Support Vector Regression using Deflected Subgradient Methods

Elia Piccoli
Nicola Gugole

March 29, 2022

A project presented for the
Computational Mathematics for Learning and Data Analysis
course

University of Pisa
Artificial Intelligence
A.Y. 2020/2021

Contents

Abstract

Project aim is developing the implementation of a model which follows an SVR-type approach including various different kernels. The implementation uses as optimization algorithm a dual approach with appropriate choices of the constraints to be dualized, where the Lagrangian Dual is solved by an algorithm of the class of deflected subgradient methods.

1 Introduction

SVR objective is predicting a uni-dimensional real-valued output y through the use of an *objective function* built by optimization using an ε-insensitive loss function. Another fundamental aspect about SVR is keeping the function *as flat as possible* through the tuning of a C parameter in order to avoid overfitting and generating a correct trade-off between accuracy and generalization.
The resulting function can be generically described as:

$$f(x) = wx + b \tag{1}$$

Keeping the above function *as flat as possible* is equivalent to an optimization problem formulated as having minimum $\|w\|$, or, for a more convenient mathematical derivation, minimum $\|w\|^2$, not changing the semantics of the problem.
This brings us to a convex minimization problem, which will be called *primal problem*:

$$\min_{w,\xi_i,\xi_i^*} \frac{1}{2}\|w\|^2 + C\sum_i(\xi_i + \xi_i^*) \tag{2}$$

Where ξ and ξ^* are called *slack variables*, used in conjunction with C to create a *regularization factor* and consequently a *penalty measure* to elements which are not part of the ε-tube. Slack variables allow the definition of constraints applicable to (2):

$$y_i - w^T\phi(x_i) - b \leq \varepsilon + \xi_i, \tag{3a}$$
$$b + w^T\phi(x_i) - y_i \leq \varepsilon + \xi_i, \tag{3b}$$
$$\xi_i, \xi_i^* \geq 0 \tag{3c}$$

x_i input, y_i output

2 Dual Representation

As expressed in the abstract, the implementation will follow a dual approach, which in SVR models is preferred due to the applicability and efficiency of the use of *kernels*. *Dual problem* formulation can be achieved defining the *Lagrangian* function:

$$\begin{aligned}\mathcal{L}(\alpha, \alpha^*, \mu, \mu^*) = &\frac{1}{2}\left\|w\right\|^2 \\ &+ C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ &+ \sum_{i=1}^{m}(\alpha_i(y_i - w^T\phi(x_i) - b - \varepsilon - \xi_i)) \\ &+ \sum_{i=1}^{m}(\alpha_i^*(w^T\phi(x_i) + b - y_i - \varepsilon - \xi_i^*)) \\ &- \sum_{i=1}^{m}(\mu_i\xi_i + \mu_i^*\xi_i^*)\end{aligned} \tag{4}$$

From which the following optimization problem can be obtained (full derivation shown in 8):

$$\begin{aligned}\max_{\alpha_i, \alpha_i^*} &- \frac{1}{2}\sum_i\sum_j(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K(x_i, x_j) \\ &- \varepsilon\sum_i(\alpha_i + \alpha_i^*) \\ &+ \sum_i y_i(\alpha_i - \alpha_i^*)\end{aligned} \tag{5}$$

With constraints:

$$\begin{aligned}&\forall i\ \alpha_i, \alpha_i^* \geq 0 && (KKT\ condition) && \text{(6a)}\\ &\forall i\ \alpha_i, \alpha_i^* \in [0, C] && (from\ derivation) && \text{(6b)}\\ &\forall i\ \sum(\alpha_i - \alpha_i^*) = 0 && (from\ derivation) && \text{(6c)}\\ &\forall i\ \alpha_i\alpha_i^* = 0 && (from\ model\ construction) && \text{(6d)}\end{aligned}$$

At this point a reformulation of (5) is necessary to follow the task objective, which is solving the *Lagrangian Dual* maximization with a subgradient method, therefore requiring a *non-differentiable function.* Such function is achievable with a simple variable substitution:

$$\begin{aligned}\beta_i &\longleftarrow (\alpha_i - \alpha_i^*)\\ |\beta_i| &\longleftarrow (\alpha_i + \alpha_i^*)\end{aligned}$$

Bringing the definitive dual problem definition:

$$\begin{aligned}\max_{\beta_i} &- \frac{1}{2}\sum_i \sum_j \beta_i \beta_j K(x_i, x_j)\\ &- \varepsilon \sum_i |\beta_i|\\ &+ \sum_i y_i \beta_i\\ &\qquad With\ the\ constraints \qquad \begin{cases}\sum_i \beta_i = 0\\ \beta_i \in [-C,\ C]\end{cases}\end{aligned} \tag{7}$$

It is important to notice how the above formulation defines a convex non-differentiable problem which still maintains the *strong duality* propriety, assuring that the optimal solution of the dual problem (*computationally less intensive*) coincides with the one of the primal problem.

3 Deflected Subgradient Algorithm

In order to solve the problem defined in (7) we need to use an algorithm among the family of *subgradients methods*. The approach that we are going to analyze is a *Constrained Deflected Subgradient Method* using *Target Value Stepsize* with a *Non-Vanishing Threshold*.
Let's briefly analyze all the elements that characterize the approach:

- *Constrained*: as we can see in (7) the dual problem variable β is subject to linear and box constraints that the algorithm must respect at each step.

- *Deflected*: at each step of the algorithm the direction will be a convex combination wrt to the previous direction and the current subgradient.

$$d_k = \alpha g_k + (1-\alpha) d_{k-1} \qquad \alpha \in [0,\ 1] \tag{8}$$

- *Target Value Stepsize* with a *Non-Vanishing Threshold*: since f^* is unknown, we will use a *target level* approach where f^* is approximated by an estimate that is updated as the algorithm proceeds. The estimate is defined wrt two values: f^k_{ref} which is the *reference value*, and δ_k which is the *threshold*. This two values will be used to approximate f^* in the formulation of the stepsize. In particular the stepsize has to follow a constraint between the α and ψ parameter (*stepsize restriction*) to assure convergence.

$$0 \leq \nu_k = \psi_k \frac{f_k - f^k_{ref} + \delta_k}{\|d_k\|^2} \qquad 0 \leq \psi_k \leq \alpha_k \leq 1 \tag{9}$$

  As far as concerns the *non-vanishing threshold*, it will assure that at each step of the algorithm δ will always be grater than zero.

$$\forall_k \quad \delta_k > 0 \tag{10}$$

Here is described a general algorithm for solving (7), which can be easily transformed into a *minimization problem.*

