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

March, 2021

Contents

List of Figures

List of Tables

1 Track

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

- `(A1.1.1)` is the *AdaGrad* algorithm [1], a *deflected subgradient* method for solving the SVC in its *primal* formulation.
- `(A1.1.2)` is the *Sequential Minimal Optimization (SMO)* algorithm [2] (see [3] for improvements), 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 [1], 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, an *accelerated gradient* method for solving the SVC in its *primal* formulation.

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

- `(A2.1.1)` is the *AdaGrad* algorithm [1], a *deflected subgradient* method for solving the SVR in its *primal* formulation.
- `(A2.1.2)` is the *Sequential Minimal Optimization (SMO)* algorithm [4] (see [5] for improvements), an ad hoc *active set* method for training a SVR in its *Wolfe dual* formulation with *linear*, *polynomial* and *gaussian* kernels.
- `(A2.1.3)` is the *AdaGrad* algorithm [1], a *deflected subgradient* method for solving the SVR in its *Lagrangian dual* formulation with *linear*, *polynomial* and *gaussian* kernels.

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

- `(A2.2.1)` is a *momentum descent* approach, an *accelerated gradient* method for solving the SVR in its *primal* formulation.

2 Abstract

A *Support Vector Machine* is a learning model used both for *classification* and *regression* tasks whose goal is to constructs 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* 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 performace of different *kernels*, also by comparing the *custom* results with *sklearn* SVM implementations, i.e, *liblinear* and *libsvm* implementations, and *cvxopt* 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 missclassified 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 $\dfrac{1}{2}\|w\|^2$, but in this form we will deal with a convex optimization problem 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.

3.1 Hinge loss

The *hinge* loss is defined as:

$$\mathcal{L}_1 = \begin{cases} 0 & \text{if } y(w^T x + b) \geq 1 \\ 1 - y(w^T x + b) & \text{otherwise} \end{cases} \tag{6}$$

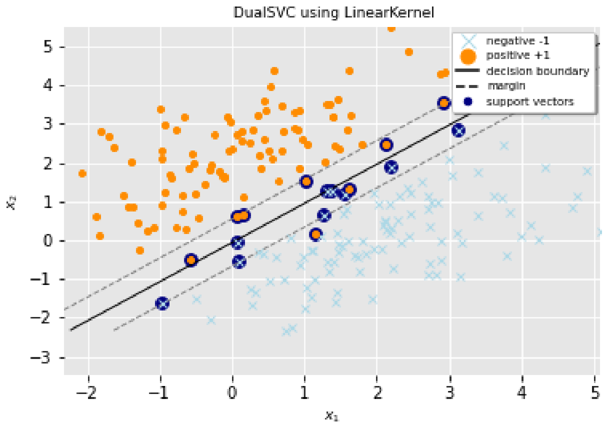


Figure 1: Linear SVC hyperplane

or, equivalently:

$$\mathcal{L}_1 = \max(0, 1 - y(w^T x + b)) \tag{7}$$

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

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

3.1.1 Primal formulation

The general primal unconstrained formulation takes the form:

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

where $\mathcal{R}(w,b)$ 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).

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.

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 10 and 41 as follows:

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

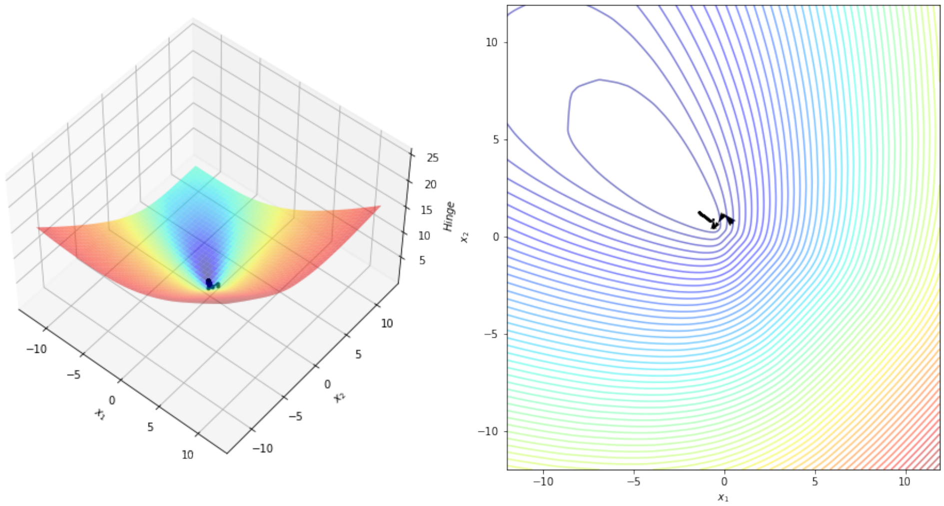


Figure 2: SVC Hinge loss with optimization steps

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:

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

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.

Notice that in terms of numerical optimization the formulations 10 and 41 are not equivalent to 11 or 12 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 [6] 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 $\alpha_i \geq 0, \mu_i \geq 0 \; \forall_i$:

$$\max_{\alpha,\mu} \min_{w,b,\xi} \mathcal{W}(w,b,\xi,\alpha,\mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^{n}\mu_i\xi_i \tag{13}$$

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:

$$\frac{\partial \mathcal{W}}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i \Rightarrow w = \sum_{i=1}^{n}\alpha_i y_i x_i \tag{14}$$

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{15}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \mu_i \tag{16}$$

Substituting 14 and 15 into 13 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:

$$\begin{aligned} \max_{\alpha} \mathcal{W}(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i Q_{ij} \alpha_j \text{ where } Q_{ij} = y_i y_j \langle x_i, x_j \rangle \\ &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \alpha^T Q \alpha \text{ subject to } 0 \leq \alpha_i \leq C \; \forall_i, \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{17}$$

or, equivalently:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha + q^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \; \forall_i \\ & y^T \alpha = 0 \end{aligned} \tag{18}$$

where $q^T = [1, \ldots, 1]$.
By solving 18 we will know α and, from 14, we will get w, so we need to calculate b.
We know that any data point satisfying 15 which is a support vector x_s will have the form:

$$y_s(w^T x_s + b) = 1 \tag{19}$$

and, by substituting in 14, we get:

$$y_s \big(\sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle + b \big) = 1 \tag{20}$$

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:

$$y_s^2 \big(\sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle + b \big) = y_s \tag{21}$$

$$b = y_s - \sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle \tag{22}$$

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

$$b = \frac{1}{N_s} \sum_{s \in S} y_s - \sum_{m \in S} \alpha_m y_m \langle x_m, x_s \rangle \tag{23}$$

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:

$$y' = \operatorname{sgn} \big(\sum_{i=1}^{n} \alpha_i y_i \langle x_i, x' \rangle + b \big) \tag{24}$$

From 18 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 [6] that arises from the primal 11 or 12 where the bias term b is embedded into the weight vector w:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + yy^T)\alpha + q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \end{aligned} \tag{25}$$

