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 standard *gradient descent* approach 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 standard *gradient descent* approach for solving the SVR in its *primal* formulation.

2 Abstract

A *Support Vector Machine (SVM)* 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:

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

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 $\frac{1}{2}\|w\|^2$, 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 Primal Formulations

The general primal unconstrained formulation takes the form:

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

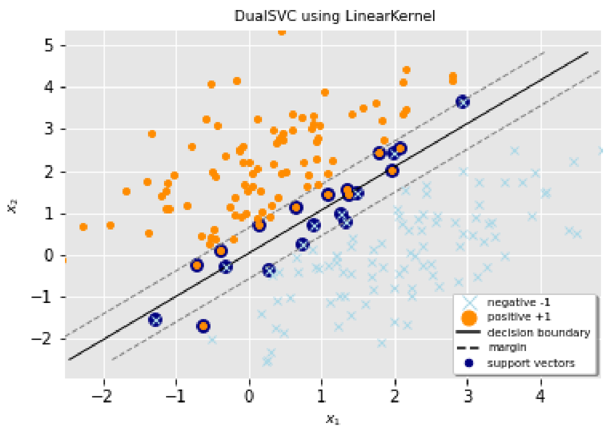


Figure 1: Linear SVC hyperplane

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).

3.1.1 Hinge loss

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:

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

3.1.2 Squared Hinge loss

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

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b))^2 \tag{11}$$

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

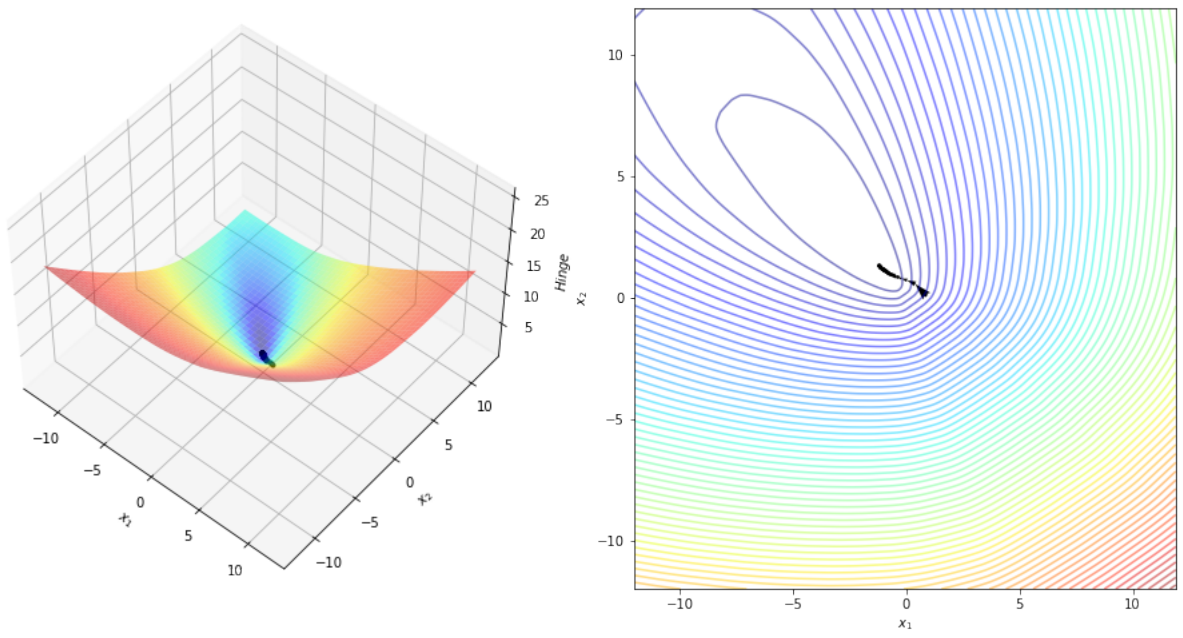


Figure 2: SVC Hinge loss

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 11 as follows:

$$\min_{w,b} \frac{1}{2}(\|w\|^2 + b^2) + C \sum_{i=1}^{n} \mathcal{L}(w; x_i, y_i) \tag{12}$$

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:

$$\begin{aligned} \min_{w} \quad & \frac{1}{2}\|\bar{w}\|^2 + C \sum_{i=1}^{n} \mathcal{L}(w; \bar{x}_i, y_i) \\ \text{where} \quad & \bar{w}^T = [w^T, b] \\ & \bar{x}_i^T = [x_i^T, 1] \end{aligned} \tag{13}$$

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 11 are not equivalent to 12 or 13 since in the first one the bias term b does not contribute to the *regularization term*, so the SVM formulation is based on an unregularized bias term b, as highlighted by the *statistical learning theory*. But, in machine learning sense, numerical experiments in [6] show that the accuracy does not vary much when the bias term b is embedded into the weight vector w.

