SVR-JBundle.jl

Solving Support Vector Regression
with Bundle Methods in Julia

Samuele Sabella, Maris Basha

October 4, 2020

Abstract

This work addresses support vector regression with *Bundle Method* convex optimization implemented using Julia programming language. We have tested different master problem stabilization techniques to overcome the main flaws of a naïve implementation and tested our bundle method implementation on a quadratic maximum problem to verify the correctness of our solver.

1 Introduction

1.1 Support Vector Regression

$\hat{y}$ We address the problem of support regression, i.e. we look for a vector $\mathbf{w} \in \mathbb{R}^{n,1}$ and a scalar b such that, given a training set $\{(\mathbf{x_i} \in \mathbb{R}^{n,1}, y_i \in \mathbb{R})\}_{1..m}$, our predictions $\{\hat{y}_1, ..., \hat{y}_m\}$ are at most ε wrong in a L1 sense [9]. In this work we use bold notation for vectors and normal font for scalars. The problem can be seen as trying to build a tube with thickness ε able to enclose all training samples and it can be written in the primal form $\mathcal{P}$ which accounts also for small errors in the model output using same technique presented in the SVM soft-margin model.

SVR Primal ($\mathcal{P}$)

$$
\begin{aligned}
\textbf{input} &: \{< \mathbf{x_1}, y_1 >, ..., < \mathbf{x_i}, y_m >\} \\
\textbf{solve} &: arg \min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \\
&\quad |y_i - (\langle \mathbf{w}, \mathbf{x_i} \rangle + b)| \leq \varepsilon + \xi_i \quad \forall i \in [1, n] \\
&\quad \xi_i \geq 0 \quad \forall i \in [1, n] \\
\textbf{output} &: \mathbf{w}, b
\end{aligned}
$$

The slack variable ξ_i manages otherwise unfeasible constraints in the optimization problem. The constant $C > 0$ determines the trade-off between the flatness of *f(x)* and the order which deviations larger than ε are tolerated.
We applied *Bundle Method* optimization (section 1.2) to an unconstrained reformulation of the primal problem $\mathcal{P}$ which comes as non-differentiable thus preventing us to use algorithms like gradient descent.

1.2 Bundle Methods

Bundle Methods are a family of iterative optimization algorithms which doesn't make assumptions on the differentiability of the target function and, at each iteration, accumulates information which is then used to build up a global model of the target function. The model is then minimized at each iteration. Given a problem, constrained or not, it requires storing at each iteration two kind of information: the gradient, if the function is differentiable, or the subgradient $\mathbf{z_i}$ at each point $\mathbf{x_i}$; the function value $f(\mathbf{x_i})$, also written as f_i. The set of all tuples collected up to iteration k, $\mathcal{B}_k = \{\langle \mathbf{x_i}, f_i, \mathbf{z_i} \rangle\}_{i=1...k}$, is called *bundle*[1]. Using the information collected, *Bundle Methods* build up a lower bound model $f_{\mathcal{B}}$ of the function using the combination of multiple hyperplanes passing through each $f(\mathbf{x_k})$ and having the sub-gradient direction (eq. 1).

$$f_{\mathcal{B}} = max\{f_b + \langle z_b, \mathbf{x} - \mathbf{x_b} \rangle : b \in \mathcal{B}\} \tag{1}$$

At each iteration k, the model $f_{\mathcal{B}}$ is rewritten as a linear constrained optimization problem (eq. 2), called master problem (MP) and the minimum $\mathbf{x_{k+1}}$ is computed using off-the-shelf solvers and used for the next iteration. The algorithm can be easily extended to constrained convex optimization problems by adding additional constraints to the MP.

$$min(f_{\mathcal{B}}) = min\{v : v \geq f_b + \langle \mathbf{z_b}, \mathbf{x} - \mathbf{x_b} \rangle \text{ with } b \in \mathcal{B}\} \tag{2}$$

Due to the fact that the algorithm does not ensure a discent direction and $f(\mathbf{x_{k+1}})$ may be grater than $f(\mathbf{x_k})$, the main iterative cycle keeps track of the best point $\bar{\mathbf{x}}_\mathbf{k}$ up to iteration k using an Armijo's like condition[2]. For seek of simplicity from now on we will drop the superscript k from $\bar{\mathbf{x}}_\mathbf{k}$ and refer to $\bar{\mathbf{x}}$ as the best result up to the last iteration computed. We call *Serious Step* (*SS*) an iteration where a new $\bar{x}$ is computed, i.e. a significant improvement is made towards the minimum of the target function. Otherwise we call that iteration a *Null Step* (*NS*).
Due to the fact that $f_{\mathcal{B}}$ is likely to be unbounded below in large spaces and, even if bounded, it is easy to trick the model and make it believe the minimum lies in regions where the function increases, we used techniques able to build up "trust regions" of the model and make it more reliable. This techniques are called *stabilizers* and in our experiments we covered two of them:

