Support Vector Regression using Deflected Subgradient Methods

Elia Piccoli
Nicola Gugole

November 18, 2021

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:

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

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:

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

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:

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

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` δ `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:*
>
> $$f_{ref}^{\infty} = -\infty = f^*$$
> $$f_{ref}^{\infty} \leq f^* + \xi\sigma^* + \delta^*, \ where \ 0 \leq \xi = \max\{1 - \delta^*\Gamma/2\sigma^*, 0\} < 1$$

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]

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

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

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

  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]

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

  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 7, 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, $\epsilon \approx 0$)

```
1 begin
2 |  ∀_i x_i ∈ [−C, C]
3 |  for i ← 0 to size(d) do
4 |  |  if (−C − x_i < ε and d_i < 0) or (C − x_i < ε and d_i > 0) then
5 |  |  |  d_i ⟵ 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 [5] 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.

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

Which by Lagrangian Relaxation leads to:

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

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 -∞ 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]).

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

Algorithm 3: Convex Separable Knapsack Problem Algorithm

```
1  begin
2      while M ≠ ∅ do
3          choose μ̂ using median of medians approach over M
4          compute h(μ̂)
5          if h(μ̂) = 0 then
6              μ* = μ̂
7              return μ*
8          else
9              if h(μ̂) > 0 then
10                 μ_L = μ̂
11                 M = {μ ∈ M : μ̂ < μ}
12             else
13                 μ_U = μ̂
14                 M = {μ ∈ M : μ̂ > μ}
15     μ* = μ_L − h(μ_L) (μ_U − μ_L) / (h(μ_U) − h(μ_L))
16     return μ*
```

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 different experiments and results of the SVR implementation developed and proposed for this project. More in detail:

- **Section 5.1** introduces qualitative tests on naive datasets, with the aim of checking the correctness of the developed model in comparison with a baseline (in this case the implementation used is the one proposed by *SkLearn* (see [4]), based on *LIBSVM* (see [3])).
- **Section 5.2** shows instead the experiments on a more complex dataset such as the one proposed for the Machine Learning course in the academic year 2020/2021. A deeper introduction to the dataset and a quantitative result are shown later.
- **Section 5.2.4** shows a comparison between the proposed solution and *SKLearn* using an higher number of iterations, reasoning about optimality and performances.
- **Section 5.3** analyzes the scalability of the proposed solution with respect to the number of inputs and number of features.

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 Baseline Comparison

A baseline comparison is a fundamental tool needed to check the correctness of the developed model and strategy. It is used for proving that the model can reach the performances (even if in a sub-optimal scenario) of a well established implementation. In the preliminary results shown in **Table 1** one can appreciate a comparison portrayed on three different kernels for which a graphical comparison is also present (**Figure 1**, **Figure 2** and **Figure 3**).

The models parameters for each input function are decided in a *Machine Learning* style, using a grid search. The grid search focuses in particular on trying various different optimization parameters, including `eps`, `max iter` and model parameters such as `box (C)` and `epsilon tube`. Once the winner model is derived, the same exact configuration is used for the SkLearn

	Proposed	**SkLearn**	**Proposed**	**SkLearn**	**Proposed**	**SkLearn**
kernel	Linear	Linear	Poly	Poly	RBF	RBF
box (C)	0.1	0.1	0.1	0.1	10.0	10.0
eps	0.1	0.1	0.05	0.05	0.05	0.05
gamma	-	-	1.0638	1.0638	1.1172	1.1172
degree	-	-	2	2	-	-
coef0	-	-	0.3834	0.3834	-	-
max iter	3e3	3e3	5e3	5e3	1e4	1e4
state	stopped	optimal	stopped	stopped	stopped	optimal
fit time	10.0950	0.0040	15.2499	0.0059	15.8779	0.0050
supp vect	100	99	101	96	63	17
f_*	-27.2319	-27.2634	-187.8195	-36.6892	-1.0550	-1.0585

Table 1: Baseline test between sklearn SVR implementation vs proposed models

model generation, so to keep the comparison meaningful.
Analyzing the results provided it is possible to assert a couple of considerations. The SVR implementation from *Scikit-Learn* (see [4]) catches immediately the eye under an important aspect: even if the performances are comparable for the most part, the execution times of the two approaches are far apart. The proposed implementation stands in fact almost four orders of magnitude back with respect to SkLearn. The responsibility can be principally given to three major aspects:

- the proposed implementation is written in `python` while SkLearn utilizes an underlying `C` running environment due to the presence of *LIBSVM*.
- the proposed implementation does not involve possible speed up adjustments that are not part of the required optimization algorithm. SkLearn does instead implement optimizations every time it is possible to speed up execution and convergence.
- the proposed implementation utilizes a subgradient optimization algorithm, much slower with respect to SkLearn, which instead uses the best possible to solve the problem in the best way.

Taking a look at the optimization results one can notice a fairly comparable performance between the proposed model and the SkLearn model. The re-

sults are in fact extremely similar looking at the f_* of the two models, with the only difference stands in the Poly function. The difference in that certain case is due to SkLearn being stopped in its optimization process, which in that case would have needed a lot more steps to converge (maintaining nevertheless a much smaller execution time).

Having given, thanks to these toy examples, a correctness check of the produced software, it is now possible to proceed to the next section, presenting the non trivial experiments.

5.2 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.

5.2.1 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 only the *first dimension* is taken into consideration, focusing on observations on the effect of a change in hyperparameters and the expected convergence behaviour.

5.2.2 Measurements and Expectations

In order to correctly plot the behaviours of the various tests a value for f_* was needed. This value was obtained by running for each kernel a separate instance of the problem using a much higher number of iterations, in particular *2e5*. 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 the aforementioned f_* and the definition reported also in lectures [see 6, 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:

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

5.2.3 Results

Experiments were carried out divided by *max iterations number*. This choice prevailed because the model never managed to converge on the proposed dataset. The algorithm proved itself to be quite slow, therefore a limit on the number of iterations proved to be a solution for bringing results.

