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

3.1.1 Primal formulation

The general primal unconstrained formulation takes the form:

$$\min_{w,b} \mathcal{R}(w, b) + C \sum_{i=1}^{n} \mathcal{L}(w, b; x_i, y_i) \tag{6}$$

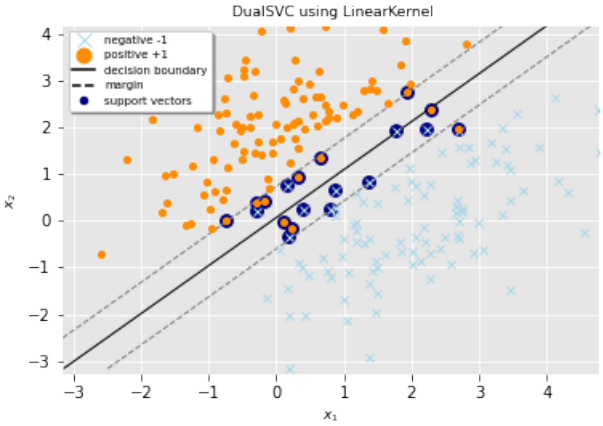


Figure 1: Linear SVC hyperplane

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:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b)) \tag{7}$$

where we make use of the *hinge* loss defined as:

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

or, equivalently:

$$\mathcal{L}_1 = \max(0, 1 - y(w^T x + b)) \tag{9}$$

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

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

$$\frac{\partial \mathcal{L}_1}{\partial w} = \begin{cases} -yx & \text{if } y(w^T x + b) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

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 7 and 38 as follows:

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

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:

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

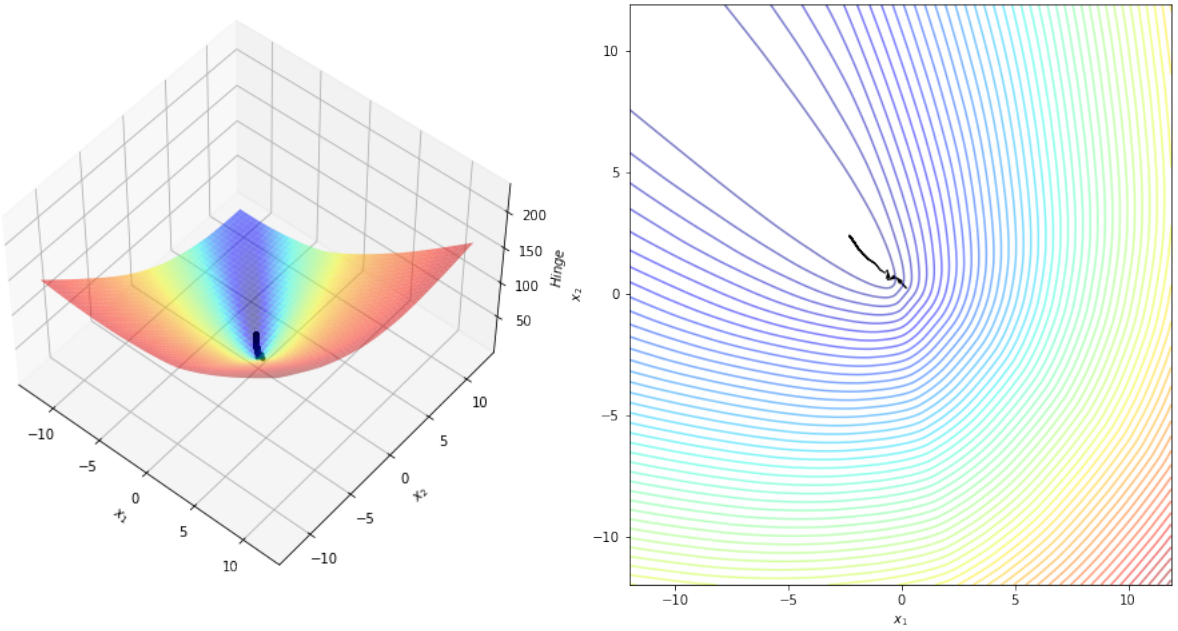


Figure 2: SVC Hinge loss with optimization steps

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 7 and 38 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:

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

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

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

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 $K_n(Q, b) = span(b, Qb, Q^2b, \ldots, Q^{n-1}b)$.

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

3.2.1 Primal formulation

Since smoothed versions of objective functions may be preferred for optimization, we can reformulate 7 as:

$$\min_{w,b}\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\max(0,1-y_i(w^Tx_i+b))^2 \tag{38}$$