[1] To simplify the notation we will assume that $\mathcal{B}$ is always referred to the last iteration computed

[2] An improvement is made whenever: $f(x_i) \leq f(\bar{x}) + m_1(f_{\mathcal{B}}(x_i) - f(\bar{x}))$ with x_i the solution of the Master Problem

- **Trust region:** it constraint $\mathbf{x_{i+1}}$ to be close to $\bar{\mathbf{x}}$ by adding a constraint to the MP: $||\mathbf{x} - \bar{\mathbf{x}_\mathbf{i}}||_\infty \leq \delta_i$.
- **Proximal:** penalizes the MP function proportionally to the distance from the best point so far: $min(f_\mathcal{B} + \frac{\mu}{2} * ||\mathbf{x} - \bar{\mathbf{x}}||_2^2)$. The problem becomes non-linear, thus a greater computational cost for each iteration is required.

At each *SS* μ is decreased by a factor of 0.99 whereas δ is increased by a factor of 1.2 until a minimum or maximum is reached, respectively: 1×10^{-3} and 1×10^2. This is justified by the fact that the model becomes gradually more accurate and the trust region can be expanded. If a NS occurs, the trust region is shrunk by increasing μ by a factor of 1.1 until a maximum is reached: 10. Using the Trust Region stabilizer we decreased δ by a factor of 0.99 until 1×10^{-3} is reached. It is necessary to say that these values were determined empirically by choosing combinations with better results both in terms of convergence and learning performance.
We have chosen to stop the algorithm whenever the improvement estimated by the model between two successive *SS*, i.e. $\Delta_i = f(\bar{x}_{i-1}) - f_\mathcal{B}(\bar{x}_i) > 0$ reaches a threshold ε as suggested by [4].

$$\Delta_i := f(\bar{x}_{i-1}) - f_\mathcal{B}(\bar{x}_i) \leq \varepsilon \tag{3}$$

Following [10] it can be shown that if proximal stabilizer is used and 3 holds: $f(\bar{x}_{i-1}) \leq f(x^*) + \frac{\mu}{2} * ||x^* - \bar{x}_{i-1}||^2 + \varepsilon$, with x_* the optimal solution to our target function. Motivated by the fact that both *Trust Region* and *Proximal Stabilizers* are deeply connected and that it can be shown both have the same optimum with appropriate choices of μ and σ, we applied the same termination condition also to trust region stabilizer to better compare the two stabilization techniques. From now on we will refer to $f_\mathcal{B}(x_i)$ as v_i, the optimum of the master problem at iteration i, computed by an off-the-shelf solver[3]. Due to the fact that in most of our experiment we have reached a $\Delta_i \simeq 1 \times 10^{-6}$ we have decided that $\Delta_i \leq 5 \times 10^{-6}$ was a reasonable termination condition.

1.3 SVR primal MP

As described by [2] and [8] the *SVR* primal problem has an unconstrained convex problem associated, reported in equation 4.

$$\min_{\mathbf{w},b} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m} L_\varepsilon(f(x_i;\mathbf{w},b), y_i) \tag{4}$$
$$\text{with} \quad L_\varepsilon(\hat{y}, y) = \max\{||\hat{y} - y||^2 - \varepsilon, 0\}$$

Equation 4 deserves few explanations. Whereas the first term accounts for the model complexity, the summation over the training set accounts for each error greater than ε computed for each training sample. Without loss of generality we

[3] If the iteration is not specified we are referring to the last one

used an $L2$ term to define the ε-tube loss (L_ε), instead of an $L1$ term, because it better resembles our final objective: a small $L2$ error. We didn't intentionally provide details about the model prediction function $f(\mathbf{x}; \mathbf{w}, b)$ because further detail has to be given. Indeed, if a linear model is what we are looking for, the hyperplane which minimizes 4 can be used together with the bias to predict a new, previously unseen sample: $f(\mathbf{x}; \mathbf{w}, b) = \langle \mathbf{w}, \mathbf{x} \rangle + b$. However we can use the kernel trick even without solving the dual problem [3] by computing each prediction as a linear combination of the kernel function, namely k, between $\mathbf{x}$ and each point in the training set $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_m\}$. Given $\mathbf{K}$, the matrix in $\mathbb{R}^{m,m}$ containing the kernel function value between each training sample pair, i.e. $K_{i,j} = k(x_i, x_j)$ we can exploit the kernel trick [4] and rewrite our model prediction as shown in equation 5.

