University of Pisa

Department of Computer Science

Computational Mathematics
Wildcard Project nr. 5 with Machine Learning
Group 35

Support Vector Machines

Author:

Donato Meoli
d.meoli@studenti.unipi.it

June, 2021

Contents

List of Figures

List of Tables

List of Algorithms

List of Theorems

1 Track

`(M1.1)` is a *Support Vector Classifier (SVC)* with the *hinge* loss.

- `(A1.1.1)` is a *momentum descent* approach [1, 2, 3], an *accelerated gradient* method for solving the SVC in its *primal* formulation.
- `(A1.1.2)` is the *Sequential Minimal Optimization (SMO)* algorithm [4, 5], an ad hoc *active set* method for training a SVC in its *Wolfe dual* formulation with *linear*, *polynomial* and *gaussian* kernels.
- `(A1.1.3)` is the *AdaGrad* algorithm [6], a *deflected subgradient* method for solving the SVC in its *Lagrangian dual* formulation with *linear*, *polynomial* and *gaussian* kernels.

`(M1.2)` is a *Support Vector Classifier (SVC)* with the *squared hinge* loss.

- `(A1.2.1)` is a *momentum descent* approach [1, 2, 3], an *accelerated gradient* method for solving the SVC in its *primal* formulation.
- `(A1.2.2)` is the *AdaGrad* algorithm [6], a *deflected subgradient* method for solving the SVC in its *Lagrangian dual* formulation with *linear*, *polynomial* and *gaussian* kernels.

`(M2.1)` is a *Support Vector Regression (SVR)* with the *epsilon-insensitive* loss.

- `(A2.1.1)` is a *momentum descent* approach [1, 2, 3], an *accelerated gradient* method for solving the SVR in its *primal* formulation.
- `(A2.1.2)` is the *Sequential Minimal Optimization (SMO)* algorithm [7, 8], an ad hoc *active set* method for training a SVR in its *Wolfe dual* formulation with *linear*, *gaussian* and *laplacian* kernels.
- `(A2.1.3)` is the *AdaGrad* algorithm [6], a *deflected subgradient* method for solving the SVR in its *Lagrangian dual* formulation with *linear*, *gaussian* and *laplacian* kernels.

`(M2.2)` is a *Support Vector Regression (SVR)* with the *squared epsilon-insensitive* loss.

- `(A2.2.1)` is a *momentum descent* approach [1, 2, 3], an *accelerated gradient* method for solving the SVR in its *primal* formulation.
- `(A2.2.2)` is the *AdaGrad* algorithm [6], a *deflected subgradient* method for solving the SVR in its *Lagrangian dual* formulation with *linear*, *gaussian* and *laplacian* kernels.

2 Abstract

A *Support Vector Machine* is a learning model used both for *classification* and *regression* tasks whose goal is to construct a *maximum margin separator*, i.e., a decision boundary with the largest distance from the nearest training data points.

The aim of this report is to compare the *primal*, the *Wolfe dual* [9] and the *Lagrangian dual* formulations of this model in terms of *numerical precision*, *accuracy* and *complexity*.

Firstly, I will provide a detailed mathematical derivation of the model for all these formulations, then I will propose two algorithms to solve the optimization problem in case of *constrained* or *unconstrained* formulation of the problem, explaining their theoretical properties, i.e., *convergence* and *complexity*.

Finally, I will show some experiments for *linearly* and *nonlinearly* separable generated datasets to compare the performance of different *kernels*, also by comparing the *custom* results with *sklearn* SVM implementations, i.e., *liblinear* [10] and *libsvm* [11] implementations, and *cvxopt* [12] QP solver.

3 Linear Support Vector Classifier

Given n training points, where each input x_i has m attributes, i.e., is of dimensionality m, and is in one of two classes $y_i = \pm 1$, i.e., our training data is of the form:

$$\{(x_i, y_i), x_i \in \Re^m, y_i = \pm 1, i = 1, \ldots, n\} \tag{1}$$

For simplicity we first assume that data are (not fully) linearly separable in the input space x, meaning that we can draw a line separating the two classes when $m = 2$, a plane for $m = 3$ and, more in general, a hyperplane for an arbitrary m.

Support vectors are the examples closest to the separating hyperplane and the aim of support vector machines is to orientate this hyperplane in such a way as to be as far as possible from the closest members of both classes, i.e., we need to maximize this margin.

This hyperplane is represented by the equation $w^T x + b = 0$. So, we need to find w and b so that our training data can be described by:

$$\begin{aligned} &w^T x_i + b \geq +1 - \xi_i, \forall y_i = +1 \\ &w^T x_i + b \leq -1 + \xi_i, \forall y_i = -1 \\ &\xi_i \geq 0 \ \forall_i \end{aligned} \tag{2}$$

where the positive slack variables ξ_i are introduced to allow misclassified points. In this way data points on the incorrect side of the margin boundary will have a penalty that increases with the distance from it.

These two equations can be combined into:

$$\begin{aligned} &y_i(w^T x_i + b) \geq 1 - \xi_i \ \forall_i \\ &\xi_i \geq 0 \ \forall_i \end{aligned} \tag{3}$$

The margin is equal to $\dfrac{1}{\|w\|}$ and maximizing it subject to the constraint in (3) while as we are trying to reduce the number of misclassifications is equivalent to finding:

$$\begin{aligned} \min_{w,b,\xi} \quad & \|w\| + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \ \forall_i \\ & \xi_i \geq 0 \ \forall_i \end{aligned} \tag{4}$$

Minimizing $\|w\|$ is equivalent to minimizing $\frac{1}{2}\|w\|^2$, but in this form we will deal with a 1-strongly convex regularization term that has more desirable convergence properties. So we need to find:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i \ \forall_i \\ & \xi_i \geq 0 \ \forall_i \end{aligned} \tag{5}$$

where the parameter C controls the trade-off between the slack variable penalty and the size of the margin.

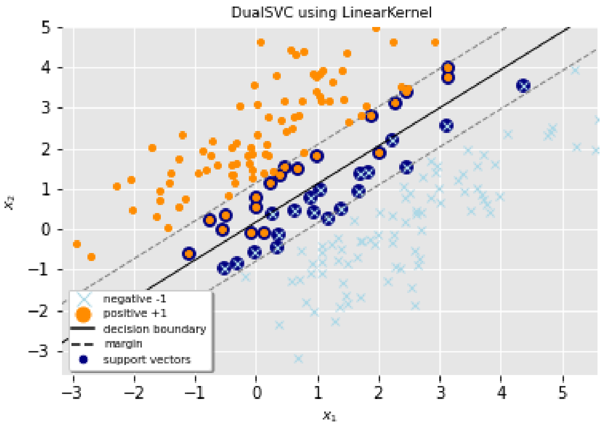


Figure 1: Linear SVC hyperplane

3.1 Hinge loss

The *hinge* loss is defined as:

$$\mathcal{L}_1 = \max(0, 1 - y(w^T x + b)) \tag{6}$$

or, equivalently:

$$\mathcal{L}_1 = \begin{cases} 0 & \text{if } y(w^T x + b) \geq 1 \\ 1 - y(w^T x + b) & \text{otherwise} \end{cases} \tag{7}$$

and it is a nondifferentiable convex function due to its nonsmoothness in 1, but has a subgradient that is given by:

$$\partial_w \mathcal{L}_1 = \begin{cases} -yx & \text{if } y(w^T x + b) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

3.1.1 Primal formulation

The general primal unconstrained formulation takes the form:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \mathcal{L}(w, b; x_i, y_i) \tag{9}$$

where $\frac{1}{2}\|w\|^2$ is the *regularization term* and $\mathcal{L}(w, b; x_i, y_i)$ is the *loss function* associated with the observation (x_i, y_i) [13].

The quadratic optimization problem (5) can be equivalently formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b)) \tag{10}$$

where we make use of the *hinge* loss (6) or (7).

The above formulation penalizes slacks ξ linearly and is called $\mathcal{L}_1$-SVC.

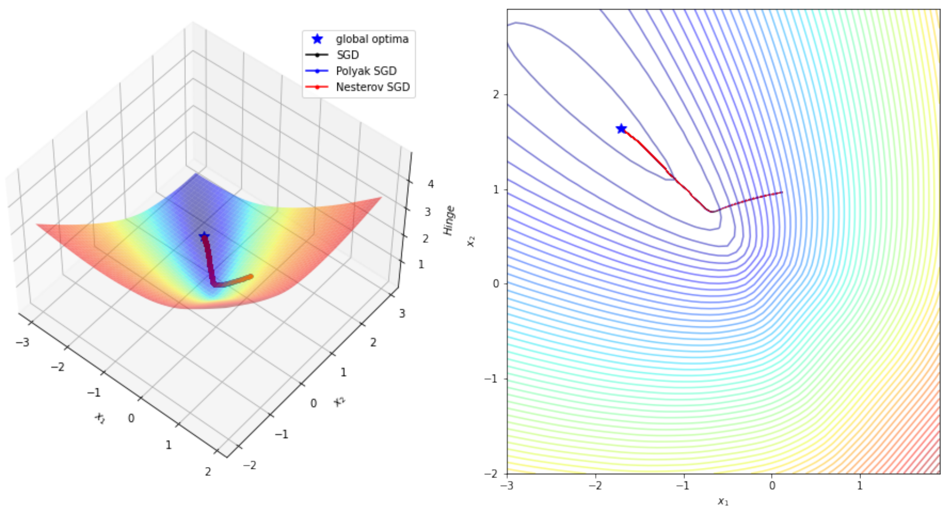


Figure 2: Hinge loss with different optimization steps

To simplify the notation and so also the design of the algorithms, the simplest approach to learn the bias term b is that of including that into the *regularization term*; so we can rewrite (9) as follows:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C \sum_{i=1}^{n} \mathcal{L}(w, b; x_i, y_i) \tag{11}$$

or, equivalently, by augmenting the weight vector w with the bias term b and each instance x_i with an additional dimension, i.e., with constant value equal to 1:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|\hat{w}\|^2 + C \sum_{i=1}^{n} \mathcal{L}(\hat{w}; \hat{x}_i, y_i) \\ \text{where} \quad & \hat{w}^T = [w^T, b] \\ & \hat{x}_i^T = [x_i^T, 1] \end{aligned} \tag{12}$$

with the advantages of having convex properties of the objective function useful for convergence analysis and the possibility to directly apply algorithms designed for models without the bias term.

In the specific case of the $\mathcal{L}_1$-SVC the objective (10) become:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b)) \tag{13}$$

Note that in terms of numerical optimization the formulation (10) is not equivalent to (13) since in the first one the bias term b does not contribute to the *regularization term*, so the SVM formulation is based on an unregularized bias term b, as highlighted by the *statistical learning theory*. But, in machine learning sense, numerical experiments in [14] show that the accuracy does not vary much when the bias term b is embedded into the weight vector w.

3.1.2 Wolfe Dual formulation

To reformulate the (5) as a *Wolfe dual*, we need to allocate the Lagrange multipliers $\alpha_i, \mu_i \geq 0 \ \forall_i$:

$$\max_{\alpha,\mu} \min_{w,b,\xi} \mathcal{W}(w,b,\xi,\alpha,\mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(w^Tx_i + b) - 1 + \xi_i) - \sum_{i=1}^{n}\mu_i\xi_i \tag{14}$$

We wish to find the w, b and ξ_i which minimizes, and the α and μ which maximizes $\mathcal{W}$, provided $\alpha_i \geq 0, \mu_i \geq 0 \ \forall_i$. We can do this by differentiating $\mathcal{W}$ wrt w and b and setting the derivatives to 0:

$$\frac{\partial \mathcal{W}}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i \Rightarrow w = \sum_{i=1}^{n}\alpha_i y_i x_i \tag{15}$$

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^{n}\alpha_i y_i \Rightarrow \sum_{i=1}^{n}\alpha_i y_i = 0 \tag{16}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \mu_i \tag{17}$$

Substituting (15) and (16) into (14) together with $\mu_i \geq 0 \ \forall_i$, which implies that $\alpha \leq C$, gives a new formulation being dependent on α. We therefore need to find:

$$\begin{aligned}
\max_{\alpha} \mathcal{W}(\alpha) &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \langle x_i, x_j\rangle \\
&= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i Q_{ij}\alpha_j \text{ where } Q_{ij} = y_i y_j \langle x_i, x_j\rangle \\
&= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\alpha^T Q\alpha \text{ subject to } 0 \leq \alpha_i \leq C \ \forall_i, \sum_{i=1}^{n}\alpha_i y_i = 0
\end{aligned} \tag{18}$$

or, equivalently:

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha \\
\text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \\
& y^T\alpha = 0
\end{aligned} \tag{19}$$

where $q^T = [1, \ldots, 1]$.
By solving (19) we will know α and, from (15), we will get w, so we need to calculate b.
We know that any data point satisfying (16) which is a support vector x_s will have the form:

$$y_s(w^T x_s + b) = 1 \tag{20}$$

and, by substituting in (15), we get:

$$y_s\big(\sum_{m\in S}\alpha_m y_m\langle x_m, x_s\rangle + b\big) = 1 \tag{21}$$

where s denotes the set of indices of the support vectors and is determined by finding the indices i where $\alpha_i > 0$, i.e., nonzero Lagrange multipliers.
Multiplying through by y_s and then using $y_s^2 = 1$ from (2):

$$y_s^2\big(\sum_{m\in S}\alpha_m y_m\langle x_m, x_s\rangle + b\big) = y_s \tag{22}$$

$$b = y_s - \sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle \tag{23}$$

Instead of using an arbitrary support vector x_s, it is better to take an average over all of the support vectors in S:

$$b = \frac{1}{N_s} \sum_{s \in S} y_s - \sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle \tag{24}$$

We now have the variables w and b that define our separating hyperplane's optimal orientation and hence our support vector machine. Each new point x' is classified by evaluating:

$$y' = \text{sign}\big(\sum_{i=1}^{n} \alpha_i y_i \langle x_i, x' \rangle + b \big) \tag{25}$$

From (19) we can notice that the equality constraint $y^T\alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. We report below the box-constrained dual formulation [14] that arises from the primal (11) or (12) where the bias term b is embedded into the weight vector w:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + yy^T)\alpha + q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \end{aligned} \tag{26}$$

3.1.3 Lagrangian Dual formulation

In order to relax the constraints in the *Wolfe dual* formulation (19) we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrange multipliers μ and $\lambda_+, \lambda_- \geq 0$:

$$\begin{aligned} \max_{\mu, \lambda_+, \lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T Q\alpha + q^T\alpha - \mu^T(y^T\alpha) - \lambda_+^T(ub - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T Q\alpha + (q - \mu y + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T ub \\ \text{subject to} \quad & \lambda_+, \lambda_- \geq 0 \end{aligned} \tag{27}$$

where the upper bound $ub^T = [C, \dots, C]$.
Taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow Q\alpha + (q - \mu y + \lambda_+ - \lambda_-) = 0 \tag{28}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu y + \lambda_+ - \lambda_-) \tag{29}$$

the gradients wrt μ, λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -y\alpha \tag{30}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{31}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{32}$$

From (19) we can notice that the equality constraint $y^T\alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. In this way the dimensionality of (27) is reduced by removing the multipliers μ which was allocated to control the equality constraint $y^T\alpha = 0$, so we will end up solving exactly the problem (26).

$$\begin{aligned} \max_{\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T(Q + yy^T)\alpha + q^T\alpha - \lambda_+^T(ub - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T(Q + yy^T)\alpha + (q + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T ub \\ \text{subject to} \quad & \lambda_+, \lambda_- \geq 0 \end{aligned} \tag{33}$$

where, again, the upper bound $ub^T = [C, \dots, C]$.
Now, taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + yy^T)\alpha + (q + \lambda_+ - \lambda_-) = 0 \tag{34}$$

With α optimal solution of the linear system:

$$(Q + yy^T)\alpha = -(q + \lambda_+ - \lambda_-) \tag{35}$$

the gradients wrt λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - ub \tag{36}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{37}$$

Note that since the Hessian matrix Q of the $\mathcal{L}_1$-SVC is not strictly positive definite, i.e., the Lagrangian function is not strictly convex since it will be linear along the eigenvectors correspondent to the null eigenvalues and so it will be unbounded below, the Lagrangian dual relaxation, i.e., 29 and 35, will be nondifferentiable, so it will have infinite solutions and for each of them it will have a different subgradient. In order to compute an approximation of the gradient, we will choose α in such a way as the one that minimizes the 2-norm since it is good almost like the gradient:

$$\min_{\alpha_n \in K_n(Q,b)} \|Q\alpha_n - b\| \tag{38}$$

Since we are dealing with a symmetric system we will choose a well-known Krylov method that performs the Lanczos iterate, i.e., symmetric Arnoldi iterate, called *minres*, i.e., symmetric *gmres*, to compute the vector α_n that minimizes the norm of the residual $r_n = Q\alpha_n - b$ among all vectors in $K_n(Q, b) = span(b, Qb, Q^2b, \dots, Q^{n-1}b)$.

Since the linear algebra methods in the ML context are crucial and also in order to deal with a per-iteration cost equals to the other algorithms described later to provide a coherent comparison of all at the end, we will solve it with a primal-dual optimization method and we modify its definition by adding a strictly convex augmentation term, i.e., a penalty term, in order to improve the practical convergence of the algorithms. So, if we consider a general quadratic optimization problem subject to linear constraints, i.e., equality and inequality constraints, defined as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha \\ \text{subject to} \quad & A\alpha = b \\ & G\alpha \leq h \\ & lb \leq \alpha \leq ub \end{aligned} \tag{39}$$

or, equivalently:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha \\ \text{subject to} \quad & A\alpha = b \\ & \hat{G}\alpha \leq \hat{h} \end{aligned} \tag{40}$$

where $\hat{G} = \begin{bmatrix} G \\ -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} h & -lb & ub \end{bmatrix}$; we give the following *augmented Lagrangian dual*:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha + \mu^T(A\alpha - b) + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|A\alpha - b\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{41}$$

with $\rho > 0$.

According to this definition, we change the formulation 27 as:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda) = & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha + \mu^T(y^T\alpha) + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|y^T\alpha\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{42}$$

and the formulation 33 as:

$$\begin{aligned} \max_{\lambda} \min_{\alpha} \mathcal{L}(\alpha,\lambda) = & \frac{1}{2}\alpha^T(Q + yy^T)\alpha + q^T\alpha + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{43}$$

where $\hat{G} = \begin{bmatrix} -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} -lb & ub \end{bmatrix}$ with $lb^T = [0, \dots, 0]$, $ub^T = [C, \dots, C]$ and $\rho > 0$.

3.2 Squared Hinge loss

The *squared hinge* loss is defined as:

$$\mathcal{L}_2 = \max(0, 1 - y(w^T x + b))^2 \tag{44}$$

or, equivalently:

$$\mathcal{L}_2 = \begin{cases} 0 & \text{if } y(w^T x + b) \geq 1 \\ (1 - y(w^T x + b))^2 & \text{otherwise} \end{cases} \tag{45}$$

It is a strictly convex function and its gradient is given by:

$$\nabla_w \mathcal{L}_2 = \begin{cases} -2\max(0, 1 - y(w^T x + b))yx & \text{if } y(w^T x + b) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

3.2.1 Primal formulation

Since smoothed versions of objective functions may be preferred for optimization, we can reformulate (10) as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b))^2 \tag{47}$$

where we make use of the *squared hinge* loss that quadratically penalized slacks ξ and is called $\mathcal{L}_2$-SVC. The $\mathcal{L}_2$-SVC objective (47) can be rewritten in form (11) or (12) as:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b))^2 \tag{48}$$

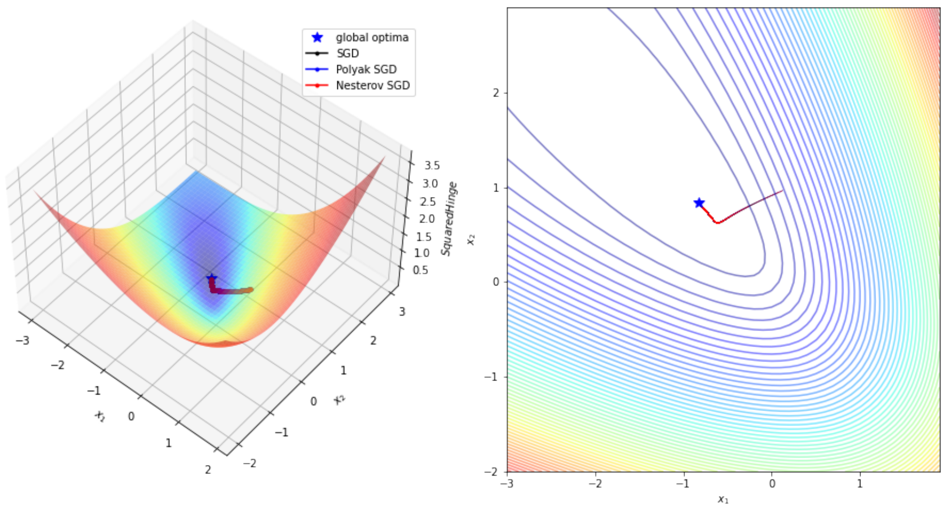


Figure 3: Squared Hinge loss with different optimization steps

3.2.2 Wolfe Dual formulation

As done for the $\mathcal{L}_1$-SVC we can derive the *Wolfe dual* formulation of the $\mathcal{L}_2$-SVC by obtaining:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q+D)\alpha + q^T\alpha \\ \text{subject to} \quad & \alpha_i \geq 0 \ \forall_i \\ & y^T\alpha = 0 \end{aligned} \tag{49}$$

or, alternatively, with the regularized bias term by obtaining:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q+yy^T+D)\alpha + q^T\alpha \\ \text{subject to} \quad & \alpha_i \geq 0 \ \forall_i \end{aligned} \tag{50}$$

where the diagonal matrix $D_{ii} = \dfrac{1}{2C} \ \forall_i$.