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

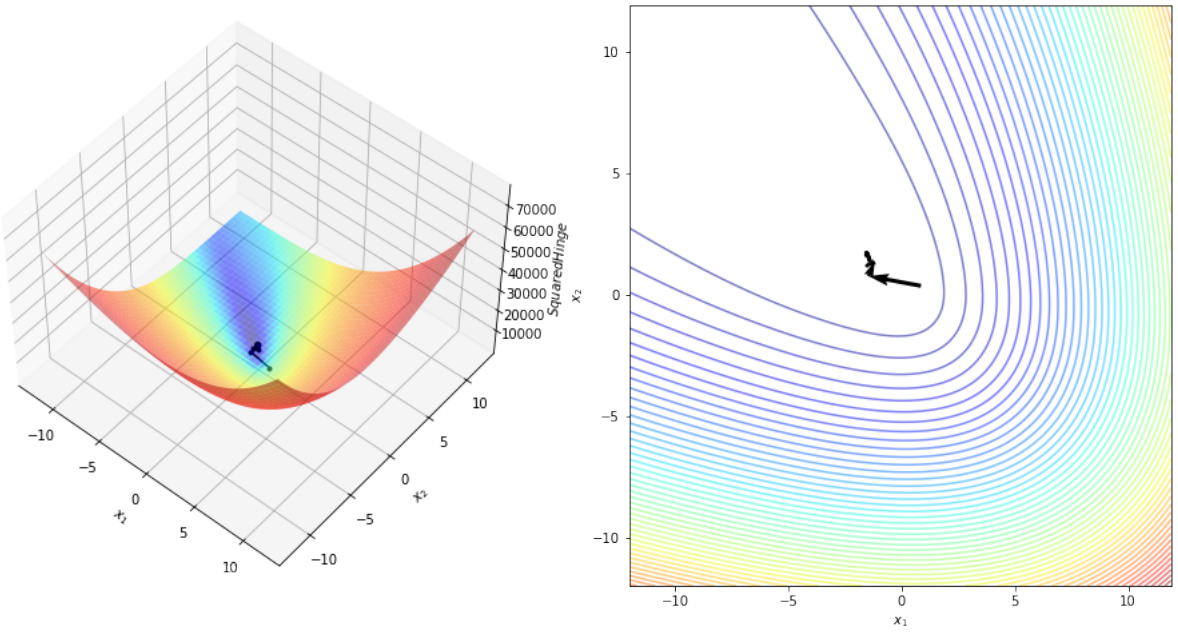


Figure 3: SVC Squared Hinge loss with optimization steps

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:

$$\{(x_i, y_i), x \in \Re^m, y_i \in \Re, i = 1, \ldots, n\} \tag{39}$$

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

$$\begin{aligned} y_i &\leq y_i' + \epsilon + \xi^+ \ \forall_i \\ y_i &\geq y_i' - \epsilon - \xi^- \ \forall_i \\ \xi_i^+, \xi_i^- &\geq 0 \ \forall_i \end{aligned} \tag{40}$$

The objective function for SVR can then be written as:

$$\begin{aligned} \min_{w,b,\xi^+,\xi^-} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) \\ \text{subject to} \quad & y_i - w^T x_i - b \leq \epsilon + \xi_i^+ \ \forall_i \\ & w^T x_i + b - y_i \leq \epsilon + \xi_i^- \ \forall_i \\ & \xi_i^+, \xi_i^- \geq 0 \ \forall_i \end{aligned} \tag{41}$$

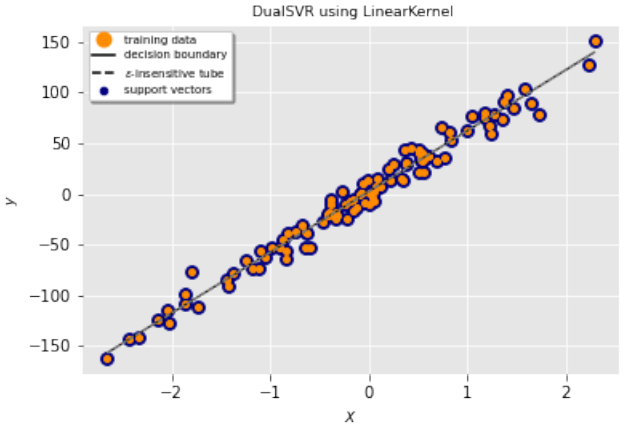


Figure 4: Linear SVR hyperplane

4.1 Epsilon-insensitive loss

4.1.1 Primal formulation

The general primal unconstrained formulation takes the same form of 6.

The quadratic optimization problem 41 can be equivalently formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, |y_i - (w^T x_i + b)| - \epsilon) \tag{42}$$

where we make use of the *epsilon-insensitive* loss defined as:

$$\mathcal{L}_\epsilon = \begin{cases} 0 & \text{if } |y - (w^T x + b)| \leq \epsilon \\ |y - (w^T x + b)| - \epsilon & \text{otherwise} \end{cases} \tag{43}$$

or, equivalently:

$$\mathcal{L}_\epsilon = \max(0, |y - (w^T x + b)| - \epsilon) \tag{44}$$

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

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

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

4.1.2 Wolfe Dual formulation

To reformulate the 41 as a *Wolfe dual*, we introduce the Lagrange multipliers $\alpha_i^+ \geq 0, \alpha_i^- \geq 0, \mu_i^+ \geq 0, \mu_i^- \geq 0\ \forall_i$:

$$\begin{aligned} \max_{\alpha^+,\alpha^-,\mu^+,\mu^-} \min_{w,b,\xi^+,\xi^-} \mathcal{W}(w,b,\xi^+,\xi^-,\alpha^+,\alpha^-,\mu^+,\mu^-) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i^+ + \xi_i^-) - \sum_{i=1}^{n}(\mu_i^+\xi_i^+ + \mu_i^-\xi_i^-) \\ - \sum_{i=1}^{n}\alpha_i^+(\epsilon + \xi_i^+ + y_i' - y_i) - \sum_{i=1}^{n}\alpha_i^-(\epsilon + \xi_i^- - y_i' + y_i) \end{aligned} \tag{46}$$