$$f(\mathbf{x}; \mathbf{w}, b) = \langle K(\mathbf{x}, \mathbf{X}), \mathbf{w} \rangle + b \quad with \quad K(\mathbf{x}, \mathbf{X}) = [k(\mathbf{x}, \mathbf{x}_1), ..., k(\mathbf{x}, \mathbf{x}_m)] \tag{5}$$

Given $\mathbf{b} \in \mathbb{R}^m$ the column vector containing the bias b in all positions and $\mathbf{y}$ the column vector containing all target samples in the training set and r_i the i-th item of the residual vector $\mathbf{r}$, our target function L_{svr} can be rewritten by expanding 4:

$$L_{svr} = \min_{\mathbf{w}, b} \frac{1}{2} \left(\mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{m} max\{||r_i||_2^2 - \varepsilon, 0\} \right) \tag{6}$$

$$\mathbf{r} = \mathbf{K}^T \mathbf{w} + \mathbf{b} - \mathbf{y} \tag{7}$$

We used bundle method optimization to directly minimize L_{svr}. The first step to achieve this goal is to compute the subgradient of L_{svr} with respect to $\mathbf{w}$ and b. We introduce the vector $\boldsymbol{\delta}$ to easily deal with the subgradient of the maximum function.

$$\delta_j = \begin{cases} 0 & if \quad ||r_j||_2^2 \leq \varepsilon \\ C & otherwise \end{cases} \quad \text{and} \quad \boldsymbol{\delta} = [\delta_1, ..., \delta_n]^T$$

[4] Depending on the Kernel chosen it may be more convenient to map each sample in the feature space and compute the dot product explicitly. In our case this does not occur due the use of infinite dimensional feature spaces.

Given $\mathbf{K}_i$ the i-th column of $\mathbf{K}$, we are able to rewrite the subgradient of L_{svr} in compact form:

$$\begin{aligned}
\frac{\partial}{\partial w_i} L_{svr} &= \frac{1}{2} \frac{\partial}{\partial w_i} \left(\mathbf{w}^T \mathbf{w} + \sum_{j=1}^{m} max\{C * (||r_j||_2^2 - \varepsilon), 0\} \right) \\
&= w_i + \frac{1}{2} \left(\sum_{j=1}^{m} \delta_j \frac{\partial}{\partial w_i} (||\mathbf{K}_j^T \mathbf{w} + \mathbf{b} - y_j||_2^2 - \varepsilon) \right) \\
&= w_i + \left(\sum_{j=1}^{m} \delta_j * K_{j,i} * r_j \right) \quad \textit{iff} \quad \nabla_w L_{svr} = \mathbf{w} + \boldsymbol{\delta} * \langle \mathbf{K}, \mathbf{r} \rangle \qquad (8) \\
\frac{\partial}{\partial b} L_{svr} &= \frac{1}{2} \left(\sum_{j=1}^{m} \delta_j \frac{\partial}{\partial b} (||\mathbf{K_j}^T \mathbf{w} + \mathbf{b} - y_j||_2^2 - \varepsilon) \right) \\
&= \sum_{j=1}^{m} \delta_j r_j = \langle \boldsymbol{\delta}, \mathbf{r} \rangle \qquad (9)
\end{aligned}$$

Substituting L_{svr}, 8 and 9 into 2 as f and z, respectively, gives us the MP formulation for a kernel based svr. Depending on the stabilizer chosen, the master problem built at each iteration has been solved using the simplex method (*trust region*) or the interior point method (*proximal*). We used Ipopt optimizer (Interior Point OPTimizer) [11] to solve the master problem which implements a primal-dual interior point method and the GLPK [6] Library (GNU Linear Programming Kit) which implements the simplex method. We had changed some default values of the hyperparameters for both optimizers which dealt with the maximum number of iterations and the solution precision tolerance to achieve good results. As the bundle size grows in size at each iteration, the MP becomes more and more complicated as the algorithm proceeds. Thus, we have constrained the bundle dimension either by resetting the bundle together with μ and σ at each SS, or by implementing the bundle as a queue with fixed size. Indeed, we didn't use techniques with theoretical fundamentals to keep the bundle size small, instead we used simple tricks which have proven to be very good in practice while solving Support Vector Regression.