3.1.3 Lagrangian Dual formulation

In order to relax the constraints in the *Wolfe dual* formulation 18 we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrangian multipliers $\mu \geq 0, \lambda_+ \geq 0, \lambda_- \geq 0$:

$$\begin{aligned} \max_{\mu,\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T Q\alpha + q^T\alpha - \mu^T(y^T\alpha) - \lambda_+^T(u - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T Q\alpha + (q - \mu y + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u \end{aligned} \tag{26}$$

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

$$\frac{\partial\mathcal{L}}{\partial\alpha} = 0 \Rightarrow Q\alpha + (q - \mu y + \lambda_+ - \lambda_-) = 0 \tag{27}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu y + \lambda_+ - \lambda_-) \tag{28}$$

the gradient wrt μ, λ_+ and λ_- are:

$$\frac{\partial\mathcal{L}}{\partial\mu} = -y\alpha \tag{29}$$

$$\frac{\partial\mathcal{L}}{\partial\lambda_+} = \alpha - u \tag{30}$$

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

If the Hessian matrix Q is indefinite, i.e., the Lagrangian function is not strictly convex since it will be linear along the eigenvectors correspondent to the null eigenvalues, the Lagrangian dual relaxation will be nondifferentiable, so it will have infinite solutions and for each of them it will have a different subgradient. In order to compute the gradient, we will choose α in such a way as the one that minimizes the residue, i.e. the least-squares solution:

$$\begin{aligned} \min_{\alpha \in K_n(Q,b)} \quad & \|Q\alpha - b\| \\ \text{where} \quad & b = -(q - \mu y + \lambda_+ - \lambda_-) \end{aligned} \tag{32}$$

Since we are dealing with a symmetric but indefinite linear 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*, which computes the vector α that minimizes $\|Q\alpha - b\|$ among all vectors in $K_n(Q, b) = span(b, Qb, Q^2b, \dots, Q^{n-1}b)$.

From 18 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 26 is reduced of 1/3 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 25.

$$\begin{aligned}\max_{\lambda_+,\lambda_-}\min_{\alpha}\mathcal{L}(\alpha,\lambda_+,\lambda_-) &= \frac{1}{2}\alpha^T(Q+yy^T)\alpha + q^T\alpha - \lambda_+^T(u-\alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+yy^T)\alpha + (q+\lambda_+-\lambda_-)^T\alpha - \lambda_+^T u\end{aligned} \tag{33}$$

where, again, the upper bound $u^T = [C,\dots,C]$.
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 gradient wrt λ_+ and λ_- are:

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

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

3.2 Squared Hinge loss

The *squared hinge* loss is defined as:

$$\mathcal{L}_2 = \begin{cases} 0 & \text{if } y(w^Tx+b) \geq 1 \\ (1-y(w^Tx+b))^2 & \text{otherwise} \end{cases} \tag{38}$$

or, equivalently:

$$\mathcal{L}_2 = \max(0, 1-y(w^Tx+b))^2 \tag{39}$$

It is a convex function and its gradient wrt w is given by:

$$\frac{\partial\mathcal{L}_2}{\partial w} = \begin{cases} -2yx & \text{if } y(w^Tx+b) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{40}$$

3.2.1 Primal formulation

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

$$\min_{w,b}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\max(0, 1-y_i(w^Tx_i+b))^2 \tag{41}$$

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

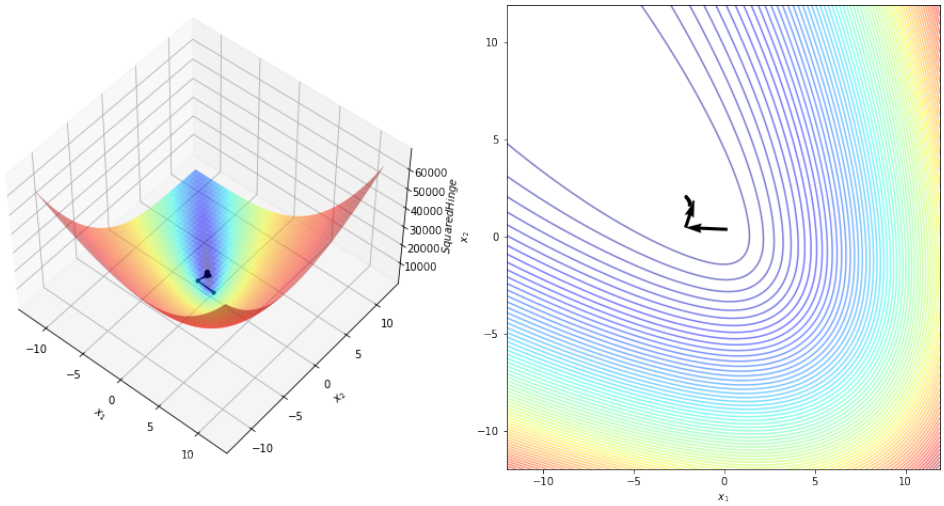


Figure 3: SVC Squared Hinge loss with optimization steps

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:

$$\{(x_i, y_i), x \in \Re^m, y_i \in \Re, i = 1, \ldots, n\} \tag{42}$$

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 $\xi^- \geq 0 \ \forall_i$:

$$\begin{aligned} &y_i \leq y'_i + \epsilon + \xi^+ \ \forall_i \\ &y_i \geq y'_i - \epsilon - \xi^- \ \forall_i \\ &\xi_i^+, \xi_i^- \geq 0 \ \forall_i \end{aligned} \tag{43}$$

The objective function for SVR can then be written as:

$$\begin{aligned} \min_{w,b,\xi^+,\xi^-} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) \\ \text{subject to} \quad & y_i - w^T x_i - b \leq \epsilon + \xi_i^+ \ \forall_i \\ & w^T x_i + b - y_i \leq \epsilon + \xi_i^- \ \forall_i \\ & \xi_i^+, \xi_i^- \geq 0 \ \forall_i \end{aligned} \tag{44}$$

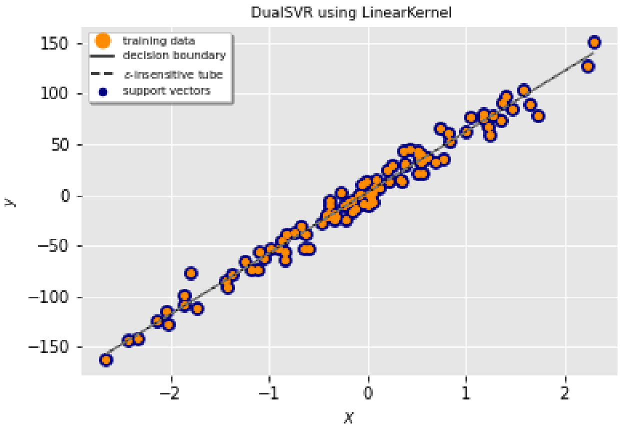


Figure 4: Linear SVR hyperplane

4.1 Epsilon-insensitive loss

The *epsilon-insensitive* loss is defined as:

$$\mathcal{L}_\epsilon = \begin{cases} 0 & \text{if } |y - (w^T x + b)| \leq \epsilon \\ |y - (w^T x + b)| - \epsilon & \text{otherwise} \end{cases} \tag{45}$$

or, equivalently:

$$\mathcal{L}_\epsilon = \max(0, |y - (w^T x + b)| - \epsilon) \tag{46}$$

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 wrt w that is given by:

$$\frac{\partial \mathcal{L}_\epsilon}{\partial w} = \begin{cases} (y - (w^T x + b))x & \text{if } |y - (w^T x + b)| > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

4.1.1 Primal formulation

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

The quadratic optimization problem 44 can be equivalently formulated as:

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

where we make use of the *epsilon-insensitive* loss 45 or 46.