Algorithm 1: Deflected Subgradient Algorithm
variable x stands for β, $\delta_{reset} \approx 0\ (> 0)$, $\rho \in [0,\ 1]$

```
begin
    $x_{ref} \longleftarrow x$
    $f_{ref} \longleftarrow \infty$
    $\delta \longleftarrow 0$
    $d_{prev} \longleftarrow 0$
    while $true$ do
        $v \longleftarrow \frac{1}{2}x'Kx + \varepsilon|x| - yx$
        $g \longleftarrow Kx + \varepsilon sgn(x) - y$
        Check if in $stopped/optimal$ condition
        // reset $\delta$ if $v$ is good or decrease it otherwise
        if $v \leq f_{ref} - \delta$ then
            $\delta \longleftarrow \delta_{reset} \cdot \max v, 1$
        else
            $\delta \longleftarrow \max(\delta\rho, eps \cdot \max(|\min(v, f_{ref})|, 1))$
        end
        // update $f_{ref}$ and $x_{ref}$ if needed
        if $v < f_{ref}$ then
            $f_{ref} \longleftarrow v$
            $x_{ref} \longleftarrow x$
        end
        $d \longleftarrow \alpha g + (1 - \alpha)d_{prev}$
        $d \longleftarrow Project(d)$                         // project $d$ (here)
        $d_{prev} \longleftarrow d$
        $\lambda \longleftarrow v - f_{ref} + \delta$
        $\nu \longleftarrow \frac{\psi \cdot \lambda}{\|d\|^2}$               // stepsize-restricted $\rightarrow \psi \leq \alpha$
        $x \longleftarrow x - \nu \cdot d$
        $x \longleftarrow Project(x)$                         // project $x$ (here)
    end
end
```

The projections required in Algorithm 1 are the ones presented in Section 4. The two projections are *easy* to perform, allowing the convergence of the *Deflected Subgradient Algorithm* as stated in [see 2, Theorem 3.6].

> *Theorem 3.6. Under conditions (2.13) and (3.5), the algorithm employing the level stepsize (3.19) with threshold condition (3.23) attains either:*
>
> $$f_{ref}^{\infty} = -\infty = f^*$$
> $$f_{ref}^{\infty} \le f^* + \xi\sigma^* + \delta^*, \ \textit{where}\ 0 \le \xi = \max\{1 - \delta^*\Gamma/2\sigma^*, 0\} < 1$$

Which in the case of a convex function, as (7), leads to the second possibility. The quoted *level stepsize* is exactly (9) and the *threshold condition* is the *non-vanishing threshold* (10).
The theorem has two conditions to ensure the convergence (note that in our notation $v_{k+1} = d_{prev}$):

- [2, Cond 2.13]

  $$\tilde{d_k} = Deflected(d_k), \qquad \hat{d_k} = Projected(d_k)$$
  $$\textit{Condition (2.12) holds if} \qquad d_k = \tilde{d_k} \implies v_{k+1} = \tilde{d_k}$$

  The above condition aims at assuring the satisfaction of (2.12):

  $$\langle d_k, x - x_k \rangle \le \langle v_{k+1}, x - x_k \rangle$$

  which in our case is correct since both $d_k = \hat{d_k}$ and $v_{k+1} = \hat{d_k}$. [see *Deflected Subgradient Algorithm*]

- [2, Cond 3.5]

  $$\lambda_k \ge 0 \implies \alpha_k \ge \psi_k \ge \psi^* > 0$$
  $$\lambda_k < 0 \implies \alpha_k = 0 (\implies \psi_k = 0)$$

  Such a condition is satisfied since at each iteration λ is always greater or equal to zero because of the algorithm structure and α_k is assured to maintain the correct ordering wrt ψ_k since for the current version they are constant. [see *Deflected Subgradient Algorithm*].

In conclusion, the convergence of the algorithm is assured by the satisfaction of the requirements. Expected convergence rate is at best the convergence rate of a SM using *Polyak stepsize*. This is derived from the fact that the proposed algorithm is a constrained approximation of Polyak using *Target Level*, suggesting a best convergence of $\mathcal{O}(\frac{1}{\epsilon^2})$
[as stated for *Polyak stepsize: efficiency* in 5, Slide 41, "*Good (bad) news:* $\mathcal{O}(\frac{1}{\epsilon^2})$ *optimal for nondifferentiable f*"].

4 Projection Algorithms

In this section the focus will be on how the two projection problems are solved.

The first projection which will be analyzed is the *direction projection* ensuring *box constraints*. This projection is pretty *easy* to achieve and can be performed *linearly* by zeroing the direction components which are leading out of the feasible area. The process is linear since it implies passing through all the direction dimensions only once.

Algorithm 2: Project Direction
(d is direction, x is current point, $\epsilon \approx 0$)

```
begin
    $\forall_i \; x_i \in [-C, C]$
    for $i \leftarrow 0$ to $size(d)$ do
        if $(-C - x_i < \epsilon \; and \; d_i < 0) \; or \; (C - x_i < \epsilon \; and \; d_i > 0)$ then
            $d_i \longleftarrow 0$
```

Convex Separable Knapsack Problem Algorithm. The constraints of the projection put it in the category of *Knapsack Problems*, which for convex and separable problems (as is (11)) a complexity of $\mathcal{O}(n \cdot log(n))$ can be promptly achieved, as stated in [3] exploiting the **Breakpoint Searching Algorithm** and its variants. In particular the following paragraphs discuss the solution of such a problem using the easiest algorithm discussed in [1]. Starting from the projection formulation.

$$\min_{\beta_{proj}} \quad \frac{1}{2}\left\|\beta - \beta_{proj}\right\|^2$$
$$With\ the\ constraints \qquad \begin{cases} \sum_i \beta^i_{proj} = 0 \\ \beta^i_{proj} \in [-C,\ C] \end{cases} \tag{11}$$

Which by Lagrangian Relaxation leads to:

$$\mathcal{L} = \min_{\beta_{proj}} \quad \frac{1}{2}\left\|\beta - \beta_{proj}\right\|^2 - \mu \sum \beta^i_{proj}$$
$$With\ the\ constraints \qquad \left\{ \beta^i_{proj} \in [-C,\ C] \right. \tag{12}$$

This allows a useful elaboration of μ and β^i_{proj} by analyzing the derivative.

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \beta^i_{proj}} &= -(\beta_i - \beta^i_{proj}) + \mu = 0 \\ \implies \qquad & \mu = \beta_i - \beta^i_{proj} \\ & \beta^i_{proj} = \beta_i - \mu \end{aligned} \tag{13}$$

In order to find the optimal value for μ we now define some elements that will be computed each iteration of Algorithm 1. These are needed in order to initialize all the elements required for the *Breakpoint Search Algorithm* (Algorithm 3). We can consider each component independently given the *separable* structure of the problem.