2 Experiments

2.1 Numerical results

We have tested our bundle method optimizer implementation on a very simple quadratic maximum ($QMax$) problem found in [5], shown in equation 10 and having a global optimum in $x = 0$. We have assumed $n = 50$ in each instance of $QMax$ problem, namely $QMax_{50}$. Even if the problem is very simple it is

suitable to verify known bundle methods properties.

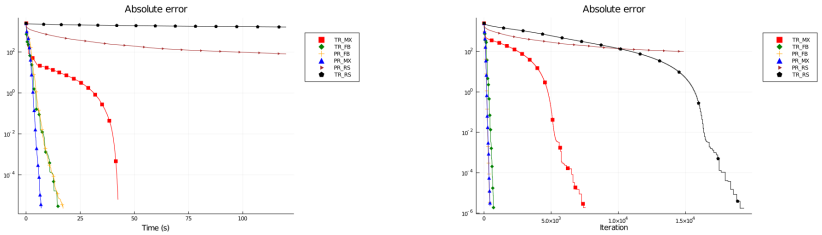
$$
\begin{aligned}
f(x) &= \max_{i=1...n} x_i^2 \\
x_i^{(1)} &= i \quad \text{for} \quad i = 1, ..., \frac{n}{2} \\
x_i^{(1)} &= -i \quad \text{for} \quad i = \frac{n}{2}, ..., n
\end{aligned}
\tag{10}
$$

Figure 1a and 1b show the improvement estimated by the model, i.e. Δ (see subsection 1.2), and the absolute error, which matches $f(\bar{x})$ exactly, computed with respect to time. We compared three different methods of containing the size of the bundle[5]. *Full bundle (FB):* meaning that no constraint is imposed on the size of the bundle; *max bundle (MX):* the bundle is implemented as a queue with max size 50 items; *reset (RS):* the bundle is emptied at each *SS*. In each experiment we have chosen the Armijo's condition parameter to be $m = 1 \times 10^{-3}$, Proximal (PR) or either Trust Region (TR)[6] as stabilizers with $\mu = 1.0$, $\sigma = 0.1$ respectively and $\varepsilon = 5 \times 10^{-6}$.
We have compared the different options also on other datasets (more details in section 2) and reported similar results. Except for $QMax$ where the optimum is in 0, for all the other datasets we had to run it first with a low $\varepsilon = 5 \times 10^{-12}$ to find f_*, then we have computed the relative error of each technique with respect to the optimum found. In most of our experiments we found a reasonably fast convergence, sometimes better than the pessimistic sub-linear upper bound of $O(1/\varepsilon^3)$ [4]. The results show us that both in Proximal and Trust Region, the Reset Bundle configuration has sub-linear convergence, mostly due to the fact that it makes lots of iterations and small improvement between iterations. The full bundle configuration does not resemble the convergence curve expected from the theory, i.e. few improvements until the bundle reaches a threshold size then fast improvement, with linear convergence overall. The best results both in terms of time efficiency and iteration number were achieved by Proximal Max Bundle, which seems to have better than sub-linear convergence but not as good as super-linear convergence.

In most of the cases we were not able to spot the convergence properties which are typical of the *Bundle methods*: few improvements when the bundle is increasing in size and a fast convergence when the bundle has accumulated enough information. Instead the algorithm seems to be steadily converging at each iteration most of the time.

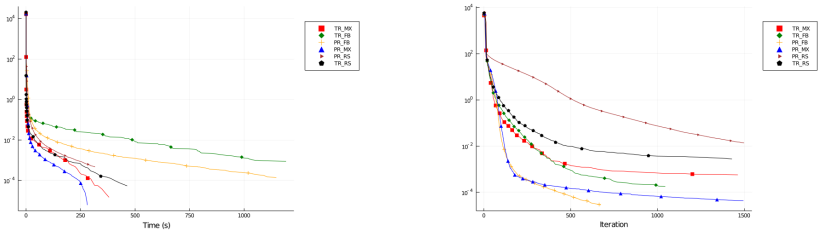
[5]The plots have been generated with the same hyper-parameters but different starting point.

[6]Figure 2 to 4 use as legend the convention "Stabilizer Name"_"Bundle Size Technique". e.g. PR_MX is Proximal with Max Bundle.