3.2.3 Lagrangian Dual formulation

In order to relax the constraints in the $\mathcal{L}_2$-SVC *Wolfe dual* formulation (49) we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrange multipliers μ and $\lambda \geq 0$:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda) &= \frac{1}{2}\alpha^T(Q+D)\alpha + q^T\alpha - \mu^T(y^T\alpha) - \lambda^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+D)\alpha + (q - \mu y - \lambda)^T\alpha \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{51}$$

Taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q+D)\alpha + (q - \mu y - \lambda) = 0 \tag{52}$$

With α optimal solution of the linear system:

$$(Q+D)\alpha = -(q - \mu y - \lambda) \tag{53}$$

the gradients wrt μ and λ are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -y\alpha \tag{54}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\alpha \tag{55}$$

From (19) we can notice that the equality constraint $y^T\alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. In this way the dimensionality of (51) is reduced by removing the multipliers μ which was allocated to control the equality constraint $y^T\alpha = 0$, so we will end up solving exactly the problem (50).

$$\begin{aligned} \max_{\lambda} \min_{\alpha} \mathcal{L}(\alpha,\lambda) &= \frac{1}{2}\alpha^T(Q+yy^T+D)\alpha + q^T\alpha - \lambda^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+yy^T+D)\alpha + (q-\lambda)^T\alpha \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{56}$$

Now, taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + yy^T + D)\alpha + (q - \lambda) = 0 \tag{57}$$

With α optimal solution of the linear system:

$$(Q + yy^T + D)\alpha = -(q - \lambda) \tag{58}$$

the gradient wrt λ is:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\alpha \tag{59}$$

Note that since the Hessian matrix Q of the $\mathcal{L}_2$-SVC is symmetric and strictly positive definite, we can find the unique solution of the Lagrangian dual relaxation, i.e., 53 and 58, by solving the system with the Cholesky factorization.

Since the linear algebra methods in the ML context are crucial and also in order to deal with a per-iteration cost equals to the other algorithms described later to provide a coherent comparison of all at the end, we will solve it with a primal-dual optimization method and we modify its definition by adding a strictly convex augmentation term, i.e., a penalty term, in order to improve the practical convergence of the algorithms. So, if we consider a general quadratic optimization problem subject to linear constraints, i.e., equality and inequality constraints, defined as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & G\alpha \leq h \\ & lb \leq \alpha \leq ub \end{aligned} \tag{60}$$

or, equivalently:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & \hat{G}\alpha \leq \hat{h} \end{aligned} \tag{61}$$

where $\hat{G} = \begin{bmatrix} G \\ -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} h & -lb & ub \end{bmatrix}$; we give the following *augmented Lagrangian dual*:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha + \mu^T (A\alpha - b) + \lambda^T (\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|A\alpha - b\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{62}$$

with $\rho > 0$.

According to this definition, we change the formulation 51 as:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda) = & \frac{1}{2}\alpha^T (Q + D) \alpha + q^T \alpha + \mu^T (y^T \alpha) + \lambda^T (\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|y^T \alpha\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{63}$$

and the formulation 56 as:

$$
\begin{aligned}
&\max_{\lambda} \min_{\alpha} \mathcal{L}(\alpha, \lambda) = \frac{1}{2}\alpha^T (Q + yy^T + D)\alpha + q^T \alpha + \lambda^T (\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\
&\quad \text{subject to} \quad \lambda \geq 0
\end{aligned} \tag{64}
$$

where $\hat{G} = \begin{bmatrix} -I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} -lb \end{bmatrix}$ with $lb^T = [0, \dots, 0]$ and $\rho > 0$.

4 Linear Support Vector Regression

In the case of regression the goal is to predict a real-valued output for y' so that our training data is of the form:

$$\{(x_i, y_i), x \in \Re^m, y_i \in \Re, i = 1, \ldots, n\} \tag{65}$$

The regression SVM use a loss function that not allocating a penalty if the predicted value y'_i is less than a distance ϵ away from the actual value y_i, i.e., if $|y_i - y'_i| \leq \epsilon$, where $y'_i = w^T x_i + b$. The region bound by $y'_i \pm \epsilon \; \forall_i$ is called an ϵ-insensitive tube. The output variables which are outside the tube are given one of two slack variable penalties depending on whether they lie above, ξ^+, or below, ξ^-, the tube, provided $\xi^+ \geq 0$ and $\xi^- \geq 0 \; \forall_i$:

$$\begin{aligned} & y_i \leq y'_i + \epsilon + \xi^+ \; \forall_i \\ & y_i \geq y'_i - \epsilon - \xi^- \; \forall_i \\ & \xi_i^+, \xi_i^- \geq 0 \; \forall_i \end{aligned} \tag{66}$$

The objective function for SVR can then be written as:

$$\begin{aligned} \min_{w,b,\xi^+,\xi^-} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) \\ \text{subject to} \quad & y_i - w^T x_i - b \leq \epsilon + \xi_i^+ \; \forall_i \\ & w^T x_i + b - y_i \leq \epsilon + \xi_i^- \; \forall_i \\ & \xi_i^+, \xi_i^- \geq 0 \; \forall_i \end{aligned} \tag{67}$$

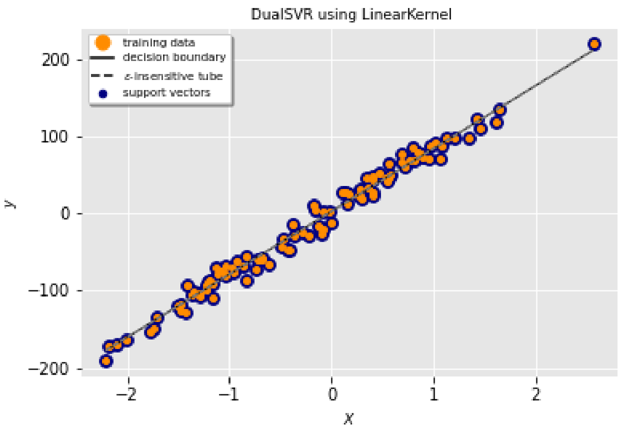


Figure 4: Linear SVR hyperplane

4.1 Epsilon-insensitive loss

The *epsilon-insensitive* loss is defined as:

$$\mathcal{L}_\epsilon^1 = \max(0, |y - (w^T x + b)| - \epsilon) \tag{68}$$

or, equivalently:

$$\mathcal{L}_\epsilon^1 = \begin{cases} 0 & \text{if } |y - (w^T x + b)| \leq \epsilon \\ |y - (w^T x + b)| - \epsilon & \text{otherwise} \end{cases} \tag{69}$$

As the *hinge* loss, also the *epsilon-insensitive* loss is a nondifferentiable convex function due to its nonsmoothness in $\pm\epsilon$, but has a subgradient that is given by:

$$\partial_w \mathcal{L}_\epsilon^1 = \begin{cases} \dfrac{y - (w^T x + b)}{|y - (w^T x + b)|} x & \text{if } |y - (w^T x + b)| \geq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{70}$$

4.1.1 Primal formulation

The general primal unconstrained formulation takes the same form of (9).

The quadratic optimization problem (67) can be equivalently formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^n \max(0, |y_i - (w^T x_i + b)| - \epsilon) \tag{71}$$

where we make use of the *epsilon-insensitive* loss (68) or (69).

The above formulation penalizes slacks ξ linearly and is called $\mathcal{L}_1$-SVR.

The $\mathcal{L}_1$-SVR objective (71) can be rewritten in form (11) or (12) as:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C \sum_{i=1}^n \max(0, |y_i - (w^T x_i + b)| - \epsilon) \tag{72}$$

4.1.2 Wolfe Dual formulation

To reformulate the (67) as a *Wolfe dual*, we introduce the Lagrange multipliers $\alpha_i^+, \alpha_i^-, \mu_i^+, \mu_i^- \geq 0 \ \forall_i$:

$$\begin{aligned} \max_{\alpha^+,\alpha^-,\mu^+,\mu^-} \min_{w,b,\xi^+,\xi^-} \mathcal{W}(w, b, \xi^+, \xi^-, \alpha^+, \alpha^-, \mu^+, \mu^-) &= \frac{1}{2}\|w\|^2 + C \sum_{i=1}^n (\xi_i^+ + \xi_i^-) - \sum_{i=1}^n (\mu_i^+ \xi_i^+ + \mu_i^- \xi_i^-) \\ &- \sum_{i=1}^n \alpha_i^+ (\epsilon + \xi_i^+ + y_i' - y_i) - \sum_{i=1}^n \alpha_i^- (\epsilon + \xi_i^- - y_i' + y_i) \end{aligned} \tag{73}$$

Substituting for y_i, differentiating wrt w, b, ξ^+, ξ^- and setting the derivatives to 0 gives:

$$\frac{\partial \mathcal{W}}{\partial w} = w - \sum_{i=1}^n (\alpha_i^+ - \alpha_i^-) x_i \Rightarrow w = \sum_{i=1}^n (\alpha_i^+ - \alpha_i^-) x_i \tag{74}$$

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^n (\alpha_i^+ - \alpha_i^-) \Rightarrow \sum_{i=1}^n (\alpha_i^+ - \alpha_i^-) = 0 \tag{75}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i^+} = 0 \Rightarrow C = \alpha_i^+ + \mu_i^+ \tag{76}$$

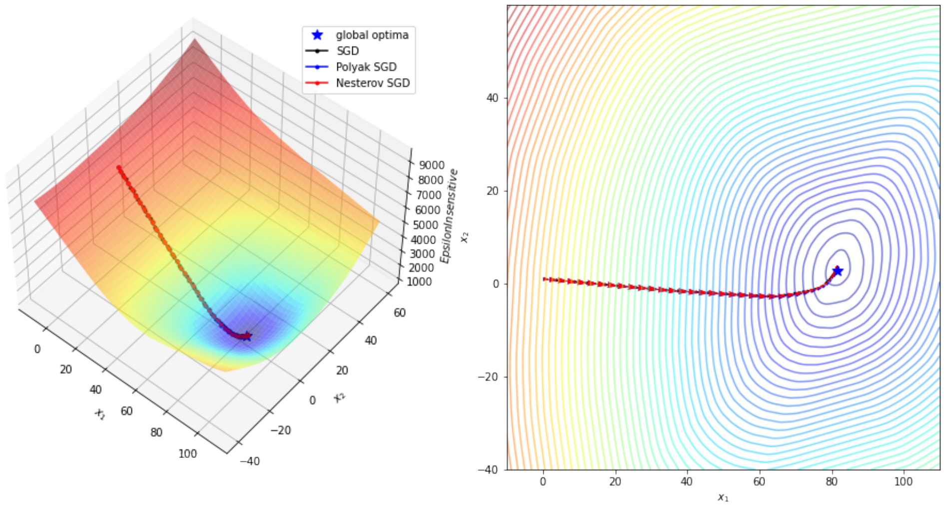


Figure 5: Epsilon-insensitive loss with different optimization steps

$$\frac{\partial \mathcal{W}}{\partial \xi_i^-} = 0 \Rightarrow C = \alpha_i^- + \mu_i^- \tag{77}$$

Substituting (74) and (75) in, we now need to maximize $\mathcal{W}$ wrt α_i^+ and α_i^-, where $\alpha_i^+ \geq 0,\ \alpha_i^- \geq 0\ \forall_i$:

$$\max_{\alpha^+,\alpha^-} \mathcal{W}(\alpha^+, \alpha^-) = \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) - \epsilon \sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) - \frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)\langle x_i, x_j\rangle(\alpha_j^+ - \alpha_j^-) \tag{78}$$

Using $\mu_i^+ \geq 0$ and $\mu_i^- \geq 0$ together with (74) and (75) means that $\alpha_i^+ \leq C$ and $\alpha_i^- \leq C$. We therefore need to find:

$$\begin{aligned} \min_{\alpha^+,\alpha^-} \quad & \frac{1}{2}(\alpha^+ - \alpha^-)^T K(\alpha^+ - \alpha^-) + \epsilon e^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-) \\ \text{subject to} \quad & 0 \leq \alpha_i^+, \alpha_i^- \leq C\ \forall_i \\ & e^T(\alpha^+ - \alpha^-) = 0 \end{aligned} \tag{79}$$

where $e^T = [1, \ldots, 1]$.
We can write the (79) in a standard quadratic form as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha - q^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C\ \forall_i \\ & e^T \alpha = 0 \end{aligned} \tag{80}$$

where the Hessian matrix $Q = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}$, $\alpha = \begin{bmatrix} \alpha^+ \\ \alpha^- \end{bmatrix}$, $q = \begin{bmatrix} -y \\ y \end{bmatrix} + \epsilon$, and $e = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$.
Each new predictions y' can be found using:

$$y' = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)\langle x_i, x'\rangle + b \tag{81}$$

A set S of support vectors x_s can be created by finding the indices i where $0 \leq \alpha \leq C$ and $\xi_i^+ = 0$ or $\xi_i^- = 0$. This gives us:

$$b = y_s - \epsilon - \sum_{m \in S} (\alpha_m^+ - \alpha_m^-) \langle x_m, x_s \rangle \tag{82}$$

As before it is better to average over all the indices i in S:

$$b = \frac{1}{N_s} \sum_{s \in S} y_s - \epsilon - \sum_{m \in S} (\alpha_m^+ - \alpha_m^-) \langle x_m, x_s \rangle \tag{83}$$

From (80) we can notice that the equality constraint $e^T \alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. We report below the box-constrained dual formulation [14] that arises from the primal (11) or (12) where the bias term b is embedded into the weight vector w:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T (Q + ee^T) \alpha + q^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \end{aligned} \tag{84}$$

4.1.3 Lagrangian Dual formulation

In order to relax the constraints in the *Wolfe dual* formulation (79) we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrange multipliers μ and $\lambda_+, \lambda_- \geq 0$:

$$\begin{aligned} \max_{\mu, \lambda_+, \lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda_+, \lambda_-) &= \frac{1}{2} \alpha^T Q \alpha + q^T \alpha - \mu^T (e^T \alpha) - \lambda_+^T (ub - \alpha) - \lambda_-^T \alpha \\ &= \frac{1}{2} \alpha^T Q \alpha + (q - \mu e + \lambda_+ - \lambda_-)^T \alpha - \lambda_+^T ub \\ \text{subject to} \quad & \lambda_+, \lambda_- \geq 0 \end{aligned} \tag{85}$$

where the upper bound $ub^T = [C, \dots, C]$.
Taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow Q\alpha + (q - \mu e + \lambda_+ - \lambda_-) = 0 \tag{86}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu e + \lambda_+ - \lambda_-) \tag{87}$$

the gradients wrt μ, λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -e\alpha \tag{88}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{89}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{90}$$

From (80) we can notice that the equality constraint $e^T \alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. In this way the dimensionality of (85) is reduced by removing the multipliers μ which was allocated to control the equality constraint $e^T\alpha = 0$, so we will end up solving exactly the problem (84).

$$\begin{aligned}
\max_{\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + q^T\alpha - \lambda_+^T(ub - \alpha) - \lambda_-^T\alpha \\
&= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T ub \\
\text{subject to} \quad & \lambda_+, \lambda_- \geq 0
\end{aligned} \tag{91}$$

where, again, the upper bound $ub^T = [C, \dots, C]$.
Now, taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-) = 0 \tag{92}$$

With α optimal solution of the linear system:

$$(Q + ee^T)\alpha = -(q + \lambda_+ - \lambda_-) \tag{93}$$

the gradients wrt λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{94}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{95}$$

Note that since the Hessian matrix Q of the $\mathcal{L}_1$-SVR is not strictly positive definite, i.e., the Lagrangian function is not strictly convex since it will be linear along the eigenvectors correspondent to the null eigenvalues and so it will be unbounded below, the Lagrangian dual relaxation, i.e., 87 and 93, will be nondifferentiable, so it will have infinite solutions and for each of them it will have a different subgradient. In order to compute an approximation of the gradient, we will choose α in such a way as the one that minimizes the 2-norm since it is good almost like the gradient:

$$\min_{\alpha_n \in K_n(Q,b)} \|Q\alpha_n - b\| \tag{96}$$

Since we are dealing with a symmetric system we will choose a well-known Krylov method that performs the Lanczos iterate, i.e., symmetric Arnoldi iterate, called *minres*, i.e., symmetric *gmres*, to compute the vector α_n that minimizes the norm of the residual $r_n = Q\alpha_n - b$ among all vectors in $K_n(Q, b) = span(b, Qb, Q^2b, \dots, Q^{n-1}b)$.

Since the linear algebra methods in the ML context are crucial and also in order to deal with a per-iteration cost equals to the other algorithms described later to provide a coherent comparison of all at the end, we will solve it with a primal-dual optimization method and we modify its definition by adding a strictly convex augmentation term, i.e., a penalty term, in order to improve the practical convergence of the algorithms. So, if we consider a general quadratic optimization problem subject to linear constraints, i.e., equality and inequality constraints, defined as:

$$\begin{aligned}
\min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha \\
\text{subject to} \quad & A\alpha = b \\
& G\alpha \leq h \\
& lb \leq \alpha \leq ub
\end{aligned} \tag{97}$$

or, equivalently:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha \\ \text{subject to} \quad & A\alpha = b \\ & \hat{G}\alpha \leq \hat{h} \end{aligned} \tag{98}$$

where $\hat{G} = \begin{bmatrix} G \\ -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} h & -lb & ub \end{bmatrix}$; we give the following *augmented Lagrangian dual*:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha + \mu^T(A\alpha - b) + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|A\alpha - b\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{99}$$

with $\rho > 0$.

According to this definition, we change the formulation 85 as:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda) = & \frac{1}{2}\alpha^T Q\alpha + q^T\alpha + \mu^T(e^T\alpha) + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|e^T\alpha\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{100}$$

and the formulation 91 as:

$$\begin{aligned} \max_{\lambda} \min_{\alpha} \mathcal{L}(\alpha,\lambda) = & \frac{1}{2}\alpha^T (Q + ee^T)\alpha + q^T\alpha + \lambda^T(\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{101}$$

where $\hat{G} = \begin{bmatrix} -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} -lb & ub \end{bmatrix}$ with $lb^T = [0, \dots, 0]$, $ub^T = [C, \dots, C]$ and $\rho > 0$.

4.2 Squared Epsilon-insensitive loss

The *squared epsilon-insensitive* loss is defined as:

$$\mathcal{L}_\epsilon^2 = \max(0, |y - (w^T x + b)| - \epsilon)^2 \tag{102}$$

or, equivalently:

$$\mathcal{L}_\epsilon^2 = \begin{cases} 0 & \text{if } |y - (w^T x + b)| \leq \epsilon \\ (|y - (w^T x + b)| - \epsilon)^2 & \text{otherwise} \end{cases} \tag{103}$$

As the *squared hinge* loss, also the *squared epsilon-insensitive* loss is a strictly convex function and its gradient is given by:

$$\nabla_w \mathcal{L}_\epsilon^2 = \begin{cases} 2\,\mathrm{sign}(y - (w^T x + b))(|y - (w^T x + b)| - \epsilon)x & \text{if } |y - (w^T x + b)| \geq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{104}$$

4.2.1 Primal formulation

To provide a continuously differentiable function the optimization problem (71) can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, |y_i - (w^T x_i + b)| - \epsilon)^2 \tag{105}$$

where we make use of the *squared epsilon-insensitive* loss that quadratically penalized slacks ξ and is called $\mathcal{L}_2$-SVR.

The $\mathcal{L}_2$-SVR objective (105) can be rewritten in form (11) or (12) as:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C\sum_{i=1}^{n} \max(0, |y_i - (w^T x_i + b)| - \epsilon)^2 \tag{106}$$

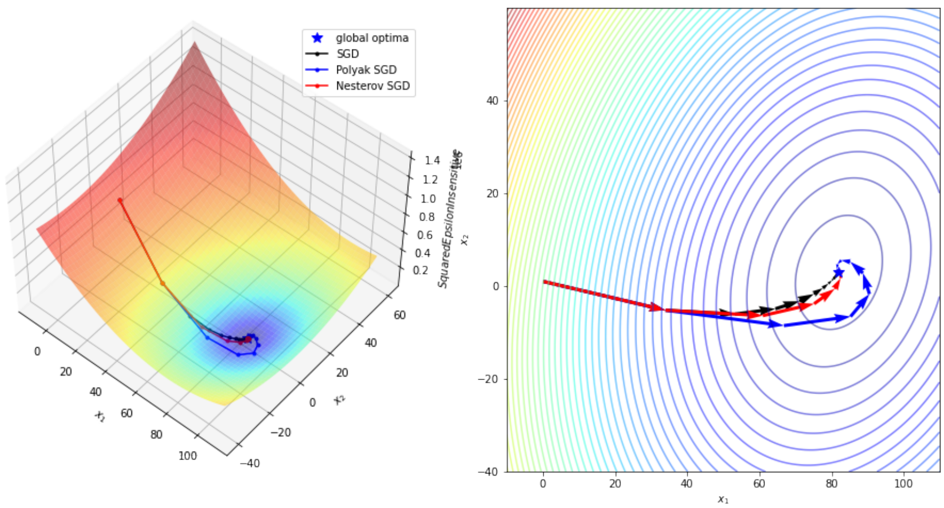


Figure 6: Squared Epsilon-insensitive loss with different optimization steps

4.2.2 Wolfe Dual formulation

As done for the $\mathcal{L}_1$-SVR we can derive the *Wolfe dual* formulation of the $\mathcal{L}_2$-SVR by obtaining:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q+D)\alpha + q^T\alpha \\ \text{subject to} \quad & \alpha_i \geq 0 \ \forall_i \\ & e^T\alpha = 0 \end{aligned} \tag{107}$$

or, alternatively, with the regularized bias term by obtaining:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q+ee^T+D)\alpha + q^T\alpha \\ \text{subject to} \quad & \alpha_i \geq 0 \ \forall_i \end{aligned} \tag{108}$$

where the diagonal matrix $D_{ii} = \dfrac{1}{2C} \ \forall_i$.