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

4.1.2 Wolfe Dual formulation

To reformulate the 44 as a *Wolfe dual*, we introduce the Lagrange multipliers $\alpha_i^+ \geq 0, \alpha_i^- \geq 0, \mu_i^+ \geq 0, \mu_i^- \geq 0 \; \forall_i$:

$$\begin{aligned} \max_{\alpha^+,\alpha^-,\mu^+,\mu^-} \min_{w,b,\xi^+,\xi^-} \mathcal{W}(w, b, \xi^+, \xi^-, \alpha^+, \alpha^-, \mu^+, \mu^-) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^n (\xi_i^+ + \xi_i^-) - \sum_{i=1}^n (\mu_i^+ \xi_i^+ + \mu_i^- \xi_i^-) \\ - \sum_{i=1}^n \alpha_i^+ (\epsilon + \xi_i^+ + y_i' - y_i) - \sum_{i=1}^n \alpha_i^- (\epsilon + \xi_i^- - y_i' + y_i) \end{aligned} \tag{49}$$

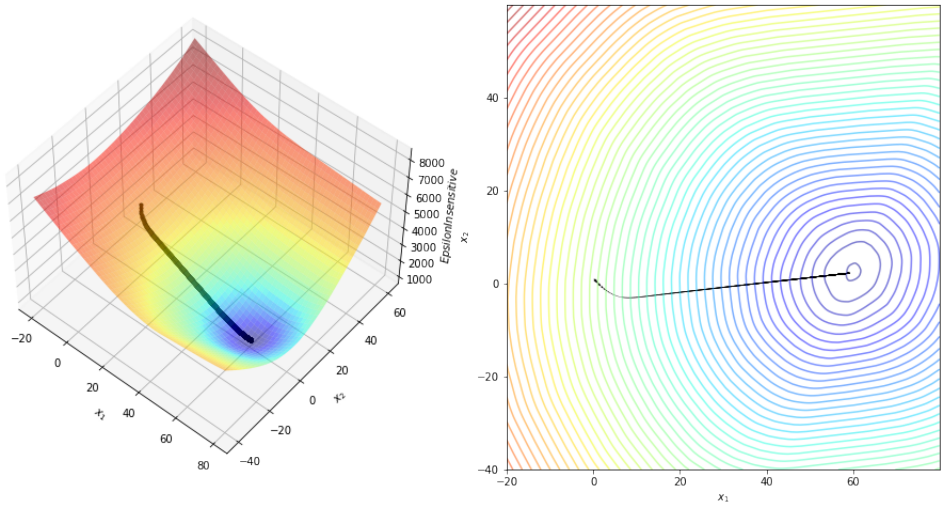


Figure 5: SVR Epsilon-insensitive loss with optimization steps

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

$$\frac{\partial \mathcal{W}}{\partial w} = w - \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)x_i \Rightarrow w = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)x_i \tag{50}$$

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-) \Rightarrow \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-) = 0 \tag{51}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i^+} = 0 \Rightarrow C = \alpha_i^+ + \mu_i^+ \tag{52}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i^-} = 0 \Rightarrow C = \alpha_i^- + \mu_i^- \tag{53}$$

Substituting 50 and 51 in, we now need to maximize $\mathcal{W}$ wrt α_i^+ and α_i^-, where $\alpha_i^+ \geq 0,\ \alpha_i^- \geq 0\ \forall_i$:

$$\max_{\alpha^+,\alpha^-} \mathcal{W}(\alpha^+, \alpha^-) = \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) - \epsilon \sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) - \frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)\langle x_i, x_j\rangle(\alpha_j^+ - \alpha_j^-) \tag{54}$$

Using $\mu_i^+ \geq 0$ and $\mu_i^- \geq 0$ together with 50 and 51 means that $\alpha_i^+ \leq C$ and $\alpha_i^- \leq C$. We therefore need to find:

$$\begin{aligned}
\min_{\alpha^+,\alpha^-} \quad & \frac{1}{2}(\alpha^+ - \alpha^-)^T K(\alpha^+ - \alpha^-) + \epsilon q^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-) \\
\text{subject to} \quad & 0 \leq \alpha_i^+, \alpha_i^- \leq C\ \forall_i \\
& q^T(\alpha^+ - \alpha^-) = 0
\end{aligned} \tag{55}$$

where $q^T = [1, \ldots, 1]$.

We can write the 55 in a standard quadratic form as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha - q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \\ & e^T\alpha = 0 \end{aligned} \tag{56}$$

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

$$y' = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)\langle x_i, x'\rangle + b \tag{57}$$

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:

$$b = y_s - \epsilon - \sum_{m \in S}(\alpha_m^+ - \alpha_m^-)\langle x_m, x_s\rangle \tag{58}$$

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

$$b = \frac{1}{N_s}\sum_{s \in S} y_s - \epsilon - \sum_{m \in S}(\alpha_m^+ - \alpha_m^-)\langle x_m, x_s\rangle \tag{59}$$

From 56 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 [6] that arises from the primal 11 or 12 where the bias term b is embedded into the weight vector w:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + ee^T)\alpha + q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \end{aligned} \tag{60}$$

4.1.3 Lagrangian Dual formulation

In order to relax the constraints in the *Wolfe dual* formulation 55 we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrangian multipliers $\mu \geq 0, \lambda_+ \geq 0, \lambda_- \geq 0$:

$$\begin{aligned} \max_{\mu,\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T Q\alpha + q^T\alpha - \mu^T(e^T\alpha) - \lambda_+^T(u - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T Q\alpha + (q - \mu e + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u \end{aligned} \tag{61}$$

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

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow Q\alpha + (q - \mu e + \lambda_+ - \lambda_-) = 0 \tag{62}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu e + \lambda_+ - \lambda_-) \tag{63}$$

the gradient wrt μ, λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -e\alpha \tag{64}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{65}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{66}$$

If the Hessian matrix Q is indefinite, i.e., the Lagrangian function is not strictly convex since it will be linear along the eigenvectors correspondent to the null eigenvalues, the Lagrangian dual relaxation will be nondifferentiable, so it will have infinite solutions and for each of them it will have a different subgradient. In order to compute the gradient, we will choose α in such a way as the one that minimizes the residue, i.e. the least-squares solution:

$$\begin{aligned} \min_{\alpha \in K_n(Q,b)} \quad & \|Q\alpha - b\| \\ \text{where} \quad & b = -(q - \mu e + \lambda_+ - \lambda_-) \end{aligned} \tag{67}$$

Since we are dealing with a symmetric but indefinite linear 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*, which computes the vector α that minimizes $\|Q\alpha - b\|$ among all vectors in $K_n(Q,b) = span(b, Qb, Q^2b, \dots, Q^{n-1}b)$.