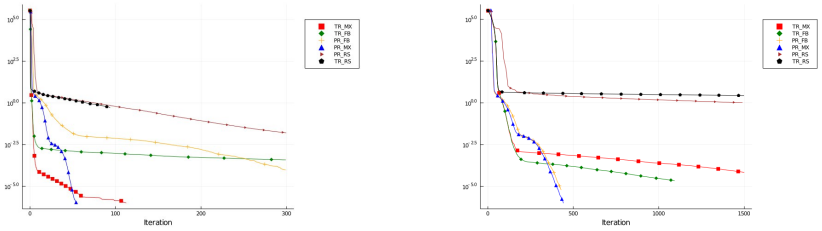
(a) Time (b) Iterations

Figure 1: Absolute Error up to 5×10^{-6} for Qmax



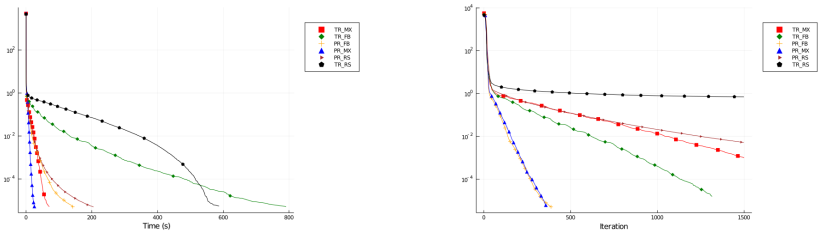
(a) Time (b) Iterations

Figure 2: Relative error up to 5×10^{-6} for *Gauss* dataset



(a) Time (b) Iterations

Figure 3: Relative error up to 5×10^{-6} for *Carbon* dataset



(a) Time (b) Iterations

Figure 4: Relative error up to 5×10^{-6} for *ML-CUP* dataset

2.2 Learning performance

We experimented two different non linear regression problems: *Gauss3*[7] and *Carbon*[1] and *ML-CUP* datasets. *Gauss3* requires to predict a non linear function in $\mathbb{R}$ whereas *Carbon* requires to predict three real value outputs given six input attributes. Instead of dealing with multi-output SVR we decided to keep just one output value for *Carbon* dataset and drop the other two. *ML-CUP* requires to predict two real value outputs given twenty input attributes. For this dataset we decided to train two single-output *SVR* and use them together. Table 1 summarizes the datasets' information.

dataset	input	output	train/test records	data domain
Gauss3	1	1	200/200	real
Carbon Nanotubes	6	1	2000/2000	real
ML-CUP	20	2	1765/412	real

Table 1

We compared our results to an off the shelf SVR trained with the same hyperparameter values to ensure the correctness of our implementation. On all datasets we run tests with *Proximal* stabilizer and kept under control the size of the bundle by resetting it upon each SS. We have chosen m, the Armijo condition parameter, to be 1×10^{-3}. We opted for this solution due to the fact that it showed to be the most performing compared to the other techniques. We run a Grid Search over these parameters: *RBF*, *Gaussian* and *Laplacian* kernels with with σ equal to 1.5 and 3.0, C with values equal to 1.0 and 2.0, ϵ-*tube* with values 1×10^{-2} and 1×10^{-3}, and μ with values 1.0 and 0.1. As termination condition we set either the convergence to be $\Delta \leq 5 \times 10^{-6}$ or the maximum time to be 20 minutes.
It's worth saying that we have chosen these hyper-parameter combinations because were considered the most interesting in order to prove the SVR implementation correctness based on the results obtained in Figures 2, 3 and 4.
Tables 4, 5 and 6 show the result obtained for the datasets with different configurations. It is to be noted that we are able to converge relatively fast for most of the configurations. The training and testing losses for Tables 4, 5 don't change that much comparing between configurations.
Figures 5,6 and 7 show for all three datasets the plots for the Serious Step number (a), the stability center value at each Serious Step (b), the relative error of the model $(f_{\bar{x}} - f_*)/f_*$ (c) and the L2 loss in the testing set (d). We have not found in our plots any clear clues about the inner working of the Bundle Method, such as low initial convergence while the algorithm is building up the bundle such as plateaus of NS where no improvement is made for a relatively long time followed by a notable improvement. We argue that this happens because of the Max Bundle condition.
It has to be noted that the plots do not resemble the best configuration for convergence, but a configuration that converges in a reasonable number of iterations and has a reasonable L2 loss on both training and testing.