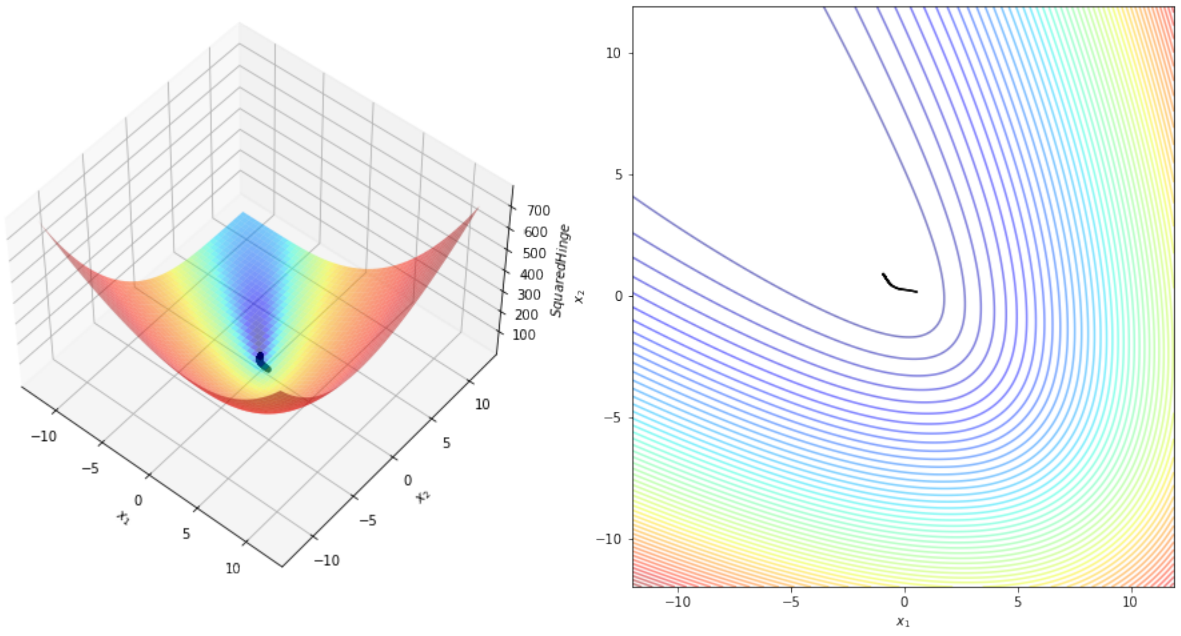


Figure 3: SVC Squared Hinge loss

3.2 Dual Formulations

3.2.1 Wolfe Dual

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

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

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

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

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

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

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

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

or, equivalently:

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

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

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

and, by substituting in 15, we get:

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

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

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

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

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

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

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

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

From 19 we can notice that the equality constraint $y^T\alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. We report below the box-constrained dual formulation [6] that arises from the primal 12 or 13 where the bias term b is embedded into the weight vector w:

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

3.2.2 Lagrangian Dual

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

$$\begin{aligned} \max_{\mu,\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda_+,\lambda_-) &= \frac{1}{2}\alpha^TQ\alpha + q^T\alpha - \mu^T(y^T\alpha) - \lambda_+^T(u-\alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^TQ\alpha + (q - \mu y + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^Tu \end{aligned} \tag{27}$$

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:

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

With α optimal solution of the linear system:

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

the gradient wrt μ, λ_+ and λ_- are:

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

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

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

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:

$$\begin{aligned} \min_{\alpha \in K_n(Q,b)} \quad & \|Q\alpha - b\| \\ \text{where} \quad & b = -(q - \mu y + \lambda_+ - \lambda_-) \end{aligned} \tag{33}$$

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

From 19 we can notice that the equality constraint $y^T\alpha = 0$ arises form the stationarity condition $\partial_b \mathcal{W} = 0$. So, again, for simplicity, we can again consider the bias term b embedded into the weight vector. In this way the dimensionality of 27 is reduced 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 26.

$$\begin{aligned}\max_{\lambda_+,\lambda_-}\min_{\alpha}\mathcal{L}(\alpha,\lambda_+,\lambda_-) &= \frac{1}{2}\alpha^T(Q+yy^T)\alpha + q^T\alpha - \lambda_+^T(u-\alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T(Q+yy^T)\alpha + (q+\lambda_+-\lambda_-)^T\alpha - \lambda_+^T u\end{aligned} \tag{34}$$

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:

$$\frac{\partial\mathcal{L}}{\partial\alpha} = 0 \Rightarrow (Q+yy^T)\alpha + (q+\lambda_+-\lambda_-) = 0 \tag{35}$$

With α optimal solution of the linear system:

$$(Q+yy^T)\alpha = -(q+\lambda_+-\lambda_-) \tag{36}$$

the gradient wrt λ_+ and λ_- are:

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

$$\frac{\partial\mathcal{L}}{\partial\lambda_-} = -\alpha \tag{38}$$

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:

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

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:

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

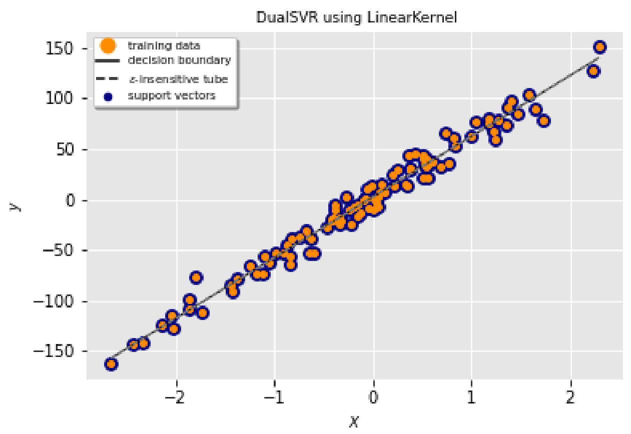


Figure 4: Linear SVR hyperplane

4.1 Primal Formulations

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

4.1.1 Epsilon-insensitive loss

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:

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

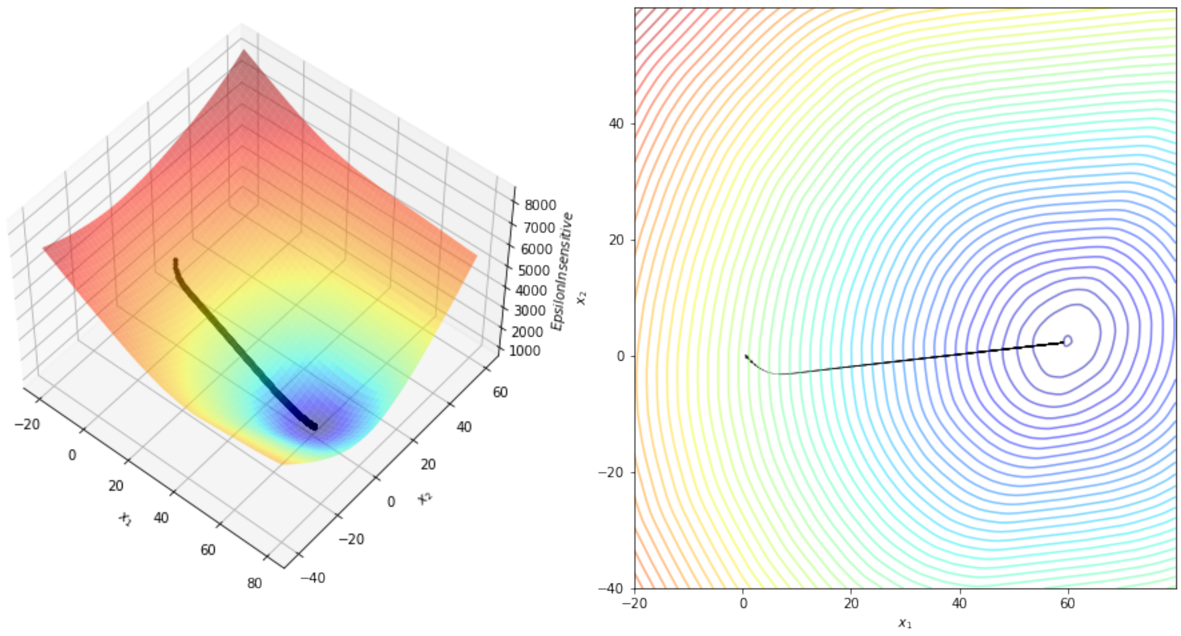


Figure 5: SVR Epsilon-insensitive loss

4.1.2 Squared Epsilon-insensitive loss

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

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, |y_i - (w^T x_i + b)| - \epsilon)^2 \tag{46}$$