The following results will be showing various configurations, varying from changing the *kernel* in use and the parameters of the optimization algorithm such as *alpha* and *eps* (*Algorithm 1*). Two experiments were carried out in particular, with the *max iter* parameter equal to *2e4* and *5e4*, each experiment tested 27 possible configurations. For each of these *max iter* possibilities, the corresponding table shows information on the fitting of the models.

Eventually two figures are also shown for each of these *max iter* possibilities, showing the *rate of convergence* as well as the *logarithmic residual error*.

Figure 4 and **Figure 5** report *convergence rate* and *logarithmic residual error* for the *2e4 max iter* scenario. The *convergence rate* expectations are met as the plot fattens at the value of 1 during the whole process, highlighting once more the expected sublinear convergence rate which can also

be appreciated by looking at the *logarithmic residual error* plot. Looking at the plot one thing that must be considered is the scale of the y axis, which represents the residual error. The values in some cases range between very small quantities, the plot was kept in its original scale to achieve a better visualization that would otherwise lead to a straight line while considering a bigger fixed scale. Focusing on the results reported in **Table 2** some interesting consideration about the behaviour of the algorithm can be drawn. In particular the value of *alpha*, which handles the deflection (8), effects the optimization process in all the experiments. The ones with a lower value of alpha, which means exploiting more the previous direction, tend to converge slower with respect to the configurations where more and more importance is given to the current gradient direction. Another characteristics which is reflected in all the experiments is related to the value of *eps*. In fact, decreasing its value the algorithm has a slower convergence, requiring more iterations to achieve the same minima as the one obtained in configuration with an higher eps. This observation leads to second scenario that was analyzed: *5e4 max iter*. **Table 3** reports results for the *5e4 max iter* case, while **Figure 6** and **Figure 7** shows the plot for the *convergence rate* and *logarithmic residual error*. As previously mentioned, in this case the various configurations of the experiments report a lower value of f_* showing that the optimization process given more step is able to converge towards a better value. As far as concerns possible consideration and analysis, the ones made for the 2e4 case hold in this scenario as well.

5.2.4 Comparison With SkLearn