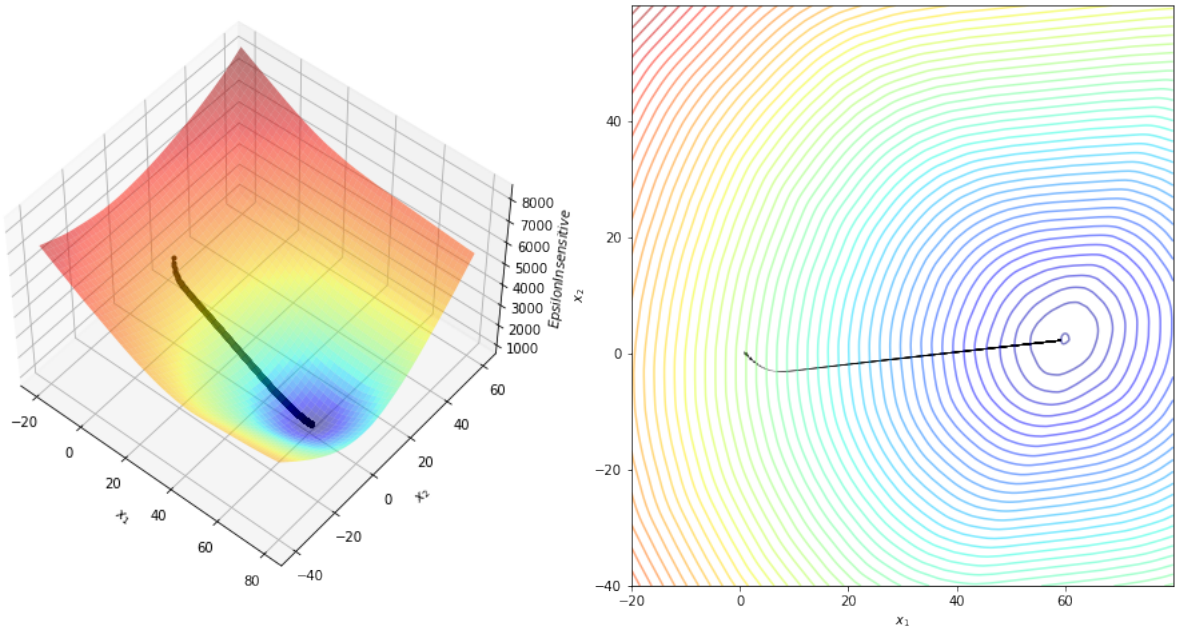


Figure 5: SVR Epsilon-insensitive loss with optimization steps

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

$$\frac{\partial \mathcal{W}}{\partial w} = w - \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)x_i \Rightarrow w = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)x_i \tag{47}$$

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-) \Rightarrow \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-) = 0 \tag{48}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i^+} = 0 \Rightarrow C = \alpha_i^+ + \mu_i^+ \tag{49}$$

$$\frac{\partial \mathcal{W}}{\partial \xi_i^-} = 0 \Rightarrow C = \alpha_i^- + \mu_i^- \tag{50}$$

Substituting 47 and 48 in, we now need to maximize $\mathcal{W}$ wrt α_i^+ and α_i^-, where $\alpha_i^+ \geq 0,\ \alpha_i^- \geq 0\ \forall_i$:

$$\max_{\alpha^+,\alpha^-} \mathcal{W}(\alpha^+, \alpha^-) = \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) - \epsilon\sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) - \frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)\langle x_i, x_j\rangle(\alpha_j^+ - \alpha_j^-) \tag{51}$$

Using $\mu_i^+ \geq 0$ and $\mu_i^- \geq 0$ together with 47 and 48 means that $\alpha_i^+ \leq C$ and $\alpha_i^- \leq C$. We therefore need to find:

$$\begin{aligned} \min_{\alpha^+,\alpha^-} \quad & \frac{1}{2}(\alpha^+ - \alpha^-)^T K(\alpha^+ - \alpha^-) + \epsilon q^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-) \\ \text{subject to} \quad & 0 \leq \alpha_i^+, \alpha_i^- \leq C\ \forall_i \\ & q^T(\alpha^+ - \alpha^-) = 0 \end{aligned} \tag{52}$$

where $q^T = [1, \dots, 1]$.
We can write the 52 in a standard quadratic form as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q \alpha - q^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \\ & e^T \alpha = 0 \end{aligned} \tag{53}$$

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:

$$y' = \sum_{i=1}^{n} (\alpha_i^+ - \alpha_i^-)\langle x_i, x' \rangle + b \tag{54}$$

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:

$$b = y_s - \epsilon - \sum_{m \in S} (\alpha_m^+ - \alpha_m^-)\langle x_m, x_s \rangle \tag{55}$$

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

$$b = \frac{1}{N_s} \sum_{s \in S} y_s - \epsilon - \sum_{m \in S} (\alpha_m^+ - \alpha_m^-)\langle x_m, x_s \rangle \tag{56}$$

From 53 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:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T (Q + ee^T)\alpha + q^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \ \forall_i \end{aligned} \tag{57}$$

4.1.3 Lagrangian Dual formulation

In order to relax the constraints in the *Wolfe dual* formulation 52 we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrangian multipliers $\mu \geq 0, \lambda_+ \geq 0, \lambda_- \geq 0$:

$$\begin{aligned} \max_{\mu, \lambda_+, \lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \mu, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T Q \alpha + q^T \alpha - \mu^T (e^T \alpha) - \lambda_+^T (u - \alpha) - \lambda_-^T \alpha \\ &= \frac{1}{2}\alpha^T Q \alpha + (q - \mu e + \lambda_+ - \lambda_-)^T \alpha - \lambda_+^T u \end{aligned} \tag{58}$$

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:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow Q\alpha + (q - \mu e + \lambda_+ - \lambda_-) = 0 \tag{59}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu e + \lambda_+ - \lambda_-) \tag{60}$$

the gradient wrt μ, λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -e\alpha \tag{61}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{62}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{63}$$

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:

$$\begin{aligned} \min_{\alpha \in K_n(Q,b)} \quad & \|Q\alpha - b\| \\ \text{where} \quad & b = -(q - \mu e + \lambda_+ - \lambda_-) \end{aligned} \tag{64}$$

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 $K_n(Q, b) = span(b, Qb, Q^2b, \ldots, Q^{n-1}b)$.

From 53 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 58 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 57.

$$\begin{aligned} \max_{\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha, \lambda_+, \lambda_-) &= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + q^T\alpha - \lambda_+^T(u - \alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T(Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u \end{aligned} \tag{65}$$

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

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-) = 0 \tag{66}$$

With α optimal solution of the linear system:

$$(Q + ee^T)\alpha = -(q + \lambda_+ - \lambda_-) \tag{67}$$

the gradient wrt λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \lambda_+} = \alpha - u \tag{68}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{69}$$

4.2 Squared Epsilon-insensitive loss

4.2.1 Primal formulation

To provide a continuously differentiable function the optimization problem 42 can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, |y_i - (w^T x_i + b)| - \epsilon)^2 \tag{70}$$