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

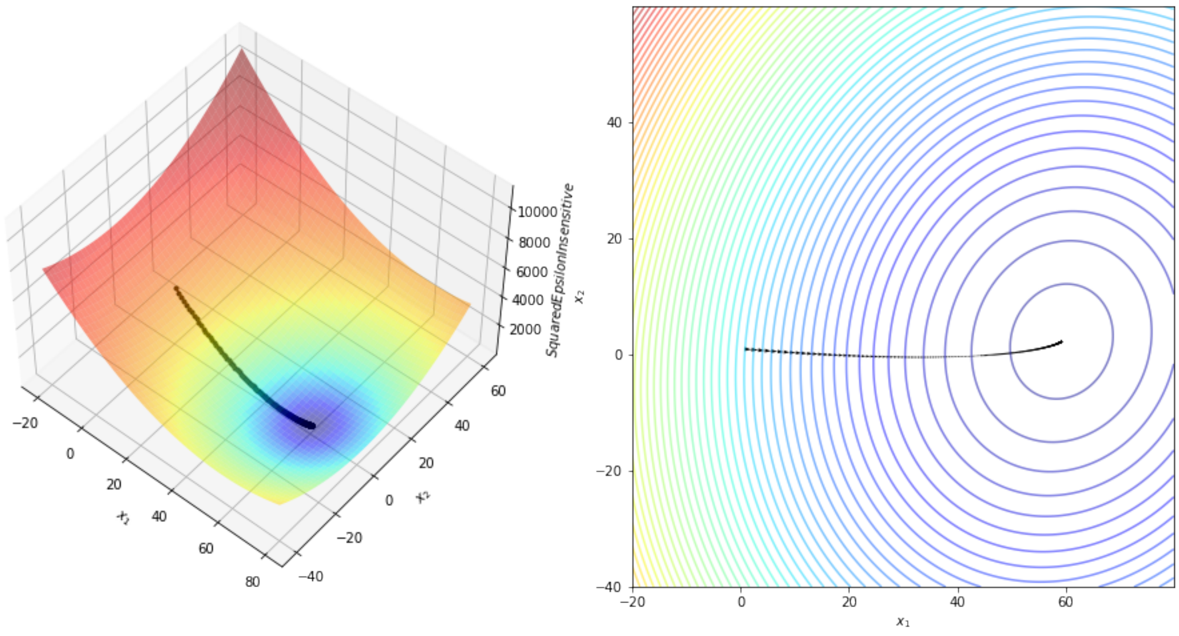


Figure 6: SVC Squared Epsilon-insensitive loss

4.2 Dual Formulations

4.2.1 Wolfe Dual

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

$$\begin{aligned}\max_{\alpha^+,\alpha^-,\mu^+,\mu^-} \min_{w,b,\xi^+,\xi^-} \mathcal{W}(w,b,\xi^+,\xi^-,\alpha^+,\alpha^-,\mu^+,\mu^-) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n(\xi_i^+ + \xi_i^-) - \sum_{i=1}^n(\mu_i^+\xi_i^+ + \mu_i^-\xi_i^-) \\ - \sum_{i=1}^n \alpha_i^+(\epsilon + \xi_i^+ + y_i' - y_i) - \sum_{i=1}^n \alpha_i^-(\epsilon + \xi_i^- - y_i' + y_i)\end{aligned} \tag{47}$$

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

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

$$\frac{\partial \mathcal{W}}{\partial b} = -\sum_{i=1}^n(\alpha_i^+ - \alpha_i^-) \Rightarrow \sum_{i=1}^n(\alpha_i^+ - \alpha_i^-) = 0 \tag{49}$$

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

$$\frac{\partial \mathcal{W}}{\partial \xi_i^-} = 0 \Rightarrow C = \alpha_i^- + \mu_i^- \tag{51}$$

Substituting 48 and 49 in, we now need to maximize $\mathcal{W}$ wrt α_i^+ and α_i^-, where $\alpha_i^+ \geq 0,\ \alpha_i^- \geq 0\ \forall_i$:

$$\max_{\alpha^+,\alpha^-} \mathcal{W}(\alpha^+,\alpha^-) = \sum_{i=1}^{n} y_i(\alpha_i^+ - \alpha_i^-) - \epsilon\sum_{i=1}^{n}(\alpha_i^+ + \alpha_i^-) - \frac{1}{2}\sum_{i,j}(\alpha_i^+ - \alpha_i^-)\langle x_i, x_j\rangle(\alpha_j^+ - \alpha_j^-) \tag{52}$$

Using $\mu_i^+ \geq 0$ and $\mu_i^- \geq 0$ together with 48 and 49 means that $\alpha_i^+ \leq C$ and $\alpha_i^- \leq C$. We therefore need to find:

$$\begin{aligned} \min_{\alpha^+,\alpha^-} \quad & \frac{1}{2}(\alpha^+ - \alpha^-)^T K(\alpha^+ - \alpha^-) + \epsilon q^T(\alpha^+ + \alpha^-) - y^T(\alpha^+ - \alpha^-) \\ \text{subject to} \quad & 0 \leq \alpha_i^+, \alpha_i^- \leq C\ \forall_i \\ & q^T(\alpha^+ - \alpha^-) = 0 \end{aligned} \tag{53}$$

where $q^T = [1, \dots, 1]$.
We can write the 53 in a standard quadratic form as:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T Q\alpha - q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C\ \forall_i \\ & e^T\alpha = 0 \end{aligned} \tag{54}$$

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:

$$y' = \sum_{i=1}^{n}(\alpha_i^+ - \alpha_i^-)\langle x_i, x'\rangle + b \tag{55}$$

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:

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

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

$$b = \frac{1}{N_s}\sum_{s \in S} y_s - \epsilon - \sum_{m \in S}(\alpha_m^+ - \alpha_m^-)\langle x_m, x_s\rangle \tag{57}$$

From 54 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 12 or 13 where the bias term b is embedded into the weight vector w:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\alpha^T(Q + ee^T)\alpha + q^T\alpha \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C\ \forall_i \end{aligned} \tag{58}$$

4.2.2 Lagrangian Dual

In order to relax the constraints in the *Wolfe dual* formulation 53 we define the problem as a *Lagrangian dual* relaxation by embedding them into objective function, so we need to allocate the Lagrangian multipliers

$\mu \geq 0, \lambda_+ \geq 0, \; \lambda_- \geq 0$:

$$\begin{aligned}\max_{\mu,\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha,\mu,\lambda_+,\lambda_-) &= \frac{1}{2}\alpha^T Q\alpha + q^T\alpha - \mu^T(e^T\alpha) - \lambda_+^T(u-\alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T Q\alpha + (q - \mu e + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u\end{aligned} \tag{59}$$

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:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow Q\alpha + (q - \mu e + \lambda_+ - \lambda_-) = 0 \tag{60}$$