- *Upper* and *lower* bound of μ: for each component we will compute the maximum and minimum value of μ_i assigning to β^i_{proj} the two extreme values -C/C in (13).
$$\begin{aligned} \forall_i \quad \mu^u_i = \beta_i - C \\ \forall_i \quad \mu^l_i = \beta_i + C \end{aligned} \tag{14}$$

- Definition of β^i_{proj} wrt μ: a piecewise linear and non-increasing function based on (13) and fundamental for checking for early algorithm termination. Also once the algorithm terminates we can compute the correct value for each β^i_{proj} given the value of μ^* .

$$\beta^i_{proj}(\mu) = \begin{cases} C & if\ \mu < \mu^u_i \\ \beta_i - \mu & if\ \mu^u_i \leq \mu \leq \mu^l_i \\ -C & if\ \mu > \mu^l_i \end{cases} \tag{15}$$

- h: we define h to be the function representing the linear constraint over the variables. This function is also a piecewise linear non-increasing function given the nature of its summation components. It will be evaluated in the algorithm to check if μ^* was found; otherwise it will work as oracle to guide the restriction of the set of possible values of μ.

$$h(\mu) = \sum_i \beta^i_{proj}(\mu) \tag{16}$$

- M: set of all the possible values that μ can assume. Is initialized as the union of all breakpoints for each β_{proj}. At each step of Algorithm 3 M is reduced, removing all values of μ that for sure won't satisfy the linear constraint.

$$M_0 = \mu^l_i \cup \mu^u_i \qquad i = 1 : size(\beta_{proj}) \tag{17}$$

- μ_L and μ_U: these two values will represent the current estimate of the optimal upper/lower value of μ respectively. In Algorithm 3, μ_L and μ_U will be initialized to $+\infty$ and $-\infty$ respectively. At each iteration one of the two value will be reassigned in order to decrease the range of possible values of μ. An interesting observation derivable from the formulation of the algorithm is: $\{\mu^i_L\}$ will be a sequence of *nondecreasing underestimates* of μ^*_L, and $\{\mu^i_U\}$ will be a sequence of *nonincreasing overestimates* of μ^*_U.
  Joining together the definition of the previous point (17) and the current one we can define the optimal set of μ and the optimal upper/lower bounds (as stated in[1]).

$$M^* = [\mu^*_L,\ \mu^*_U] \quad where \quad \begin{aligned} \mu^*_L &= \inf\{\mu : h(\mu) = 0\} \\ \mu^*_U &= \sup\{\mu : h(\mu) = 0\} \end{aligned} \tag{18}$$

Algorithm 3: Convex Separable Knapsack Problem Algorithm

```
begin
    while $M \neq \emptyset$ do
        choose $\hat{\mu}$ using median of medians approach over M
        compute $h(\hat{\mu})$
        if $h(\hat{\mu}) = 0$ then
            $\mu^* = \hat{\mu}$
            return $\mu^*$
        else
            if $h(\hat{\mu}) > 0$ then
                $\mu_L = \hat{\mu}$
                M $= \{\mu \in M : \hat{\mu} < \mu\}$
            else
                $\mu_U = \hat{\mu}$
                M $= \{\mu \in M : \hat{\mu} > \mu\}$
    $\mu^* = \mu_L - h(\mu_L)\frac{\mu_U - \mu_L}{h(\mu_U) - h(\mu_L)}$
    return $\mu^*$
```

The algorithm is quite simple. At each iteration a $\hat{\mu}$ is chosen from M and the stopping condition is checked, returning $\hat{\mu}$ in the positive case. If we are not in stopping condition then M is restricted appropriately. Eventually the algorithm terminates by either finding μ^* or by emptying M.
In the second case (line 15) the emptiness of M stands for having found the best possible approximation of μ_L^* and μ_U^* and no other breakpoints are left in the middle. Therefore the range $[\mu_L,\ \mu_U]$ is a segment which formulation we can get and exploit (see Appendix B).

The convergence of Algorithm 3 is strictly dependent on the choosing approach of $\hat{\mu}$. In the proposed pseudo-implementation the *median of medians* algorithm is exploited, giving a double benefit. The algorithm allows for a linear time choice of $\hat{\mu}$ and an halving of M per iteration. In conclusion this leads to an $\mathcal{O}(n)$ cost per iteration (*median of medians*) and an $\mathcal{O}(log(n))$ number of iterations (halving of M), for an overall $\mathcal{O}(n \cdot log(n))$.

5 Experiments

The following subsections show experiments and results of the SVR implementation developed and proposed for this project. **Section 5.1** and **Section 5.2** present a complex dataset such as the one proposed for the Machine Learning course in the academic year 2020/2021. A deeper introduction to the measurements, expectations and quantitative results are shown later.

All the training times refer to running on a laptop with an *Intel(R) Core(TM) i9-9880H* CPU and 32 GB of memory - all test are run in parallel at the same time. The Python source code is available here: `http://github.com/EliaPiccoli/ML-CM-Project`

5.1 ML Cup Results

Following experiments are about testing the optimization algorithm with different parameters and analyze its results. The non trivial dataset of choice is the set of data proposed by professor A. Micheli for the *Machine Learning course project (AY 2020/2021)* at University of Pisa.

5.2 Dataset Presentation

Dataset of interest is composed of two different files, with the first one including a labeled dataset, intended for training and validation of an hypothetical neural network, and the second one including a blind dataset, used in the *Machine Learning project* as non-cheatable test. Going more in detail, the dataset is divided into an input vector (composed of **10** features) and a **2**-dimensional output vector. The presence of a multidimensional output calls for the need of a multiregressor, composed of one SVR per output dimension because of the SVR model nature. In the following analysis both dimensions will be taken into consideration in order to study if the behaviour of the optimization process is stable.

5.3 Measurements and Expectations

The results obtained and shown in the next subsection are both in a tabular configuration, where parameters and quantitative measurements are reported, and in a graphical configuration, which takes advantage of the *rate* *of convergence* as well as the *logarithmic residual error*. Both can be derived from the model fitting:

- *Rate of convergence:* using f_* and the definition reported also in lectures [see 4, Slide 6]:

  $$\lim_{i \to +\infty}(f(x^{i+1}) - f_*)/(f(x^i) - f_*)^p = R$$

  Since the model proposed follows a subgradient approach with the convergence speed of $\mathcal{O}(\frac{1}{\epsilon^2})$ (as reported in Section 3), the theoretical expectations suggest that with $p = 1$ the resulting convergence rate should be $R = 1$, a sublinear convergence rate.