From 56 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 61 is reduced of 1/3 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 60.

$$\begin{aligned} \max_{\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + q^T\alpha - \lambda_+^T(u - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u \end{aligned} \tag{68}$$

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

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-) = 0 \tag{69}$$

With α optimal solution of the linear system:

$$(Q + ee^T)\alpha = -(q + \lambda_+ - \lambda_-) \tag{70}$$

the gradient wrt λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{71}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{72}$$

4.2 Squared Epsilon-insensitive loss

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

$$\mathcal{L}_\epsilon^2 = \begin{cases} 0 & \text{if } |y - (w^T x + b)| \le \epsilon \\ (|y - (w^T x + b)| - \epsilon)^2 & \text{otherwise} \end{cases} \tag{73}$$

or, equivalently:

$$\mathcal{L}_\epsilon^2 = \max(0, |y - (w^T x + b)| - \epsilon)^2 \tag{74}$$

As the *squared hinge* loss, also the *squared epsilon-insensitive* loss is a convex function and it has a gradient wrt w that is given by:

$$\frac{\partial \mathcal{L}_\epsilon^2}{\partial w} = \begin{cases} 2((y - (w^T x + b))x) & \text{if } |y - (w^T x + b)| > \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{75}$$

4.2.1 Primal formulation

To provide a continuously differentiable function the optimization problem 48 can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \max(0, |y_i - (w^T x_i + b)| - \epsilon)^2 \tag{76}$$

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

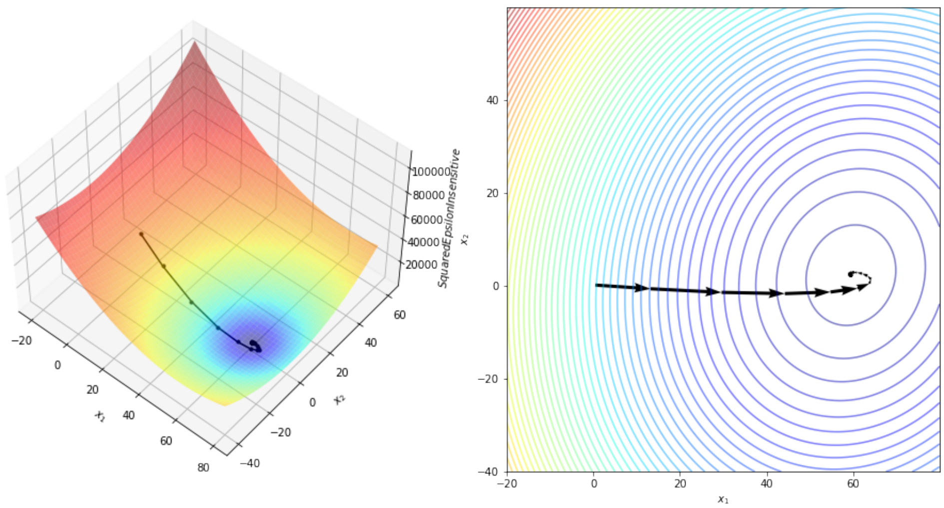


Figure 6: SVC Squared Epsilon-insensitive loss with optimization steps

5 Nonlinear Support Vector Machines

When applying our SVC to *linearly separable* data in 17, we have started by creating a matrix Q from the dot product of our input variables:

$$Q_{ij} = y_i y_j k(x_i, x_j) \tag{77}$$

or, a matrix K from the dot product of our input variables in the SVR case 55:

$$K_{ij} = k(x_i, x_j) \tag{78}$$

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

$$k(x_i, x_j) = \langle x_i, x_j \rangle = x_i^T x_j \tag{79}$$

is known as *linear* kernel.

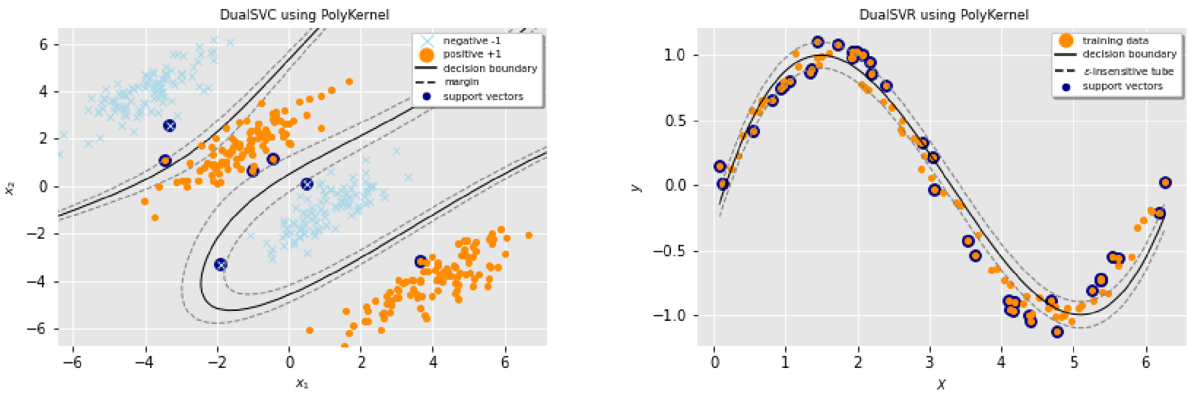
The reason that this *kernel trick* is useful is that there are many classification/regression problems that are nonlinearly separable/regressable in the *input space*, which might be in a higher dimensionality *feature space* given a suitable mapping $x \to \phi(x)$.