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

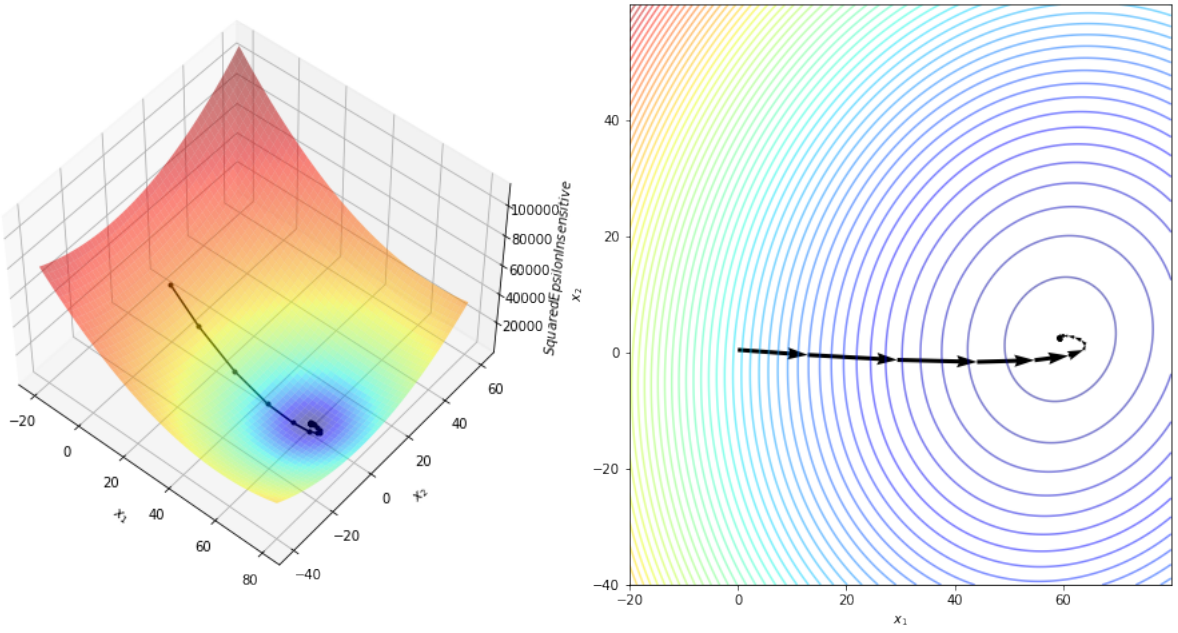


Figure 6: SVC Squared Epsilon-insensitive loss with optimization steps

5 Nonlinear Support Vector Machines

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

$$Q_{ij} = y_i y_j k(x_i, x_j) \tag{71}$$

or, a matrix K from in the SVR case:

$$K_{ij} = k(x_i, x_j) \tag{72}$$

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

$$k(x_i, x_j) = \langle x_i, x_j \rangle = x_i^T x_j \tag{73}$$

is known as *linear* kernel.

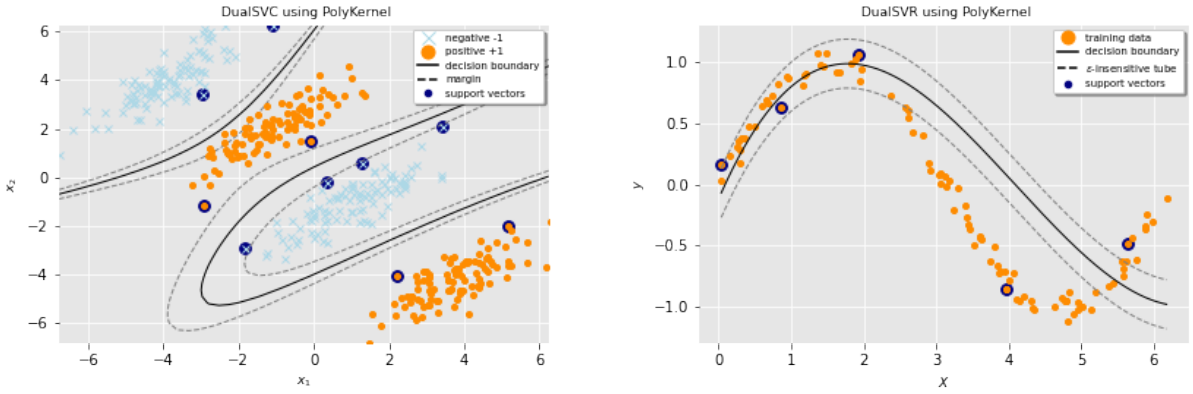
The reason that this *kernel trick* is useful is that there are many classification/regression problems that are nonlinearly separable/regressable in the *input space*, which might be in a higher dimensionality *feature space* given a suitable mapping $x \to \phi(x)$.

5.1 Polynomial kernel

The *polynomial* kernel is defined as:

$$k(x_i, x_j) = (\gamma \langle x_i, x_j \rangle + r)^d \tag{74}$$

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

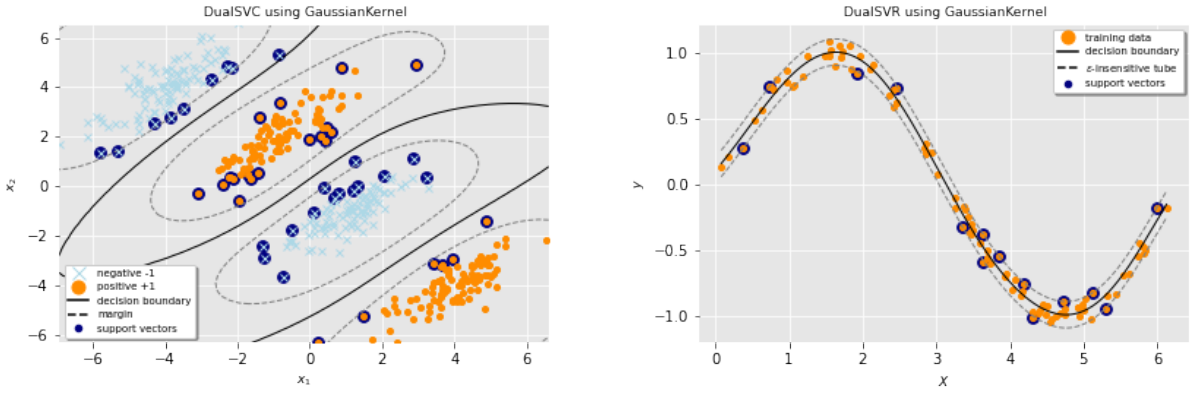
The *gaussian* kernel is defined as:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{75}$$

or, equivalently:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{76}$$

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

6.1 Momentum

6.1.1 Standard

Algorithm 1 Standard Momentum Accelerated Gradient Descent. The learning rate η, the α term and the maximum number of iterations are given.