The previous analysis focused on looking at the effects of varying optimization parameters, a fundamental process to fully understand if the implemented model respects the mathematical theories behind the algorithm, rejoicing for every agreement between theory and practice, giving motivations for any incongruence.

This section does instead focus on a different analysis, a comparison with the well established SkLearn SVR implementation, needed to assess a performance and utility test for the proposed model against a ground truth.

In particular, the previous analysis gave the opportunity to extract the best model per kernel, allowing for a longer fitting of those models, deriving therefore the three *champions* of the proposed implementation. The three champions are compared with identically parametrized SkLearn models, so

max iter	**kernel**	**eps**	**alpha**	**fit time (sec)**	**# supp vect**	f_*
2e4	linear	0.1	0.3	566	1123	-461.1444
			0.5	619	1212	-2560.6844
			0.7	668	1219	**-3071.3413**
		0.01	0.3	699	1201	-13.7185
			0.5	677	1199	-23.9650
			0.7	636	1191	**-33.2939**
		0.005	0.3	648	1184	-3.4673
			0.5	627	1192	-10.0076
			0.7	624	1205	**-16.8539**
	poly	0.1	0.3	759	1219	-4602.1950
			0.5	731	1220	-11515.5291
			0.7	738	1220	**-18767.8895**
		0.01	0.3	656	1215	-62.9900
			0.5	639	1218	-308.7982
			0.7	583	1220	**-488.0803**
		0.005	0.3	615	1213	-5.2767
			0.5	628	1215	-29.1122
			0.7	633	1219	**-120.1288**
	rbf	0.1	0.3	704	1213	-2786.5179
			0.5	719	1219	-6362.9727
			0.7	688	1220	**-12116.4381**
		0.01	0.3	583	1219	-69.2946
			0.5	567	1219	-521.5314
			0.7	618	1220	**-766.8852**
		0.005	0.3	622	1213	-5.3133
			0.5	632	1214	-30.3707
			0.7	616	1220	**-145.6790**

Table 2: Proposed implementation fitting results, 2e4 max iter

to assess correctness and more.
The comparison can be appreciated in **Table 4**, giving rise to some discussion. First of all, it can be noticed how every SkLearn model did not need all the $2e5$ iterations, reaching the optimal state before reaching `max iter`. Second of all, once again, as previously reported in **Section 5.1**, there is a huge execution time difference between the two models, here reaching almost six orders of magnitude of difference. The plausible motivations stand therefore in the different programming languages used, the use of various

max iter	**kernel**	**eps**	**alpha**	**fit time (sec)**	**# supp vect**	f_*
5e4	linear	0.1	0.3	1523	1212	-3092.3348
			0.5	1725	1216	-3134.4915
			0.7	1657	1218	**-3137.3875**
		0.01	0.3	1555	1182	-35.8766
			0.5	1525	1129	-68.8610
			0.7	1427	1119	**-120.5657**
		0.005	0.3	1291	1203	-18.2082
			0.5	1300	1194	-29.6284
			0.7	1274	1169	**-42.8427**