Table 2 compares the time efficiency for all datasets, using the off-the-shelf SVR versus our implementation.
Figure 8 shows the time performance comparison of our Bundle Method SVR compared to the off-the-shelf SVR on the Gauss dataset when the number of example grows with a tolerance set to 1×10^{-2}. Both are quasi-linear in scalability, but it's worth noting that the off-the-shelf SVR is faster, this due to the fact that they solve a different problem, and that the Bundle Method solver uses more time as the bundle size grows to compute the constraints defined by the MP.

Dataset	off-the-shelf SVR [s]	Bundle Method [s]
Gauss	0.05446	51.43
Carbon Nanotubes	2.74	151.80
ML-CUP	0.36	236.10

Table 2: Best time efficiency results with different parameters, for off-the-shelf SVR and Bundle Method

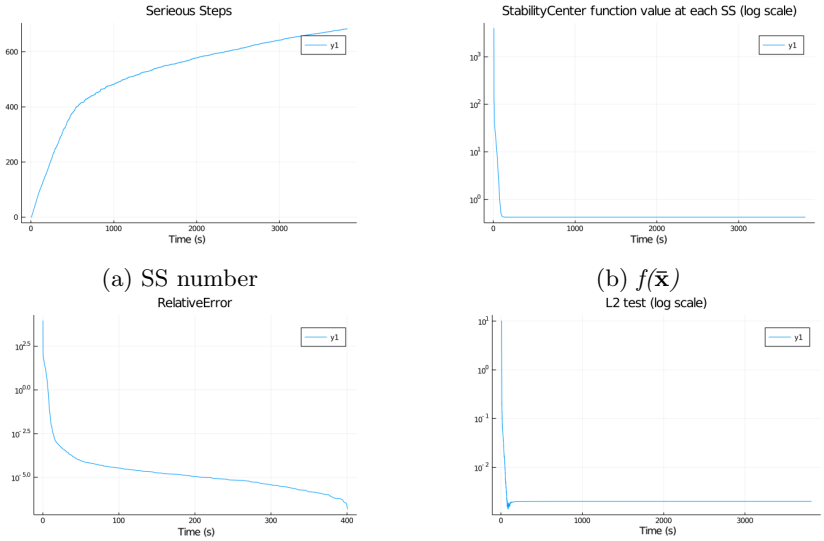
	Hyper-parameters						
Dataset	Stabilizer	ϵ-tube	C	μ	Kernel	L2-TR	L2-TS
Bundle SVR	Proximal	1×10^{-3}	2.0	0.1	Laplacian	0.010 08	0.0391
off-the-shelf SVR	-	0.1	10	-	RBF	9.32×10^{-5}	1.05×10^{-4}

Table 3: Best L2 performance of Bundle method SVR vs off-the-shelf SVR on the *ML-CUP* dataset

Table 3 shows the L2 loss result for both training and testing for *ML-CUP* dataset. It is notable that Bundle SVR is better performing than the off-the-shelf SVR as per best results on training and testing L2.

Hyper-parameters								
ϵ-tube	C	μ	Kernel	Kernel ς	L2-TR	L2-TS	Converged	Time (s)
1×10^{-3}	2.0	1.0	RBF	3.0	6.305×10^{-4}	7.549×10^{-4}	Yes	60.33
1×10^{-2}	1.0	1.0	RBF	3.0	6.6602×10^{-4}	8.6359×10^{-4}	Yes	52.71
1×10^{-3}	2.0	1.0	Laplacian	3.0	5.3343×10^{-4}	1.475×10^{-3}	Yes	51.43
1×10^{-2}	1.0	0.1	Laplacian	3.0	5.621×10^{-4}	1.737×10^{-3}	Yes	51.47
1×10^{-2}	2.0	0.1	RBF	3.0	6.936×10^{-4}	2.6941×10^{-3}	Yes	53.46
1×10^{-3}	1.0	1.0	RBF	1.5	1.056×10^{-3}	3.912×10^{-3}	Yes	51.54

Table 4: Convergence results of SVR Bundle method on the *Gauss3* dataset with $\epsilon = 5 \times 10^{-6}$, Proximal Stabilizer Max Bundle = 50.