5.1 Polynomial kernel

The *polynomial* kernel is defined as:

$$k(x_i, x_j) = (\gamma \langle x_i, x_j \rangle + r)^d \tag{80}$$

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 RBF kernel

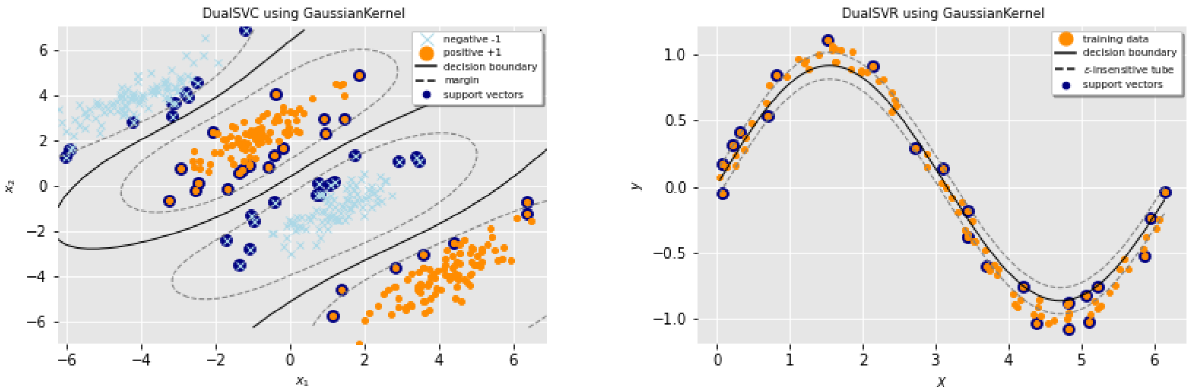
The *gaussian* kernel is defined as:

$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2}) \tag{81}$$

or, equivalently:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{82}$$

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

6 Gradient Descent

Algorithm 1 Gradient Descent

Require: Learning rate or step size $\eta > 0$

procedure GRADIENT DESCENT
    Initialize weight vector $\mathbf{w}_0$
    $k \leftarrow 0$
    while *not_convergence* **do**
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla \mathcal{L}(\mathbf{w}_k)$
        $k \leftarrow k + 1$
    end while
end procedure

6.1 Momentum

6.1.1 Standard

Algorithm 2 Standard Momentum Accelerated Gradient Descent

Require: Learning rate or step size $\eta > 0$
Require: Momentum $\beta \in [0, 1]$

procedure ACCELERATED GRADIENT DESCENT
    Initialize weight vector $\mathbf{w}_1 = \mathbf{w}_0$
    $k \leftarrow 1$
    while *not_convergence* **do**
        $\mathbf{v}_k \leftarrow \beta(\mathbf{w}_k - \mathbf{w}_{k-1}) - \eta \nabla \mathcal{L}(\mathbf{w}_k)$
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_k$
        $k \leftarrow k + 1$
    end while
end procedure

6.1.2 Nesterov

Algorithm 3 Nesterov Momentum Accelerated Gradient Descent

Require: Learning rate $\eta > 0$
Require: Momentum $\beta \in [0, 1]$

procedure NESTEROV ACCELERATED GRADIENT DESCENT
    Initialize weight vector $\mathbf{w}_1 = \mathbf{w}_0$ and velocity vector $\mathbf{v}_0 = 0$
    $k \leftarrow 1$
    while *not_convergence* **do**
        $\mathbf{v}_{k+1} \leftarrow \mathbf{w}_k - \eta \nabla \mathcal{L}(\mathbf{w}_k)$
        $\mathbf{w}_{k+1} \leftarrow \mathbf{v}_{k+1} + \beta(\mathbf{v}_{k+1} - \mathbf{v}_k)$
        $k \leftarrow k + 1$
    end while
end procedure

7 AdaGrad

Due to the nondifferentiability of the *hinge* loss, we might end up in a situation where some components of the gradient are very small and others large. 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.

AdaGrad [1] addresses this problem by introducing the aggregate of the squares of previously observed gradients to adjust the learning rate. This has two benefits: first, we no longer need to decide just when a gradient is large enough. Second, it scales automatically 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.

Algorithm 4 AdaGrad

Require: Learning rate or step size $\eta > 0$
Require: Offset $\epsilon > 0$ to ensures not divide by 0

procedure AdaGrad
    Initialize weight vector $\mathbf{w}_0$
    $k \leftarrow 0$
    while *not_convergence* **do**
        $\mathbf{g}_k \leftarrow \nabla \mathcal{L}(\mathbf{w}_k)$
        $\mathbf{s}_k \leftarrow \mathbf{s}_{k-1} + \mathbf{g}_k^2$
        $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k - \dfrac{\eta}{\sqrt{\mathbf{s}_k + \epsilon}} \cdot \mathbf{g}_k$
        $k \leftarrow k + 1$
    end while
end procedure

8 Sequential Minimal Optimization

The *Sequential Minimal Optimization (SMO)* [2] 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, i.e., $y^T\alpha = 0$, so the Lagrange multipliers can be solved analitically.

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 bound 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.

8.1 Classification

The ends of the diagonal line segment in terms of α_2 can be espressed as follow if the target $y_1 \neq y_2$:

$$\begin{aligned} L &= max(0, \alpha_2 - \alpha_1) \\ H &= min(C, C + \alpha_2 - \alpha_1) \end{aligned} \tag{83}$$

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

$$\begin{aligned} L &= max(0, \alpha_2 + \alpha_1 - C) \\ H &= min(C, \alpha_2 + \alpha_1) \end{aligned} \tag{84}$$

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

$$\eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2) \tag{85}$$

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:

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{86}$$

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

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

$$\alpha_2^{new,clipped} = \begin{cases} H & \text{if } \alpha_2^{new} \geq H \\ \alpha_2^{new} & \text{if } L < \alpha_2^{new} < H \\ L & \text{if } \alpha_2^{new} \leq L \end{cases} \tag{87}$$

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

$$\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped}) \tag{88}$$

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 problem 18 are:

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i u_i \geq 1 \\ 0 < \alpha_i < C &\Leftrightarrow y_i u_i = 1 \\ \alpha_i = C &\Leftrightarrow y_i u_i \leq 1 \end{aligned} \tag{89}$$

the steps described above will be iterate as long as there will be an example that violates these KKT conditions.

8.2 Regression

9 Experiments

The following experiments refer to 3-fold cross-validation over *linearly* and *nonlinearly* separable generated datasets of size 100, so the reported results are to considered as a mean over the 3 folds.

9.1 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.

9.1.1 Hinge loss

Primal formulation The experiments results shown in 1 referred to *AdaGrad* algorithm are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5. Training is stopped if after 5 iterations the training loss is not lower than the best found so far.

Table 1: SVC Primal formulation results with Hinge loss

		fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
solver	C						
adagrad	1	0.042778	68	0.979987	0.974898	13	8
	10	0.179957	398	0.979987	0.954922	8	6
	100	0.201493	435	0.979987	0.949947	7	5
liblinear	1	0.001300	627	0.982512	0.985075	11	6
	10	0.001481	775	0.987506	0.980024	7	3
	100	0.001523	1000	0.984981	0.945047	8	5

Linear Dual formulations The experiments results shown in 3 are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5 for the *AdaGrad* algorithm.

Table 2: Linear SVC Wolfe Dual formulation results with Hinge loss

		fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
solver	C						
cvxopt	1	0.022609	10	0.987525	0.974974	12	12
	10	0.017216	10	0.987525	0.974974	9	9
	100	0.031388	10	0.982512	0.974974	7	7
smo	1	0.094136	105	0.987525	0.974974	12	12
	10	0.121012	144	0.987525	0.974974	8	8
	100	0.660060	2050	0.982512	0.974974	7	7
libsvm	1	0.002635	257	0.982512	0.970074	14	14
	10	0.004053	522	0.982512	0.965099	9	9
	100	0.003277	3866	0.985019	0.955073	7	7

Table 3: Linear SVC Lagrangian Dual formulation results with Hinge loss

dual	C	fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
bcqp	1	0.006318	1	0.980006	0.970074	129	129
	10	0.006567	1	0.980006	0.970074	129	129
	100	0.005008	1	0.980006	0.970074	129	129
qp	1	0.007371	1	0.982494	0.970074	131	131
	10	0.005883	1	0.982494	0.970074	131	131
	100	0.005801	1	0.982494	0.970074	131	131

Nonlinear Dual formulations The experiments results shown in 4 and 5 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 RBF* kernels. Moreover, the experiments results shown in 5 are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5 for the *AdaGrad* algorithm.