4.2.3 Lagrangian Dual formulation

In order to relax the constraints in the $\mathcal{L}_2$-SVR *Wolfe dual* formulation (107) we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrange multipliers μ and $\lambda \geq 0$:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda) &= \frac{1}{2}\alpha^T(Q+D)\alpha + q^T\alpha - \mu^T(e^T\alpha) - \lambda^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+D)\alpha + (q-\mu e-\lambda)^T\alpha \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{109}$$

Taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q+D)\alpha + (q-\mu e-\lambda) = 0 \tag{110}$$

With α optimal solution of the linear system:

$$(Q+D)\alpha = -(q-\mu e-\lambda) \tag{111}$$

the gradients wrt μ and λ are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -e\alpha \tag{112}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\alpha \tag{113}$$

From (80) we can notice that the equality constraint $e^T\alpha = 0$ arises form the stationarity condition $\partial_b\mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. In this way the dimensionality of (109) is reduced by removing the multipliers μ which was allocated to control the equality constraint $e^T\alpha = 0$, so we will end up solving exactly the problem (108).

$$\begin{aligned} \max_{\lambda} \min_{\alpha} \mathcal{L}(\alpha,\lambda) &= \frac{1}{2}\alpha^T(Q+ee^T+D)\alpha + q^T\alpha - \lambda^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+ee^T+D)\alpha + (q-\lambda)^T\alpha \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{114}$$

Now, taking the derivative of the Lagrangian $\mathcal{L}$ wrt α and settings it to 0 gives:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + ee^T + D)\alpha + (q - \lambda) = 0 \tag{115}$$

With α optimal solution of the linear system:

$$(Q + ee^T + D)\alpha = -(q - \lambda) \tag{116}$$

the gradient wrt λ is:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -\alpha \tag{117}$$

Note that since the Hessian matrix Q of the $\mathcal{L}_2$-SVR is symmetric and strictly positive definite, we can find the unique solution of the Lagrangian dual relaxation, i.e., 111 and 116, by solving the system with the Cholesky factorization.

Since the linear algebra methods in the ML context are crucial and also in order to deal with a per-iteration cost equals to the other algorithms described later to provide a coherent comparison of all at the end, we will solve it with a primal-dual optimization method and we modify its definition by adding a strictly convex augmentation term, i.e., a penalty term, in order to improve the practical convergence of the algorithms. So, if we consider a general quadratic optimization problem subject to linear constraints, i.e., equality and inequality constraints, defined as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & G\alpha \leq h \\ & lb \leq \alpha \leq ub \end{aligned} \tag{118}$$

or, equivalently:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha \\ \text{subject to} \quad & A\alpha = b \\ & \hat{G}\alpha \leq \hat{h} \end{aligned} \tag{119}$$

where $\hat{G} = \begin{bmatrix} G \\ -I \\ I \end{bmatrix}$ and $\hat{h} = \begin{bmatrix} h & -lb & ub \end{bmatrix}$; we give the following *augmented Lagrangian dual*:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha + q^T \alpha + \mu^T (A\alpha - b) + \lambda^T (\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|A\alpha - b\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{120}$$

with $\rho > 0$.

According to this definition, we change the formulation 109 as:

$$\begin{aligned} \max_{\mu,\lambda} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda) = & \frac{1}{2}\alpha^T (Q + D)\alpha + q^T \alpha + \mu^T (e^T \alpha) + \lambda^T (\hat{G}\alpha - \hat{h}) + \frac{\rho}{2}\|e^T \alpha\|^2 + \frac{\rho}{2}\|\hat{G}\alpha - \hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0 \end{aligned} \tag{121}$$

and the formulation 114 as:

$$\begin{aligned}\max_{\lambda}\min_{\alpha}\mathcal{L}(\alpha,\lambda) &= \frac{1}{2}\alpha^T(Q+ee^T+D)\alpha+q^T\alpha+\lambda^T(\hat{G}\alpha-\hat{h})+\frac{\rho}{2}\|\hat{G}\alpha-\hat{h}\|^2 \\ \text{subject to} \quad & \lambda \geq 0\end{aligned} \tag{122}$$

where $\hat{G} = \begin{bmatrix}-I\end{bmatrix}$ and $\hat{h} = \begin{bmatrix}-lb\end{bmatrix}$ with $lb^T = [0,\dots,0]$ and $\rho > 0$.

5 Nonlinear Support Vector Machines

When applying our SVC to *linearly separable* data in (18), we have started by creating a matrix Q from the dot product of our input variables:

$$Q_{ij} = y_i y_j k(x_i, x_j) \tag{123}$$

or, a matrix K from the dot product of our input variables in the SVR case (79):

$$K_{ij} = k(x_i, x_j) \tag{124}$$

where $k(x_i, x_j)$ is an example of a family of functions called *kernel functions*.

For any positive definite kernel function k (a so called Mercer kernel), it is guaranteed that there exists a mapping ϕ into a Hilbert space $\mathcal{H}$, such that:

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \phi(x_i)^T \phi(x_j) \tag{125}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in the Hilbert space and $\phi(\cdot)$ is the identity function.

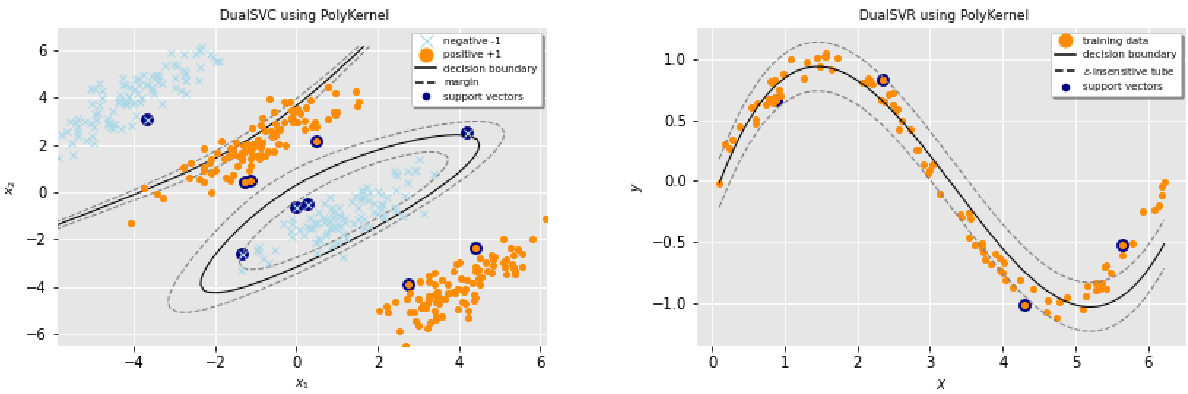
The reason that this *kernel trick* is useful is that there are many classification/regression problems that are nonlinearly separable/regressable in the *input space*, which might be in a higher dimensionality *feature space* given a suitable mapping $x \to \phi(x)$.

5.1 Polynomial kernel

The *polynomial* kernel is defined as:

$$k(x_i, x_j) = (\gamma \langle x_i, x_j \rangle + r)^d \tag{126}$$

where γ define how far the influence of a single training example reaches (low values meaning 'far' and high values meaning 'close').



(a) Polynomial SVC hyperplane

(b) Polynomial SVR hyperplane

Figure 7: Polynomial SVM hyperplanes

5.2 Gaussian kernel

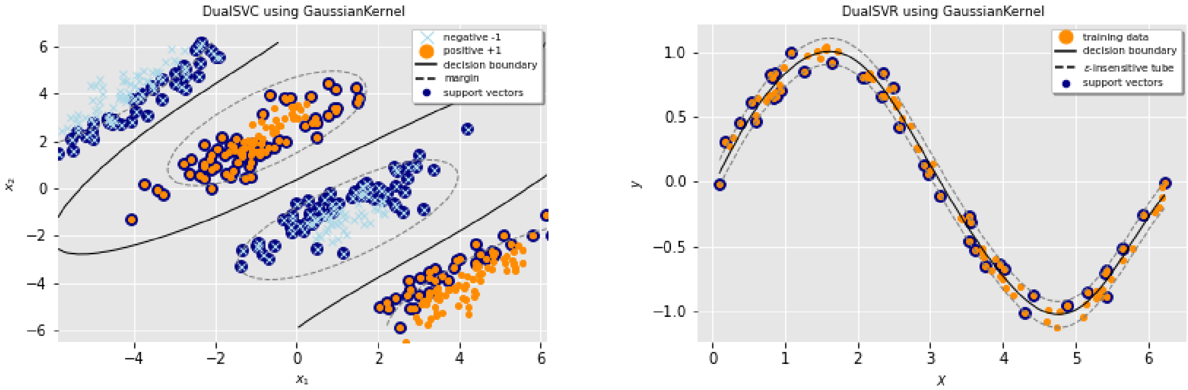
The *gaussian* kernel is defined as:

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}) \tag{127}$$

or, equivalently:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|_2^2) \tag{128}$$

where $\gamma = \dfrac{1}{2\sigma^2}$ define how far the influence of a single training example reaches (low values meaning 'far' and high values meaning 'close').



(a) Gaussian SVC hyperplane

(b) Gaussian SVR hyperplane

Figure 8: Gaussian SVM hyperplanes

5.3 Laplacian kernel

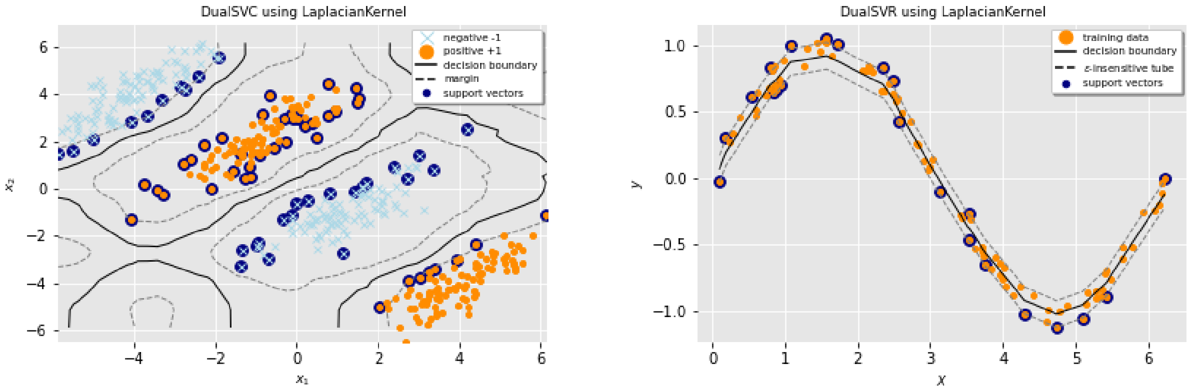
The *laplacian* kernel is defined as:

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|_1}{2\sigma^2}) \tag{129}$$

or, equivalently:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|_1) \tag{130}$$

where $\gamma = \dfrac{1}{2\sigma^2}$ define how far the influence of a single training example reaches (low values meaning 'far' and high values meaning 'close').



(a) Laplacian SVC hyperplane

(b) Laplacian SVR hyperplane

Figure 9: Laplacian SVM hyperplanes

6 Optimization Methods

In order to explain the *convergence rates* of the following optimization methods, we need to introduce some preliminary definitions about *convexity* and the *Lipschitz continuity* of a function [15].

Definition 1 (Convexity).

(i) We say that a function $f : \Re^m \rightarrow \Re$ is convex if:

$$(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y) \;\forall\; x, y \in \Re^m, \lambda \in [0, 1]$$

(ii) We say that a differentiable function $f : \Re^m \rightarrow \Re$ is convex if:

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle \;\forall\; x, y \in \Re^m$$

(iii) We say that a twice differentiable function $f : \Re^m \rightarrow \Re$ is convex iff:

$$\nabla^2 f(x) \succeq 0 \;\forall\; x \in \Re^m$$

i.e., the Hessian matix is *positive semidefinite*.

Definition 2 (Strict Convexity).

(i) We say that a function $f : \Re^m \rightarrow \Re$ is strictly convex if:

$$(\lambda x + (1 - \lambda) y) < \lambda f(x) + (1 - \lambda) f(y) \;\forall\; x, y \in \Re^m, x \neq y, \lambda \in (0, 1)$$

(ii) We say that a differentiable function $f : \Re^m \rightarrow \Re$ is strictly convex if:

$$f(y) > f(x) + \langle \nabla f(x), y - x \rangle \;\forall\; x, y \in \Re^m, x \neq y$$

(iii) We say that a twice differentiable function $f : \Re^m \rightarrow \Re$ is strictly convex iff:

$$\nabla^2 f(x) \succ 0 \;\forall\; x \in \Re^m$$

i.e., the Hessian matix is *positive definite*.

Definition 3 (Strong Convexity)**.** We say that a function $f : \Re^m \to \Re$ is μ-strongly convex if the function:

$$g(x) = f(x) - \frac{\mu}{2}\|x\|^2$$

is convex for any $\mu > 0$. If f is differentiable this is also equivalent to:

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2 \;\forall\; x, y \in \Re^m$$

and, if f is a twice differentiable function then f is μ-strongly convex iff:

$$\nabla^2 g(x) \succ 0 \;\forall\; x \in \Re^m$$

i.e., the Hessian matix is *positive definite*, which is:

$$\nabla^2 f(x) \succeq \mu I \;\forall\; x \in \Re^m$$

i.e., all the eigenvalues of the Hessian matrix are lowerbounded by μI.

Definition 4 (L_f-Lipschitz continuity)**.** We say that a function $f : \Re^m \to \Re$ is L_f-Lipschitz continuous if:

$$|f(x) - f(y)| \leq L_f\|x - y\| \;\forall\; x, y \in \Re^m$$

meaning that f is bounded above and below by a linear function.

Intuitively, L is a measure of how fast the function can change.

Finally, we say that a function $f : \Re^m \to \Re$ is locally L_f-Lipschitz continuous if for every x in $\Re^m$ there exists a neighborhood U of x such that f restricted to U is L_f-Lipschitz continuous. Every convex function is locally L_f-Lipschitz continuous.

Definition 5 (L-Lipschitz continuity)**.** We say that a function $f : \Re^m \to \Re$ is L-Lipschitz gradient continuous if f is differentiable and:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \;\forall\; x, y \in \Re^m$$

that is equivalent to:

$$|f(y) - f(x) - \langle \nabla f(x), y - x \rangle| \leq \frac{L}{2}\|y - x\|^2 \;\forall\; x, y \in \Re^m$$

meaning that f is bounded above and below by a quadratic function.

Also, if f is a twice differentiable function this is equivalent to:

$$\nabla^2 f(x) \preceq LI \;\forall\; x \in \Re^m$$

i.e., all the eigenvalues of the Hessian matrix are upperbounded by L.

Note that if f is a μ-strongly convex function, we give the following Hessian bounds:

$$0 \prec \mu I \preceq \nabla^2 f(x) \preceq LI \;\forall\; x \in \Re^m$$

i.e., all the eigenvalues of the Hessian matrix are lowerbounded by μI and upperbounded by L.

Finally, we say that a function $f : \Re^m \to \Re$ is locally L-Lipschitz gradient continuous if for every x in $\Re^m$ there exists a neighborhood U of x such that f restricted to U is L-Lipschitz gradient continuous.

Definition 6 (Subgradient)**.** Given a function $f : \Re^m \to \Re$ and $x \in \Re^m$, we define a subgradient $g \in \Re^m$ at x to be any point satisfying:

$$f(y) \geq f(x) + \langle g, y - x \rangle \;\forall\; y \in \Re^m$$

Subgradients always exist for convex function.

Theorem 7 (L_f-Lipschitz continuity for convex functions)**.** *Let $f : \Re^m \to \Re$ be a convex function and let K be a closed and bounded set contained in the relative interior of the domain of f, i.e., $K \subset \Re^m$. Then f is L_f-Lipschitz continuous on K, i.e.,:*

$$|f(x) - f(y)| \leq L_f \|x - y\| \;\forall\; x, y \in K$$

In particular, f is bounded on K.

Proof. Let x and y be any two points in the set K. Since $\partial f(x)$ is nonempty, by using the subgradient inequality 6, it follows that:

$$f(y) \geq f(x) + \langle g, y - x\rangle \;\forall\; g \in \partial f(x)$$

implying that:

$$f(x) - f(y) \leq \|g\|\|x - y\| \;\forall\; g \in \partial f(x)$$

By definition, the set $\cup_{x\in K}\partial f(x)$ is nonempty and bounded, so that for some constant $L > 0$, we have:

$$\|g\| \leq L_f \quad \forall g \in \partial f(x) \quad \forall x \in K$$

and therefore:

$$f(x) - f(y) \leq L_f \|x - y\|$$

By exchanging the roles of x ad y, we similarly obtain:

$$f(y) - f(x) \leq L_f \|x - y\|$$

and by combining the preceding two relations, we see that:

$$|f(x) - f(y)| \leq L_f \|x - y\|$$

showing that f is L_f-Lipschitz continuous over K. □

Note that this proof shows how to determine the Lipschitz constant L_f: it is the maximum subgradient norm, over all subgradients in $\cup_{x\in K}\partial f(x)$.

Strong convexity and L-Lipschitz continuity are related by Fenchel duality according to the following theorem, which proof is given in [16].

Theorem 8 (μ-strong convexity and L-Lipschitz continuity for convex functions)**.** *A function f and its Fenchel dual f^* satisfy the following assertions:*

(i) if f is μ-strongly convex, then f^* is $\dfrac{1}{\mu}$-Lipschitz continuous.

(ii) if f is convex and L-Lipschitz continuous, then f^* is $\dfrac{1}{L}$-strongly convex.

Note that since f is convex and its epigraph is a closed convex set, $f^* = f$.

6.1 Gradient Descent for Primal formulations

The Gradient Descent algorithm is the simplest *first-order optimization* method that exploits the orthogonality of the gradient wrt the level sets to take a descent direction. In particular, it performs the following iterations:

Algorithm 1 Gradient Descent

Require: Function f to minimize
Require: Learning rate or step size $\alpha > 0$
  function GRADIENTDESCENT(f, α)
    Initialize weight vector x_0
    $t = 0$
    while *not_convergence* **do**
      $x_{t+1} = x_t - \alpha \partial f(x_t)$ ▷ if f is differentiable then $\partial f(x_t) = \nabla f(x_t)$
      $t = t + 1$
    end while
    return x_t
  end function

Gradient Descent is based on full gradients, since at each iteration we compute the average gradient on the whole dataset:

$$\partial f(x) = \frac{1}{n} \sum_{i=1}^{n} \partial f_i(x)$$

The downside is that every step is very computationally expensive, $\mathcal{O}(nm)$ per iteration, where n is the number of samples in our dataset and m is the number of dimensions.

Since *Gradient Descent* becomes impractical when dealing with large datasets we introduce a stochastic version, called *Stochastic Gradient Descent*, which does not use the whole set of examples to compute the gradient at every step. By doing so, we can reduce computation all the way down to $\mathcal{O}(m)$ per iteration.

Algorithm 2 Stochastic Gradient Descent

Require: Function f to minimize
Require: Learning rate or step size $\alpha > 0$
Require: Batch size k
  function STOCHASTICGRADIENTDESCENT(f, α, k)
    Initialize weight vector x_0
    $t \leftarrow 0$
    while *not_convergence* **do**
      Sample $(i_1, \ldots, i_k) \sim \mathcal{U}^k(1, \ldots, n)$
      $x_{t+1} \leftarrow x_t - \alpha \frac{1}{k} \sum_{j=1}^{k} \partial f_{i_j}(x_t)$ ▷ if f is differentiable then $\partial f_{i_j}(x_t) = \nabla f_{i_j}(x_t)$
      $t \leftarrow t + 1$
    end while
    return x_t
  end function

Note that in expectation, we converge like GD, since $\mathbb{E}_{i \sim \mathcal{U}(1,\ldots,n)}[\partial f_i(x_t)] = \partial f(x_t)$, therefore, the expected iterate of SGD converges to the optimum.

Now, consider the SGD algorithm introduced previously but where each iteration is projected into the ball $\mathcal{B}(0, R)$ with radius $R > 0$ fixed. So, the following lower bounds on convergence rates are given.

Theorem 9 (Stochastic Gradient Descent convergence for convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous convex function and assume that exists $b > 0$ satisfying:*

$$\|f_i(x)\| \leq b \ \forall \ x \in \mathcal{B}(0, R)$$

Besides, assume that all minima of f belong to $\mathcal{B}(0, R)$. Then the Stochastic Gradient Descent with step size $\alpha = \dfrac{2R}{b\sqrt{k}}$ satisfies:

$$\mathbb{E}\left[f\left(\frac{1}{k}\sum_{t=1}^{k} x_t\right)\right] - f(x^*) \leq \frac{3Rb}{\sqrt{k}}$$

Theorem 10 (Stochastic Gradient Descent convergence for strongly convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous, μ-strongly convex function and assume that exists $b > 0$ satisfying:*

$$\|f_i(x)\| \leq b \ \forall \ x \in \mathcal{B}(0, R)$$

Besides, assume that all minima of f belong to $\mathcal{B}(0, R)$. Then the Stochastic Gradient Descent with step size $\alpha = \dfrac{2}{\mu(k+1)}$ satisfies:

$$\mathbb{E}\left[f\left(\frac{2}{k(k+1)}\sum_{t=1}^{k} t x_{t-1}\right)\right] - f(x^*) \leq \frac{2b^2}{\mu(k+1)}$$

SGD's *convergence rate* for L-Lipschitz continuous convex functions is $\mathcal{O}\left(\dfrac{1}{\sqrt{t}}\right)$ and $\mathcal{O}\left(\dfrac{1}{t}\right)$ for L-Lipschitz continuous and strongly convex functions. More iterations are needed to reach the same accuracy as GD, but the iterations are far cheaper.

6.1.1 Nonsmooth

First, consider a nonsmooth, i.e., nondifferentiable, convex function. So, the following lower bounds on convergence rates are given.