(a) SS number

(b) $f(\bar{\mathbf{x}})$

(c) Relative Error

(d) L2 distance on testing

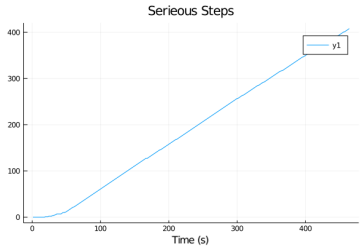
Figure 5: ϵ-*tube* $= (1 \times 10^{-3})$, $C = (2.0)$, *RBF* kernel and *Proximal* stabilizer with Max Bundle = 50, dataset *Gauss3*

Hyper-parameters								
ϵ-tube	C	μ	Kernel	Kernel ς	L2-TR	L2-TS	Converged	Time (s)
1×10^{-3}	2.0	0.1	RBF	3.0	8.3245×10^{-4}	1.40×10^{-3}	Yes	171.57
1×10^{-2}	1.0	0.1	RBF	1.5	2.656×10^{-3}	1.672×10^{-3}	Yes	162.49
1×10^{-2}	2.0	1.0	RBF	3.0	1.148×10^{-3}	1.692×10^{-3}	Yes	160.59
1×10^{-3}	1.0	1.0	RBF	1.5	5.2507×10^{-4}	3.122×10^{-3}	Yes	151.80
1×10^{-3}	2.0	0.1	Gaussian	3.0	5.253×10^{-4}	3.254×10^{-3}	Yes	216.75
1×10^{-2}	2.0	1.0	Gaussian	3.0	6.765×10^{-3}	4.771×10^{-3}	Yes	210.60

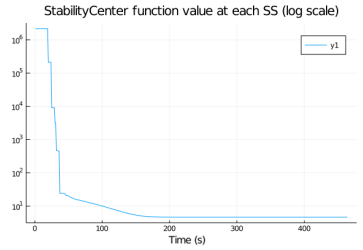
Table 5: Convergence results of SVR Bundle method on the *Carbon Nanotubes* dataset with $\epsilon = 5 \times 10^{-6}$, Proximal Stabilizer with Max Bundle = 50

Hyper-parameters								
ϵ-tube	C	μ	Kernel	Kernel ς	L2-TR	L2-TS	Converged	Time (s)
1×10^{-3}	2.0	0.1	Laplacian	3.0	0.010 013	0.039 127	Yes	250.18
1×10^{-2}	2.0	0.1	Laplacian	3.0	0.0122	0.041	Yes	294.59
1×10^{-3}	1.0	1.0	Laplacian	3.0	0.0188	0.050 09	Yes	236.10
1×10^{-2}	1.0	1.0	Laplacian	3.0	0.020 644	0.051 67	Yes	268.012
1×10^{-3}	2.0	1.0	RBF	3.0	0.046 73	0.067 45	Yes	324.95
1×10^{-2}	2.0	1.0	RBF	1.5	0.047 24	0.068 223	Yes	320.634

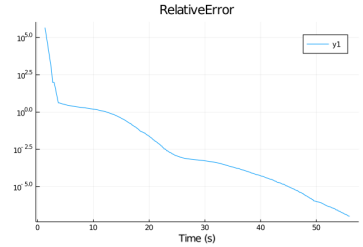
Table 6: Convergence results of SVR Bundle method on the *ML-CUP* dataset with $\epsilon = 5 \times 10^{-6}$, Proximal Stabilizer Max Bundle = 50



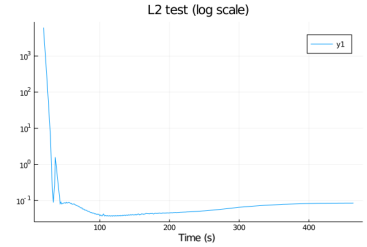
(a) SS number



(b) $f(\bar{\mathbf{x}})$

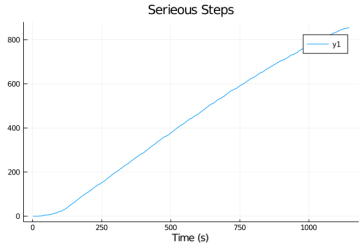


(c) Relative Error

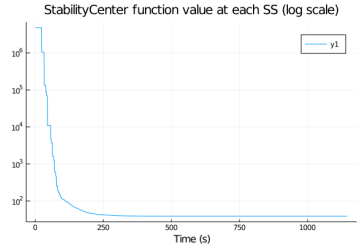


(d) L2 distance on testing

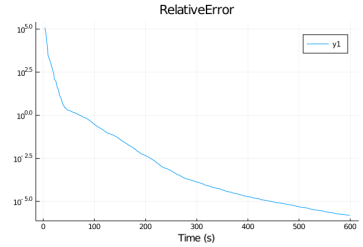
Figure 6: ϵ-*tube* $= (1 \times 10^{-3})$, $C = (2.0)$, *RBF* kernel and *Proximal* stabilizer with Max Bundle = 50, dataset *Carbon Nanotubes*



