zero weight to the deflection

applied before. We are stunned (and not in a good way): we have a clearer understanding of how deflection works and also of how a knapsack problem can be solved. We find ourselves in a *saddle point* trying to figure out how to merge the two.

3 Reference

1. Breakpoint searching algorithms for the continuous quadratic knapsack problem - Krzysztof C. Kiwiel