- *Logarithmic residual error:* this measure gives a hint of how fast the model is reducing its error with respect to the optimal function value, following the formula:

  $$log(|f(x^i) - f_*|/|f_*|)$$

The implementation at this moment allows for the fitting to stop in three different possible scenarios:

- *optimal*: if the gradient norm gets extremely near to a stall then we have reached a stable point (we set $1e^{-10}$ as reference value)
- *acceptable*: a ground truth $f*$ value might be set together with an error with respect to it. When the fitted function value `f_ref` goes below the error threshold then this exit scenario is reached. We have set the error to $1e^{-3}$.
- *stopped*: when the number of iterations exceeds the number defined by the `max_iter` parameter

5.4 Results

The results we propose are focused on fixating the model parameters (thanks to all previous tests and results proposed in previous versions of the report) while searching for the best performing model optimization-wise through many different *algorithmic parameters* configurations.

5.4.1 1st Dimension Linear Kernel

As already mentioned in Section 5.2 the dataset has **two output dimensions**. In the following analysis only the *first dimension* is taken into consideration, focusing on observations on the effect of a change in hyperparameters and the expected convergence behaviour.

Going more into details we set the model parameters to **kernel** *linear* with both the **C** and **eps** parameters equal to 1.

Due to the implementation poor performances in time and memory occupation we restricted the search to few selected configurations of algorithmic parameters, shown in **Table 1**.

Model	*alpha*	*psi*	*eps*	*rho*	*deltares*
1	0.3	0.075	0.01	0.6	0.001
2	0.3	0.15	0.1	0.8	0.0001
3	0.3	0.15	0.01	0.9	0.00001
4	0.3	0.3	0.005	0.99	0.000001
5	0.5	0.075	0.1	0.6	0.00001
6	0.5	0.15	0.1	0.8	0.00005
7	0.5	0.3	0.1	0.9	0.0001
8	0.5	0.5	0.1	0.99	0.001
9	0.5	0.075	0.01	0.6	0.00001
10	0.5	0.15	0.01	0.8	0.00005
11	0.5	0.3	0.01	0.9	0.0001
12	0.5	0.5	0.01	0.99	0.001
13	0.5	0.15	0.005	0.6	0.00001
14	0.5	0.3	0.005	0.99	0.001
15	0.7	0.15	0.1	0.6	0.00001
16	0.7	0.3	0.1	0.8	0.00005
17	0.7	0.5	0.1	0.9	0.0001
18	0.7	0.7	0.1	0.99	0.001
19	0.7	0.15	0.01	0.6	0.00001
20	0.7	0.3	0.01	0.8	0.00005
21	0.7	0.5	0.01	0.9	0.0001
22	0.7	0.7	0.01	0.99	0.001
23	0.7	0.15	0.005	0.6	0.00001
24	0.7	0.7	0.005	0.99	0.001

Table 1: Configuration attempted varying Algorithmic Parameters

Since every one of the proposed models requires approximately one hour to fit the data (empirically we noticed that the *stopped* scenario is the most frequent in this study case, given a high number of `max_iter`), we reasoned the configurations in order to achieve the highest variability in 24 models.

We varied therefore various parameters of algorithm 1. Starting from the `alpha` parameter we went for three possible scenarios, varying from models highly dependent on the old gradient (0.3), passing through a balanced combination (0.5) and eventually trying a high dependence on the new gradient information (0.7). For each one of the `alpha` values we varied also the `psi`, coefficient important for the stepsize magnitude, varying it in a range from small stepsizes ($\frac{1}{4}$ of `alpha`) to the maximum stepsize possible (with `psi` equal to `alpha`). One can appreciate the scarcer configuration with `alpha` 0.3, this is due to the empirical understanding of how configurations with this setting will underperform with respect to others. For the sake of testing we nevertheless carried on tests including these parameter settings.

We certainly varied also the other algorithmic parameters:

- the `rho` parameter takes responsibility in how fast the accepted function reference value `f_ref` decreases for defining a new valid `f_ref` (before restarting the `f_ref` thanks to the `deltares` parameter). We varied the `rho` parameter from 0.6 to 0.99.
- the `deltares` parameter, responsible for the rescaling of the `delta` once a new *good* minimum is found. This parameter was also attempted in different combos together with `rho`.
- the `eps` parameter also takes similar responsibility as the `rho` parameter and is tipically set to values lower than 1 (to allow for a tighter function reference value). We attempted values in the set 0.1, 0.01, 0.005.

After running the grid search over the aforementioned configurations we could finally visualize the models behaviours by plotting the metrics proposed in 5.3. In order to correctly plot the behaviours of the various models a value for f_* was needed for the said metrics.

Choosing f_best: Since we could not empirically obtain the true optimum value through our models, we opted for obtaining an f_{best} given the models we

Model	*best func value*
17	-3447.6904
7	-3447.6568
16	-3447.6534
6	-3447.5219
15	-3447.5202

Table 2: Best resulting *linear* models

have. To obtain such value we decided to first select the top 5 models, chosen by minimum function values, which configuration number and best function values are shown in **Table 2**. After getting the 5 best models, we ran the best performing model (conf 17) for 50k iterations, to obtain an f_{best} which is rapidly obtained by the best configuration but that is also reachable in an understandable amount of time by the other configurations. f_{best} obtained with conf 17 using 50k iterations resulted to be **-3924.1776**.

Discussing best models: The best models show some similarities and recurrences which we can discuss, hinting at which algorithmic parameter configurations give best performances. In particular, all 5 best models have an `alpha` value which either halves the contribution to the next direction between old and new gradient or gives more weight to the new gradient, always keeping a `psi` value lower than `alpha`. Another interesting fact is that all best models show to prefer the highest `eps` value and the top 2 show to favour a high `rho` as well. This last fact can be interpreted as the optimization getting better results when keeping a *more optimistic* Target Value. Having high values for these two parameters leads to a slower decrease of the `delta` parameter, keeping the Target Value from increasing rapidly towards a faster update of `f_ref` and a consequent `delta` reset.

Results on *convergence rate* are eventually shown in **Figure 1** while results on *logarithmic residual error* are shown in **Figure 2**. Convergence rate results confirm the behaviour theoretically expected by a subgradient method such as the proposed one. When looking instead at **Figure 2** (line 17 is alone, line 16 and 7 are just-apposed and the same goes for line 6 and 15) one can notice how the best 5 models show extremely similar behaviours, some slower than others. This difference in pace can be explained by looking at the parametric diversity in the models. The fact that catches the eye is the relevance of the `psi` parameter, as its difference in value fits well the behaviour of the models. In fact model 17 (which has a `psi` of 0.5) converges in almost half iterations with respect to models 16/7 (which have a `psi` of 0.3) and these two models converge in half iterations with respect to models 15/6 (which have a `psi` of 0.15).

Another eye-catching graphical fact is that all models show some inconsistent behaviour with what is expected, having all in fact a graph ending which makes some eyebrows raise.