Table 4: Nonlinear SVC Wolfe Dual formulation results with Hinge loss

solver	kernel	C	fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
cvxopt	poly	1	0.092079	10	0.832507	0.675831	32	32
		10	0.111197	10	0.956159	0.830734	11	11
		100	0.097903	10	0.993734	0.985056	8	8
	rbf	1	0.072104	10	0.998747	1.000000	48	48
		10	0.069883	10	1.000000	0.997494	16	16
		100	0.077572	10	1.000000	0.997494	12	12
libsvm	poly	1	0.002903	382	0.997498	0.992481	30	30
		10	0.005636	344	0.998752	0.992481	11	11
		100	0.002283	285	1.000000	0.989975	7	7
	rbf	1	0.003181	99	0.998752	1.000000	44	44
		10	0.003128	238	1.000000	1.000000	15	15
		100	0.002410	234	1.000000	0.995006	9	9
smo	poly	1	0.391564	137	0.831259	0.675831	32	32
		10	0.311969	141	0.948640	0.810852	11	11
		100	0.293261	241	0.993734	0.987562	8	8
	rbf	1	0.253831	38	0.998747	1.000000	44	44
		10	0.234757	39	1.000000	0.997494	15	15
		100	0.164737	47	1.000000	0.997494	11	11

Table 5: Nonlinear SVC Lagrangian Dual formulation results with Hinge loss

dual	kernel	C	fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
bcqp	poly	1	1.300455	347	0.781213	0.531328	206	206
		10	1.221536	347	0.781213	0.531328	206	206
		100	1.109463	347	0.781213	0.531328	206	206
	rbf	1	0.023433	1	1.000000	1.000000	235	235
		10	0.021801	1	1.000000	1.000000	235	235
		100	0.020279	1	1.000000	1.000000	235	235
qp	poly	1	0.708087	153	0.773723	0.518797	179	179
		10	0.744719	153	0.773723	0.518797	179	179
		100	0.743838	153	0.773723	0.518797	179	179
	rbf	1	1.063100	180	0.772526	0.543560	190	190
		10	0.805252	144	0.837515	0.663300	198	198
		100	2.084545	679	0.782584	0.615681	151	151

9.1.2 Squared Hinge loss

Primal formulation The experiments results shown in 6 referred to *Gradient Descent* algorithm are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.001 and β, i.e, the *momentum*, equal to 0.4. Training is stopped if after 5 iterations the training loss is not lower than the best found so far.

Table 6: SVC Primal formulation results with Squared Hinge loss

solver	C	momentum	fit_time	n_iter	train_accuracy	val_accuracy	train_n_sv	val_n_sv
gd	1	none	0.481737	804	0.962462	0.959822	48	24
		standard	0.305842	520	0.965006	0.959822	48	26
		nesterov	0.328955	526	0.962500	0.954847	48	25
	10	none	0.300511	496	0.969962	0.969848	22	11
		standard	0.157195	266	0.972468	0.969848	21	11
		nesterov	0.127041	256	0.972468	0.969848	22	11
	100	none	0.036143	68	0.969962	0.974823	19	9
		standard	0.031577	56	0.972468	0.964873	16	7
		nesterov	0.023003	50	0.969962	0.969848	18	9
liblinear	1	-	0.001004	189	0.985000	0.984999	14	8
	10	-	0.001673	983	0.985000	0.984999	10	4
	100	-	0.001649	1000	0.987487	0.984999	9	3

9.2 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.

9.2.1 Epsilon-insensitive loss

Primal formulation The experiments results shown in 7 referred to *AdaGrad* algorithm are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5. Training is stopped if after 5 iterations the training loss is not lower than the best found so far.

Table 7: SVR Primal formulation results with Epsilon-insensitive loss

solver	C	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
adagrad	1	0.1	0.483018	873	0.919206	0.915684	66	33
		0.2	0.501837	897	0.919990	0.916504	66	33
		0.3	0.498535	880	0.920085	0.916655	65	33
	10	0.1	2.049033	3542	0.977834	0.972868	65	32
		0.2	1.913584	3511	0.977801	0.972839	65	32
		0.3	1.882657	3478	0.977783	0.972878	65	32
	100	0.1	2.263548	4000	0.978120	0.974239	66	32
		0.2	2.248653	4000	0.978118	0.974263	66	32
		0.3	1.818380	4000	0.978120	0.974189	66	32
liblinear	1	0.1	0.000683	14	0.918827	0.916841	66	33
		0.2	0.000565	12	0.918820	0.916672	65	32
		0.3	0.000755	11	0.919212	0.916977	65	32
	10	0.1	0.000760	103	0.977852	0.972051	65	33
		0.2	0.000651	188	0.977844	0.971971	65	33
		0.3	0.000603	105	0.977865	0.972111	64	33
	100	0.1	0.000908	719	0.977723	0.974270	66	33
		0.2	0.001058	689	0.977628	0.973889	65	33
		0.3	0.001008	807	0.977658	0.974038	65	33

Linear Dual formulations The experiments results shown in 9 are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5 for the *AdaGrad* algorithm.

Table 8: Linear SVR Wolfe Dual formulation results with Epsilon-insensitive loss

solver	C	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
cvxopt	1	0.1	0.021536	9	0.917772	0.914479	67	67
		0.2	0.022979	9	0.918341	0.915058	67	67
		0.3	0.019355	10	0.918942	0.915614	66	66
	10	0.1	0.021882	9	0.977920	0.972466	67	67
		0.2	0.013598	9	0.977926	0.972474	67	67
		0.3	0.012306	10	0.977954	0.972562	66	66
	100	0.1	0.012041	9	0.977788	0.974150	67	67
		0.2	0.027912	9	0.977742	0.974033	67	67
		0.3	0.009634	9	0.977737	0.973956	67	67
smo	1	0.1	0.014631	15	0.917773	0.914442	66	66
		0.2	0.014218	13	0.918341	0.915019	66	66
		0.3	0.040505	60	0.918942	0.915576	66	66
	10	0.1	0.063922	56	0.977920	0.972445	66	66
		0.2	0.146936	219	0.977926	0.972457	65	65
		0.3	0.053513	38	0.977953	0.972544	65	65
	100	0.1	0.595028	1508	0.977788	0.974139	66	66
		0.2	0.327064	394	0.977742	0.974022	66	66
		0.3	0.515121	900	0.977737	0.973939	66	66
libsvm	1	0.1	0.002211	63	0.917627	0.915448	66	66
		0.2	0.002491	102	0.918194	0.915985	66	66
		0.3	0.002069	54	0.918786	0.916554	66	66
	10	0.1	0.001865	282	0.977852	0.972051	66	66
		0.2	0.002066	193	0.977851	0.972025	65	65
		0.3	0.002003	593	0.977870	0.972135	65	65
	100	0.1	0.003146	2621	0.977723	0.974270	66	66
		0.2	0.003566	2709	0.977673	0.974122	66	66
		0.3	0.003817	4141	0.977655	0.974045	66	66