Theorem 11 (Subgradient Descent convergence for convex functions with Polyak's stepsize)**.** *Let $f : \Re^m \to \Re$ be a L_f-Lipschitz continuous convex function. Then the Subgradient Descent with Polyak's step size $\alpha_t = \dfrac{f(x_t) - f(x^*)}{\|g_t\|^2}$ satisfies:*

$$f(x_t) - f(x^*) \leq \frac{L\|x_0 - x^*\|^2}{\sqrt{t+1}}$$

Unfortunately, Polyak's stepsize rule requires knowledge of $f(x^*)$, which is often unknown a priori, so we might often need simpler rule for setting stepsizes.

Theorem 12 (Subgradient Descent convergence for convex functions)**.** *Let $f : \Re^m \to \Re$ be a L_f-Lipschitz continuous convex function. Then the Subgradient Descent with step size $\alpha_t = \dfrac{1}{\sqrt{t}}$ satisfies:*

$$f(x_t) - f(x^*) \leq \frac{\|x_0 - x^*\|^2 + L^2 \log t}{\sqrt{t}}$$

Theorem 13 (Subgradient Descent convergence for strongly convex functions)**.** *Let $f : \Re^m \to \Re$ be a L_f-Lipschitz continuous and μ-strongly convex function. Then the Subgradient Descent with step size $\alpha_t = \dfrac{2}{\mu(t+1)}$ satisfies:*

$$f(x_t) - f(x^*) \leq \frac{2L^2}{\mu}\frac{1}{t+1}$$

In summary, the following *convergence rates* and *iterations complexities* are given:

Table 1: Subgradient Descent convergence rates and iterations complexities

	stepsize rule	convergence rate	iteration complexity
convex and L_f-Lipschitz	$\alpha = \dfrac{1}{\sqrt{t}}$	$\mathcal{O}\left(\dfrac{1}{\sqrt{t}}\right)$	$\mathcal{O}\left(\dfrac{1}{\epsilon^2}\right)$
strongly convex and L_f-Lipschitz	$\alpha = \dfrac{1}{t}$	$\mathcal{O}\left(\dfrac{1}{t}\right)$	$\mathcal{O}\left(\dfrac{1}{\epsilon}\right)$

Among algorithms that only use subgradient, these *convergence rates* cannot be futher improved.

6.1.2 Smooth

Now, consider a smooth, i.e., differentiable, convex function. So, the following lower bounds on convergence rates are given.

Theorem 14 (Gradient Descent convergence for convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous convex function. Then the Gradient Descent with step size $\alpha = 1/L$ satisfies:*

$$f(x_t) - f(x^*) \leq \frac{L\|x_0 - x^*\|^2}{2t}$$

Theorem 15 (Gradient Descent convergence for strongly convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous and μ-strongly convex function. Then the Gradient Descent with step size $\alpha = 1/L$ satisfies:*

$$\begin{aligned} f(x_t) - f(x^*) \leq& \left(1 - \frac{\mu}{L}\right)^t \|f(x_0) - f(x^*)\|^2 \\ =& \left(1 - \frac{1}{\kappa}\right)^t \|f(x_0) - f(x^*)\|^2 \end{aligned}$$

where $\kappa = L/\mu$.

Theorem 16 (Gradient Descent convergence for convex quadratic functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous and μ-strongly convex quadratic function. Then the Gradient Descent with step size $\alpha = \dfrac{2}{L+\mu}$ and momentum $\beta = \max\left\{|1-\alpha\mu|, |1-\alpha L|\right\}$ satisfies:*

$$\|x_t - x^*\| = \left(\frac{\kappa - 1}{\kappa + 1}\right)^t \|x_0 - x^*\|$$

where $\kappa = L/\mu$.

In summary, the following *convergence rates* and *iterations complexities* are given:

Table 2: Gradient Descent convergence rates and iterations complexities

	stepsize rule	convergence rate	iteration complexity
convex and L-Lipschitz	$\alpha = \dfrac{1}{L}$	$\mathcal{O}\left(\dfrac{1}{t}\right)$	$\mathcal{O}\left(\dfrac{1}{\epsilon}\right)$
strongly convex and L-Lipschitz	$\alpha = \dfrac{1}{L}$	$\mathcal{O}\left(\left(1 - \dfrac{1}{\kappa}\right)^t\right)$	$\mathcal{O}\left(\kappa \log \dfrac{1}{\epsilon}\right)$

6.1.3 Momentum

To mitigate the pathological zig-zagging by speeding up the *convergence rate* of the SGD method, we introduce two accelerated methods [1] and [2, 3] that exploits information from the history, i.e., past iterates, to add some inertia, i.e., the momentum, to yield smoother trajectory.

In the Polyak's method [1] the velocity vector v_t is calculated by applying the β momentum to the previous v_{t-1} displacement, and subtracting the gradient step to x_t.

Figure 10: Polyak's and Nesterov's Momentum

Algorithm 3 Polyak's Accelerated Gradient Descent or Polyak Heavy-Ball method

Require: Function f to minimize
Require: Learning rate or step size $\alpha > 0$
Require: Momentum $\beta \in [0, 1)$
  function POLYAKACCELERATEDGRADIENTDESCENT(f, α, β)
    Initialize weight vector $x_1 \leftarrow x_0$ and velocity vector $v_0 \leftarrow 0$
    $t \leftarrow 1$
    while *not_convergence* **do**
      $v_t = \beta v_{t-1} + \alpha \nabla f(x_t)$
      $x_{t+1} = x_t - v_t$
      $t \leftarrow t + 1$
    end while
    return x_t
  end function

Theorem 17 (Polyak's Accelerated Gradient Descent convergence for convex quadratic functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous and μ-strongly convex quadratic function. Then the Polyak's Accelerated Gradient Descent with step size $\alpha = \dfrac{4}{(\sqrt{L} + \sqrt{\mu})^2}$ and momentum $\beta = \max\{|1 - \sqrt{\alpha\mu}|, |1 - \sqrt{\alpha L}|\}^2$ satisfies:*

$$\|x_t - x^*\| = \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^t \|x_0 - x^*\|$$

where $\kappa = L/\mu$.

Leveraging the idea of momentum introduced by Polyak, Nesterov introduced a slightly altered update rule that has been shown to converge not only for quadratic functions, but for general convex functions. In the Nesterov's method [2], instead, the velocity vector v_t is calculated by applying the β momentum to the previous v_{t-1} displacement, and subtracting the gradient step to $x_t + \beta v_{t-1}$, which is the point where the momentum term leads from x_t.

Algorithm 4 Nesterov's Accelerated Gradient Descent or Nesterov Heavy-Ball method

Require: Function f to minimize
Require: Learning rate $\alpha > 0$
Require: Momentum $\beta \in [0, 1)$

```
function NesterovAcceleratedGradientDescent(f, α, β)
    Initialize weight vector x_1 ← x_0 and velocity vector v_0 ← 0
    t ← 1
    while not_convergence do
        x̂_t ← x_t + β v_{t−1}
        v_t ← β v_{t−1} + α ∇f(x̂_t)
        x_{t+1} ← x_t − v_t
        t ← t + 1
    end while
    return x_t
end function
```

Comparing the algorithm 3 with the algorithm 4, we can see that Polyak's method evaluates the gradient before adding momentum, whereas Nesterov's algorithm evaluates it after applying momentum, which intuitively brings us closer to the minimum x^*, as showb in figure 10.

Theorem 18 (Nesterov's Accelerated Gradient Descent convergence for convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous convex function. Then the Nesterov's Accelerated Gradient Descent with step size $\alpha = 1/L$ and momentum $\beta_{t+1} = t/(t+3)$ satisfies:*

$$f(x_t) - f(x^*) \leq \frac{2L\|x_0 - x^*\|^2}{(t+1)^2}$$

Theorem 19 (Nesterov's Accelerated Gradient Descent convergence for strongly convex functions)**.** *Let $f : \Re^m \to \Re$ be a L-Lipschitz continuous and μ-strongly convex function. Then the Nesterov's Accelerated Gradient Descent with step size $\alpha = 1/L$ and momentum $\beta = \dfrac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ satisfies:*

$$\begin{aligned} f(x_t) - f(x^*) \leq& \left(1 - \sqrt{\frac{\mu}{L}}\right)^t \left(f(x_0) - f(x^*) + \frac{\mu\|x_0 - x^*\|^2}{2}\right) \\ =& \left(1 - \frac{1}{\sqrt{\kappa}}\right)^t \left(f(x_0) - f(x^*) + \frac{\mu\|x_0 - x^*\|^2}{2}\right) \end{aligned}$$

where $\kappa = L/\mu$.

In summary, the following *convergence rates* and *iterations complexities* are given:

Table 3: Nesterov's Accelerated Gradient Descent convergence rates and iterations complexities

	stepsize rule	convergence rate	iteration complexity
convex and L-Lipschitz	$\alpha = \frac{1}{L}$	$\mathcal{O}\left(\frac{1}{t^2}\right)$	$\mathcal{O}\left(\frac{1}{\sqrt{\epsilon}}\right)$
strongly convex and L-Lipschitz	$\alpha = \frac{1}{L}$	$\mathcal{O}\left(\left(1 - \frac{1}{\sqrt{\kappa}}\right)^t\right)$	$\mathcal{O}\left(\sqrt{\kappa}\log\frac{1}{\epsilon}\right)$

Note that in case of L-Lipschitz continuous and strongly convex functions, Nesterov's momentum gives the acceleration that we had with Polyak's momentum for quadratic functions. This is great because we get the guarantee for a more general class of functions, but these *convergence rates* cannot be futher improved only using first-order information.

6.2 Sequential Minimal Optimization for Wolfe Dual formulations

The *Sequential Minimal Optimization (SMO)* [4] method is the most popular approach for solving the SVM QP problem without any extra Q matrix storage required by common QP methods. The advantage of SMO lies in the fact that it performs a series of two-point optimizations since we deal with just one equality constraint, so the Lagrange multipliers can be solved analitically.

6.2.1 Classification

At each iteration, SMO chooses two α_i to jointly optimize, let α_1 and α_2, finds the optimal values for these multipliers and update the SVM to reflect these new values. In order to solve for two Lagrange multipliers, SMO first computes the constraints over these and then solves for the constrained minimum. Since there are only two multipliers, the box-constraints cause the Lagrange multipliers to lie within a box, while the linear equality constraint causes the Lagrange multipliers to lie on a diagonal line inside the box. So, the constrained minimum must lie there as shown in 11.

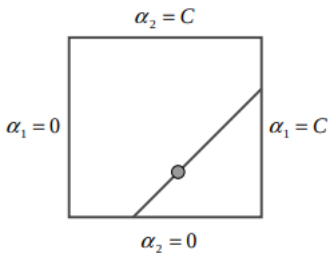


Figure 11: SMO for two Lagrange multipliers

In case of classification the ends of the diagonal line segment, i.e., the lower and upper bounds, can be espressed as follow if the target $y_1 \neq y_2$:

$$\begin{aligned} L &= max(0, \alpha_2 - \alpha_1) \\ H &= min(C, C + \alpha_2 - \alpha_1) \end{aligned} \tag{131}$$

or, alternatively, if the target $y_1 = y_2$:

$$\begin{aligned} L &= max(0, \alpha_2 + \alpha_1 - C) \\ H &= min(C, \alpha_2 + \alpha_1) \end{aligned} \tag{132}$$

The second derivative of the objective quadratic function along the diagonl line can be expressed as:

$$\eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2) \tag{133}$$

that will be grather than zero if the kernel matrix will be positive definite, so there will be a minimum along the linear equality constraints that will be:

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{134}$$

where $E_i = y_i - y_i'$ is the error on the i-th training example and y_i' is the output of the SVC for the same.

Then, the box-constrained minimum is found by clipping the unconstrained minimum to the ends of the line segment:

$$\alpha_2^{new,clipped} = \begin{cases} H & \text{if } \alpha_2^{new} \geq H \\ \alpha_2^{new} & \text{if } L < \alpha_2^{new} < H \\ L & \text{if } \alpha_2^{new} \leq L \end{cases} \tag{135}$$

Finally, the value of α_1 is computed from the new clipped α_2 as:

$$\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped}) \tag{136}$$

where $s = y_1 y_2$.

Since the *Karush-Kuhn-Tucker* conditions are necessary and sufficient conditions for optimality of a positive definite QP problem and the KKT conditions for the classification problem (19) are:

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i y_i' \geq 1 \\ 0 < \alpha_i < C &\Leftrightarrow y_i y_i' = 1 \\ \alpha_i = C &\Leftrightarrow y_i y_i' \leq 1 \end{aligned} \tag{137}$$

the steps described above will be iterate as long as there will be an example that violates them.

After optimizing α_1 and α_2, we select the threshold b such that the KKT conditions are satisfied for x_1 and x_2. If, after optimization, α_1 is not at the bounds, i.e., $0 < \alpha_1 < C$, then the following threshold b_{up} is valid, since it forces the SVC to output y_1 when the input is x_1:

$$b_{up} = E_1 + y_1(\alpha_1^{new} - \alpha_1)K(x_1, x_1) + y_2(\alpha_2^{new,clipped} - \alpha_2)K(x_1, x_2) + b \tag{138}$$

similarly, the following threshold b_{low} is valid if $0 < \alpha_2 < C$:

$$b_{low} = E_2 + y_1(\alpha_1^{new} - \alpha_1)K(x_1, x_2) + y_2(\alpha_2^{new,clipped} - \alpha_2)K(x_2, x_2) + b \tag{139}$$

If, after optimization, both $0 < \alpha_1 < C$ and $0 < \alpha_2 < C$ then both these thresholds are valid, and they will be equal; else, if both α_1 and α_2 are at the bounds, i.e., $\alpha_1 = 0$ or $\alpha_1 = C$ and $\alpha_2 = 0$ or $\alpha_2 = C$, then all the thresholds between b_{up} and b_{low} satisfy the KKT conditions, so we choose the threshold to be halfway in between b_{up} and b_{low}. This gives the complete equation for b:

$$b = \begin{cases} b_{up} & \text{if } 0 < \alpha_1 < C \\ b_{low} & \text{if } 0 < \alpha_2 < C \\ \dfrac{b_{up} + b_{low}}{2} & \text{otherwise} \end{cases} \tag{140}$$

Algorithm 5 Sequential Minimal Optimization for Classification

Require: Training examples matrix $X \in \Re^{n \times m}$
Require: Training target vector $y \in \pm 1^n$
Require: Kernel matrix $K \in \Re^{n \times n}$
Require: Regularization parameter $C > 0$
Require: Tolerance value *tol* for stopping criterion

```
function SMOClassifier(X, y, K, C, tol)
    Initialize the Lagrange multipliers vector α ∈ ℜ^n, α ← 0
    Initialize the empty set I0 ← {i : 0 < α_i < C}
    Initialize the set I1 ← {i : y_i = +1, α_i = 0} to contain all the indices of the training examples of class +1
    Initialize the empty set I2 ← {i : y_i = −1, α_i = C}
    Initialize the empty set I3 ← {i : y_i = +1, α_i = C}
    Initialize the set I4 ← {i : y_i = −1, α_i = 0} to contain all the indices of the training examples of class −1
    Initialize b_up ← −1
    Initialize b_low ← +1
    Initialize the error cache vector errors ∈ ℜ^n, errors ← 0
    while num_changed > 0 or examine_all = True do
        num_changed ← 0
        examine_all ← True
        if examine_all = True then
            for i ← 0 to n do                                   ▷ loop over all training examples
                num_changed ← num_changed + ExamineExample(i)
            end for
        else
            for i in I0 do          ▷ loop over examples where α_i are not already at their bounds
                num_changed ← num_changed + ExamineExample(i)
                if b_up > b_low − 2tol then                   ▷ check if optimality on I0 is attained
                    num_changed ← 0
                    break
                end if
            end for
        end if
        if examine_all = True then
            examine_all ← False
        else if num_changed = 0 then
            examine_all ← True
        end if
    end while
    Compute b by (140)
    return α, b
end function
```

Require: $i2$-th Lagrange multiplier
  function EXAMINEEXAMPLE($i2$)
    if $i2$ in $I0$ **then**
      $E_2 \leftarrow errors_{i2}$
    else
      Compute E_2
      $errors_{i2} \leftarrow E_2$
      Update (b_{low}, i_{low}) or (b_{up}, i_{up}) using $(E_2, i2)$
    end if
    if optimality is attained using current b_{low} and b_{up} **then**
      return 0
    else
      Find an index $i1$ to do joint optimization with $i2$
      if TAKESTEP($i1, i2$) = True **then**
        return 1
      else
        return 0
      end if
    end if
  end function

Require: $i1$-th Lagrange multiplier
Require: $i2$-th Lagrange multiplier
  function TAKESTEP($i1, i2$)
    if $i1 = i2$ **then**
      return False
    end if
    Compute L and H using (131) or (132)
    if $L = H$ **then**
      return False
    end if
    Compute η by (133) ▷ we assume that $\eta > 0$, i.e., the kernel matrix K is positive definite
    if $\eta < 0$ **then**
      Choose $\alpha_2^{new,clipped}$ between L and H according to the largest value of the objective function at these points
    else
      Compute α_2^{new} by (134)
      Compute $\alpha_2^{new,clipped}$ by (135)
    end if
    if changes in $\alpha_2^{new,clipped}$ are larger than some eps **then**
      Compute α_1^{new} by (136)
      Update $\alpha_2^{new,clipped}$ and α_1^{new}
      for i in $I0$ **do**
        Update $errors_i$ using new Lagrange multipliers
      end for
      Update α using new Lagrange multipliers
      Update $I0, I1, I2, I3$ and $I4$
      Update $errors_{i1}$ and $errors_{i2}$
      for i in $I0 \cup \{i1, i2\}$ **do**
        Compute (i_{low}, b_{low}) by $b_{low} = \max\{errors_i : i \in I0 \cup I3 \cup I4\}$
        Compute (i_{up}, b_{up}) by $b_{up} = \min\{errors_i : i \in I0 \cup I1 \cup I2\}$
      end for
      return True
    else
      return False
    end if
  end function

6.2.2 Regression

In case of regression the bounds and the new multipliers $\alpha_1^{+,new}$ and $\alpha_2^{+,new}$ can be expressed as follows if $(\alpha_1^+ > 0$ or $(\alpha_1^- = 0$ and $E_1 - E_2 > 0))$ and $(\alpha_2^+ > 0$ or $(\alpha_2^- = 0$ and $E_1 - E_2 < 0))$:

$$\begin{aligned} L &= max(0, \gamma - C) \\ H &= min(C, \gamma) \end{aligned} \tag{141}$$

$$\alpha_2^{+,new} = \alpha_2^+ - \frac{E_1 - E_2}{\eta} \tag{142}$$

$$\alpha_1^{+,new} = \alpha_1^+ - (\alpha_2^{+,new,clipped} - \alpha_2^+) \tag{143}$$

or, if $(\alpha_1^+ > 0$ or $(\alpha_1^- = 0$ and $E_1 - E_2 > 2\epsilon))$ and $(\alpha_2^- > 0$ or $(\alpha_2^+ = 0$ and $E_1 - E_2 > 2\epsilon))$:

$$\begin{aligned} L &= max(0, -\gamma) \\ H &= min(C, -\gamma + C) \end{aligned} \tag{144}$$

$$\alpha_2^{-,new} = \alpha_2^- + \frac{(E_1 - E_2) - 2\epsilon}{\eta} \tag{145}$$

$$\alpha_1^{+,new} = \alpha_1^+ + (\alpha_2^{-,new,clipped} - \alpha_2^-) \tag{146}$$

or, if $(\alpha_1^- > 0$ or $(\alpha_1^+ = 0$ and $E_1 - E_2 < -2\epsilon))$ and $(\alpha_2^+ > 0$ or $(\alpha_2^- = 0$ and $E_1 - E_2 < -2\epsilon))$:

$$\begin{aligned} L &= max(0, \gamma) \\ H &= min(C, C + \gamma) \end{aligned} \tag{147}$$

$$\alpha_2^{+,new} = \alpha_2^+ - \frac{(E_1 - E_2) + 2\epsilon}{\eta} \tag{148}$$

$$\alpha_1^{-,new} = \alpha_1^- + (\alpha_2^{+,new,clipped} - \alpha_2^+) \tag{149}$$

or, finally, if $(\alpha_1^- > 0$ or $(\alpha_1^+ = 0$ and $E_1 - E_2 < 0))$ and $(\alpha_2^- > 0$ or $(\alpha_2^+ = 0$ and $E_1 - E_2 > 0))$:

$$\begin{aligned} L &= max(0, -\gamma - C) \\ H &= min(C, -\gamma) \end{aligned} \tag{150}$$

$$\alpha_2^{-,new} = \alpha_2^- + \frac{E_1 - E_2}{\eta} \tag{151}$$

$$\alpha_1^{-,new} = \alpha_1^- - (\alpha_2^{-,new,clipped} - \alpha_2^-) \tag{152}$$

where $\gamma = \alpha_1^+ - \alpha_1^- + \alpha_2^+ - \alpha_2^-$. Note that η and $\alpha_2^{+,new,clipped}$ or $\alpha_2^{-,new,clipped}$ are identical to (133) and (135) respectively.

The KKT conditions for the regression problem (79) are:

$$\begin{aligned} \alpha_i^+ - \alpha_i^- = 0 &\Leftrightarrow |y_i - y_i'| < \epsilon \\ -C < \alpha_i^+ - \alpha_i^- < C &\Leftrightarrow |y_i - y_i'| = \epsilon \\ \alpha_i^+ + \alpha_i^- = C &\Leftrightarrow |y_i - y_i'| > \epsilon \end{aligned} \tag{153}$$

so, the steps described above will be iterate as long as there will be an example that violates them.