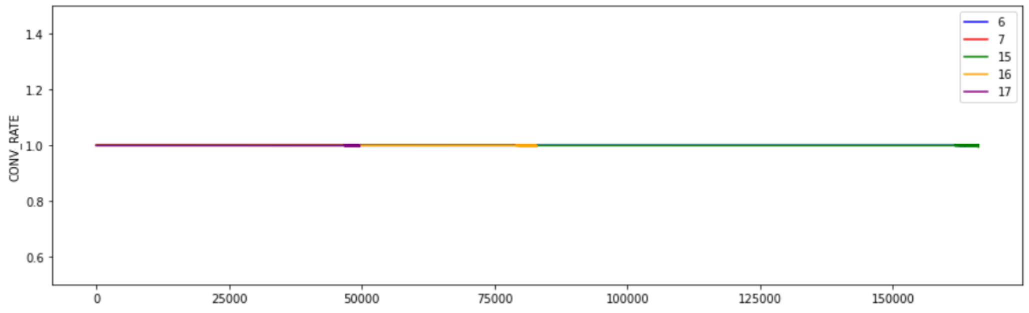

Figure 1: 1st Linear - Convergence rate of top 5 performing.

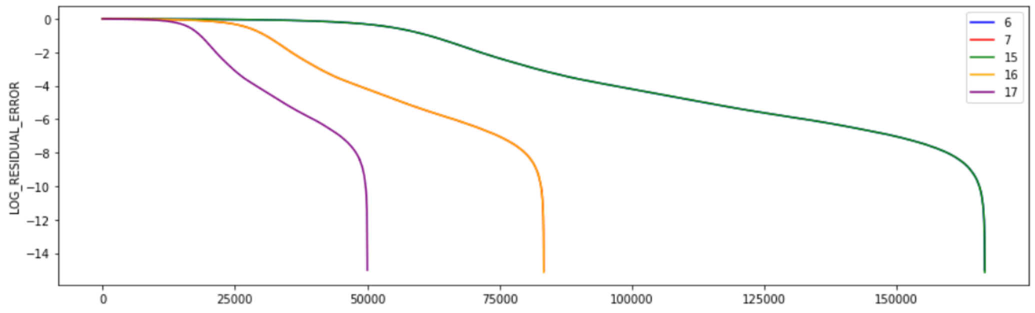

Figure 2: 1st Linear - Logarithmic residual error of top 5 performing.

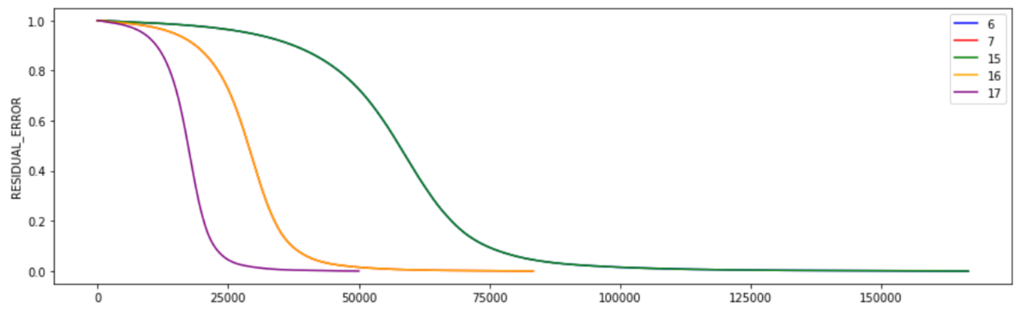

Figure 3: 1st Linear - Residual error of top 5 performing.

The models optimization curves do in fact start with a slow optimization, keeping a subgradient behaviour until the very last iterations before reaching *acceptable* convergence (described in **5.3**), where there appears to be what seems a superlinearity. We noticed this behaviour and we want to give a motivation for it.

We started motivating this behaviour by looking at the logarithmic residual error scale as we approach convergence. One can in fact appreciate how even before the suspicious *superlinearity* the logarithmic residual error gets already around 10^{-8}. When we saw the aforementioned weird behaviour we started monitoring how the `f_ref` value neared the f_{best} in these last iterations, not seeing an incredible speedup in the decreasing of `f_ref`, but simply noticing a residual error which continued to near the optimum in an extremely slow manner, actually being almost a flat line as shown in **Figure 3**.

To sum it up, this leads us to the conclusion that the optimization graph does not show a true *superlinearity*. The noticeable artifact is in our opinion present because the logarithmic residual error gets scales of degrees nearer to the desired value once the convergence gets incredibly near to f_{best}, seemingly accelerating the convergence even though the actual residual error does not show a speed up in converging. The artifact highlights how choosing an f_{best} in the manner we decided to choose it (again motivated by having a slow and heavy implementation, therefore requiring a cut-off towards models that need an understandable amount of time to train) leads to such an ending graph situation, which without motivation can certainly pave the way to some confusion.

5.4.2 2nd Dimension Linear Kernel

In this section and the following ones, we fixed the *model configuration* and used as range of possible configurations of *algorithmic parameters* the five best ones analyzed in the previous part. We limited the number of experiments and we were able to provide consideration about the stability of the optimization process and results in different scenarios wtih respect to the one analyzed in **Section 5.4.1**.

In particular in this setting, we analyse the *second dimension*, setting the model parameters once again to **kernel** *linear* with both **C** and **eps** equal to 1. In order to achieve the f_{best} value we used the same approach discussed before. We ran configuration 17 for 50k iterations, resulting in a f_{best} value of **-3447.4201**.

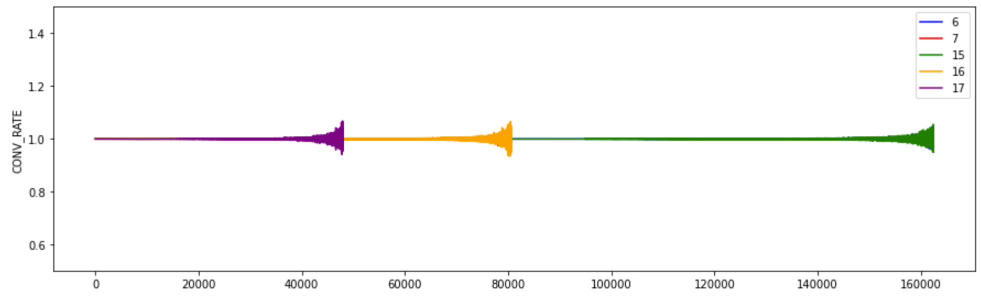

Figure 4: 2nd Linear - Convergence rate of top 5 performing.