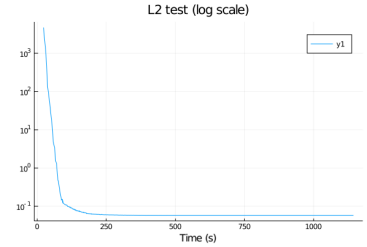
(a) SS number



(b) $f(\bar{\mathbf{x}})$



(c) Relative Error



(d) L2 distance on testing

Figure 7: ϵ-*tube* $= (0.1)$, $C = (1.0)$, *Laplacian* kernel and *Proximal* stabilizer with reset upon SS, dataset *ML-CUP*

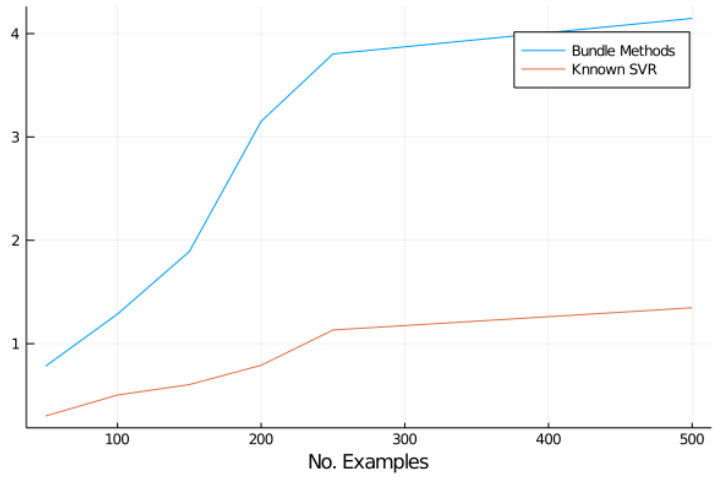


Figure 8: Bundle Method SVR model scalability vs. off-the-shelf SVR on Gauss3

3 Conclusions

In this paper we presented the Julia implementation of a Support Vector Regression optimized through Bundle Methods. We tested the optimizer correctness on a quadratic maximum problem with known optima. In this experiments we achieved very good precision (up to 1×10^{-8}) with few iterations (at most 500), disproving the worst case scenario: sublinear convergence. Encouraged by this results we applied our optimizer to a support vector regression formulation exploiting also the kernel trick to tackle non-linear functions. After running a grid search we have achieved both good learning performance (i.e. L2 validation error up to 0.003 16) and solution quality (i.e. $\Delta = 5 \times 10^{-8}$) in our main target dataset: *MLCup*. While running numerical experiments we found that the convergence rate suggested by the theory, i.e. sublinear, is rather pessimistic and in most of our experiments we found better convergence speed, with best case scenario given by quadratic convergence. We have also shown that as long as the learning performance is the main goal, our implementation is able to achieve good results both in predictions score and dataset scalability comparing to an off-the-shelf SVR implementation.

References

[1] Mehmet Acı and Mutlu Avcı. Artificial neural network approach for atomic coordinate prediction of carbon nanotubes. *Applied Physics A*, 122(7):631, 2016.

[2] Dominik Brugger, Wolfgang Rosenstiel, and Martin Bogdan. Online svr training by solving the primal optimization problem. *Journal of Signal Processing Systems*, 65(3):391–402, 2011.

[3] Olivier Chapelle. Training a support vector machine in the primal. *Neural computation*, 19(5):1155–1178, 2007.

[4] Antonio Frangioni. Standard bundle methods: untrusted models and duality. pages 61–116, 2020.

[5] Marjo Haarala, Kaisa Miettinen, and Marko M Mäkelä. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, 2004.

[6] Andrew Makhorin. Glpk (gnu linear programming kit). *http://www. gnu. org/s/glpk/glpk. html*, 2008.

[7] nist. Nist nonlinear regression. *https://www.itl.nist.gov/div898/strd/nls/data /LINKS/DATA/Gauss3.dat*.

[8] Anagha Puthiyottil, S Balasundaram, and Yogendra Meena. L1-norm support vector regression in primal based on huber loss function. In *Proceedings of ICETIT 2019*, pages 195–205. Springer, 2020.

[9] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[10] Choon Hui Teo, SVN Vishwanathan, Alex Smola, and Quoc V Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11(1), 2010.

[11] Andreas Wächter and L Biegler. Ipopt-an interior point optimizer, 2009.