In case of regression we select the threshold b as follows:

$$b_{up} = E_1 + ((\alpha_1^+ - \alpha_1^-) - (\alpha_1^{+,new} - \alpha_1^{-,new}))K(x_1, x_1) + ((\alpha_2^+ - \alpha_2^-) - (\alpha_2^{+,new,clipped} - \alpha_2^{-,new,clipped}))K(x_1, x_2) + b \tag{154}$$

$$b_{low} = E_2 + ((\alpha_1^+ - \alpha_1^-) - (\alpha_1^{+,new} - \alpha_1^{-,new}))K(x_1, x_2) + ((\alpha_2^+ - \alpha_2^-) - (\alpha_2^{+,new,clipped} - \alpha_2^{-,new,clipped}))K(x_2, x_2) + b \tag{155}$$

$$b = \begin{cases} b_{up} & \text{if } 0 < \alpha_1^+, \alpha_1^- < C \\ b_{low} & \text{if } 0 < \alpha_2^+, \alpha_2^- < C \\ \dfrac{b_{up} + b_{low}}{2} & \text{otherwise} \end{cases} \tag{156}$$

The improvements described in [5, 8] for classification and regression respectively are about the definition of subsets of multipliers to efficiently update them at each iteration by separating the multipliers at the bounds from those who can be further minimized.

Algorithm 6 Sequential Minimal Optimization for Regression

Require: Training examples matrix $X \in \Re^{n \times m}$
Require: Training target vector $y \in \Re^{n}$
Require: Kernel matrix $K \in \Re^{n \times n}$
Require: Regularization parameter $C > 0$
Require: Epsilon-tube value $\epsilon \geq 0$ within which no penalty is associated in the epsilon-insensitive loss function
Require: Tolerance value tol for stopping criterion

```
function SMORegression(X, y, K, C, ϵ, tol)
    Initialize the Lagrange multipliers vector α⁺ ∈ ℜⁿ, α⁺ ← 0
    Initialize the Lagrange multipliers vector α⁻ ∈ ℜⁿ, α⁻ ← 0
    Initialize the empty set I0 ← {i : 0 < α_i⁺, α_i⁻ < C}
    Initialize the set I1 ← {i : α_i⁺ = 0, α_i⁻ = 0} to contain all the indices of the training examples
    Initialize the empty set I2 ← {i : α_i⁺ = 0, α_i⁻ = C}
    Initialize the empty set I3 ← {i : α_i⁺ = C, α_i⁻ = 0}
    Initialize i_up ← 0                    ▷ or any other target index i_up from the training examples
    Initialize i_low ← 0                   ▷ or any other target index i_low from the training examples
    Initialize b_up ← y_{i_up} + ϵ
    Initialize b_low ← y_{i_low} − ϵ
    Initialize the error cache vector errors ∈ ℜⁿ, errors ← 0
    while num_changed > 0 or examine_all = True do
        num_changed ← 0
        examine_all ← True
        if examine_all = True then
            for i ← 0 to n do                                  ▷ loop over all training examples
                num_changed ← num_changed + ExamineExample(i)
            end for
        else
            for i in I0 do         ▷ loop over examples where α_i⁺ and α_i⁻ are not already at their bounds
                num_changed ← num_changed + ExamineExample(i)
                if b_up > b_low − 2tol then                    ▷ check if optimality on I0 is attained
                    num_changed ← 0
                    break
                end if
            end for
        end if
        if examine_all = True then
            examine_all ← False
        else if num_changed = 0 then
            examine_all ← True
        end if
    end while
    Compute b by (156)
    return α⁺, α⁻, b
end function
```

Require: $i1$-th Lagrange multiplier
Require: $i2$-th Lagrange multiplier
  function TAKESTEP($i1, i2$)
    if $i1 = i2$ **then**
      return False
    end if
    $finished = False$
    while not $finished$ **do**
      Compute L and H using (141), (144), (147) or (150)
      if $L < H$ **then**
        Compute η by (133) ▷ we assume that $\eta > 0$, i.e., the kernel matrix K is positive definite
        if $\eta < 0$ **then**
          Choose $\alpha_2^{+,new,clipped}$ or $\alpha_2^{-,new,clipped}$ between L and H according to the largest value of the objective function at these points
        else
          Compute $\alpha_2^{+,new}$ or $\alpha_2^{-,new}$ using (142), (148) or (145), (151) respectively
          Compute $\alpha_2^{+,new,clipped}$ or $\alpha_2^{-,new,clipped}$ by (135)
        end if
        Compute $\alpha_1^{+,new}$ or $\alpha_1^{-,new}$ using (143), (146) or (149), (152) respectively
        if changes in $\alpha_2^{+,new,clipped}, \alpha_2^{-,new,clipped}, \alpha_1^{+,new}$ or $\alpha_1^{-,new}$ are larger than some eps **then**
          Update $\alpha_2^{+,new,clipped}, \alpha_2^{-,new,clipped}, \alpha_1^{+,new}$ or $\alpha_1^{-,new}$
        end if
      else
        $finished = True$
      end if
    end while
    if changes in $\alpha_2^{+,new,clipped}, \alpha_2^{-,new,clipped}, \alpha_1^{+,new}$ or $\alpha_1^{-,new}$ are larger than some eps **then**
      for i in $I0$ **do**
        Update $errors_i$ using new Lagrange multipliers
      end for
      Update α^+ and α^- using new Lagrange multipliers
      Update $I0, I1, I2$ and $I3$
      Update $errors_{i1}$ and $errors_{i2}$
      for i in $I0 \cup \{i1, i2\}$ **do**
        Compute (i_{low}, b_{low}) by $b_{low} = \max\{errors_i : i \in I0 \cup I1 \cup I2\}$
        Compute and (i_{up}, b_{up}) by $b_{up} = \min\{errors_i : i \in I0 \cup I1 \cup I3\}$
      end for
      return True
    else
      return False
    end if
  end function

6.3 AdaGrad for Lagrangian Dual formulations

Due to the sparsity of the weight vector of the *Lagrangian dual*, i.e., the Lagrange multipliers, we might end up in a situation where some components of the gradient are very small and others large. This, in terms of *conditioning number*, i.e., $\kappa = L/\mu \gg 1$, means that the level sets of f are ellipsoid, i.e., we are dealing with an ill-conditioned problem. So, given a learning rate, a standard gradient descent approach might end up in a situation where it decreases too quickly the small weights or too slowly the large ones.

Another method, that is usually deprecated in ML applications due to its increased computational complexity, is Newton's method. Newton's method favors a much faster *convergence rate*, i.e., number of iterations, at the cost of being more expensive per iteration. For convex problems, the recursion is similar to the gradient descent algorithm:

$$x_{t+1} = x_t - \alpha H^{-1} \nabla f(x_t)$$

where α is often close to one (damped-Newton) or one, and H^{-1} denotes the Hessian of f at the current point, i.e., $\nabla^2 f(x_t)$.

The above suggest a general rule in optimization: find any preconditioner, in convex optimization it has to be positive semidefinite, that improves the performance of gradient descent in terms of iterations, but without wasting too much time to compute that precoditioner. The above result into:

$$x_{t+1} = x_t - \alpha P^{-1} \nabla f(x_t)$$

where P is the preconditioner. This idea is the basis of the BFGS quasi-Newton method.

The *AdaGrad* [6] algorithm is just a variant of preconditioned gradient descent, where P is selected to be a diagonal preconditioner matrix and is updated using the gradient information, in particular it is the diagonal approximation of the inverse of the square roots of gradient outer products, until the k-th iteration. The above lead to the algorithm:

Algorithm 7 AdaGrad

Require: Function f to minimize
Require: Learning rate or step size $\alpha > 0$
Require: Offset $\epsilon > 0$ to ensures not divide by 0
  function AdaGrad(f, α, ϵ)
    Initialize weight vector x_0 and the squared accumulated gradients vector $s_t \leftarrow 0$
    $t = 1$
    while *not_convergence* **do**
      $g_t \leftarrow \partial f(x_t)$ ▷ if f is differentiable then $\partial f(x_t) = \nabla f(x_t)$
      $s_t \leftarrow s_{t-1} + g_t^2$
      $x_{t+1} \leftarrow x_t - \alpha P_t^{-1} g_t = x_t - \dfrac{\alpha}{\sqrt{s_t + \epsilon}} \odot g_t$ where $P_t \leftarrow diag(s_t + \epsilon)^{1/2}$
      $t \leftarrow t + 1$
    end while
    return x_t
  end function

In practical terms, *AdaGrad* addresses the problem of the sparse optimal by adaptively scaling the learning rate for each dimension with the magnitude of the gradients. Coordinates that routinely correspond to large gradients are scaled down significantly, whereas others with small gradients receive a much more gentle treatment.

6.4 Losses properties

Several losses and objectives have been presented in section 3 and 4. In our experiments, we will consider the following.

For what about the loss functions, two of them are nonsmooth convex functions, i.e., the *hinge* and the *epsilon-insensitive* losses for *classification* and *regression* tasks respectively, and linearly penalizes the misclassified points, i.e., $\mathcal{L}_1$-SVM, meanwhile, their two *squared* versions are smooth, i.e., $\mathcal{L}_2$-SVM, and quadratically penalizes the misclassified points.

Also, both the *margin-based* losses, i.e., the *hinge* and the *squared hinge* losses, are L_f-Lipschitz continuous; meanwhile, among the *distance-based* losses, the *epsilon-insensitive* loss is L_f-Lipschitz continuous but the *squared epsilon-insensitive* is not L_f-Lipschitz continuous, however it is convex and for this reason is locally L_f-Lipschitz continuous.

Also the regularization term, i.e., $\frac{1}{2}\|w\|^2$, is not L_f-Lipschitz continuous since it becomes arbitrarily steep as w approaches infinity, but it is strictly convex and for this reason is locally L_f-Lipschitz continuous. Clearly, its gradient, i.e., w, is not bounded since, again, they go to infinity as w goes to infinity, so this function is not L-Lipschitz continuous.

Since for our purposes, we need to show that our $\mathcal{L}_1$-SVM objectives are L_f-Lipschitz continuous and the $\mathcal{L}_2$-SVM objectives are L-Lipschitz continuous for the applicability of the convergence theorems, we will use the theorem 7 and 8 respectively.

In general, if the objective function of a quadratic programming problem is strictly convex, i.e., the associated Hessian matrix is positive definite, the solution is unique. Meanwhile, if the objective function is convex, there may be cases where the solution is nonunique.

Assume that the hard margin SVM has a solution, i.e., the given problem is separable in the feature space. Then, since the objective function of the primal problem is $\frac{1}{2}\|w\|^2$, which is strongly convex, the primal problem has a unique solution for w and b.

Since the $\mathcal{L}_1$-SVM linearly penalizes the misclassified points, the primal objective function is convex. Likewise, the Hessian matrix of the dual objective function is positive semidefinite. Thus the primal and dual solutions may be nonunique. Meanwhile, the objective function of the primal problem for the $\mathcal{L}_2$-SVM is strictly convex, due to the quadratic penalization of the misclassified points. Therefore, w and b are uniquely determined if we solve the primal or dual problem.

In summary, the following properties for the SVM's objectives are given:

Table 4: SVM's objectives properties for primal formulations

objective	smooth	Lipschitz continuous	convexity
$\mathcal{L}_1$-SVC (13)	no	L_f-Lipschitz	convex
$\mathcal{L}_2$-SVC (48)	yes	L-Lipschitz	strongly convex
$\mathcal{L}_1$-SVR (72)	no	L_f-Lipschitz	convex
$\mathcal{L}_2$-SVR (106)	yes	L-Lipschitz	strongly convex

And, according to the theoretical analysis, the following *convergence rates* are given for the primal and *Lagrangian dual* formulations respectively:

Table 5: SVM's objectives convergence rates for primal formulations

objective	SGD convergence rate	Polyak SGD convergence rate	Nesterov SGD convergence rate
$\mathcal{L}_1$-SVM (13, 72)	$\mathcal{O}\left(\frac{m}{\sqrt{t}}\right)$	$\mathcal{O}\left(\frac{m}{\sqrt{t}}\right)$	$\mathcal{O}\left(\frac{m}{\sqrt{t}}\right)$
$\mathcal{L}_2$-SVM (48, 106)	$\mathcal{O}\left(\frac{m}{t}\right)$	$\mathcal{O}\left(\frac{m}{t}\right)$	$\mathcal{O}\left(\frac{m}{t^2}\right)$

Table 6: SVM's objectives convergence rate for Lagrangian dual formulations

objective	AdaGrad convergence rate
$\mathcal{L}_1$-SVM (42, 100) or (43, 101)	$\mathcal{O}\left(\frac{nm}{\sqrt{t}}\right)$
$\mathcal{L}_2$-SVM (63, 121) or (64, 122)	$\mathcal{O}\left(\frac{nm}{t}\right)$

7 Experiments

The following experiments refer to *linearly* and *nonlinearly* separable generated datasets of size 100. All the training times refer to running on a laptop with an Intel i7-6700HQ (8) @ 3.500GHz and 31.2 GB of memory.

The Python source code is available at: `github.com/dmeoli/optiml`.

7.1 Support Vector Classifier

Below experiments are about the SVC for which I tested different values for the regularization hyperparameter C, i.e., from *soft* to *hard margin*, and in case of nonlinearly separable data also different *kernel functions* mentioned above.

The experiments about SVCs are available at:
`github.com/dmeoli/optiml/blob/master/notebooks/optimization/CM_SVC_report_experiments.ipynb`.

7.1.1 Hinge loss

Primal formulation The experiments results shown in 7 referred to *Stochastic Gradient Descent* algorithm are obtained with α, i.e., the *learning rate* or *step size*, setted to 0.02 and β, i.e., the *momentum*, equal to 0.5. The optimization process is stopped if after 5 iterations the function value does not improve by at least 1e$-$8.

Table 7: Primal $\mathcal{L}_1$-SVC results

			fit_time	accuracy	n_iter	n_sv
solver	momentum	C				
sgd	none	0.1	7.062483	0.975	7196	36
		1.0	3.250063	0.985	4092	15
		10.0	0.795309	0.980	947	10
	polyak	0.1	4.450933	0.975	3589	37
		1.0	1.794078	0.985	2098	16
		10.0	0.367355	0.980	467	10
	nesterov	0.1	3.677246	0.975	4344	37
		1.0	1.788335	0.985	2115	16
		10.0	0.414720	0.980	472	10
liblinear	-	0.1	0.007178	0.980	31	37
		1.0	0.004740	0.985	332	16
		10.0	0.002364	0.985	1183	7

The results provided from the *custom* implementation, i.e., the SGD with different momentum settings, are strongly similar to those of *sklearn* implementation, i.e., *liblinear* [10] implementation, in terms of *accuracy* score. More training data points are selected as *support vectors* from the SGD solver but it always requires lower iterations, i.e., epochs, to achieve the same *numerical precision*. *Polyak* and *Nesterov* momentums always perform lower iterations as expected from the theoretical analysis of the convergence rate.

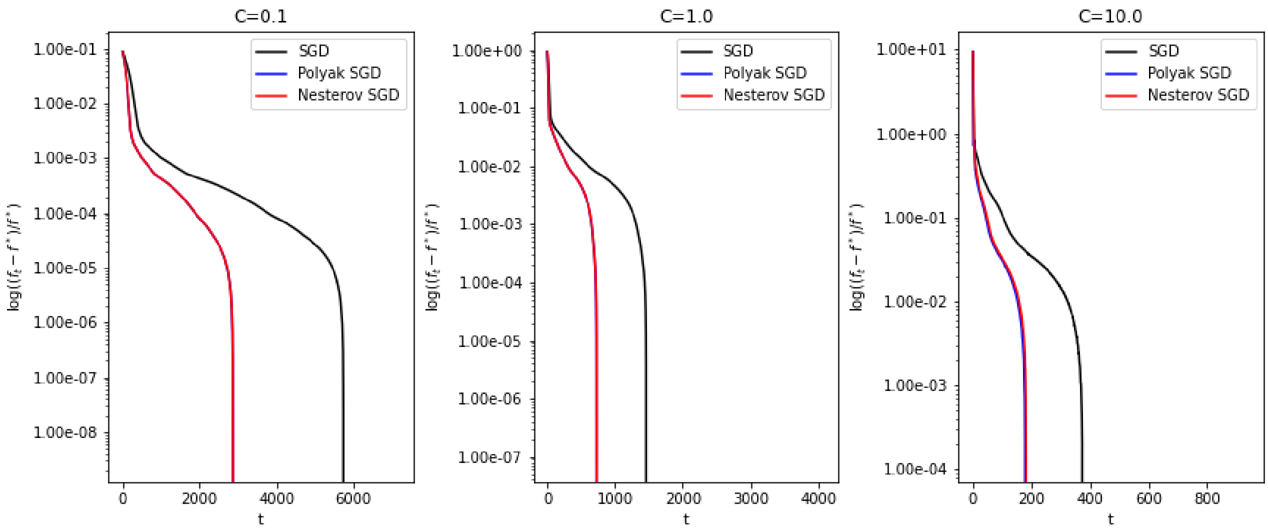


Figure 12: SGD Convergence for the Primal formulation of the $\mathcal{L}_1$-SVC

Linear Dual formulations The experiments results shown in 9 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (42) and (43) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 8: Wolfe Dual linear $\mathcal{L}_1$-SVC results

solver	C	fit_time	accuracy	n_iter	n_sv
smo	0.1	0.158289	0.985	33	38
	1.0	0.209209	0.980	62	17
	10.0	0.352789	0.980	295	10
libsvm	0.1	0.007530	0.985	37	38
	1.0	0.008367	0.985	243	17
	10.0	0.003456	0.985	194	10
cvxopt	0.1	0.079530	0.985	9	38
	1.0	0.073912	0.980	10	17
	10.0	0.071737	0.980	10	11

For what about the linear *Wolfe dual* formulation we can immediately notice as higher *regularization hyperparameter C* makes the model harder, so the *custom* implementation of the SMO algorithm and also the *sklearn* implementation, i.e., *libsvm* [11] implementation, needs to perform more iterations to achieve the same *numerical precision*; meanwhile the *cvxopt* [12] seems to be insensitive to the increasing complexity of the model. The results in terms of *accuracy* and number of *support vectors* are strongly similar to each others.

Table 9: Lagrangian Dual linear $\mathcal{L}_1$-SVC results

dual	C	fit_time	accuracy	n_iter	n_sv
reg_bias	0.1	74.882949	0.985	62016	37
	1.0	124.619140	0.980	63970	17
	10.0	132.066754	0.980	71086	10
unreg_bias	0.1	198.488607	0.985	93132	38
	1.0	127.705429	0.980	67696	17
	10.0	136.623035	0.980	74848	10

For what about the linear *Lagrangian dual* formulation we can see as it seems to be insensitive to the increasing complexity of the model in terms of number of *iterations* but it tends to select many training data points as *support vectors*.

Nonlinear Dual formulations The experiments results shown in 10 and 11 are obtained with d and r hyperparameters equal to 3 and 1 respectively for the *polynomial* kernel; *gamma* is setted to *'scale'* for both *polynomial* and *gaussian* kernels. The experiments results shown in 11 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (42) and (43) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e$-$8 between two consecutive iterations.

Table 10: Wolfe Dual nonlinear $\mathcal{L}_1$-SVC results

solver	kernel	C	fit_time	accuracy	n_iter	n_sv
smo	gaussian	0.1	0.878962	1.0000	65	222
		1.0	0.860773	1.0000	76	48
		10.0	0.525582	1.0000	29	13
	poly	0.1	1.096614	0.8675	121	142
		1.0	0.999406	0.6825	143	30
		10.0	0.680124	0.9475	65	10
libsvm	gaussian	0.1	0.011811	1.0000	131	222
		1.0	0.007152	1.0000	252	50
		10.0	0.003799	1.0000	134	13
	poly	0.1	0.022929	1.0000	210	143
		1.0	0.010823	1.0000	233	30
		10.0	0.003304	1.0000	118	10
cvxopt	gaussian	0.1	0.426778	1.0000	10	222
		1.0	0.568489	1.0000	10	49
		10.0	0.557587	1.0000	10	14
	poly	0.1	0.487677	0.8575	10	143
		1.0	0.693849	0.6775	10	31
		10.0	0.610578	0.9475	10	10

Table 11: Lagrangian Dual nonlinear $\mathcal{L}_1$-SVC results

dual	kernel	C	fit_time	accuracy	n_iter	n_sv
reg_bias	gaussian	0.1	243.423625	1.0000	48809	222
		1.0	824.143136	1.0000	165224	50
		10.0	620.770894	1.0000	121122	13
	poly	0.1	503.107014	0.8575	96857	143
		1.0	444.329111	0.6775	104275	31
		10.0	440.088633	0.9475	103227	10
unreg_bias	gaussian	0.1	229.974389	1.0000	49604	222
		1.0	1296.649206	1.0000	250943	50
		10.0	999.276160	1.0000	202406	13
	poly	0.1	538.226101	0.8600	102946	143
		1.0	506.097667	0.6775	124476	31
		10.0	466.560731	0.9475	108691	10

The same considerations made for the previous linear *Wolfe dual* and *Lagrangian dual* formulations are confirmed also in the nonlinearly separable case. In this setting the complexity of the model coming with higher C regularization values seems to be not paying a tradeoff in terms of the number of *iterations* of the algorithm and, moreover, the *reg_bias Lagrangian dual* formulation seems to perform better wrt the *unreg_bias* formulation, both tends to select even more training data points as *support vectors*.