With α optimal solution of the linear system:

$$Q\alpha = -(q - \mu e + \lambda_+ - \lambda_-) \tag{61}$$

the gradient wrt μ, λ_+ and λ_- are:

$$\frac{\partial \mathcal{L}}{\partial \mu} = -e\alpha \tag{62}$$

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

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

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:

$$\begin{aligned}\min_{\alpha \in K_n(Q,b)} \quad & \|Q\alpha - b\| \\ \text{where} \quad & b = -(q - \mu e + \lambda_+ - \lambda_-)\end{aligned} \tag{65}$$

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

From 54 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 59 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 58.

$$\begin{aligned}\max_{\lambda_+,\lambda_-} \min_{\alpha} \mathcal{L}(\alpha,\lambda_+,\lambda_-) &= \frac{1}{2}\alpha^T (Q + ee^T)\alpha + q^T\alpha - \lambda_+^T(u-\alpha) - \lambda_-^T\alpha \\ &= \frac{1}{2}\alpha^T (Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-)^T\alpha - \lambda_+^T u\end{aligned} \tag{66}$$

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:

$$\frac{\partial \mathcal{L}}{\partial \alpha} = 0 \Rightarrow (Q + ee^T)\alpha + (q + \lambda_+ - \lambda_-) = 0 \tag{67}$$

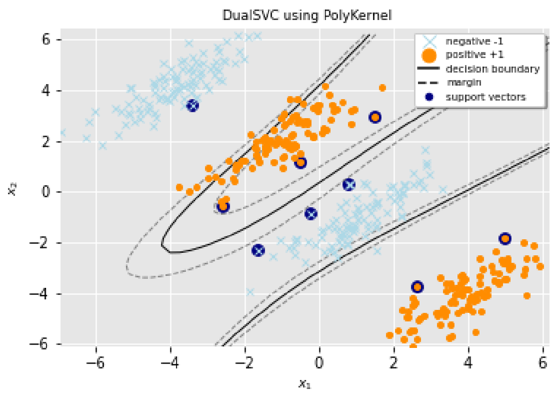
With α optimal solution of the linear system:

$$(Q + ee^T)\alpha = -(q + \lambda_+ - \lambda_-) \tag{68}$$

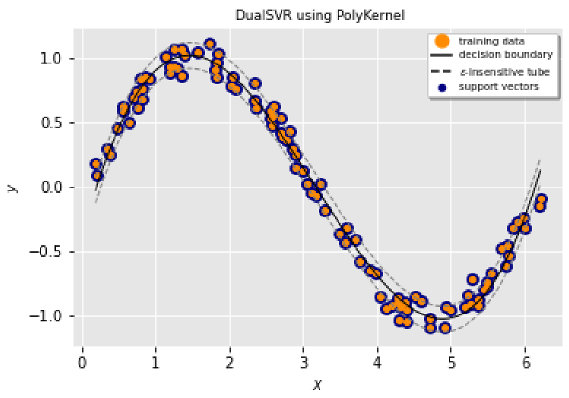
the gradient wrt λ_+ and λ_- are:

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

$$\frac{\partial \mathcal{L}}{\partial \lambda_-} = -\alpha \tag{70}$$



(a) Polynomial SVC hyperplane



(b) Polynomial SVR hyperplane

Figure 7: Polynomial SVM hyperplanes

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 not linearly separable/regressable in the space of the inputs x, 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’).

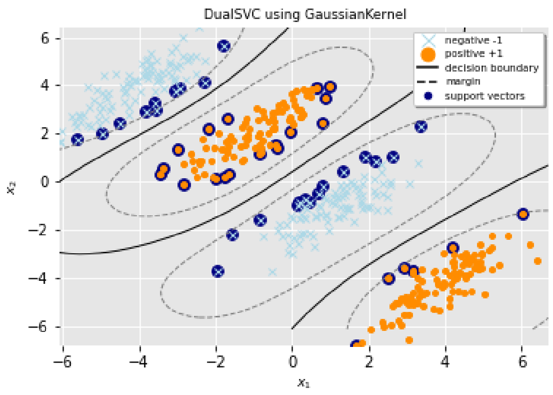
5.2 Gaussian kernel

The *gaussian* kernel is defined as:

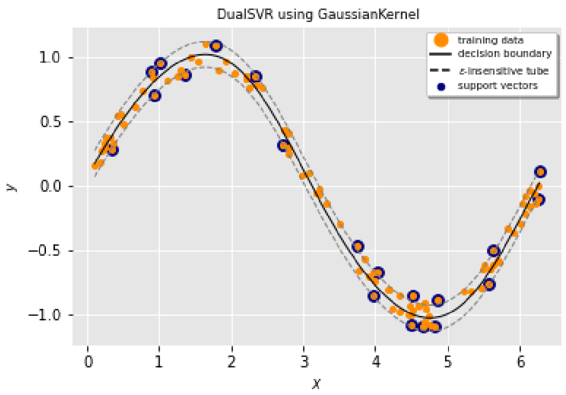
$$k(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2}) \tag{75}$$

or, equivalently, as:

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



(a) Gaussian SVC hyperplane



(b) Gaussian SVR hyperplane

Figure 8: Gaussian SVM hyperplanes

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').

6 Stochastic Gradient Descent

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.