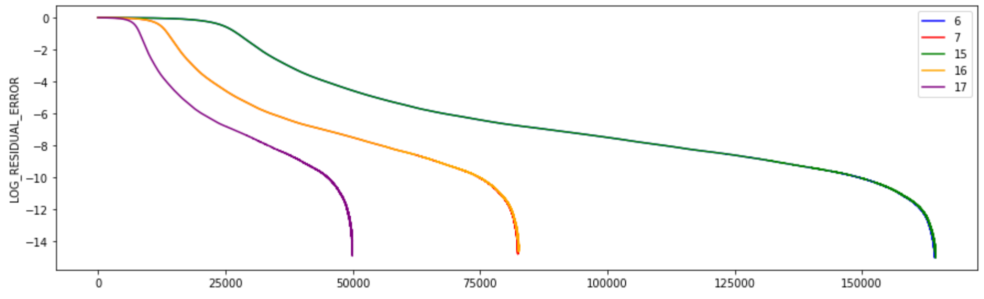

Figure 5: 2nd Linear - Logarithmic residual error of top 5 performing.

Taking a look at the results, the logarithmic residual error graph shown in **Figure 5** goes for a similar comportment as **Figure 2**, showing coherence in a subgradient behaviour.

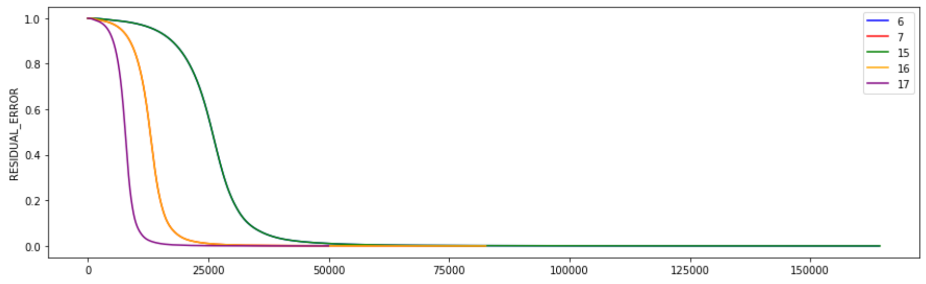

Figure 6: 2nd Linear - Residual error of top 5 performing.

More peculiar is the convergence rate shown in **Figure 4**: the rate is coherent with theoretical results until very last iterations, where it starts to have small but increasing oscillations around the value 1. To motivate this we should take a look at the equation for *convergence rate* shown in **Section 5.3**. The figures we show for convergence rate are created by fixing the p parameter to 1 since we want to check the theoretical coherence, while the f_* value is set to the aforementioned f_{best}. The motivation we can give for this scenario's peculiar behaviour regards therefore the numerical increasing instability that once again arises when the function values $f(x^i)$ get nearer and nearer to f_{best}. In particular, in this scenario the function values show some oscillating behaviour in last iterations, meaning that $f(x^i)$ and $f(x^{i+1})$ frequently are wobbling around f_{best} but at the same time are getting closer and closer to that value (although oscillating), creating such a graph finale.

5.4.3 1st Dimension Poly Kernel

We then tried to change the kernel used by the SVR from *linear* to *polynomial*. In this scenario we expect the algorithm's *convergence rate* and *log residual error* to behave similarly to the previous experiments.

The model parameters are obtained by a grid search among different configurations in order to find the best hyper parameters. The SVR paramters are **kernel** *polynomial*, with the following kernel parameters: **degree** 3, **gamma** 0.075 and **coefficient** 0.28, and **C** equal to 10.

To collect the f_{best} value we ran the model for 50k iterations using as parameters configuration 17, in this case f_{best} is equal to **-27823.29**.

The first results we achieved were not what we expected, or better not

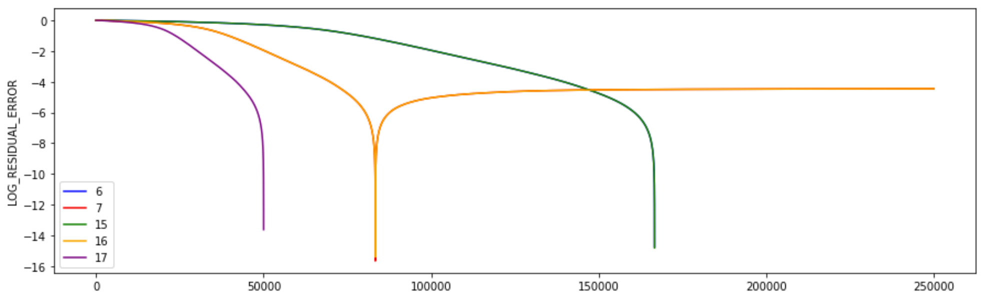

Figure 7: 1st Poly (attempt)- Logarithmic residual error of top 5 performing.

exactly. The strangeness (and also the fastest motivation) is portrayed at best by the logarithmic residual error shown in **Figure 7**. In the figure it is in fact noticeable how the *orange plot* (models 16 and 7) move extremely near to f_{best} before *diverging*. What the plot truly represent (looking at the logs) is that the model function values get extremely close to f_{best} without reaching the error threshold required for entering the *accepted* convergence scenario. In fact, even though the logarithmic residual error goes to almost 10^{-16}, the difference at the nearest point between function value and f_{best} only reaches a $4 * 10^{-3}$ error, not enough for reaching the convergence scenario.

Apart from this artifact we noticed how the plot highly resembles the previous ones, so we resolved the issue by highering the *accepted* threshold from $1 * 10^{-3}$ to $5 * 10^{-3}$, getting therefore the figures shown in **Figure 9** and **Figure 8**.

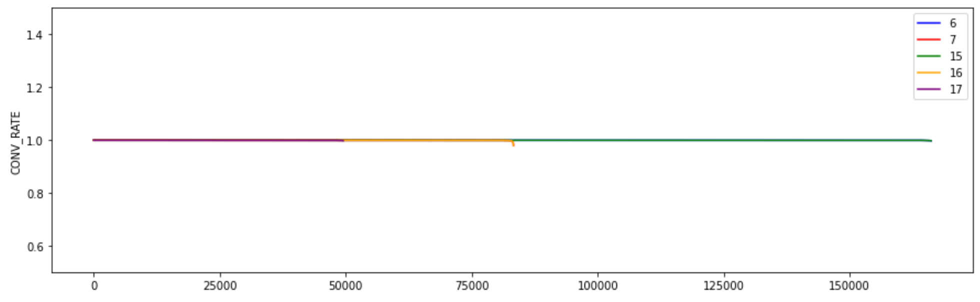

Figure 8: 1st Poly - Convergence rate of top 5 performing.