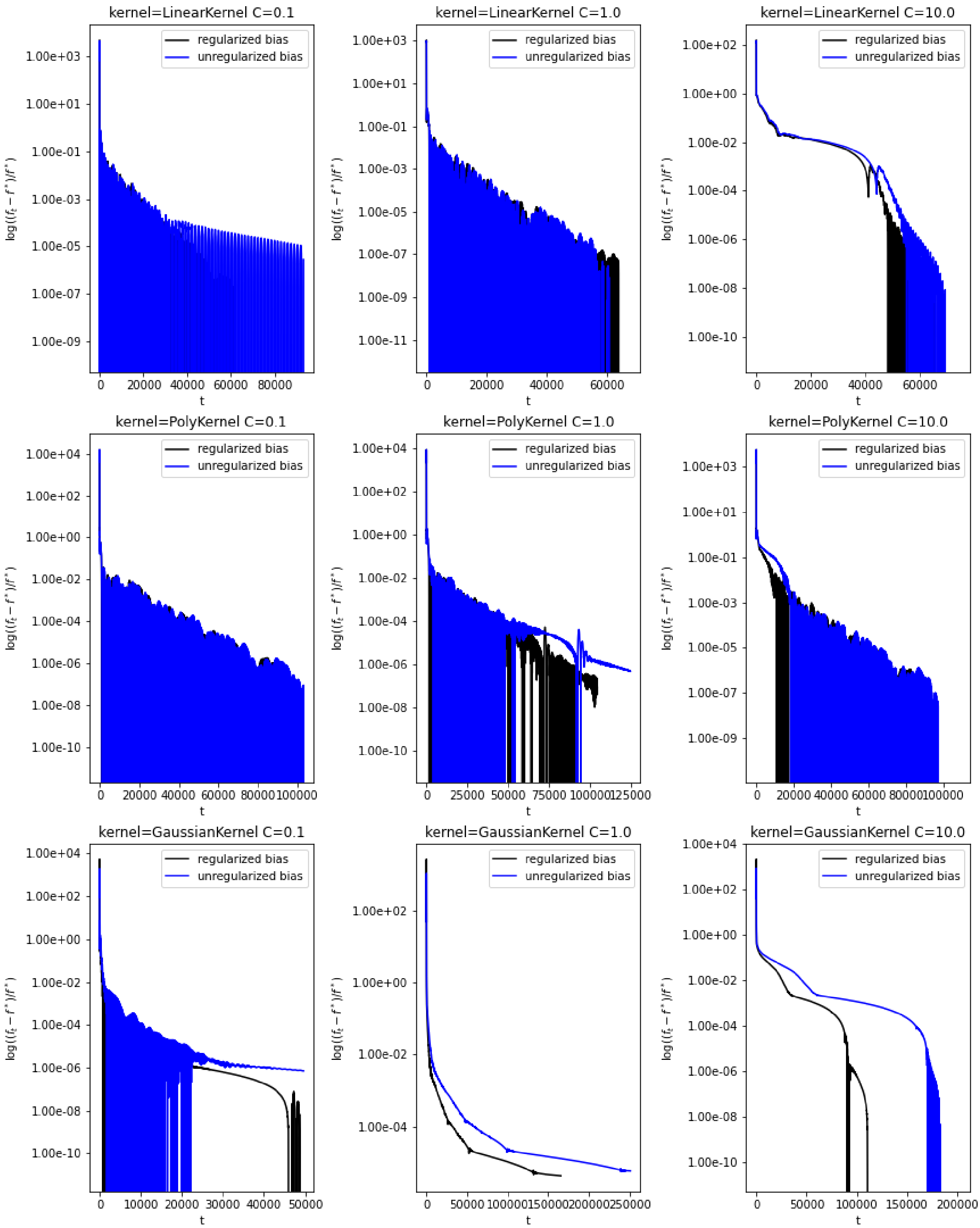


Figure 13: AdaGrad convergence for the Lagrangian Dual formulation of the nonlinear $\mathcal{L}_1$-SVC

7.1.2 Squared Hinge loss

Primal formulation The experiments results shown in 12 referred to *Stochastic Gradient Descent* algorithm are obtained with α, i.e., the *learning rate* or *step size*, setted to 0.02 and β, i.e., the *momentum*, equal to 0.5. The optimization process is stopped if after 5 iterations the function value does not improve by at least 1e−8.

Table 12: Primal $\mathcal{L}_2$-SVC results

solver	momentum	C	fit_time	accuracy	n_iter	n_sv
sgd	none	0.1	4.478252	0.980	5661	49
		1.0	4.060005	0.980	5165	25
		10.0	2.564839	0.980	1895	19
	polyak	0.1	2.756118	0.980	3287	49
		1.0	2.298601	0.980	2827	25
		10.0	0.023973	0.975	10	20
	nesterov	0.1	2.714415	0.980	3291	49
		1.0	2.367481	0.980	2835	25
		10.0	0.731884	0.980	849	19
liblinear	-	0.1	0.003543	0.980	52	46
		1.0	0.005556	0.980	563	25
		10.0	0.008653	0.980	3129	19

Again, the results provided from the *custom* implementation, i.e., the SGD with different momentum settings, are strongly similar to those of *sklearn* implementation, i.e., *liblinear* [10] implementation, in terms of *accuracy* score. More training data points are selected as *support vectors* from the SGD solver but it always requires even lower iterations, i.e., epochs, to achieve the same *numerical precision*. *Polyak* and *Nesterov* momentums always perform lower iterations as expected from the theoretical analysis of the convergence rate.

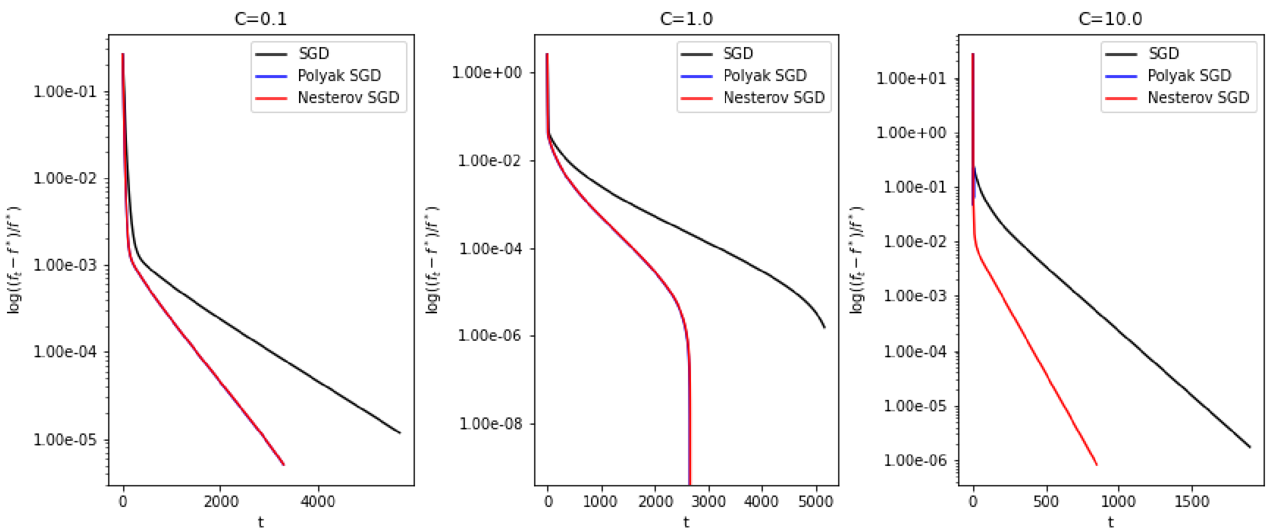


Figure 14: SGD convergence for the Primal formulation of the $\mathcal{L}_2$-SVC

Linear Dual formulations The experiments results shown in 13 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (63) and (64) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 13: Lagrangian Dual linear $\mathcal{L}_2$-SVC results

dual	C	fit_time	accuracy	n_iter	n_sv
reg_bias	0.1	8.659919	0.98	6020	46
	1.0	50.128582	0.98	31786	25
	10.0	230.757967	0.98	120271	19
unreg_bias	0.1	10.267597	0.98	6391	47
	1.0	51.611019	0.98	32856	25
	10.0	234.691407	0.98	127862	19

For what about the linear *Lagrangian dual* formulation we can see as it seems to be insensitive to the increasing complexity of the model in terms of number of *iterations* but it tends to select many training data points as *support vectors*.

Nonlinear Dual formulations The experiments results shown in 14 are obtained with d and r hyperparameters equal to 3 and 1 respectively for the *polynomial* kernel; *gamma* is setted to *'scale'* for both *polynomial* and *gaussian* kernels. The experiments results shown in 11 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (63) and (64) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 14: Lagrangian Dual nonlinear $\mathcal{L}_2$-SVC results

dual	kernel	C	fit_time	accuracy	n_iter	n_sv
reg_bias	gaussian	0.1	3.462003	1.0000	1693	345
		1.0	36.790274	1.0000	11169	130
		10.0	233.673823	1.0000	59626	33
	poly	0.1	27.774890	0.8550	10137	233
		1.0	191.140481	0.6950	51914	80
		10.0	371.644588	0.7300	91925	16
unreg_bias	gaussian	0.1	8.036577	1.0000	3443	344
		1.0	69.095447	1.0000	20156	130
		10.0	377.351724	1.0000	96154	33
	poly	0.1	28.834911	0.8625	10859	234
		1.0	201.957567	0.6950	54832	80
		10.0	388.242444	0.7300	97414	16

The same considerations made for the previous linear *Lagrangian dual* formulations are confirmed also in the nonlinearly separable case. In this setting the complexity of the model coming with higher C regularization values seems to be not paying a tradeoff in terms of the number of *iterations* of the algorithm and, moreover, the *reg_bias Lagrangian dual* formulation seems to perform better wrt the *unreg_bias* formulation, both tends to select even more training data points as *support vectors*.

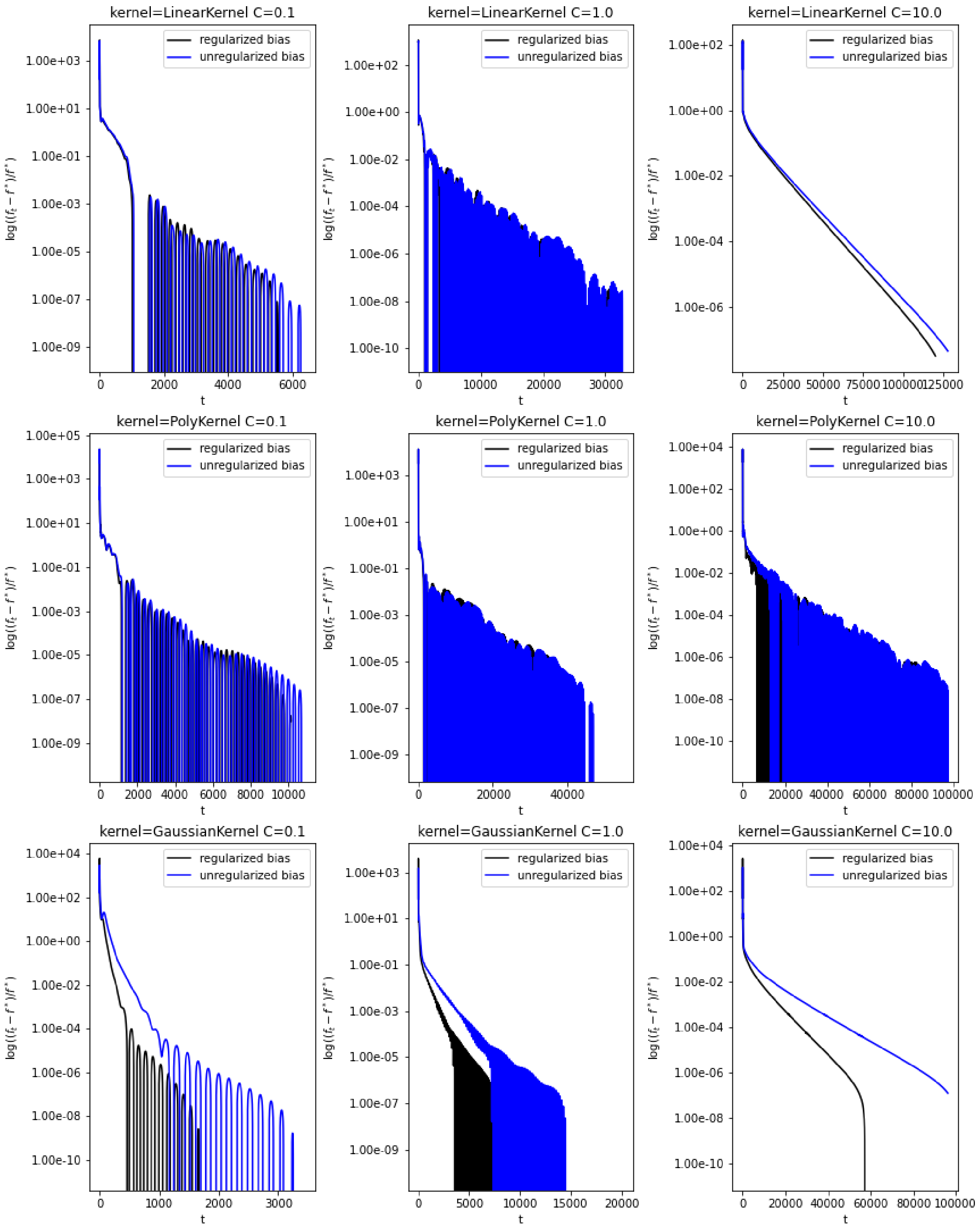


Figure 15: AdaGrad convergence for the Lagrangian Dual formulation of the nonlinear $\mathcal{L}_2$-SVC

7.2 Support Vector Regression

Below experiments are about the SVR for which I tested different values for regularization hyperparameter C, i.e., from *soft* to *hard margin*, the ϵ penalty value and in case of nonlinearly separable data also different *kernel functions* mentioned above.

The experiments about SVRs are available at:
`github.com/dmeoli/optiml/blob/master/notebooks/optimization/CM_SVR_report_experiments.ipynb`.

7.2.1 Epsilon-insensitive loss

Primal formulation The experiments results shown in 15 referred to *Stochastic Gradient Descent* algorithm are obtained with α, i.e., the *learning rate* or *step size*, setted to 0.02 and β, i.e., the *momentum*, equal to 0.4. The optimization process is stopped if after 5 iterations the function value does not improve by at least 1e$-$8.

Table 15: Primal $\mathcal{L}_1$-SVR results

solver	momentum	C	epsilon	fit_time	r2	n_iter	n_sv
sgd	none	1	0.1	13.207407	0.954298	16161	100
			0.2	9.523358	0.954544	12267	99
			0.3	11.594773	0.955424	13665	99
		10	0.1	0.690761	0.983893	806	98
			0.2	0.722907	0.983891	884	98
			0.3	0.756686	0.983884	958	97
		100	0.1	0.121053	0.984034	85	97
			0.2	0.088140	0.984047	96	98
			0.3	0.156799	0.984056	109	98
	polyak	1	0.1	7.895537	0.954321	9874	100
			0.2	5.816864	0.954549	7400	99
			0.3	6.935173	0.955424	8200	99
		10	0.1	0.489926	0.983893	487	97
			0.2	0.503936	0.983891	535	98
			0.3	0.483785	0.983885	569	98
		100	0.1	0.090085	0.984030	48	98
			0.2	0.108489	0.984046	56	98
			0.3	0.114756	0.984055	61	97
	nesterov	1	0.1	8.996545	0.954310	9785	100
			0.2	6.001156	0.954546	7382	99
			0.3	7.318327	0.955424	8198	99
		10	0.1	0.803194	0.983892	489	97
			0.2	0.457146	0.983890	533	97
			0.3	0.561535	0.983884	579	98
		100	0.1	0.097744	0.984031	61	98
			0.2	0.120314	0.984047	58	98
			0.3	0.107943	0.984057	62	98
liblinear	-	1	0.1	0.019717	0.954684	12	100
			0.2	0.001534	0.955112	10	99
			0.3	0.001302	0.955415	10	97
		10	0.1	0.002194	0.983893	57	99
			0.2	0.001483	0.983890	69	98
			0.3	0.001968	0.983906	142	97
		100	0.1	0.001972	0.984023	980	97
			0.2	0.006431	0.984028	1340	97
			0.3	0.002328	0.984051	2886	97

The results provided from the *custom* implementation, i.e., the SGD with different momentum settings, are strongly similar to those of *sklearn* implementation, i.e., *liblinear* [10] implementation, in terms of *r2* score, except in case of C regularization hyperparameter equals to 1 for which those of SGD are lower. Moreover, the SGD solver always requires lower iterations, i.e., epochs, for higher C regularization values, i.e., for C equals to 10 or 100, to achieve the same *numerical precision.* Again, *Polyak* and *Nesterov* momentums always perform lower iterations as expected from the theoretical analysis of the convergence rate. The results in terms of *support vectors* are strongly similar to each others.

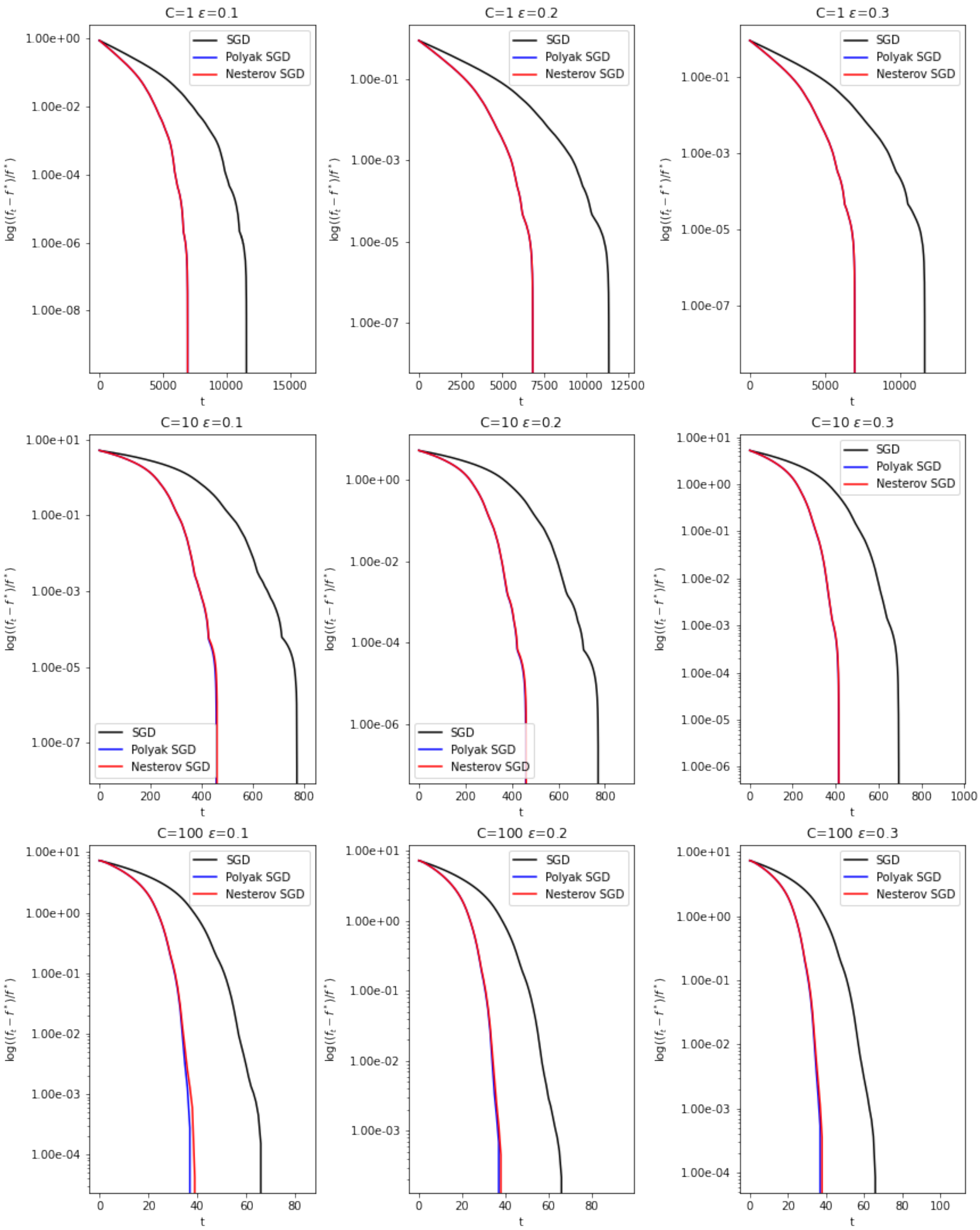


Figure 16: SGD convergence for the Primal formulation of the $\mathcal{L}_1$-SVR

Linear Dual formulations The experiments results shown in 17 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (100) and (101) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 16: Wolfe Dual linear $\mathcal{L}_1$-SVR results

			fit_time	r2	n_iter	n_sv
solver	C	epsilon				
smo	1	0.1	0.049352	0.954396	10	100
		0.2	0.026155	0.954546	15	100
		0.3	0.090050	0.955429	13	99
	10	0.1	0.201986	0.983893	44	99
		0.2	0.091500	0.983893	48	99
		0.3	0.084329	0.983893	41	99
	100	0.1	0.826075	0.984071	623	98
		0.2	0.409304	0.984088	157	98
		0.3	0.521488	0.984103	334	98
libsvm	1	0.1	0.009594	0.954393	79	100
		0.2	0.005095	0.954543	82	100
		0.3	0.029558	0.955424	78	99
	10	0.1	0.033350	0.983892	206	99
		0.2	0.005173	0.983890	219	99
		0.3	0.009250	0.983885	216	99
	100	0.1	0.019970	0.984028	2239	98
		0.2	0.006288	0.984041	1189	98
		0.3	0.003961	0.984051	1366	98
cvxopt	1	0.1	0.122198	0.954685	9	100
		0.2	0.152970	0.954849	9	100
		0.3	0.066905	0.955429	10	100
	10	0.1	0.144911	0.983893	9	100
		0.2	0.056698	0.983893	8	100
		0.3	0.045109	0.983893	8	100
	100	0.1	0.094987	0.984071	9	100
		0.2	0.070957	0.984088	9	100
		0.3	0.095619	0.984103	8	100

For what about the linear *Wolfe dual* formulation we can immediately notice as higher *regularization hyper-parameter C* and lower ϵ values makes the model harder, so the *custom* implementation of the SMO algorithm and also the *sklearn* implementation, i.e., *libsvm* [11] implementation, needs to perform more iterations to achieve the same *numerical precision*; meanwhile, again, the *cvxopt* [12] seems to be insensitive to the increasing complexity of the model. The results in terms of *r2* and number of *support vectors* are strongly similar to each others.