Table 9: Linear SVR Lagrangian Dual formulation results with Epsilon-insensitive loss

dual	C	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
bcqp	1	0.1	0.656280	522	0.731073	0.721200	67	67
		0.2	0.658018	524	0.731073	0.721199	67	67
		0.3	0.602584	526	0.731073	0.721199	67	67
	10	0.1	0.614027	539	0.733638	0.723925	67	67
		0.2	0.668057	541	0.733638	0.723924	67	67
		0.3	0.637837	543	0.733638	0.723924	67	67
	100	0.1	0.733603	539	0.733638	0.723925	67	67
		0.2	0.627369	541	0.733638	0.723924	67	67
		0.3	0.409460	543	0.733638	0.723924	67	67
qp	1	0.1	0.674449	653	0.876534	0.870926	67	67
		0.2	0.678396	653	0.876534	0.870927	67	67
		0.3	0.722974	653	0.876534	0.870927	67	67
	10	0.1	0.498502	519	0.731825	0.722021	67	67
		0.2	0.563454	524	0.731825	0.722021	67	67
		0.3	0.537692	530	0.731825	0.722020	67	67
	100	0.1	0.652838	519	0.731825	0.722021	67	67
		0.2	0.604621	524	0.731825	0.722021	67	67
		0.3	0.550931	530	0.731825	0.722020	67	67

Nonlinear Dual formulations The experiments results shown in 10 and 11 are obtained with d and r hyperparameters both equal to 3 for the *polynomial* kernel; *gamma* is setted to *'scale'* for both *polynomial* and *gaussian RBF* kernels. Moreover, the experiments results shown in 5 are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.5 for the *AdaGrad* algorithm.

Table 10: Nonlinear SVR Wolfe Dual formulation results with Epsilon-insensitive loss

solver	kernel	C	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
cvxopt	poly	1	0.1	0.015057	10	0.848547	-5.090164	23	23
			0.2	0.017265	10	-3.609617	-8.632746	6	6
			0.3	0.011037	10	-0.786399	-8.517434	4	4
		10	0.1	0.010415	10	0.968145	-5.331473	25	25
			0.2	0.022692	10	-3.609013	-8.638973	4	4
			0.3	0.011880	10	-0.780783	-8.483884	4	4
		100	0.1	0.012144	10	0.945501	-5.188429	27	27
			0.2	0.014013	10	-3.608986	-8.638995	4	4
			0.3	0.012482	10	-0.780887	-8.484199	4	4
	rbf	1	0.1	0.014662	10	0.979933	0.327020	14	14
			0.2	0.021198	10	0.961792	-0.943080	6	6
			0.3	0.017357	9	0.890164	-1.687411	5	5
		10	0.1	0.014854	10	0.977704	0.640986	14	14
			0.2	0.016555	10	0.952051	-0.963789	6	6
			0.3	0.014812	10	0.882145	-1.696148	4	4
		100	0.1	0.015500	10	0.974118	0.715636	15	15
			0.2	0.018510	10	0.965122	-0.935738	7	7
			0.3	0.018584	10	0.882145	-1.696145	4	4
smo	poly	1	0.1	80.259683	175472	0.850381	-6.479953	23	23
			0.2	4.845339	6682	-5.669765	-15.026022	6	6
			0.3	0.531903	909	-2.663418	-16.100682	4	4
		10	0.1	660.696393	1352666	0.958635	-6.309580	23	23
			0.2	3.283288	5413	-5.659396	-15.048805	4	4
			0.3	4.703530	7008	-2.652290	-16.086707	4	4
		100	0.1	3838.989156	9325434	0.956258	-6.351469	23	23
			0.2	3.581371	5413	-5.659396	-15.048805	4	4
			0.3	2.518382	7008	-2.652290	-16.086707	4	4
	rbf	1	0.1	0.035963	30	0.979269	0.337883	14	14
			0.2	0.014002	14	0.953442	-0.706423	6	6
			0.3	0.009471	9	0.880480	-1.956576	5	5
		10	0.1	0.294246	198	0.979879	0.614804	14	14
			0.2	0.018293	20	0.943679	-0.729439	5	5
			0.3	0.010107	11	0.872418	-1.965274	4	4
		100	0.1	0.869753	1199	0.976323	0.735233	14	14
			0.2	0.019184	20	0.943679	-0.729439	5	5
			0.3	0.008680	11	0.872418	-1.965274	4	4
libsvm	poly	1	0.1	0.065331	202636	0.981648	-29.427063	20	20
			0.2	0.008808	5634	0.971457	-44.854253	5	5
			0.3	0.012103	1133	0.921669	-67.995416	4	4
		10	0.1	0.577278	2329808	0.981723	-28.867737	18	18
			0.2	0.012278	4967	0.972091	-44.851795	4	4
			0.3	0.001801	925	0.922233	-67.994190	3	3
		100	0.1	1.439916	6416597	0.980670	-14.594558	24	24
			0.2	0.009960	4967	0.972091	-44.851795	4	4
			0.3	0.021700	925	0.922233	-67.994190	3	3
	rbf	1	0.1	0.015557	71	0.986549	-2.969459	16	16
			0.2	0.018801	33	0.964555	-4.149278	5	5
			0.3	0.002971	8	0.912691	-4.815179	4	4
		10	0.1	0.004562	474	0.987401	-2.477526	15	15
			0.2	0.011069	34	0.964563	-4.149231	5	5
			0.3	0.001300	8	0.913367	-4.813685	4	4
		100	0.1	0.006711	2712	0.987693	-1.428809	13	13
			0.2	0.002964	34	0.964563	-4.149231	5	5
			0.3	0.002766	8	0.913367	-4.813685	4	4