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 (KKT)* conditions are necessary and sufficient conditions for optimality of a positive definite QP problem and the KKT conditions for the problem 19 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

9.1 Support Vector Classifier

9.1.1 Hinge loss

Primal formulation etc

solver	C	fit_time	train_accuracy	val_accuracy	n_iter	nr_train_sv	nr_val_sv
adagrad	1	0.151886	0.972506	0.975049	73	15	8
liblinear	1	0.001718	0.985000	0.974974	125	12	7
adagrad	10	0.300753	0.980025	0.970074	229	10	5
liblinear	10	0.002889	0.982494	0.979949	443	8	5
adagrad	100	0.374231	0.977518	0.975049	491	9	4
liblinear	100	0.003613	0.985000	0.984999	829	6	4

Dual formulations

Linear Wolfe etc

solver	C	fit_time	train_accuracy	val_accuracy	nr_train_sv	nr_val_sv
cvxopt	1	0.063277	0.985019	0.980100	11	11
libsvm	1	0.003978	0.980025	0.969998	13	13
smo	1	0.194799	0.985019	0.980100	11	11
cvxopt	10	0.050291	0.987506	0.980100	7	7
libsvm	10	0.004888	0.980006	0.974974	9	9
smo	10	0.243180	0.987506	0.975049	6	6
cvxopt	100	0.023014	0.990012	0.980100	6	6
libsvm	100	0.006158	0.980006	0.969998	8	8
smo	100	0.600229	0.985000	0.975049	6	6

Linear Lagrangian etc

ld	C	fit_time	train_accuracy	val_accuracy	nr_train_sv	nr_val_sv
bcqp	1	0.018653	0.992481	0.994949	127	127
qp	1	0.013991	0.974993	0.980024	131	131
bcqp	10	0.018454	0.992481	0.994949	127	127
qp	10	0.013773	0.974993	0.980024	131	131
bcqp	100	0.018384	0.992481	0.994949	127	127
qp	100	0.016275	0.974993	0.980024	131	131

Nonlinear Wolfe etc

solver	kernel	C	fit_time	train_accuracy	val_accuracy	nr_train_sv	nr_val_sv
cvxopt	poly	1	0.224446	0.878657	0.696293	25	25
libsvm	poly	1	0.008651	1.000000	0.997494	24	24
smo	poly	1	1.467505	0.881154	0.691318	25	25
cvxopt	rbf	1	0.076681	1.000000	1.000000	42	42
libsvm	rbf	1	0.007161	1.000000	1.000000	40	40
smo	rbf	1	0.396867	1.000000	1.000000	42	42
cvxopt	poly	10	0.091701	0.884965	0.728482	10	10
libsvm	poly	10	0.008077	1.000000	0.997494	9	9
smo	poly	10	1.106135	0.886218	0.725975	10	10
cvxopt	rbf	10	0.085812	1.000000	1.000000	15	15
libsvm	rbf	10	0.007587	1.000000	1.000000	13	13
smo	rbf	10	0.259829	1.000000	1.000000	15	15
cvxopt	poly	100	0.092667	0.966259	0.920323	8	8
libsvm	poly	100	0.008358	1.000000	0.997494	8	8
smo	poly	100	0.919512	0.966259	0.917817	8	8
cvxopt	rbf	100	0.081729	1.000000	1.000000	12	12
libsvm	rbf	100	0.005648	1.000000	1.000000	11	11
smo	rbf	100	0.235654	1.000000	1.000000	12	12

Nonlinear Lagrangian etc

ld	kernel	C	fit_time	train_accuracy	val_accuracy	nr_train_sv	nr_val_sv
bcqp	poly	1	0.078685	0.750007	0.501253	217	217
qp	poly	1	0.737430	0.872504	0.750627	138	138
bcqp	rbf	1	0.028115	1.000000	0.997512	241	241
qp	rbf	1	1.608392	0.800071	0.635656	188	188
bcqp	poly	10	0.073986	0.750007	0.501253	217	217
qp	poly	10	0.741292	0.872504	0.750627	138	138
bcqp	rbf	10	0.021263	1.000000	0.997512	241	241
qp	rbf	10	1.405860	0.857500	0.718400	199	199
bcqp	poly	100	0.063981	0.750007	0.501253	217	217
qp	poly	100	0.571038	0.872504	0.750627	138	138
bcqp	rbf	100	0.025343	1.000000	0.997512	241	241
qp	rbf	100	0.761562	0.782584	0.608218	154	154

9.1.2 Squared Hinge loss

Primal formulation etc

solver	C	fit_time	train_accuracy	val_accuracy	n_iter	nr_train_sv	nr_val_sv
liblinear	1	0.001661	0.980006	0.980024	368	15	8
sgd	1	0.265811	0.985000	0.985075	196	25	13
liblinear	10	0.001666	0.980006	0.980024	1000	10	5
sgd	10	0.215871	0.987525	0.985075	155	12	6
liblinear	100	0.001558	0.985019	0.980024	1000	12	5
sgd	100	0.095087	0.987525	0.980024	27	7	4