Table 17: Lagrangian Dual linear $\mathcal{L}_1$-SVR results

dual	C	epsilon	fit_time	r2	n_iter	n_sv
reg_bias	1	0.1	19.435027	0.954685	22445	100
		0.2	26.781448	0.954845	22235	100
		0.3	20.560702	0.955429	21493	99
	10	0.1	21.476722	0.983893	24700	99
		0.2	22.552869	0.983893	26586	99
		0.3	21.729551	0.983893	26076	99
	100	0.1	88.206115	0.984071	105273	98
		0.2	119.215992	0.984088	141626	98
		0.3	270.843667	0.984103	284365	98
unreg_bias	1	0.1	28.822030	0.954396	24597	100
		0.2	60.522108	0.954546	62678	100
		0.3	275.789893	0.955429	332397	99
	10	0.1	49.331111	0.983893	55824	99
		0.2	47.654717	0.983893	56708	99
		0.3	52.723625	0.983893	62786	99
	100	0.1	407.311823	0.984071	491963	98
		0.2	450.761096	0.984088	541088	98
		0.3	836.699374	0.984103	674048	98

For what about the linear *Lagrangian dual* formulation we can see as it seems to be insensitive to the increasing complexity of the model in terms of number of *iterations* and require many *iterations* wrt the *Wolfe dual* formulation.

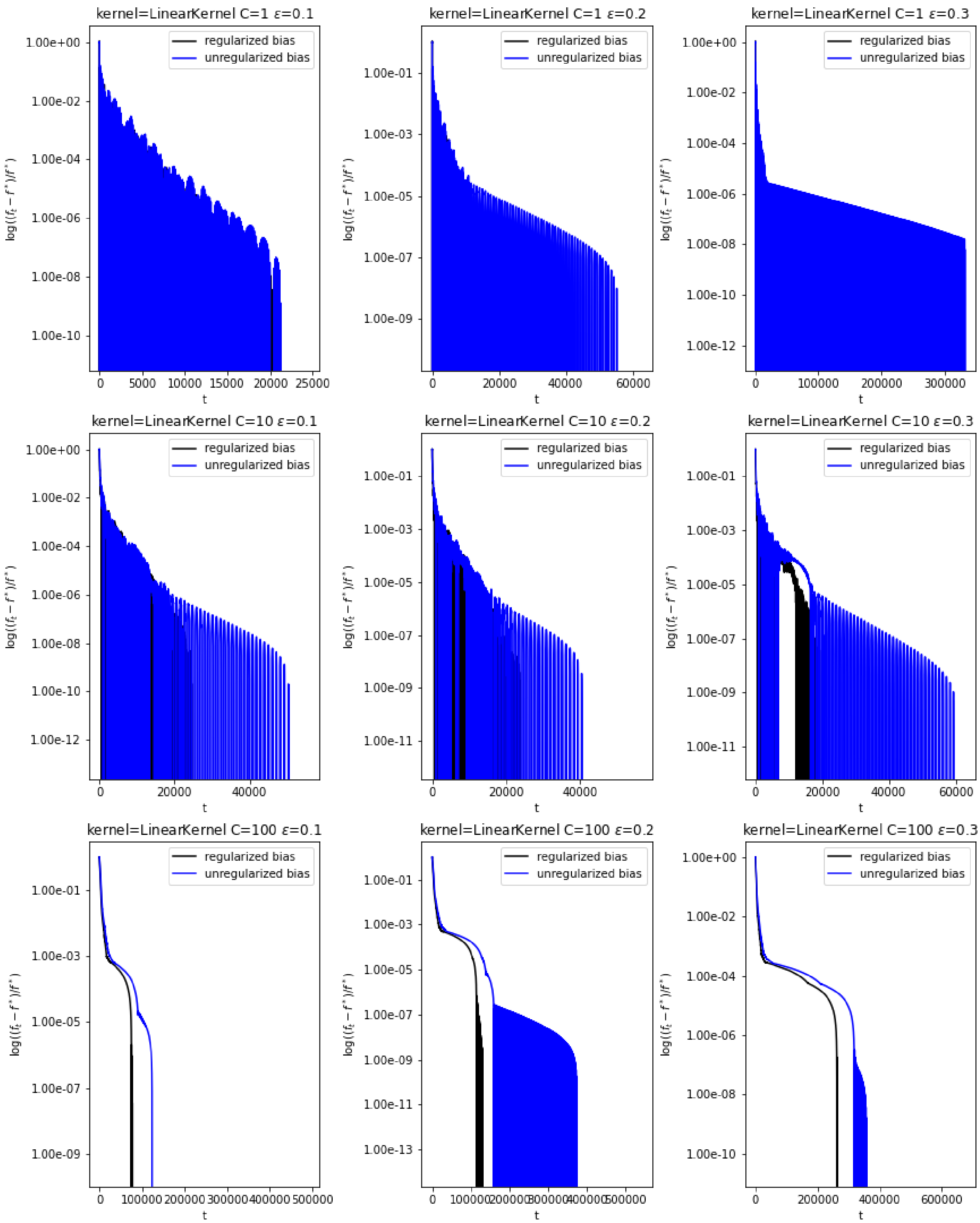


Figure 17: AdaGrad convergence for the Lagrangian Dual formulation of the linear $\mathcal{L}_1$-SVR

Nonlinear Dual formulations The experiments results shown in 18 and 19 are obtained with d and r hyperparameters both equal to 3 for the *polynomial* kernel; *gamma* is setted to *'scale'* for both *gaussian* and *laplacian* kernels. The experiments results shown in 11 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (100) and (101) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e$-$8 between two consecutive iterations.

Table 18: Wolfe Dual nonlinear $\mathcal{L}_1$-SVR formulation results

solver	kernel	C	epsilon	fit_time	r2	n_iter	n_sv
smo	gaussian	1	0.1	0.134567	0.988249	61	17
			0.2	0.098772	0.924439	18	7
			0.3	0.058488	0.882880	17	5
		10	0.1	0.628905	0.989828	289	18
			0.2	0.073629	0.924770	27	6
			0.3	0.034380	0.883067	13	5
		100	0.1	10.473704	0.899765	4835	17
			0.2	0.087404	0.924770	27	6
			0.3	0.023395	0.883067	13	5
	laplacian	1	0.1	0.190778	0.972858	23	23
			0.2	0.103656	0.942216	21	13
			0.3	0.073561	0.866739	17	9
		10	0.1	0.308953	0.989399	19	22
			0.2	0.110892	0.941932	17	13
			0.3	0.087622	0.866472	13	9
		100	0.1	0.237147	0.989399	19	22
			0.2	0.083979	0.941932	17	13
			0.3	0.080471	0.866472	13	9
libsvm	gaussian	1	0.1	0.009212	0.990088	96	17
			0.2	0.007524	0.977763	36	7
			0.3	0.002327	0.945601	24	5
		10	0.1	0.006896	0.990493	616	18
			0.2	0.008819	0.980673	39	6
			0.3	0.002239	0.945601	24	5
		100	0.1	0.010395	0.990496	9854	18
			0.2	0.002149	0.980673	39	6
			0.3	0.001906	0.945601	24	5
	laplacian	1	0.1	0.019776	0.990050	47	23
			0.2	0.006970	0.969067	28	13
			0.3	0.005510	0.924296	22	9
		10	0.1	0.002124	0.990777	47	23
			0.2	0.007089	0.969103	31	13
			0.3	0.002430	0.924237	22	9
		100	0.1	0.005186	0.990777	47	23
			0.2	0.002676	0.969103	31	13
			0.3	0.005486	0.924237	22	9
cvxopt	gaussian	1	0.1	0.106479	0.988117	10	17
			0.2	0.060167	0.924679	10	7
			0.3	0.048079	0.883386	10	5
		10	0.1	0.126220	0.989956	10	18
			0.2	0.085939	0.925595	10	6
			0.3	0.109721	0.883386	10	5
		100	0.1	0.097486	0.990216	10	40
			0.2	0.076151	0.925595	10	6
			0.3	0.095276	0.883386	10	5
	laplacian	1	0.1	0.094410	0.977836	9	24
			0.2	0.096828	0.942110	9	13
			0.3	0.145580	0.866633	9	9
		10	0.1	0.112691	0.984378	10	24
			0.2	0.133206	0.942110	10	13
			0.3	0.098581	0.866633	10	9
		100	0.1	0.071575	0.984378	10	24
			0.2	0.086546	0.955697	10	14
			0.3	0.076675	0.888440	10	10

Table 19: Lagrangian Dual nonlinear $\mathcal{L}_1$-SVR results

dual	kernel	C	epsilon	fit_time	r2	n_iter	n_sv
reg_bias	gaussian	1	0.1	67.644551	0.986553	72482	18
			0.2	78.462785	0.924632	54744	7
			0.3	14.232657	0.883390	16244	5
		10	0.1	2322.455135	0.989959	2344114	18
			0.2	17.755198	0.925601	21899	6
			0.3	13.567137	0.883389	16246	5
		100	0.1	8270.649641	0.980120	10000000	18
			0.2	18.380509	0.925601	21899	6
			0.3	13.919278	0.883389	16246	5
	laplacian	1	0.1	74.508785	0.972770	70484	23
			0.2	45.027107	0.942106	41886	13
			0.3	47.641032	0.866627	36338	9
		10	0.1	108.923422	0.980896	138874	23
			0.2	38.575338	0.942106	41946	13
			0.3	32.220732	0.866627	36479	9
		100	0.1	112.747609	0.980896	138874	23
			0.2	35.052471	0.942106	41946	13
			0.3	30.797087	0.866627	36479	9
unreg_bias	gaussian	1	0.1	143.920365	0.986528	133972	18
			0.2	81.246749	0.924628	75812	7
			0.3	30.468117	0.883584	31649	5
		10	0.1	3153.690065	0.989944	2211354	18
			0.2	31.067626	0.925928	38849	6
			0.3	26.399761	0.883584	31608	5
		100	0.1	8060.317186	0.882186	10000000	18
			0.2	31.917725	0.925928	38849	6
			0.3	24.978174	0.883584	31608	5
	laplacian	1	0.1	121.508376	0.977780	105230	24
			0.2	73.975619	0.942111	65682	13
			0.3	74.175946	0.866660	56971	9
		10	0.1	137.601343	0.980911	174106	23
			0.2	52.520926	0.942111	65627	13
			0.3	46.564164	0.866660	57235	9
		100	0.1	140.732585	0.980911	174106	23
			0.2	55.761176	0.942111	65627	13
			0.3	47.950730	0.866660	57235	9

The same considerations made for the previous linear *Wolfe dual* and *Lagrangian dual* formulations are confirmed also in the nonlinearly separable case. In this setting, the complexity of the model coming with higher C regularization hyperparameters and lower ϵ values pays a larger tradeoff in terms of the number of *iterations* of the algorithm.

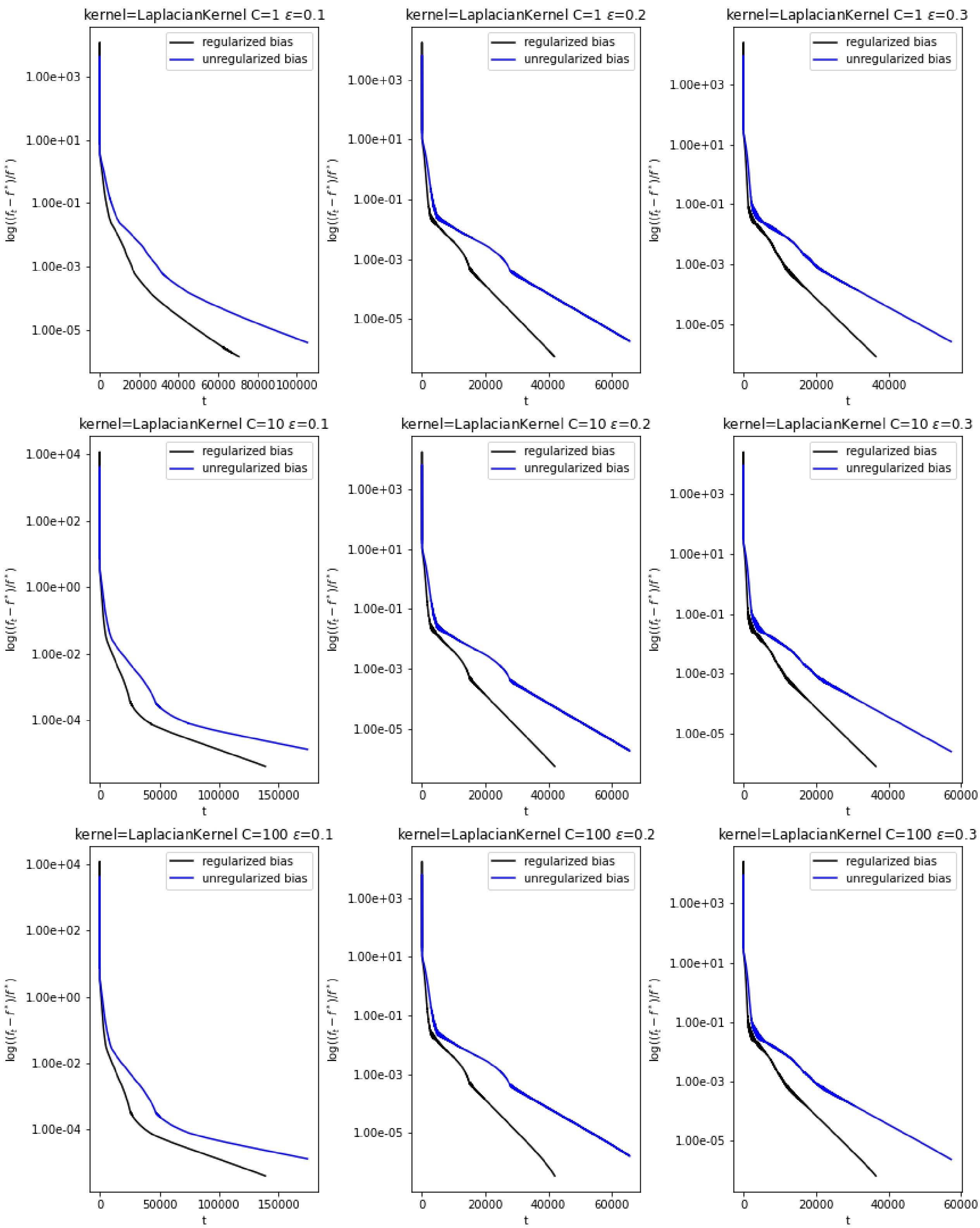


Figure 18: AdaGrad convergence for the Lagrangian Dual formulation of the Laplacian $\mathcal{L}_1$-SVR

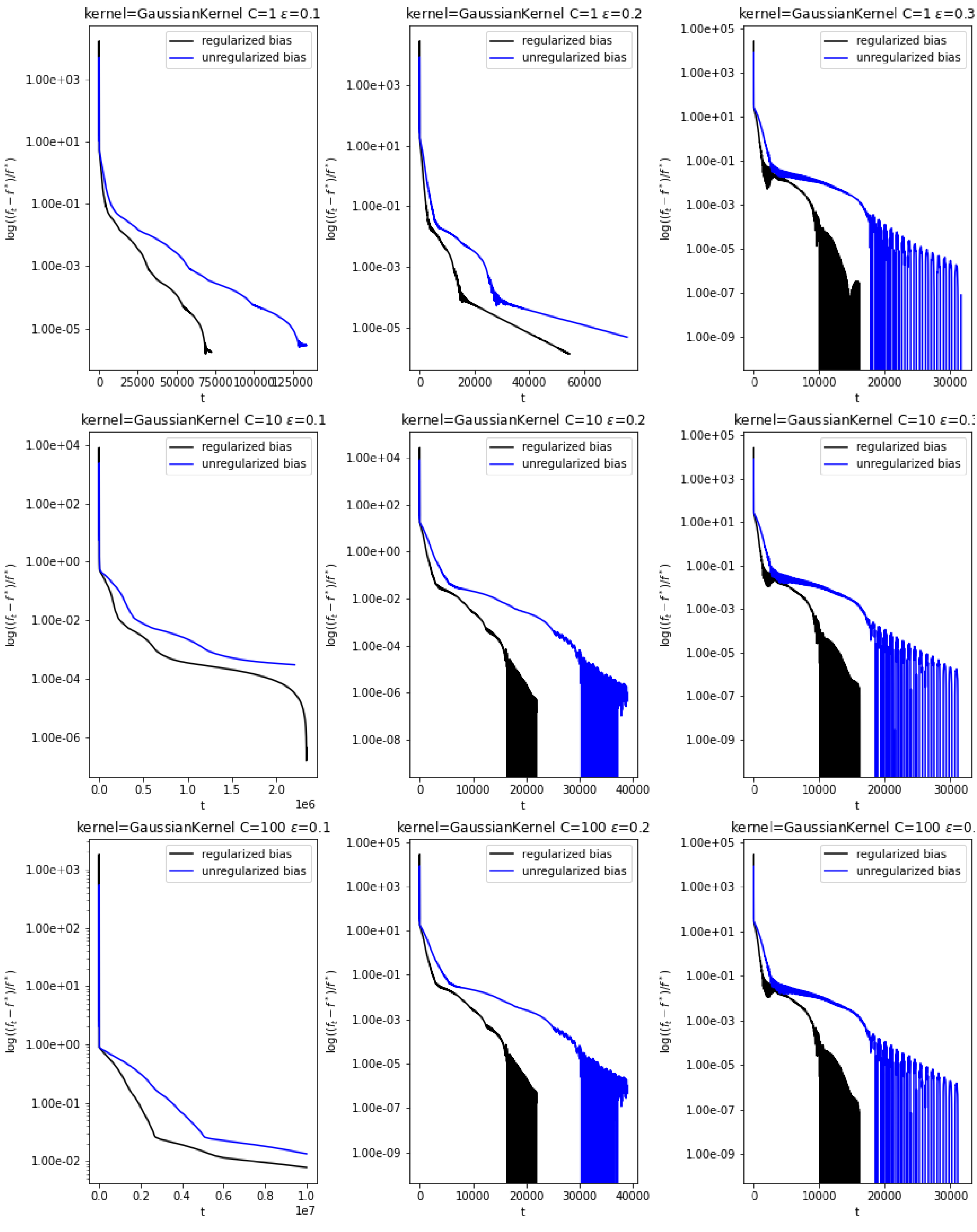


Figure 19: AdaGrad convergence for the Lagrangian Dual formulation of the Gaussian $\mathcal{L}_1$-SVR

7.2.2 Squared Epsilon-insensitive loss

Primal formulation The experiments results shown in 20 referred to *Stochastic Gradient Descent* algorithm are obtained with α, i.e., the *learning rate* or *step size*, setted to 0.02 and β, i.e., the *momentum*, equal to 0.4. The optimization process is stopped if after 5 iterations the function value does not improve by at least 1e−8.

Table 20: Primal $\mathcal{L}_2$-SVR results

solver	momentum	C	epsilon	fit_time	r2	n_iter	n_sv
sgd	none	1	0.1	2.610538	0.984109	3283	100
			0.2	2.536275	0.984109	3294	100
			0.3	2.622454	0.984109	3321	98
		10	0.1	0.373240	0.984133	409	98
			0.2	0.373901	0.984133	410	98
			0.3	0.406998	0.984133	411	98
		100	0.1	0.086858	0.984133	47	98
			0.2	0.064771	0.984133	47	98
			0.3	0.034043	0.984133	47	98
	polyak	1	0.1	1.642638	0.984109	2011	100
			0.2	1.685593	0.984109	2018	100
			0.3	1.856611	0.984109	2035	98
		10	0.1	0.233500	0.984133	241	98
			0.2	0.233849	0.984133	242	98
			0.3	0.260583	0.984133	243	98
		100	0.1	0.062701	0.984133	40	98
			0.2	0.062025	0.984133	40	98
			0.3	0.062099	0.984133	40	98
	nesterov	1	0.1	2.522627	0.984109	2015	100
			0.2	1.593459	0.984109	2022	100
			0.3	1.696347	0.984109	2039	98
		10	0.1	0.224427	0.984133	247	98
			0.2	0.231077	0.984133	248	98
			0.3	0.251711	0.984133	248	98
		100	0.1	0.041334	0.984133	27	98
			0.2	0.036721	0.984133	27	98
			0.3	0.057086	0.984133	27	98
liblinear	-	1	0.1	0.002193	0.984109	84	100
			0.2	0.003775	0.984109	84	100
			0.3	0.006009	0.984109	84	98
		10	0.1	0.004759	0.984133	778	98
			0.2	0.005728	0.984133	773	98
			0.3	0.005014	0.984133	773	98
		100	0.1	0.055547	0.984133	7296	99
			0.2	0.062751	0.984133	7434	98
			0.3	0.038260	0.984133	7262	98

Again, the results provided from the *custom* implementation, i.e., the SGD with different momentum settings, are strongly similar to those of *sklearn* implementation, i.e., *liblinear* [10] implementation, in terms of *r2* score. SGD solver always requires even lower iterations, i.e., epochs, for higher C regularization values, i.e., for C equals to 10 or 100, to achieve the same *numerical precision. Polyak* and *Nesterov* momentums always perform lower iterations as expected from the theoretical analysis of the convergence rate.