Require: Learning rate η and momentum parameter α
Require: Maximum number of iteration and error threshold
procedure MOMENTUM DESCENT
  Initialize $\mathbf{w}$ and $\mathbf{v}$
  $k \leftarrow 0$
  while $k < max\_iterations$ && $error\_th < e$ **do**
    if Nesterov Momentum **then**
      $\tilde{\mathbf{W}} \leftarrow \mathbf{w} + \alpha\mathbf{v}$
    end if
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{n}\nabla \sum_i L(\tilde{\mathbf{W}})$
    Compute velocity update: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta\mathbf{g}$
    Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$
  end while
end procedure

6.1.2 Nesterov

Algorithm 2 Nesterov Momentum Accelerated Gradient Descent. The learning rate η, the α term and the maximum number of iterations are given.

Require: Learning rate η and momentum parameter α
Require: Maximum number of iteration and error threshold
procedure MOMENTUM DESCENT
  Initialize $\mathbf{w}$ and $\mathbf{v}$
  $k \leftarrow 0$
  while $k < max\_iterations$ && $error\_th < e$ **do**
    if Nesterov Momentum **then**
      $\tilde{\mathbf{W}} \leftarrow \mathbf{w} + \alpha\mathbf{v}$
    end if
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{n}\nabla \sum_i L(\tilde{\mathbf{W}})$
    Compute velocity update: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \eta\mathbf{g}$
    Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$
  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.

We use the variable s_t to accumulate past gradient variance as follows:

$$\begin{aligned} g_t &= \partial_{w_t} \mathcal{L}(y_t, f(x_t, w)) \\ s_t &= s_{t-1} + g_t^2 \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \cdot g_t \end{aligned} \tag{77}$$

where ϵ is an additive constant that ensures that we do not divide by 0.

Algorithm 3 AdaGrad Algorithm. The learning rate η, the α term and the maximum number of iterations are given.

Require: Learning rate η and momentum parameter α
Require: Maximum number of iteration and error threshold
procedure MOMENTUM DESCENT
  Initialize $\mathbf{w}$ and $\mathbf{v}$
  $k \leftarrow 0$
  while $k < max\_iterations$ && $error\_th < e$ **do**
    if Nesterov Momentum **then**
      $\tilde{\mathbf{W}} \leftarrow \mathbf{w} + \alpha \mathbf{v}$
    end if
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{n} \nabla \sum_i L(\tilde{\mathbf{W}})$
    Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \mathbf{g}$
    Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$
  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 $y_1 \neq y_2$:

$$\begin{aligned} L &= max(0, \alpha_2 - \alpha_1) \\ H &= min(C, C + \alpha_2 - \alpha_1) \end{aligned} \tag{78}$$

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

$$\begin{aligned} L &= max(0, \alpha_2 + \alpha_1 - C) \\ H &= min(C, \alpha_2 + \alpha_1) \end{aligned} \tag{79}$$

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

$$\eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2) \tag{80}$$

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:

$$\alpha_2^{new} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta} \tag{81}$$

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:

$$\alpha_2^{new,clipped} = \begin{cases} H & \text{if } \alpha_2^{new} \geq H \\ \alpha_2^{new} & \text{if } L < \alpha_2^{new} < H \\ L & \text{if } \alpha_2^{new} \leq L \end{cases} \tag{82}$$

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

$$\alpha_1^{new} = \alpha_1 + s(\alpha_2 - \alpha_2^{new,clipped}) \tag{83}$$

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:

$$\begin{aligned} &\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1 \\ &0 < \alpha_i < C \Leftrightarrow y_i u_i = 1 \\ &\alpha_i = C \Leftrightarrow y_i u_i \leq 1 \end{aligned} \tag{84}$$

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 C complexity hyperparameter, 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

Table 1: SVC Primal formulation results with Hinge loss

solver	C	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
adagrad	1	0.045228	0.967475	0.969848	66	20	10
	10	0.247653	0.972468	0.974898	398	12	7
	100	0.301816	0.972468	0.974898	593	10	7
liblinear	1	0.001328	0.982494	0.980024	153	12	5
	10	0.001203	0.987487	0.989974	633	7	4
	100	0.001864	0.985000	0.984999	831	7	3

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

solver	C	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
cvxopt	1	0.015791	0.972487	0.959897	1	15	15
	10	0.025156	0.972487	0.959897	1	11	11
	100	0.028108	0.972487	0.959897	1	38	38
smo	1	0.042172	0.972487	0.959897	23	15	15
	10	0.114690	0.972487	0.959897	76	11	11
	100	0.281406	0.972487	0.959897	411	10	10
libsvm	1	0.002600	0.987506	0.985075	209	12	12
	10	0.004442	0.990012	0.980100	213	8	8
	100	0.002549	0.990012	0.980100	757	7	7

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

dual	C	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
bcqp	1	0.006401	0.992519	0.99005	1	130	130
	10	0.006864	0.992519	0.99005	1	130	130
	100	0.005661	0.992519	0.99005	1	130	130
qp	1	0.005798	0.995006	0.99005	1	132	132
	10	0.006755	0.995006	0.99005	1	132	132
	100	0.006650	0.995006	0.99005	1	132	132