Table 11: Nonlinear SVR Lagrangian Dual formulation results with Epsilon-insensitive loss

dual	kernel	C	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
bcqp	poly	1	0.1	0.482641	345	0.643482	-10.313337	67	67
			0.2	0.507596	348	0.632323	-8.680716	67	67
			0.3	0.980019	669	0.623699	-7.643487	67	67
		10	0.1	0.517057	345	0.643482	-10.313337	67	67
			0.2	0.458200	348	0.632323	-8.680716	67	67
			0.3	0.920299	669	0.623699	-7.643487	67	67
		100	0.1	0.557809	345	0.643482	-10.313337	67	67
			0.2	0.492806	348	0.632323	-8.680716	67	67
			0.3	0.654942	669	0.623699	-7.643487	67	67
	rbf	1	0.1	0.043660	19	0.740194	-1.880483	67	67
			0.2	0.192294	92	0.740177	-1.882909	67	67
			0.3	0.290547	164	0.603015	-3.956481	67	67
		10	0.1	0.037314	19	0.740194	-1.880483	67	67
			0.2	0.150209	92	0.740177	-1.882909	67	67
			0.3	0.290903	164	0.603015	-3.956481	67	67
		100	0.1	0.044858	19	0.740194	-1.880483	67	67
			0.2	0.180130	92	0.740177	-1.882909	67	67
			0.3	0.264654	164	0.603015	-3.956481	67	67
qp	poly	1	0.1	0.471979	347	0.641413	-10.283007	67	67
			0.2	0.667717	348	0.633767	-8.692657	67	67
			0.3	0.634864	349	0.626387	-7.655867	67	67
		10	0.1	0.421276	347	0.641413	-10.283007	67	67
			0.2	0.454034	348	0.633767	-8.692657	67	67
			0.3	0.492867	349	0.626387	-7.655867	67	67
		100	0.1	0.474953	347	0.641413	-10.283007	67	67
			0.2	0.574175	348	0.633767	-8.692657	67	67
			0.3	0.406300	349	0.626387	-7.655867	67	67
	rbf	1	0.1	0.191649	102	0.721454	-2.320027	67	67
			0.2	0.407150	157	0.675573	-2.863975	67	67
			0.3	0.479133	213	0.627853	-3.010361	67	67
		10	0.1	0.128122	42	0.723816	-2.306795	67	67
			0.2	0.227605	95	0.675184	-2.866411	67	67
			0.3	0.292192	163	0.614802	-3.217847	67	67
		100	0.1	0.080129	42	0.723816	-2.306795	67	67
			0.2	0.244637	95	0.675184	-2.866411	67	67
			0.3	0.393030	163	0.614802	-3.217847	67	67

9.2.2 Squared Epsilon-insensitive loss

Primal formulation The experiments results shown in 12 referred to *Gradient Descent* algorithm are obtained with η, i.e, the *learning rate* or *step size*, setted to 0.001 and β, i.e, the *momentum*, equal to 0.4. Training is stopped if after 5 iterations the training loss is not lower than the best found so far.

Table 12: SVR Primal formulation results with Squared Epsilon-insensitive loss

solver	C	momentum	epsilon	fit_time	n_iter	train_r2	val_r2	train_n_sv	val_n_sv
gd	1	none	0.1	0.626564	1000	0.963072	0.958845	66	33
			0.2	0.572553	1000	0.962855	0.958554	66	33
			0.3	0.534237	1000	0.962802	0.958604	66	33
		standard	0.1	0.668645	1000	0.976795	0.972792	66	33
			0.2	0.560162	1000	0.976805	0.972807	66	33
			0.3	0.579397	1000	0.976804	0.972802	65	33
		nesterov	0.1	0.636513	1000	0.976770	0.972801	66	33
			0.2	0.562291	1000	0.976789	0.972792	66	33
			0.3	0.646051	1000	0.976796	0.972796	65	33
	10	none	0.1	0.267765	412	0.978183	0.973954	66	33
			0.2	0.285989	405	0.978183	0.973952	66	33
			0.3	0.223693	399	0.978183	0.973950	66	32
		standard	0.1	0.134979	250	0.978183	0.973955	66	33
			0.2	0.164567	246	0.978183	0.973953	66	33
			0.3	0.137312	242	0.978183	0.973951	66	32
		nesterov	0.1	0.159334	253	0.978183	0.973955	66	33
			0.2	0.135648	249	0.978183	0.973953	66	33
			0.3	0.118718	245	0.978183	0.973951	66	32
	100	none	0.1	0.028912	51	0.978184	0.973950	66	33
			0.2	0.022776	49	0.978184	0.973947	66	33
			0.3	0.024241	47	0.978184	0.973946	66	33
		standard	0.1	0.016042	29	0.978184	0.973950	66	33
			0.2	0.017194	25	0.978184	0.973948	66	33
			0.3	0.014390	25	0.978184	0.973947	66	33
		nesterov	0.1	0.015756	28	0.978184	0.973951	66	33
			0.2	0.014655	29	0.978184	0.973950	66	33
			0.3	0.012350	29	0.978184	0.973950	66	33
liblinear	1	-	0.1	0.000834	85	0.978134	0.974007	67	32
			0.2	0.000932	86	0.978132	0.974007	66	32
			0.3	0.000998	88	0.978130	0.974014	66	32
	10	-	0.1	0.002740	776	0.978183	0.973961	66	33
			0.2	0.002714	777	0.978183	0.973968	66	33
			0.3	0.002906	771	0.978183	0.973985	66	32
	100	-	0.1	0.003736	1000	0.977164	0.972436	67	32
			0.2	0.002975	1000	0.978023	0.975378	66	33
			0.3	0.003194	1000	0.977903	0.972687	65	32

10 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, $n >> m$; meanwhile the *dual formulation*, is more suitable in case the number of examples n is less than the number of features m, $n < m$, since the complexity of the model is dominated by the number of examples.

From all these experiments we can see as, for what about the *primal* formulations, the results provided from the *custom* implementations are strongly similar to those of *sklearn* implementations, i.e., *liblinear* implementations, with a slight exception about the time gap obviously due to the different core implementation languages, Python and C respectively.

Meanwhile, for what about the *dual* formulations we can notice as *cvxopt* underperforms the *sklearn* implementations, i.e., *libsvm* implementations, 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. Despite this, the *custom* implementations does not overperform the *cvxopt* probably due to the gap generated from the different core implementation languages, again Python and C respectively. For these reasons, *sklearn* provides better results in terms of time wrt the other implementations since it is designed to work in a large-scale context and its core is implemented in C. Furthermore, in the SVC example with the polynomial kernel of degree 5, we can see that the time gap is significatively, properly two different orders of magnitude ($\simeq$ 29min vs. $\simeq$ 19ms), and this could not depend just only by the different implementation languages; it's probable that *liblinear* adopts some heuristics, i.e., low rank approssimations of the kernel matrix, to deal with the polynomial kernel in case of high degree.

Important consideration involves the number of support vector machines: the *Lagrangian dual* formulation tends to select all the data points as support vectors, so it makes the model complex and it tends to give low scores wrt the equivalent *Wolfe dual* formulation. In particular, the *Lagrangian relaxation* resulting from the *Wolfe dual* always gives rise to a nonsmooth optimization with an exception for the SVC with a Gaussian kernel where the two formulations solve exactly the same problem. In all the other cases the goodness of the solution depends on the residue in the solution of the *Lagrangian dual* at each step; one of the wrost results certainly concerns the SVC with the polynomial kernel of degree 3, where the residue is in the order of +02/03 and so the approssimation is horrible. Finally, 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 relaxation* of the same problem with the bias term embedded into the weight matrix.

References

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

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

[3] 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.

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

[5] 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.

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