	poly	0.1	0.3	1782	1219	-19736.4481
			0.5	1544	1220	-22223.8363
			0.7	1497	1220	**-22408.9065**
		0.01	0.3	1385	1219	-537.9957
			0.5	1400	1219	-1190.4383
			0.7	1385	1218	**-2051.9656**
		0.005	0.3	1446	1218	-155.4863
			0.5	1257	1219	-418.4509
			0.7	1254	1220	**-674.8039**
	rbf	0.1	0.3	1351	1218	-13455.1430
			0.5	1484	1219	-18315.4798
			0.7	1446	1220	**-18778.9937**
		0.01	0.3	1426	1220	-808.6225
			0.5	1434	1216	-1157.5535
			0.7	1325	1217	**-1524.8936**
		0.005	0.3	1369	1220	-201.2898
			0.5	1362	1220	-696.6714
			0.7	1534	1220	**-902.5939**

Table 3: Proposed implementation fitting results, 5e4 max iter

heuristics and optimization from SkLearn and the different optimization algorithm (therefore with a much different convergence rate). Finally, taking a look at the optimal values reached, the comparison between the models clearly show how the proposed model is valid, giving incredibly near results to the ones derived by SkLearn.

	Proposed	**SkLearn**	**Proposed**	**SkLearn**	**Proposed**	**SkLearn**
kernel	Linear	Linear	Poly	Poly	RBF	RBF
box (C)	1.0	1.0	10.0	10.0	10.0	10.0
eps	1.0	1.0	0.1	0.1	0.1	0.1
gamma	-	-	0.075	0.075	0.1	0.1
degree	-	-	3	3	-	-
coef0	-	-	0.28	0.28	-	-
max iter	2e5	2e5	2e5	2e5	2e5	2e5
state	stopped	optimal	stopped	optimal	stopped	optimal
fit time	5152.0997	0.054	4630.1994	0.095	4919.8894	0.0869
supp vect	1216	1017	1220	1183	1218	1179
f*	-3138.9241	-3138.9592	-22463.3411	-22463.8118	-18916.4973	-18917.8941

Table 4: Performances of sklearn SVR implementation vs best performing proposed models

5.3 Scalability Analysis

Before drawing the conclusions it is possible to analyze another aspect of the proposed implementation: the scalability with respect to different amount of inputs and inputs dimensionalities.

Starting from the scalability analysis on the amount of inputs, **Table 5** and **Figure 8 (a)** show the experiment results. For each tested amount of inputs the reported time is an average taken from five executions, to have a more robust result. It can be noticed how the model fitting time scales linearly with the increasing of inputs amount.

inputs	**fit time**
1e2	3.3326
2e2	5.6530
5e2	13.3949
1e3	29.6346
2e3	62.7350
5e3	177.0587

Table 5: Scalability test - inputs

Looking instead at the scalability analysis on the input dimensionality,