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

solver	kernel	C	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
cvxopt	poly	1	0.128965	0.852492	0.728519	1	30	30
		10	0.117962	0.923686	0.760857	1	10	10
		100	0.090554	0.956197	0.835615	1	8	8
	rbf	1	0.132866	1.000000	0.997494	1	43	43
		10	0.123202	1.000000	0.997494	1	15	15
		100	0.094838	1.000000	0.994987	1	12	12
libsvm	poly	1	0.009136	1.000000	0.992481	287	30	30
		10	0.005259	1.000000	0.984962	504	12	12
		100	0.003439	1.000000	0.974937	460	8	8
	rbf	1	0.005949	1.000000	0.997512	121	46	46
		10	0.003889	1.000000	1.000000	214	15	15
		100	0.002700	1.000000	1.000000	105	11	11
smo	poly	1	0.452768	0.851243	0.730988	114	29	29
		10	0.383016	0.922437	0.763364	71	10	10
		100	0.343767	0.947429	0.803296	132	9	9
	rbf	1	0.287587	1.000000	0.997494	41	42	42
		10	0.259508	1.000000	0.994987	43	13	13
		100	0.319339	1.000000	0.994987	101	12	12

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

dual	kernel	C	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
bcqp	poly	1	0.070312	0.806196	0.553791	4	211	211
		10	0.051106	0.806196	0.553791	4	211	211
		100	0.033508	0.806196	0.553791	4	211	211
	rbf	1	0.057051	1.000000	0.994987	1	246	246
		10	0.024727	1.000000	0.994987	1	246	246
		100	0.023880	1.000000	0.994987	1	246	246
qp	poly	1	0.601361	0.804934	0.551378	102	178	178
		10	0.598967	0.804934	0.551378	102	178	178
		100	0.455699	0.804934	0.551378	102	178	178
	rbf	1	1.002897	0.776332	0.600363	126	151	151
		10	0.731749	0.891207	0.745670	155	197	197
		100	0.297122	0.860001	0.728089	49	160	160

Table 6: SVC Primal formulation results with Squared Hinge loss

solver	C	momentum	fit_time	train_accuracy	val_accuracy	n_iter	train_n_sv	val_n_sv
gd	1	none	0.123503	0.984962	0.969923	123	30	18
		standard	0.071623	0.984962	0.974898	81	26	13
		nesterov	0.128726	0.979987	0.974898	101	24	13
	10	none	0.086649	0.977462	0.974898	125	16	9
		standard	0.061827	0.984962	0.974898	124	14	8
		nesterov	0.026249	0.984962	0.969923	37	15	8
	100	none	0.034297	0.979969	0.984848	45	8	3
		standard	0.014864	0.982475	0.974898	20	6	3
		nesterov	0.012460	0.982475	0.974898	18	5	2
liblinear	1	-	0.001369	0.965025	0.960124	290	23	11
	10	-	0.002461	0.965025	0.955148	1001	22	11
	100	-	0.002265	0.970018	0.945198	1001	22	11

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

solver	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
adagrad	1	0.1	0.552443	0.919151	0.915619	872	66	33
liblinear	1	0.1	0.000486	0.918844	0.916863	13	66	33
adagrad	10	0.1	2.037144	0.977824	0.972910	3561	65	32
liblinear	10	0.1	0.000525	0.977848	0.972085	90	65	33
adagrad	100	0.1	2.211481	0.978119	0.974239	3999	66	32
liblinear	100	0.1	0.001154	0.977744	0.974259	793	66	33
adagrad	1	0.2	0.594297	0.920057	0.916558	893	66	33
liblinear	1	0.2	0.000647	0.918822	0.916660	12	65	32
adagrad	10	0.2	2.045412	0.977798	0.972835	3509	65	32
liblinear	10	0.2	0.000551	0.977847	0.972059	146	65	33
adagrad	100	0.2	2.201760	0.978122	0.974169	3999	66	32
liblinear	100	0.2	0.001270	0.977638	0.973877	654	65	33
adagrad	1	0.3	0.542826	0.920114	0.916683	882	65	33
liblinear	1	0.3	0.000537	0.919322	0.917117	10	65	32
adagrad	10	0.3	1.931806	0.977783	0.972878	3474	65	32
liblinear	10	0.3	0.000492	0.977871	0.972147	108	64	33
adagrad	100	0.3	1.872259	0.978120	0.974223	3999	66	32
liblinear	100	0.3	0.001098	0.977658	0.974011	899	66	33

9.1.2 Squared Hinge loss

9.2 Support Vector Regression

9.2.1 Epsilon-insensitive loss

solver	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
cvxopt	1	0.1	0.015898	0.917772	0.914479	-	67	67
libsvm	1	0.1	0.003156	0.917627	0.915448	63	66	66
smo	1	0.1	0.014242	0.917773	0.914442	14	66	66
cvxopt	10	0.1	0.016083	0.977920	0.972466	-	67	67
libsvm	10	0.1	0.002189	0.977852	0.972051	282	66	66
smo	10	0.1	0.071679	0.977920	0.972445	55	66	66
cvxopt	100	0.1	0.012125	0.977788	0.974150	-	67	67
libsvm	100	0.1	0.004521	0.977723	0.974270	2621	66	66
smo	100	0.1	0.569793	0.977788	0.974139	1507	66	66
cvxopt	1	0.2	0.020107	0.918341	0.915058	-	67	67
libsvm	1	0.2	0.004471	0.918194	0.915985	102	66	66
smo	1	0.2	0.012662	0.918341	0.915019	12	66	66
cvxopt	10	0.2	0.016285	0.977926	0.972474	-	67	67

ld	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
bcqp	1	0.1	0.599320	0.731073	0.721200	521	67	67
qp	1	0.1	0.673257	0.876534	0.870926	652	67	67
bcqp	10	0.1	0.642036	0.733638	0.723925	538	67	67
qp	10	0.1	0.513933	0.731825	0.722021	518	67	67
bcqp	100	0.1	0.540965	0.733638	0.723925	538	67	67
qp	100	0.1	0.512248	0.731825	0.722021	518	67	67
bcqp	1	0.2	0.653856	0.731073	0.721199	523	67	67
qp	1	0.2	0.683596	0.876534	0.870927	652	67	67
bcqp	10	0.2	0.603952	0.733638	0.723924	540	67	67
qp	10	0.2	0.523205	0.731825	0.722021	523	67	67
bcqp	100	0.2	0.494582	0.733638	0.723924	540	67	67
qp	100	0.2	0.469498	0.731825	0.722021	523	67	67
bcqp	1	0.3	0.587615	0.731073	0.721199	525	67	67
qp	1	0.3	0.702774	0.876534	0.870927	652	67	67
bcqp	10	0.3	0.600687	0.733638	0.723924	542	67	67
qp	10	0.3	0.518083	0.731825	0.722020	529	67	67
bcqp	100	0.3	0.441069	0.733638	0.723924	542	67	67
qp	100	0.3	0.372234	0.731825	0.722020	529	67	67