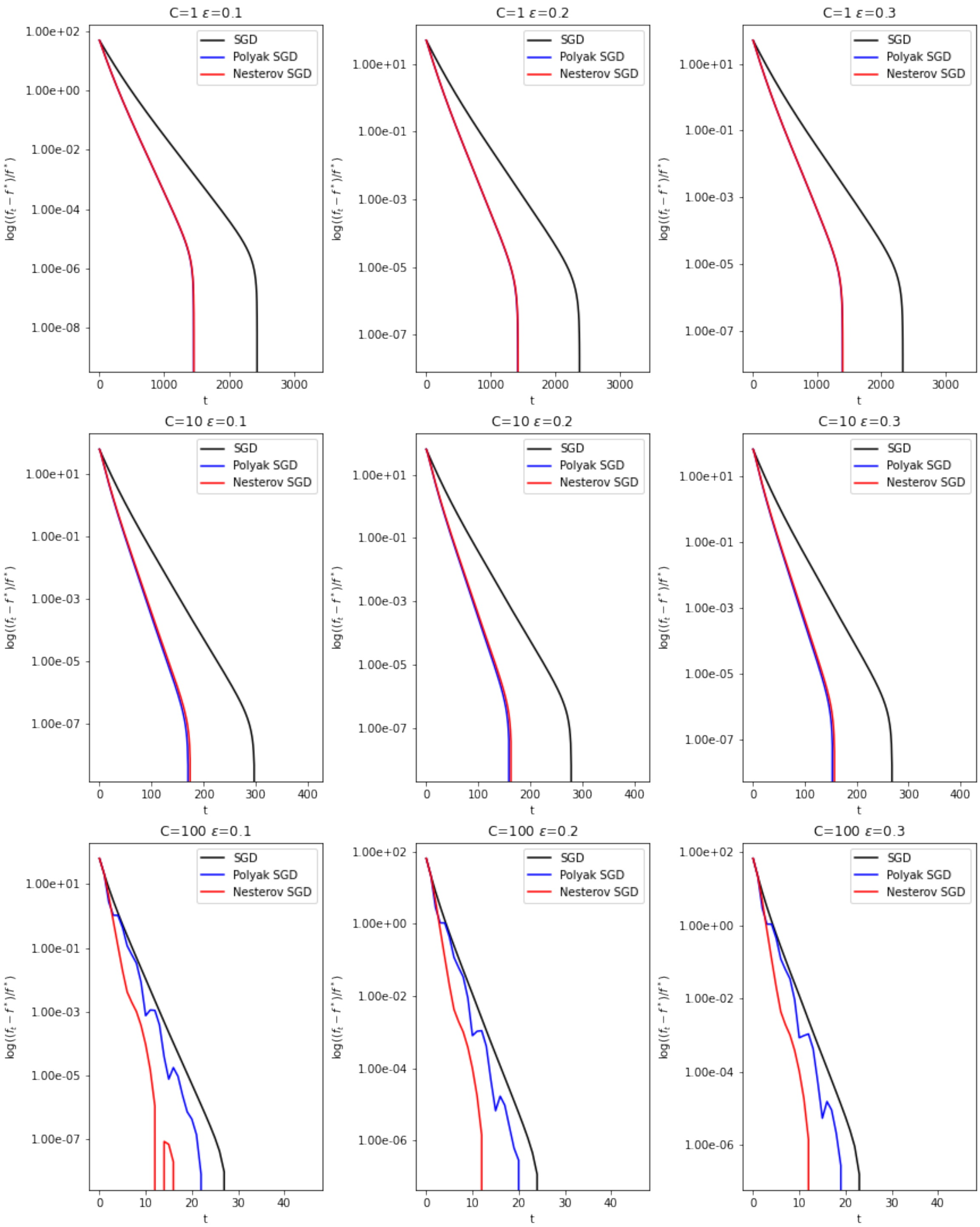


Figure 20: SGD convergence for the Primal formulation of the $\mathcal{L}_2$-SVR

Linear Dual formulations The experiments results shown in 21 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (121) and (122) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 21: Lagrangian Dual linear $\mathcal{L}_2$-SVR results

			fit_time	r2	n_iter	n_sv
dual	C	epsilon				
reg_bias	1	0.1	9.710948	0.984109	9163	100
		0.2	6.555949	0.984109	9161	100
		0.3	5.224398	0.984109	9172	98
	10	0.1	311.821745	0.984133	486760	98
		0.2	330.416619	0.984133	482844	98
		0.3	334.450082	0.984133	478956	98
	100	0.1	7054.383372	0.984133	10000000	99
		0.2	7088.717244	0.984133	10000000	98
		0.3	9802.507830	0.984133	10000000	98
unreg_bias	1	0.1	7.992880	0.984109	10989	100
		0.2	7.131316	0.984109	11052	100
		0.3	6.363277	0.984109	11014	98
	10	0.1	332.598610	0.984133	501047	98
		0.2	344.210789	0.984133	497149	98
		0.3	342.279607	0.984133	493280	98
	100	0.1	7181.725315	0.984133	10000000	99
		0.2	7179.237001	0.984133	10000000	98
		0.3	7533.460378	0.984133	10000000	98

For what about the linear *Lagrangian dual* formulation we can see as it seems to be insensitive to the increasing complexity of the model in terms of number of *iterations* and require many *iterations* wrt the *Wolfe dual* formulation.

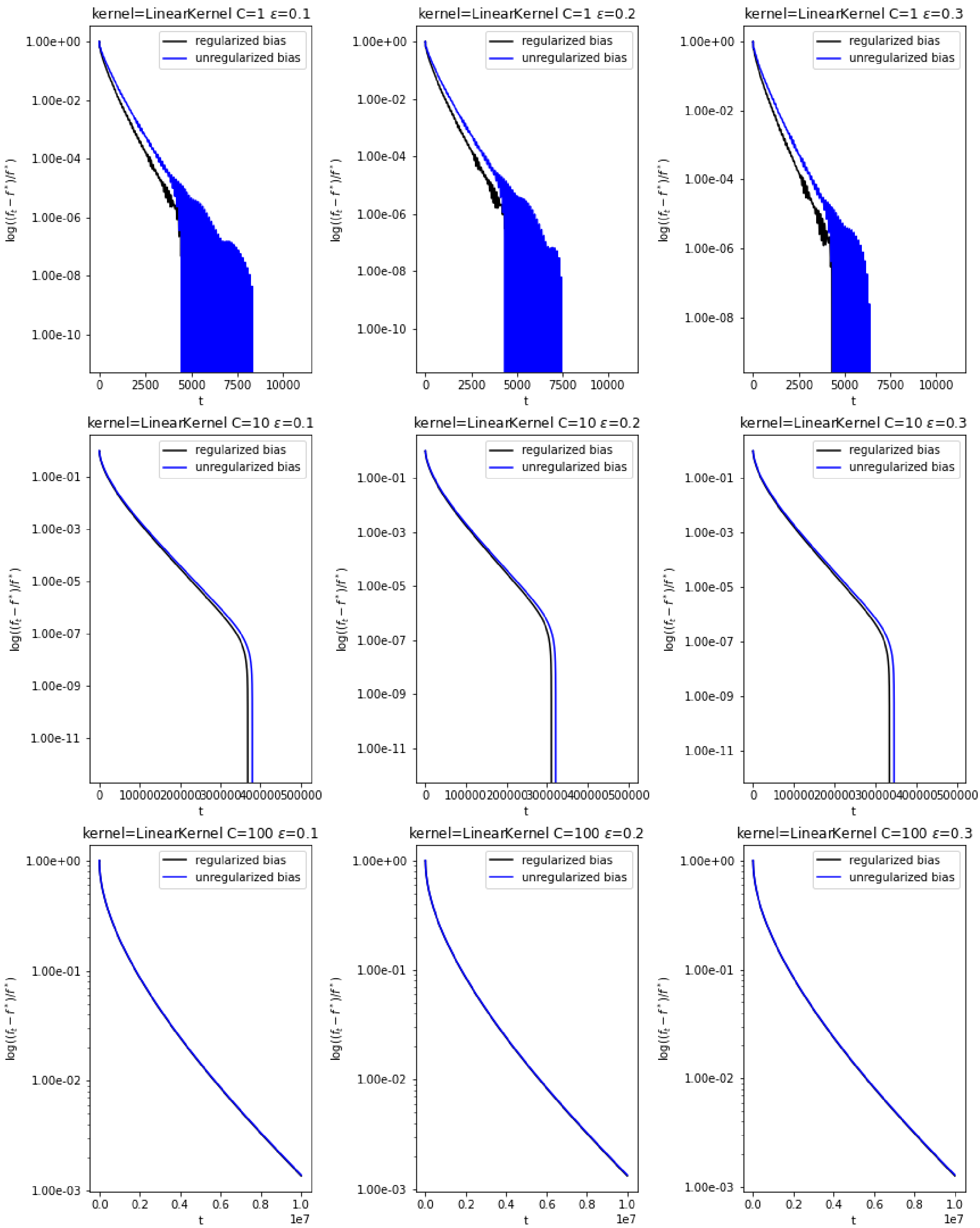


Figure 21: AdaGrad convergence for the Lagrangian Dual formulation of the linear $\mathcal{L}_2$-SVR

Nonlinear Dual formulations The experiments results shown in 22 are obtained with d and r hyperparameters both equal to 3 for the *polynomial* kernel; *gamma* is setted to *'scale'* for both *gaussian* and *laplacian* kernels. The experiments results shown in 22 are obtained with α, i.e., the *learning rate* or *step size*, setted to 1 for the *AdaGrad* algorithm. Note that the *unreg_bias* and *reg_bias* duals refers to the augmented dual formulations (121) and (122) respectively with ρ equals to 1. The optimization process is stopped if the primal-dual weight vector does not change by at least 1e−8 between two consecutive iterations.

Table 22: Lagrangian Dual nonlinear $\mathcal{L}_2$-SVR results

dual	kernel	C	epsilon	fit_time	r2	n_iter	n_sv
reg_bias	gaussian	1	0.1	4.736382	0.971405	5127	35
			0.2	4.254038	0.932771	5292	28
			0.3	5.544147	0.897683	5668	16
		10	0.1	18.900264	0.980109	22838	18
			0.2	17.255836	0.915557	20244	9
			0.3	17.989189	0.896923	20371	8
		100	0.1	184.046951	0.985273	222103	20
			0.2	23.545632	0.924209	27625	6
			0.3	13.160150	0.881664	14839	5
	laplacian	1	0.1	4.561488	0.968637	5723	51
			0.2	4.625473	0.934767	6050	41
			0.3	5.090187	0.888289	6500	33
		10	0.1	22.081973	0.983604	24246	24
			0.2	14.658179	0.934553	17711	18
			0.3	14.524363	0.910527	16626	13
		100	0.1	55.568179	0.980212	66718	23
			0.2	23.727008	0.941098	27112	13
			0.3	22.440146	0.865076	25357	9
unreg_bias	gaussian	1	0.1	5.560852	0.971405	6861	35
			0.2	6.412180	0.932774	7041	28
			0.3	6.538553	0.897714	8189	16
		10	0.1	22.557080	0.980097	28208	18
			0.2	21.429779	0.915599	26415	9
			0.3	25.689793	0.897110	28352	8
		100	0.1	236.043319	0.985307	282823	20
			0.2	40.174325	0.924497	45881	6
			0.3	18.592879	0.881849	21247	5
	laplacian	1	0.1	7.534001	0.968636	8369	51
			0.2	6.578958	0.934768	8228	41
			0.3	8.370220	0.888284	9795	33
		10	0.1	29.307292	0.983603	35413	24
			0.2	18.693548	0.934550	22472	18
			0.3	20.471616	0.910527	24848	13
		100	0.1	94.831595	0.980227	115240	23
			0.2	30.138026	0.941106	34499	13
			0.3	32.429797	0.865113	36854	9

The same considerations made for the previous linear *Lagrangian dual* formulations are confirmed also in the nonlinearly separable case. In this setting, the complexity of the model coming with higher C regularization hyperparameters and lower ϵ values pays a larger tradeoff in terms of the number of *iterations* of the algorithm.

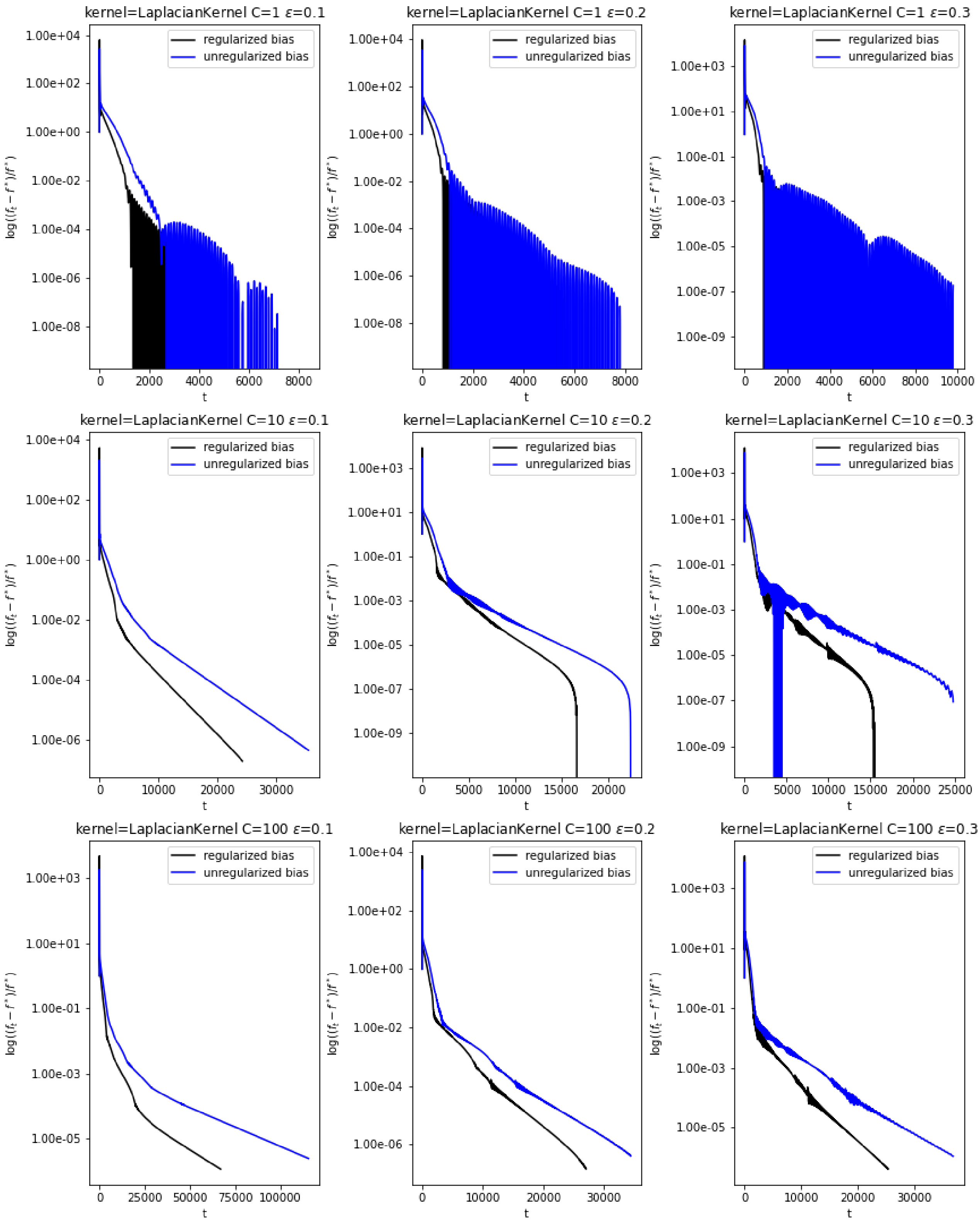


Figure 22: AdaGrad convergence for the Lagrangian Dual formulation of the Laplacian $\mathcal{L}_2$-SVR

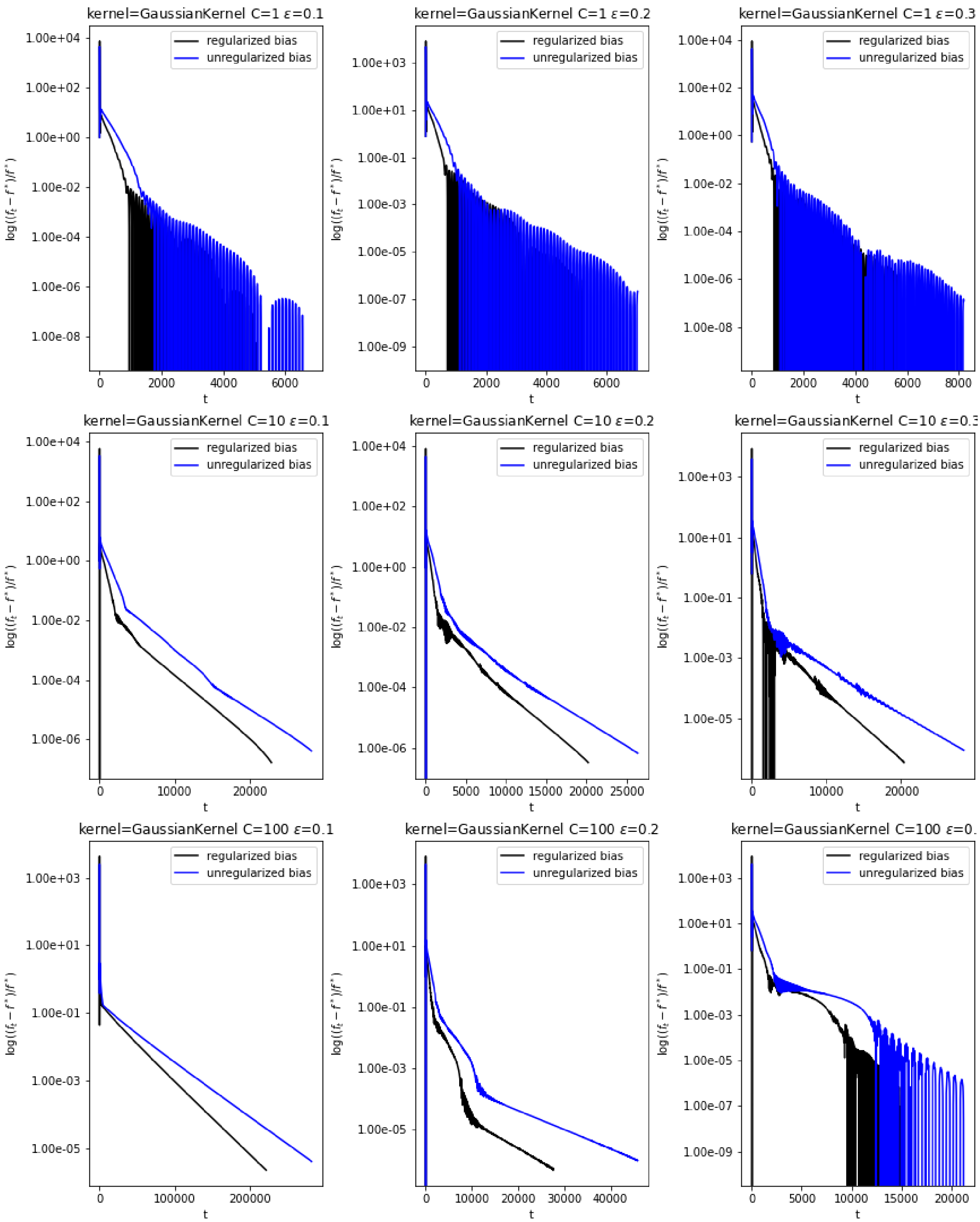


Figure 23: AdaGrad convergence for the Lagrangian Dual formulation of the Gaussian $\mathcal{L}_2$-SVR

8 Conclusions

For what about the SVM formulations, it is known, in general, that the *primal formulation*, is suitable for large linear training since the complexity of the model grows with the number of features or, more in general, when the number of examples n is much larger than the number of features m, i.e., $n \gg m$; meanwhile the *dual formulation*, is more suitable in case the number of examples n is less than the number of features m, i.e., $n < m$, since the complexity of the model is dominated by the number of examples, or more in general when the training data are not linearly separable in the input space.

From all these experiments we can see as all the *custom* implementations underperforms all the others, i.e., both *cvxopt* [12] and *sklearn* implementations, i.e., *liblinear* [10] and *libsvm* [11] implementations, in terms of *time* obviously due to the different core implementation languages, i.e., Python and C respectively.

In the *primal* formulations the *liblinear* [10] implementation uses an optimization method called *Coordinate Gradient Descent* which minimizes one coordinate at a time.

Meanwhile, for what about the *Wolfe dual* formulations we can notice as *cvxopt* [12] underperforms the *sklearn* implementation, i.e., *libsvm* [11] implementation, in terms of *time* since it is a general-purpose QP solver and it does not exploit the structure of the problem, as SMO does. An interesting consideration can be made about the number of *iterations* of *custom* SMO implementation wrt that in *libsvm* which seems to be always lower thanks to the improvements described in [5, 8] for classification and regression respectively.

Finally, in the *Lagrangian dual* formulations, we can see as fitting the intercept in an explicit way, i.e., by adding Lagrange multipliers to control the equality constraint always get lower scores wrt the *Lagrangian dual* of the same problem with the bias term embedded into the weight matrix.

References

[1] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[2] Yurii Nesterov. Introductory lectures on convex programming volume i: Basic course. *Lecture notes*, 3(4):5, 1998.

[3] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

[4] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[5] S. Sathiya Keerthi, Shirish Krishnaj Shevade, Chiranjib Bhattacharyya, and Karuturi Radha Krishna Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural computation*, 13(3):637–649, 2001.

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[7] Gary William Flake and Steve Lawrence. Efficient SVM regression training with SMO. *Machine Learning*, 46(1):271–290, 2002.

[8] SK Shevade, SS Keerthi, C Bhattacharyya, and KRK Murthy. Improvements to SMO algorithm for SVM regression (Tech. Rep. No. CD-99-16). *Singapore: Control Division Department of Mechanical and Production Engineering*, 1999.

[9] Tristan Fletcher. Support vector machines explained. *Tutorial paper., Mar*, page 28, 2009.

[10] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *the Journal of machine Learning research*, 9:1871–1874, 2008.

[11] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[12] Lieven Vandenberghe. The CVXOPT linear and quadratic cone program solvers. *Online: http://cvxopt. org/documentation/coneprog. pdf*, 2010.

[13] Veronica Piccialli and Marco Sciandrone. Nonlinear optimization and support vector machines. *4OR*, 16(2):111–149, 2018.

[14] Chih-Wei Hsu and Chih-Jen Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46(1):291–314, 2002.

[15] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[16] Xingyu Zhou. On the fenchel duality between strong convexity and lipschitz continuous gradient. *arXiv preprint arXiv:1803.06573*, 2018.