Table 6 and **Figure 8 (b)** show the experiment results. Once again, for each tested input dimensionality the reported time is an average taken from five executions, to have a more robust result. One might get a little startled at first by the reported results. The model fitting time does in fact not scale with the dimensionality, remaining pretty much the same varying from 1 input dimension to 5000 input dimensions. This can be easily explained with the optimization algorithm: since the proposed implementation follows a dual approach, the deflected subgradient does not directly use the inputs but their lagrangian multipliers (in our case not even directly the lagrangian multipliers but an agglomeration of lagrangian multiplier couples to achieve the function non-linearity), which are unidimensional no matter the input dimensionality. This mapping makes it so that no impact is left on the fitting, whichever the original input dimensionality is.

features	**fit time**
1	12.5656
2	12.6406
5	12.6993
1e1	12.8860
2e1	12.9466
5e1	12.7858
1e2	12.6711
2e2	12.4675
5e2	12.3403
1e3	13.1008
2e3	12.9759
5e3	13.2751

Table 6: Scalability test - features

6 Conclusions

The proposed implementation proves itself valid, by passing a correctness check on some trivial data, showing anyhow its slowness with respect to a well established distributed software such as *SkLearn*. These differences are mostly related to a more optimized implementation, that with a refactor of the code (i.e. using `C++`) could possibly be overcome. However, in **Section 5.2.4** the comparison of the optimal value between the two different implementation showed that the proposed algorithm is able to reach a minima which is very close to the one of SKLearn. This suggests that this methodology can possibly become a valid alternative in terms of optimality, obviously it needs a faster implementation in order to be truly competitive.
For all the experiments, the convergence rate of the *Constrained Deflected Subgradient Method* using *Target Value Stepsize* with a *Non-Vanishing Threshold* is met, upholding the theoretical analysis provided in **Section 3**.
To wrap it up on a personal note, such a mathematically-oriented implementation has been a great challenge with respect to working on modelling just for the sake of results. The caution, care and precision needed to follow the rigorousness lead us to the development of a different point of view even when working on other projects, leading the way to an effective personal growth.

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] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[4] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[5] 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.

[6] 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.

[7] 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