solver	kernel	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
cvxopt	poly	1	0.1	0.012785	0.414638	-26.864749	-	18	18
libsvm	poly	1	0.1	0.046200	0.979295	-8.978090	134487	21	21
smo	poly	1	0.1	52.024504	0.309358	-30.592107	101185	18	18
cvxopt	rbf	1	0.1	0.011190	0.971967	-1.895743	-	12	12
libsvm	rbf	1	0.1	0.001463	0.985722	-0.927310	136	17	17
smo	rbf	1	0.1	0.030710	0.975638	-2.110523	27	11	11
cvxopt	poly	10	0.1	0.010302	0.756741	-37.096432	-	19	19
libsvm	poly	10	0.1	0.415621	0.979466	-9.594846	1475764	20	20
smo	poly	10	0.1	732.300189	0.506437	-42.271049	1596205	18	18
cvxopt	rbf	10	0.1	0.008236	0.978627	-1.546229	-	9	9
libsvm	rbf	10	0.1	0.003731	0.986915	-0.380431	631	14	14
smo	rbf	10	0.1	0.084678	0.975736	-1.668655	127	9	9
cvxopt	poly	100	0.1	0.009482	0.872392	-32.869690	-	40	40
libsvm	poly	100	0.1	2.734114	0.968475	-12.778407	14132099	29	29
smo	poly	100	0.1	8520.397550	0.630464	-43.759998	23550227	18	18
cvxopt	rbf	100	0.1	0.012793	0.984049	-1.601824	-	9	9
libsvm	rbf	100	0.1	0.003926	0.986620	-0.615667	7463	15	15
smo	rbf	100	0.1	0.206550	0.977478	-1.678005	190	9	9
cvxopt	poly	1	0.2	0.019310	-24.725734	-14.560019	-	5	5
libsvm	poly	1	0.2	0.012645	0.969960	-20.242246	5020	5	5
smo	poly	1	0.2	1.905395	-78.501086	-28.105470	2702	6	6
cvxopt	rbf	1	0.2	0.013063	0.950854	-3.290344	-	5	5
libsvm	rbf	1	0.2	0.003546	0.967046	-1.575738	25	6	6
smo	rbf	1	0.2	0.016769	0.941880	-3.730226	17	5	5
cvxopt	poly	10	0.2	0.011569	-1.399837	-13.286394	-	4	4
libsvm	poly	10	0.2	0.026896	0.970775	-20.240331	21778	4	4
smo	poly	10	0.2	3.858863	-5.246133	-24.474814	5490	4	4
cvxopt	rbf	10	0.2	0.012855	0.945045	-3.298715	-	5	5
libsvm	rbf	10	0.2	0.019481	0.968151	-1.575546	22	5	5
smo	rbf	10	0.2	0.014923	0.936370	-3.740143	15	5	5
cvxopt	poly	100	0.2	0.011042	-1.370225	-13.318003	-	4	4
libsvm	poly	100	0.2	0.078896	0.970781	-20.239827	365018	4	4
smo	poly	100	0.2	19.640497	-5.246629	-24.476366	27815	4	4
cvxopt	rbf	100	0.2	0.013807	0.945045	-3.298716	-	5	5
libsvm	rbf	100	0.2	0.006708	0.968151	-1.575546	22	5	5
smo	rbf	100	0.2	0.014204	0.936370	-3.740143	15	5	5
cvxopt	poly	1	0.3	0.010673	-2.454608	-71.364588	-	4	4
libsvm	poly	1	0.3	0.003224	0.927252	-51.573460	1072	4	4
smo	poly	1	0.3	0.359517	-4.782670	-73.217161	571	4	4
cvxopt	rbf	1	0.3	0.013750	0.910926	-5.458921	-	4	4
libsvm	rbf	1	0.3	0.009628	0.923272	-2.106825	21	4	4
smo	rbf	1	0.3	0.007615	0.895258	-6.100397	10	3	3
cvxopt	poly	10	0.3	0.010410	-0.787233	-70.831075	-	3	3
libsvm	poly	10	0.3	0.005132	0.929397	-51.567032	2441	4	4
smo	poly	10	0.3	0.359631	-3.095494	-72.674875	543	4	4
cvxopt	rbf	10	0.3	0.013062	0.910926	-5.458915	-	4	4
libsvm	rbf	10	0.3	0.001174	0.923272	-2.106825	21	4	4
smo	rbf	10	0.3	0.010678	0.895258	-6.100397	10	3	3
cvxopt	poly	100	0.3	0.011879	-0.787409	-70.807854	-	3	3
libsvm	poly	100	0.3	0.002220	0.929397	-51.567032	2441	4	4
smo	poly	100	0.3	0.348538	-3.095494	-72.674875	543	4	4
cvxopt	rbf	100	0.3	0.014641	0.920401	-5.237672	-	3	3
libsvm	rbf	100	0.3	0.009147	0.923272	-2.106825	21	4	4
smo	rbf	100	0.3	0.010081	0.895258	-6.100397	10	3	3