9.2 Support Vector Regression

9.2.1 Epsilon-insensitive loss

Primal formulation etc

solver	C	epsilon	fit_time	train_r2	val_r2	n_iter	nr_train_sv	nr_val_sv
adagrad	1	0.1	0.743824	0.919139	0.915610	872	66	33
liblinear	1	0.1	0.000909	0.918828	0.916845	12	66	33
adagrad	10	0.1	2.723583	0.977832	0.972861	3546	65	32
liblinear	10	0.1	0.001047	0.977848	0.972083	122	66	33
adagrad	100	0.1	3.106563	0.978115	0.974270	3999	66	32
liblinear	100	0.1	0.001624	0.977723	0.974270	735	65	33
adagrad	1	0.2	0.738437	0.919988	0.916497	885	66	33
liblinear	1	0.2	0.000924	0.918753	0.916601	13	65	32
adagrad	10	0.2	2.604949	0.977798	0.972835	3514	65	32
liblinear	10	0.2	0.001276	0.977852	0.972041	164	64	33
adagrad	100	0.2	3.065408	0.978128	0.974186	3999	66	32
liblinear	100	0.2	0.001437	0.977641	0.973867	693	66	33
adagrad	1	0.3	0.698630	0.920124	0.916702	878	65	33
liblinear	1	0.3	0.001215	0.919287	0.917030	10	66	32
adagrad	10	0.3	2.715705	0.977781	0.972876	3458	65	32
liblinear	10	0.3	0.001184	0.977871	0.972149	129	63	33
adagrad	100	0.3	2.610475	0.978122	0.974216	3999	66	32
liblinear	100	0.3	0.001536	0.977641	0.973903	808	65	33

Dual formulations

Linear Wolfe etc

solver	C	epsilon	fit_time	train_r2	val_r2	nr_train_sv	nr_val_sv
cvxopt	1	0.1	0.078818	0.917772	0.914479	67	67
libsvm	1	0.1	0.001410	0.917627	0.915448	66	66
smo	1	0.1	0.062739	0.917773	0.914442	66	66
cvxopt	10	0.1	0.028650	0.977920	0.972466	67	67
libsvm	10	0.1	0.001434	0.977852	0.972051	66	66
smo	10	0.1	0.117902	0.977920	0.972445	66	66
cvxopt	100	0.1	0.014835	0.977788	0.974150	67	67
libsvm	100	0.1	0.002421	0.977723	0.974270	66	66
smo	100	0.1	0.612424	0.977788	0.974139	66	66
cvxopt	1	0.2	0.083481	0.918341	0.915058	67	67
libsvm	1	0.2	0.000902	0.918194	0.915985	66	66
smo	1	0.2	0.070250	0.918341	0.915019	66	66
cvxopt	10	0.2	0.022306	0.977926	0.972474	67	67
libsvm	10	0.2	0.001295	0.977851	0.972025	65	65
smo	10	0.2	0.185026	0.977926	0.972457	65	65
cvxopt	100	0.2	0.014931	0.977742	0.974033	67	67
libsvm	100	0.2	0.002740	0.977673	0.974122	66	66
smo	100	0.2	0.342281	0.977742	0.974022	66	66
cvxopt	1	0.3	0.072449	0.918942	0.915614	66	66
libsvm	1	0.3	0.001214	0.918786	0.916554	66	66
smo	1	0.3	0.090543	0.918942	0.915576	66	66
cvxopt	10	0.3	0.013097	0.977954	0.972562	66	66
libsvm	10	0.3	0.001567	0.977870	0.972135	65	65
smo	10	0.3	0.069342	0.977953	0.972544	65	65
cvxopt	100	0.3	0.012269	0.977737	0.973956	67	67
libsvm	100	0.3	0.002715	0.977655	0.974045	66	66
smo	100	0.3	0.487434	0.977737	0.973939	66	66

Linear Lagrangian etc

ld	C	epsilon	fit_time	train_r2	val_r2	nr_train_sv	nr_val_sv
bcqp	1	0.1	0.840952	0.731073	0.721200	67	67
qp	1	0.1	1.064520	0.876534	0.870926	67	67
bcqp	10	0.1	0.849611	0.733638	0.723925	67	67
qp	10	0.1	0.817001	0.731825	0.722021	67	67
bcqp	100	0.1	0.698911	0.733638	0.723925	67	67
qp	100	0.1	0.690645	0.731825	0.722021	67	67
bcqp	1	0.2	0.880100	0.731073	0.721199	67	67
qp	1	0.2	1.118590	0.876534	0.870927	67	67
bcqp	10	0.2	0.778322	0.733638	0.723924	67	67
qp	10	0.2	0.758636	0.731825	0.722021	67	67
bcqp	100	0.2	0.695293	0.733638	0.723924	67	67
qp	100	0.2	0.608646	0.731825	0.722021	67	67
bcqp	1	0.3	0.884336	0.731073	0.721199	67	67
qp	1	0.3	0.981236	0.876534	0.870927	67	67
bcqp	10	0.3	0.762222	0.733638	0.723924	67	67
qp	10	0.3	0.758262	0.731825	0.722020	67	67
bcqp	100	0.3	0.647260	0.733638	0.723924	67	67
qp	100	0.3	0.455725	0.731825	0.722020	67	67