$$\begin{aligned}\mathcal{L} = \frac{1}{2}\left\|w\right\|^2 + C\sum_i(\xi_i + \xi_i^*) &+ \sum_i \alpha_i(y_i - w\phi_i - b - \varepsilon - \xi_i) \\ &+ \sum_i \alpha_i(-y_i + w\phi_i - b - \varepsilon - \xi_i^*) \\ &- \sum_i \mu_i\xi_i \\ &- \sum_i \mu_i^*\xi_i^* \\ \textit{where} \quad \forall_i\, \xi_i\xi_i^* \geq 0\end{aligned} \tag{19}$$

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.

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

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

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

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

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

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

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

$$\begin{aligned}
\mathcal{L}(\alpha, \alpha^*)\ =\ &-\frac{1}{2}\sum_i\sum_j(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\phi_i\phi_j \\
&+ \sum_i(\alpha_i - \alpha_i^*)y_i \\
&- \sum_i(\alpha_i + \alpha_i^*)\varepsilon \\
&\textit{With the constraints} \qquad \begin{cases} \sum_i(\alpha_i^* - \alpha_i) = 0 \\ \alpha_i \in [0,\ C] \\ \alpha_i^* \in [0,\ C] \end{cases}
\end{aligned}$$

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^*

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

10 Figures

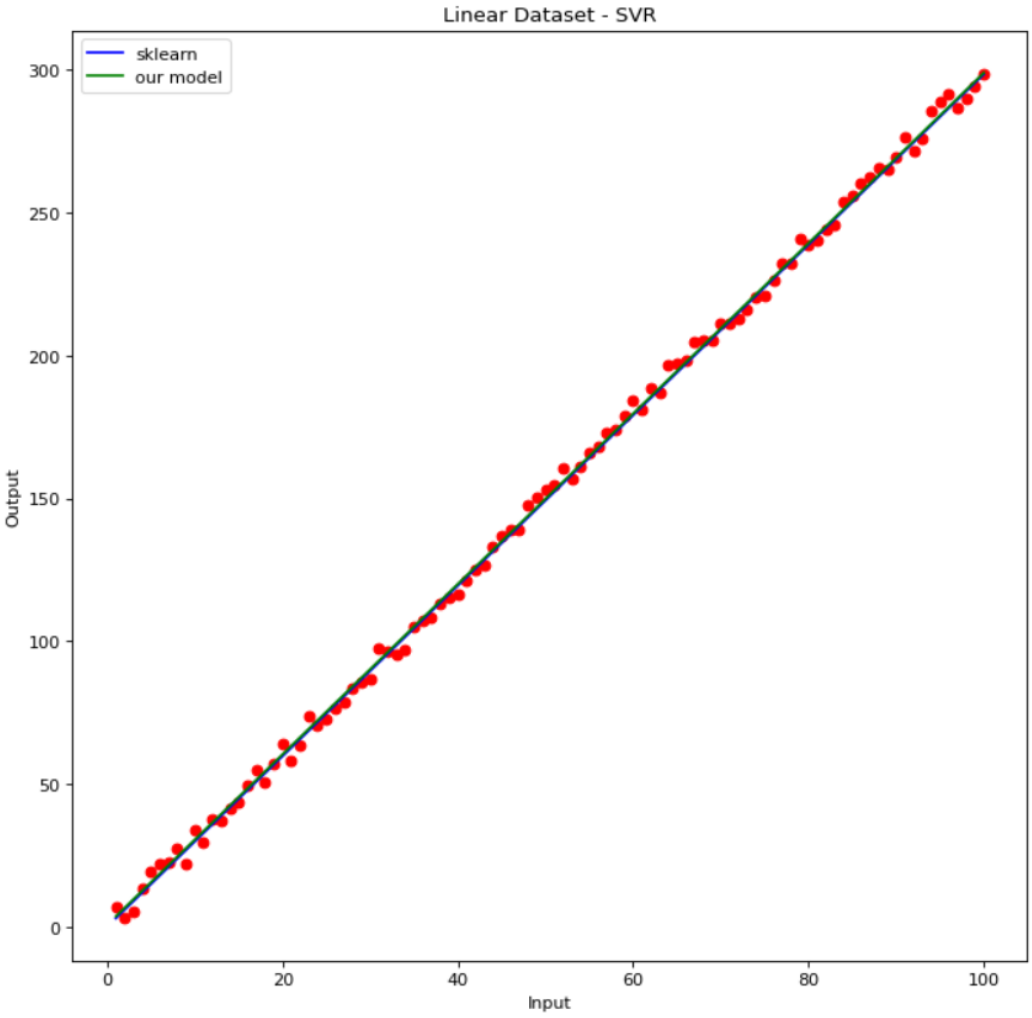


Figure 1: Comparison between the proposed SVR implementation and SKLearn on an easy dataset - proving the implementation is working

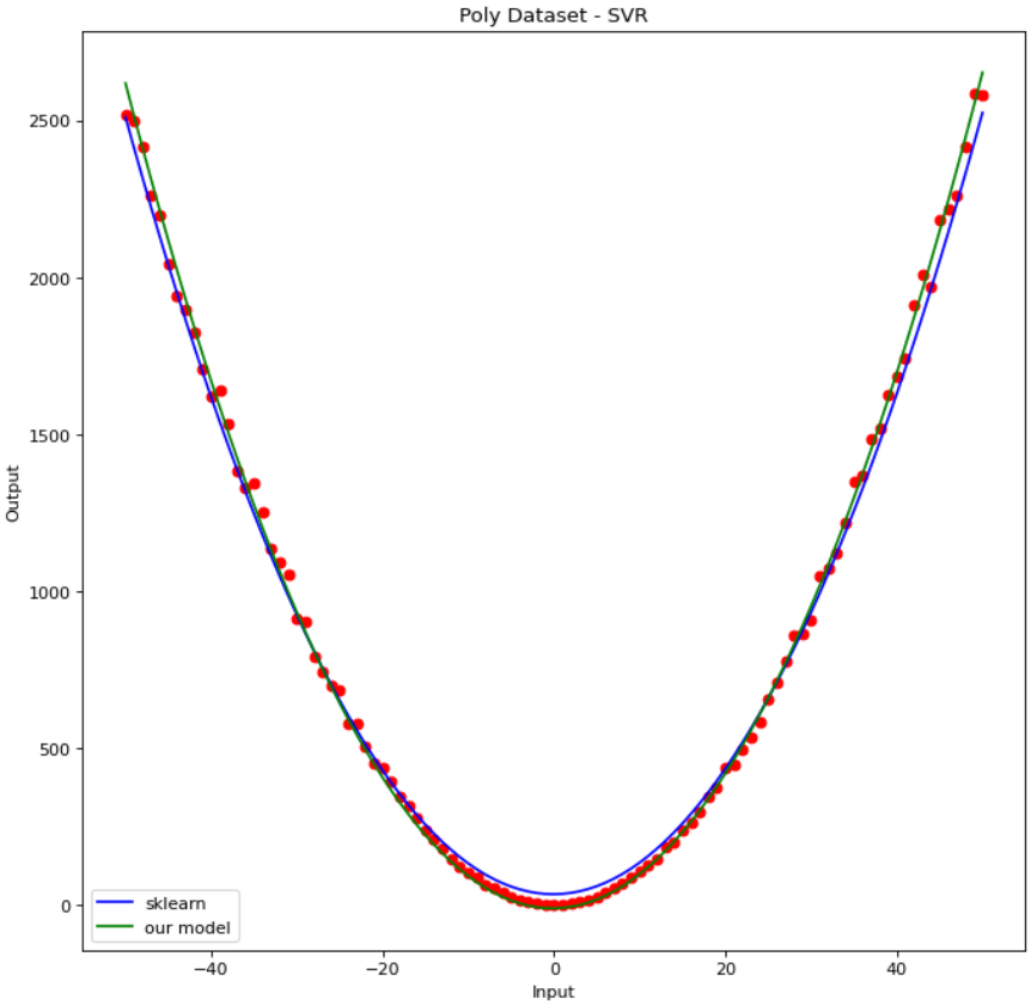


Figure 2: Comparison between the proposed SVR implementation and SKLearn on an easy dataset - proving the implementation is working

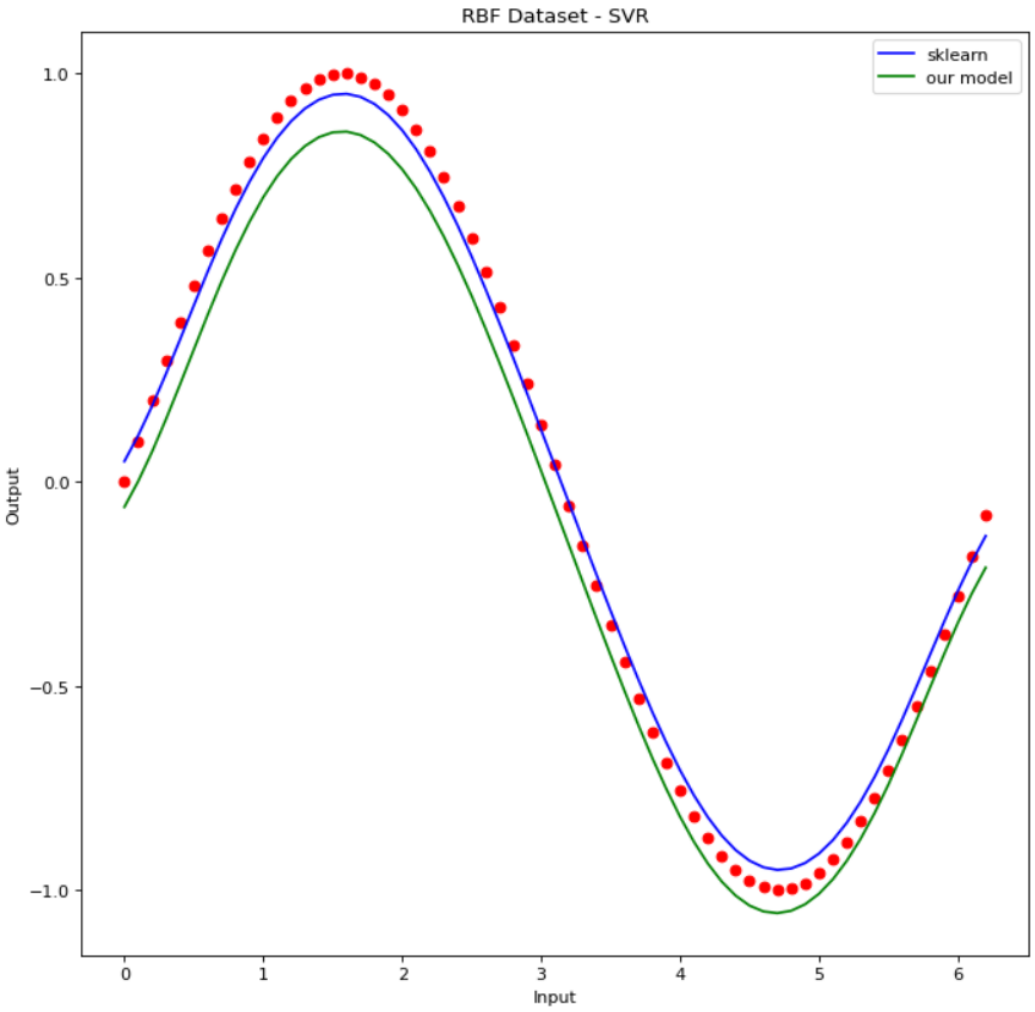


Figure 3: Comparison between the proposed SVR implementation and SKLearn on an easy dataset - proving the implementation is working

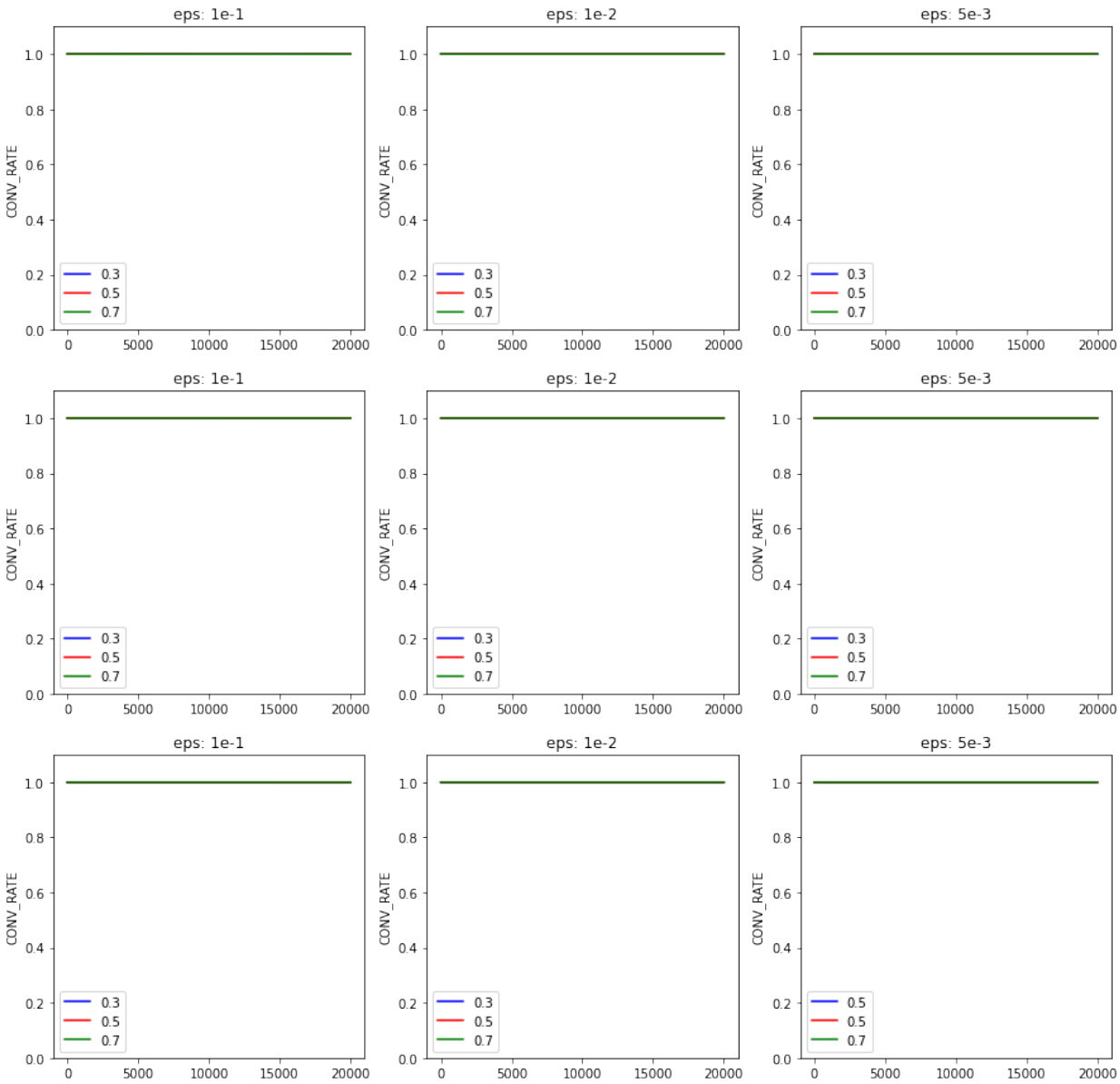


Figure 4: Convergence rate - 2e4 max iter
Each row uses one possible kernel, top to bottom: *Linear*, *Poly* and *RBF*

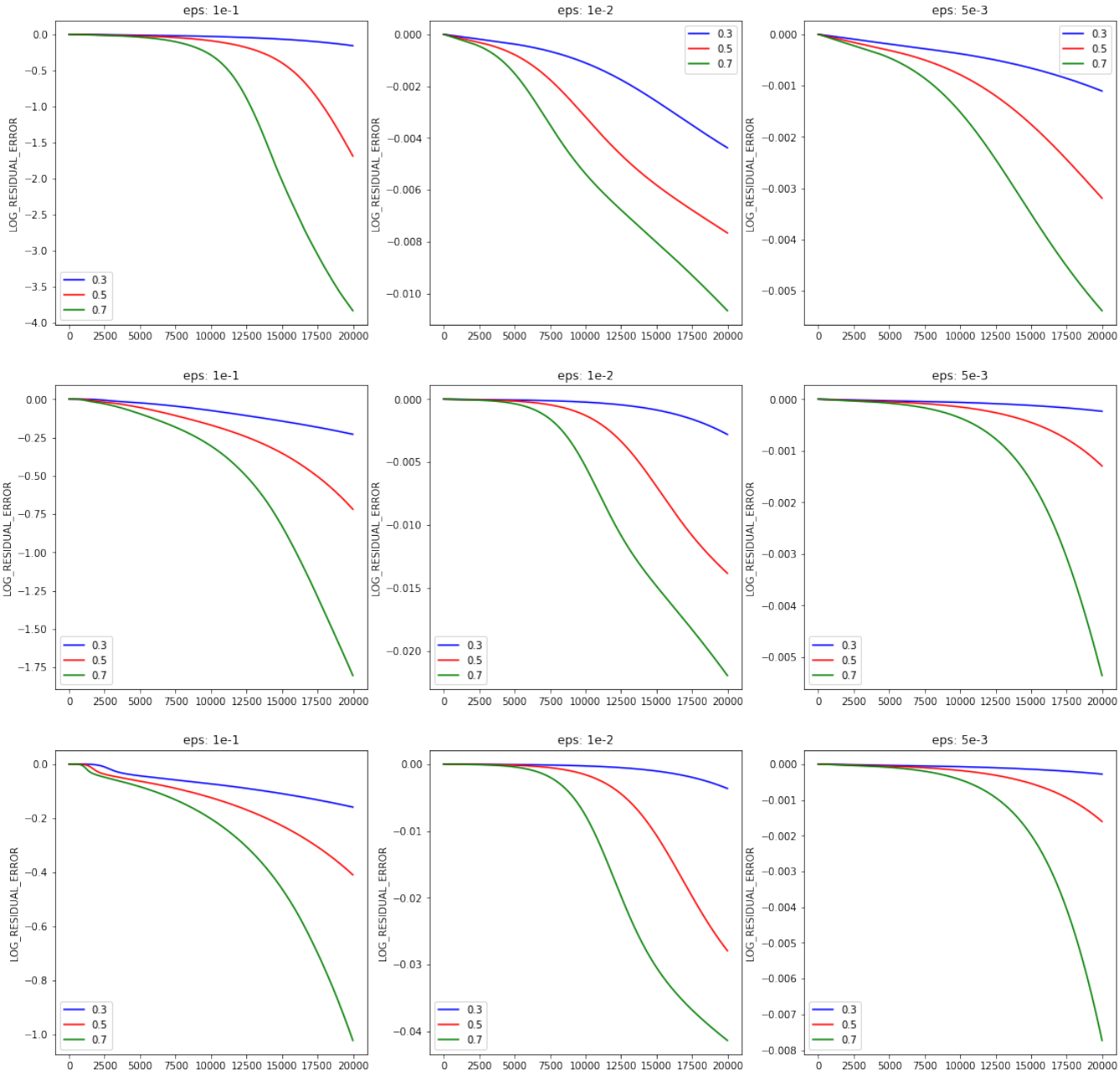


Figure 5: Logarithmic residual error - 2e4 max iter
Each row uses one possible kernel, top to bottom: *Linear*, *Poly* and *RBF*

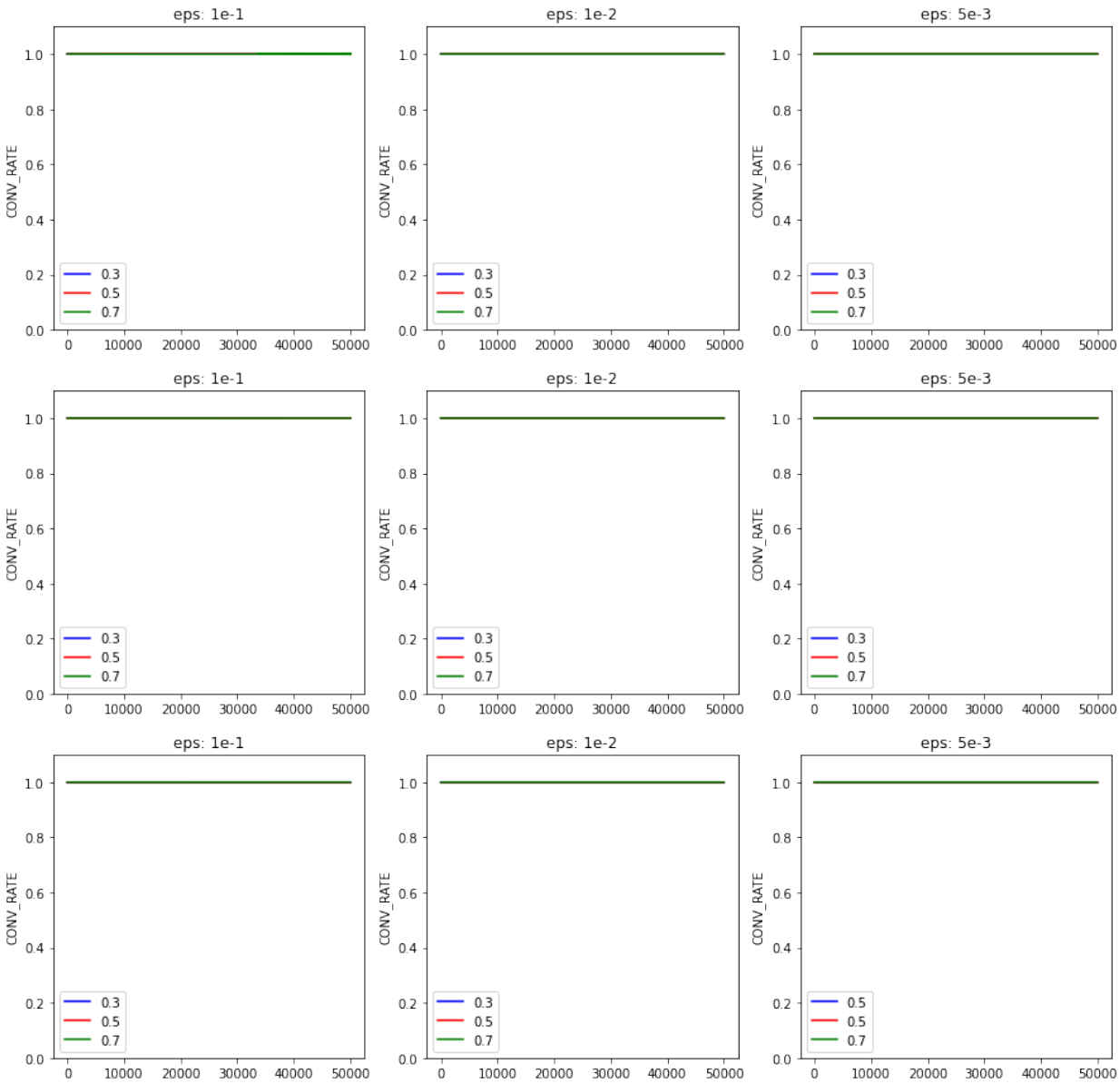


Figure 6: Convergence rate - 5e4 max iter
Each row uses one possible kernel, top to bottom: *Linear*, *Poly* and *RBF*

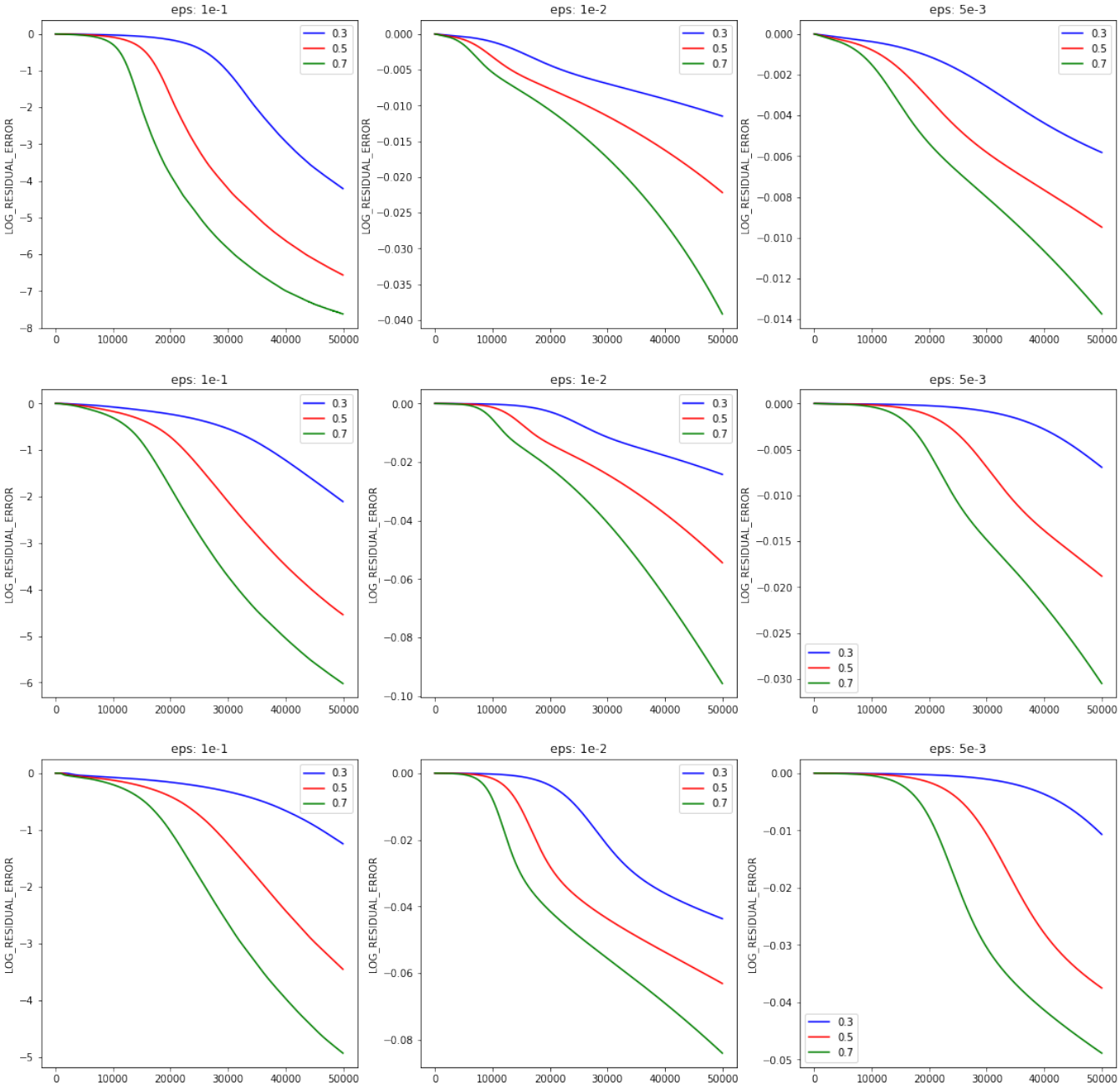
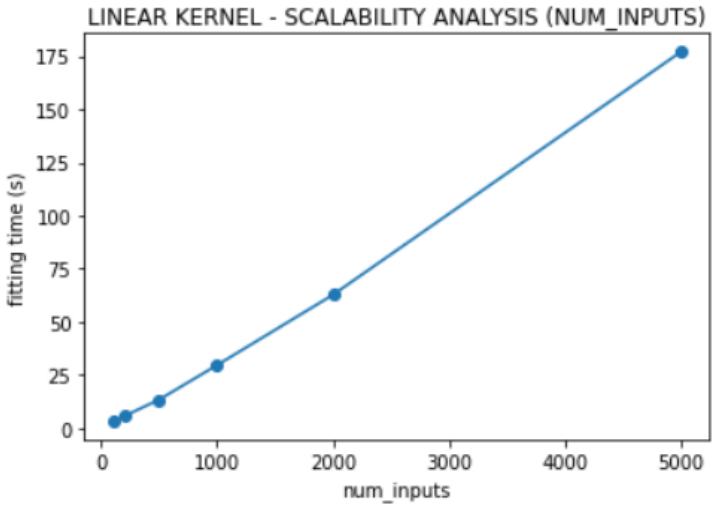
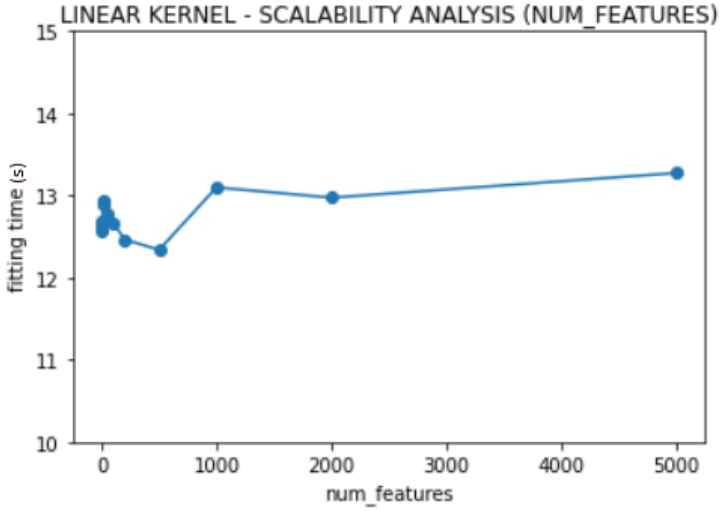


Figure 7: Logarithmic residual error - 5e4 max iter
Each row uses one possible kernel, top to bottom: *Linear*, *Poly* and *RBF*



(a) Scalability on inputs



(b) Scalability on input dimensionality

Figure 8: Scalability plots