ld	kernel	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
bcqp	poly	1	0.1	0.017984	0.643732	-15.426192	13	67	67
qp	poly	1	0.1	0.150542	0.421344	-9.329338	131	66	66
bcqp	rbf	1	0.1	0.062317	0.738736	-3.372788	34	67	67
qp	rbf	1	0.1	0.267464	0.718313	-2.765181	138	67	67
bcqp	poly	10	0.1	0.017602	0.643732	-15.426192	13	67	67
qp	poly	10	0.1	0.140099	0.421344	-9.329338	131	66	66
bcqp	rbf	10	0.1	0.060944	0.738736	-3.372788	34	67	67
qp	rbf	10	0.1	0.078848	0.717282	-2.842064	42	67	67
bcqp	poly	100	0.1	0.018668	0.643732	-15.426192	13	67	67
qp	poly	100	0.1	0.141352	0.421344	-9.329338	131	66	66
bcqp	rbf	100	0.1	0.058455	0.738736	-3.372788	34	67	67
qp	rbf	100	0.1	0.076331	0.717282	-2.842064	42	67	67
bcqp	poly	1	0.2	0.037193	0.638981	-13.631038	22	67	67
qp	poly	1	0.2	0.310099	0.410610	-9.274233	289	65	65
bcqp	rbf	1	0.2	0.205713	0.686113	-4.392405	108	67	67
qp	rbf	1	0.2	0.311129	0.708151	-3.092656	143	67	67
bcqp	poly	10	0.2	0.028625	0.638981	-13.631038	22	67	67
qp	poly	10	0.2	0.319094	0.410610	-9.274233	289	65	65
bcqp	rbf	10	0.2	0.174781	0.686113	-4.392405	108	67	67
qp	rbf	10	0.2	0.155609	0.694267	-3.177124	88	67	67
bcqp	poly	100	0.2	0.030659	0.638981	-13.631038	22	67	67
qp	poly	100	0.2	0.247892	0.410610	-9.274233	289	65	65
bcqp	rbf	100	0.2	0.122796	0.686113	-4.392405	108	67	67
qp	rbf	100	0.2	0.162911	0.694267	-3.177124	88	67	67
bcqp	poly	1	0.3	0.040232	0.619272	-13.613887	26	67	67
qp	poly	1	0.3	0.401573	0.375807	-12.796440	363	63	63
bcqp	rbf	1	0.3	0.254562	0.590783	-5.517688	146	67	67
qp	rbf	1	0.3	0.407843	0.609949	-3.976029	230	67	67
bcqp	poly	10	0.3	0.033742	0.619272	-13.613887	26	67	67
qp	poly	10	0.3	0.347158	0.375807	-12.796440	363	63	63
bcqp	rbf	10	0.3	0.203943	0.590783	-5.517688	146	67	67
qp	rbf	10	0.3	0.315084	0.637726	-3.588466	174	67	67
bcqp	poly	100	0.3	0.032668	0.619272	-13.613887	26	67	67
qp	poly	100	0.3	0.279308	0.375807	-12.796440	363	63	63
bcqp	rbf	100	0.3	0.136740	0.590783	-5.517688	146	67	67
qp	rbf	100	0.3	0.242088	0.637726	-3.588466	174	67	67

9.2.2 Squared Epsilon-insensitive loss

solver	momentum	C	epsilon	fit_time	train_r2	val_r2	n_iter	train_n_sv	val_n_sv
liblinear	-	1	0.1	0.000824	0.978134	0.973998	83	67	32
sgd	none	1	0.1	0.195629	0.978126	0.973976	351	66	32
sgd	standard	1	0.1	0.104086	0.978130	0.973982	179	66	32
sgd	nesterov	1	0.1	0.104320	0.978130	0.973982	182	66	32
liblinear	-	10	0.1	0.002591	0.978183	0.973965	774	66	33
sgd	none	10	0.1	0.025562	0.978184	0.973958	47	66	33
sgd	standard	10	0.1	0.013823	0.977869	0.975114	24	65	33
sgd	nesterov	10	0.1	0.015189	0.978184	0.973958	25	66	33
liblinear	-	100	0.1	0.003617	0.977810	0.974351	1000	67	32
sgd	none	100	0.1	0.003544	-18.792762	-18.629559	5	67	33
sgd	standard	100	0.1	0.014836	0.978184	0.973963	28	66	33
sgd	nesterov	100	0.1	0.003504	-1625.878702	-1596.359750	5	67	33
liblinear	-	1	0.2	0.000822	0.978133	0.974008	87	66	32
sgd	none	1	0.2	0.200803	0.978125	0.973974	348	66	32
sgd	standard	1	0.2	0.108426	0.978129	0.973980	177	66	32
sgd	nesterov	1	0.2	0.105084	0.978129	0.973980	181	66	32
liblinear	-	10	0.2	0.002662	0.978183	0.973968	775	66	33
sgd	none	10	0.2	0.025126	0.978184	0.973957	45	66	33
sgd	standard	10	0.2	0.013937	0.977869	0.975107	24	65	33
sgd	nesterov	10	0.2	0.013735	0.978184	0.973958	24	66	33
liblinear	-	100	0.2	0.003612	0.978173	0.973883	1000	66	33
sgd	none	100	0.2	0.003521	-18.802757	-18.640422	5	67	33
sgd	standard	100	0.2	0.016040	0.978184	0.973963	28	66	33
sgd	nesterov	100	0.2	0.003533	-1636.129126	-1608.392099	5	67	33
liblinear	-	1	0.3	0.000733	0.978130	0.974013	86	66	32
sgd	none	1	0.3	0.174313	0.978124	0.973972	345	66	32
sgd	standard	1	0.3	0.090863	0.978129	0.973977	174	66	32
sgd	nesterov	1	0.3	0.095276	0.978129	0.973977	178	66	32
liblinear	-	10	0.3	0.002766	0.978183	0.973982	754	66	32
sgd	none	10	0.3	0.025836	0.978183	0.973955	44	66	33
sgd	standard	10	0.3	0.013707	0.977875	0.975095	24	65	33
sgd	nesterov	10	0.3	0.013613	0.978184	0.973958	24	66	33
liblinear	-	100	0.3	0.003617	0.977908	0.974973	1000	66	32
sgd	none	100	0.3	0.003552	-19.157272	-18.998711	5	67	33
sgd	standard	100	0.3	0.015196	0.978184	0.973968	28	66	33
sgd	nesterov	100	0.3	0.002588	-1634.206747	-1605.710912	5	67	33

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.