These graphical results show how the change of kernel does not radically change the outcome behaviour, as we hoped and expected. It is interesting

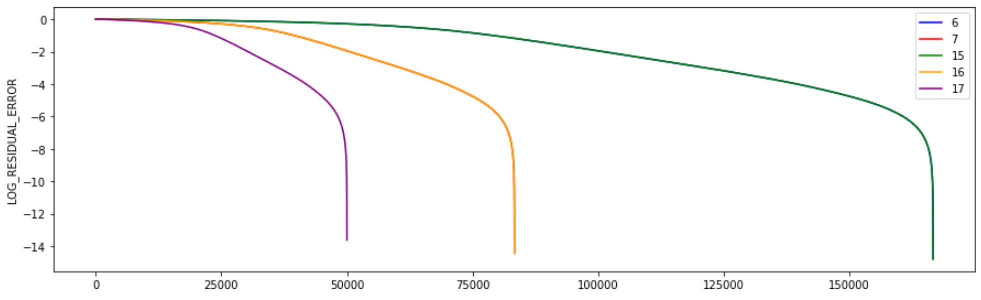

Figure 9: 1st Poly - Logarithmic residual error of top 5 performing.

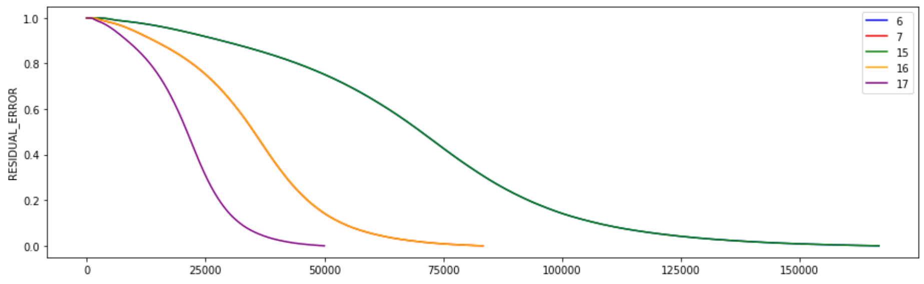

Figure 10: 1st Poly - Residual error of top 5 performing.

to notice how the number of iterations is generally comparable to the *linear* kernel scenario, with the difference residing in the logarithmic residual error having a much slower descent in the *poly* case with respect to the *linear case*.

5.4.4 2nd Dimension Poly Kernel

The model parameters are the same of Section 5.4.3. f_{best} in this experiment is equal to **-31723.1799**.

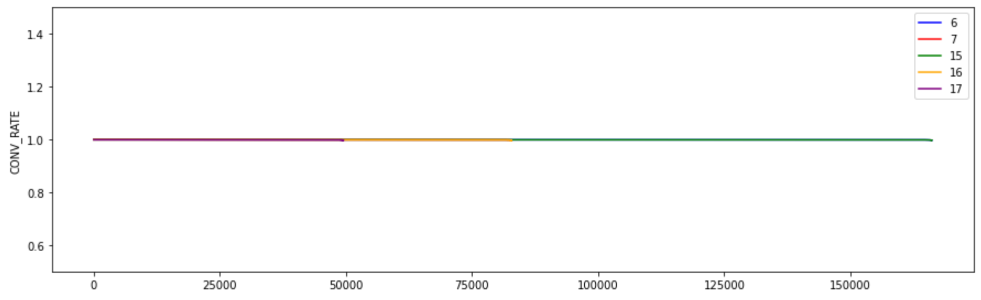

Figure 11: 2nd Poly - Convergence rate of top 5 performing.

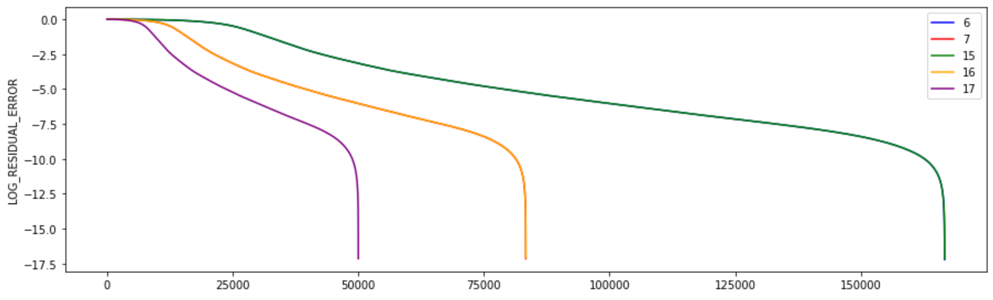

Figure 12: 2nd Poly - Logarithmic residual error of top 5 performing.

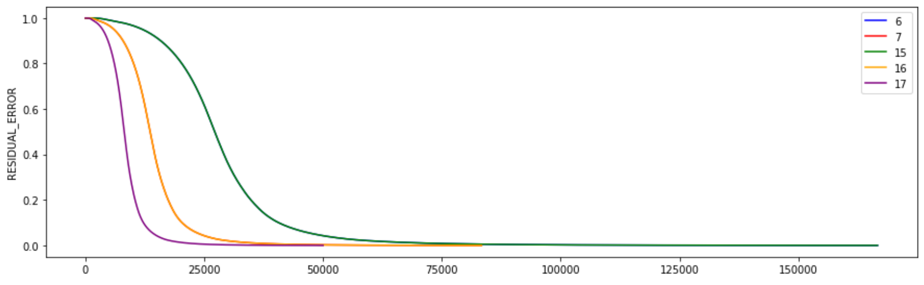

Figure 13: 2nd Poly - Residual error of top 5 performing.

We include this scenario for completeness and to show result coherence. No peculiar cases or strange behaviours occurred, with the graphical results resembling the previous scenarios outputs.

6 Conclusions

Results, reported in depth in *section 5.4.1*, demonstrate how the *deflected subgradient algorithm* follows theoretical expectations while optimizing the function. In particular, figures 1, 2 show how the convergence rate and the log residual error follow the forecasted sub-linear behaviour.

Sections 5.4.2, 5.4.3 and 5.4.4 provide other possible scenarios for the optimization task. This help us to analyze how changing the function affects the algorithm performances and behaviour as we change the output features and the kernel. Nevertheless, as shown in the results, this does not bring major changes - as we can appreciate in figures 4-12 - in fact, in all the analyzed scenarios the algorithm still meets an expected sub-linear behaviour.

As far as concerns the implementation, using an interpreted language such as *Python* brought some advantages in the drafting of the code, leading at the same time to a slow computation due to the overhead added by the programming language. As a matter of fact, simple models with just 50k iterations required a noticeably long amount of time to compute, with the computation time growing with the number of iterations, while being independent from the number of input features. An implementation in other languages - *C/C++* - would have for sure led to a faster execution, similarly to *scikit-learn* which uses *libsvm* implemented in *C*.

At the end of the day, we are happy with the results obtained in the development of this project. This work helped us to better understand and embrace a different view over optimization algorithms and Support Vector Machines.

7 References

[1] Krzysztof C. Kiwiel. "Breakpoint searching algorithms for the continuous quadratic knapsack problem". In: *Mathematical Programming* 112.2 (Apr. 2008), pp. 473–491. ISSN: 1436-4646. DOI: 10.1007/s10107-006-0050-z. URL: https://doi.org/10.1007/s10107-006-0050-z.

[2] d'Antonio Giacomo and Frangioni Antonio. "Convergence Analysis of Deflected Conditional Approximate Subgradient Methods". In: *SIUM Journal on Optimization* 20 (Jan. 2009). DOI: 10.1137/080718814.

[3] Frangioni Antonio and Enrico Gorgone. "A library for continuous convex separable quadratic knapsack problems". In: *European Journal of Operational Research* 229 (Aug. 2013), pp. 37–40. DOI: 10.1016/j.ejor.2013.02.038.

[4] Frangioni Antonio. *Unconstrained optimization I Gradient-type methods.* URL: https://elearning.di.unipi.it/pluginfile.php/41615/mod_resource/content/3/3-unconstrained%20optimization%20I.pdf.

[5] Frangioni Antonio. *Unconstrained optimization III Less-than-gradient methods.* URL: https://elearning.di.unipi.it/pluginfile.php/42987/mod_resource/content/2/5-unconstrained%20optimization%20III.pdf.

8 Appendix A

Define the Lagrangian function

$$\begin{aligned}\mathcal{L} = \frac{1}{2}\|w\|^2 + C\sum_i(\xi_i + \xi_i^*) &+ \sum_i \alpha_i(y_i - w\phi_i - b - \varepsilon - \xi_i)\\ &+ \sum_i \alpha_i(-y_i + w\phi_i - b - \varepsilon - \xi_i^*)\\ &- \sum_i \mu_i\xi_i \\ &- \sum_i \mu_i^*\xi_i^*\end{aligned} \tag{19}$$

$$\textit{where} \quad \forall_i\; \xi_i\xi_i^* \geq 0$$

Variables of the two definition of the problem:

Primal problem	$w,\ b,\ \xi_i,\ \xi_i^*$
Dual Problem	$\alpha_i,\ \alpha_i^*,\ \mu_i,\ \mu_i^*$

Next step is try to simplify the definition of the Lagrangian wrt the problem that needs to be solved. Since the objective is to find the *minimum* the developments proceeds imposing this condition.

$$\frac{\partial\mathcal{L}}{\partial w} = 0 \quad \longrightarrow \quad w + \sum_i \alpha_i(-\phi_i) + \sum_i \alpha_i^*\phi_i = 0 \tag{20a}$$

$$\frac{\partial\mathcal{L}}{\partial b} = 0 \quad \longrightarrow \quad \sum_i -\alpha_i + \sum_i \alpha_i^* = 0 \tag{20b}$$

$$\frac{\partial\mathcal{L}}{\partial \xi_i} = 0 \quad \longrightarrow \quad C - \alpha_i - \mu_i = 0 \tag{20c}$$

$$\frac{\partial\mathcal{L}}{\partial \xi_i^*} = 0 \quad \longrightarrow \quad C - \alpha_i^* - \mu_i^* = 0 \tag{20d}$$

From (20a) the definition of w can be derived

$$w = \sum_i(\alpha_i - \alpha_i^*)\phi_i \tag{21}$$

From (20b) the first constraint on the Lagrangian variables is obtained

$$\sum_i(\alpha_i^* - \alpha_i) = 0 \tag{22}$$

While from (20c)/(20d) with some further development the second constraint on the Lagrangian variables can be defined

$$\begin{aligned}
&\alpha_i,\ \alpha_i^*,\ \mu_i,\ \mu_i^*\ \geq\ 0 \quad \forall_i \\
&C = \alpha_i + \mu_i \quad \longrightarrow \quad \alpha_i = C - \mu_i \\
&\Longrightarrow \quad \alpha_i\ \in\ [0,\ C] \\
&\textit{and equivalently} \quad \alpha_i^*\ \in\ [0,\ C]
\end{aligned}$$

Simplify (19) using the substitution (21)

$$\begin{aligned}
\mathcal{L}\ =&\ \frac{1}{2}\sum_i\sum_j(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\phi_i\phi_j \\
&-\sum_i\sum_j(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\phi_i\phi_j \\
&+\sum_i(\alpha_i-\alpha_i^*)y_i+\sum_i(\alpha_i-\alpha_i^*)b-\sum_i(\alpha_i+\alpha_i^*)\varepsilon \\
&+\sum_i\alpha_i(-\xi_i)+\sum_i\alpha_i^*(-\xi_i^*) \\
&-\sum_i\mu_i\xi_i-\sum_i\mu_i^*\xi_i^* \\
&+C\sum_i\xi_i+\xi_i^*
\end{aligned}$$

Apply condition (22) and (20c) to simplify some terms and obtain the final formulation

$$\begin{aligned}
\mathcal{L}(\alpha,\alpha^*)\ =&\ -\frac{1}{2}\sum_i\sum_j(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\phi_i\phi_j \\
&+\sum_i(\alpha_i-\alpha_i^*)y_i \\
&-\sum_i(\alpha_i+\alpha_i^*)\varepsilon
\end{aligned}$$

$$\textit{With the constraints} \qquad \begin{cases}\sum_i(\alpha_i^*-\alpha_i)=0\\ \alpha_i\in[0,\ C]\\ \alpha_i^*\in[0,\ C]\end{cases}$$

9 Appendix B

If $M = \emptyset$ then we have reached a point in the algorithm in which μ_L and μ_U are two consecutive breakpoint so $h(\mu)$ is linear in the interval $[\mu_L, \mu_U]$. This can be exploited in order to compute the straight line that connects the two points.

$$\frac{h(\mu) - h(\mu_L)}{h(\mu_U) - h(\mu_L)} = \frac{\mu - \mu_L}{\mu_U - \mu_L} \tag{23}$$

Given the formulation of Algorithm 3 the following statements are true at each step of the procedure.

$$h(\mu_L) > 0 \qquad h(\mu_U) < 0 \tag{24}$$

Given the formulation for (23) and the assumptions in (24) for the *intermediate zero theorem* there exists a point $\hat{\mu}$ where $h(\hat{\mu}) = 0$. In (16), $h(\mu)$ was defined as the function representing the linear constraint. In this case the linear constraint is $h(\mu) = 0$ so the point $\hat{\mu}$ is the optimal value of μ. Substituting in (23) and isolating μ, we can define μ^*

$$\mu^* = \mu_L - [h(\mu_L) - 0]\frac{\mu_U - \mu_L}{h(\mu_U) - h(\mu_L)} \tag{25}$$