Nonlinear Wolfe etc

solver	kernel	C	epsilon	fit_time	train_r2	val_r2	nr_train_sv	nr_val_sv
cvxopt	poly	1	0.1	0.020200	0.912871	-11.755067	25	25
libsvm	poly	1	0.1	0.048495	0.969950	-31.639597	26	26
smo	poly	1	0.1	89.065522	0.912118	-12.565930	26	26
cvxopt	rbf	1	0.1	0.023099	0.981651	-0.414322	14	14
libsvm	rbf	1	0.1	0.002932	0.981854	-1.495513	16	16
smo	rbf	1	0.1	0.051685	0.981312	-0.523270	14	14
cvxopt	poly	10	0.1	0.012495	0.306857	-6.842633	25	25
libsvm	poly	10	0.1	0.271278	0.974545	-19.141421	24	24
smo	poly	10	0.1	388.475776	0.638964	-8.268856	24	24
cvxopt	rbf	10	0.1	0.018509	0.985561	0.108792	13	13
libsvm	rbf	10	0.1	0.003479	0.983351	-1.396791	15	15
smo	rbf	10	0.1	0.229888	0.985327	0.147453	13	13
cvxopt	poly	100	0.1	0.015051	0.869438	-15.010219	45	45
libsvm	poly	100	0.1	1.376607	0.974419	-17.948798	24	24
smo	poly	100	0.1	2302.380157	0.621400	-7.760303	25	25
cvxopt	rbf	100	0.1	0.013123	0.982875	0.213034	19	19
libsvm	rbf	100	0.1	0.003680	0.982791	-1.524307	17	17
smo	rbf	100	0.1	1.025222	0.984668	0.197268	13	13
cvxopt	poly	1	0.2	0.016242	-1.378366	-12.903744	6	6
libsvm	poly	1	0.2	0.009272	0.949658	-77.766723	9	9
smo	poly	1	0.2	2.303559	-3.937504	-24.237531	6	6
cvxopt	rbf	1	0.2	0.018508	0.968397	-1.008175	5	5
libsvm	rbf	1	0.2	0.008502	0.954108	-1.783851	6	6
smo	rbf	1	0.2	0.026222	0.963877	-1.133633	5	5
cvxopt	poly	10	0.2	0.013106	-0.870729	-12.712712	4	4
libsvm	poly	10	0.2	0.012017	0.959396	-76.201228	4	4
smo	poly	10	0.2	1.930811	-3.426887	-24.039817	4	4
cvxopt	rbf	10	0.2	0.015264	0.967099	-1.006287	5	5
libsvm	rbf	10	0.2	0.001138	0.955676	-1.791806	5	5
smo	rbf	10	0.2	0.026772	0.955168	-1.139310	4	4
cvxopt	poly	100	0.2	0.015552	-0.866842	-12.690245	4	4
libsvm	poly	100	0.2	0.012390	0.960116	-76.187402	4	4
smo	poly	100	0.2	1.897000	-3.426887	-24.039817	4	4
cvxopt	rbf	100	0.2	0.012336	0.959724	-1.036271	5	5
libsvm	rbf	100	0.2	0.000955	0.955676	-1.791806	5	5
smo	rbf	100	0.2	0.022408	0.955168	-1.139310	4	4
cvxopt	poly	1	0.3	0.011968	-0.947829	-62.996809	4	4
libsvm	poly	1	0.3	0.004746	0.899052	-110.044559	7	7
smo	poly	1	0.3	3.685649	-3.302843	-67.383739	4	4
cvxopt	rbf	1	0.3	0.021200	0.932699	-1.538800	5	5
libsvm	rbf	1	0.3	0.009246	0.896824	-2.223010	4	4
smo	rbf	1	0.3	0.015314	0.925682	-1.696438	5	5
cvxopt	poly	10	0.3	0.013138	-0.984655	-63.093005	3	3
libsvm	poly	10	0.3	0.005744	0.911641	-109.999548	3	3
smo	poly	10	0.3	2.395583	-3.337588	-67.521683	3	3
cvxopt	rbf	10	0.3	0.014347	0.922362	-1.547447	5	5
libsvm	rbf	10	0.3	0.001121	0.898804	-2.209810	4	4
smo	rbf	10	0.3	0.012778	0.911616	-1.712181	4	4
cvxopt	poly	100	0.3	0.014906	-0.984706	-63.094752	3	3
libsvm	poly	100	0.3	0.006325	0.911641	-109.999548	3	3
smo	poly	100	0.3	2.170626	-3.337588	-67.521683	3	3
cvxopt	rbf	100	0.3	0.015353	0.919987	-1.493444	4	4
libsvm	rbf	100	0.3	0.000902	0.898804	-2.209810	4	4
smo	rbf	100	0.3	0.012705	0.911616	-1.712181	4	4

Nonlinear Lagrangian etc

ld	kernel	C	epsilon	fit_time	train_r2	val_r2	nr_train_sv	nr_val_sv
bcqp	poly	1	0.1	0.023057	0.639114	-36.677747	67	67
qp	poly	1	0.1	0.020220	0.646376	-11.830936	67	67
bcqp	rbf	1	0.1	0.064591	0.733892	-3.633330	67	67
qp	rbf	1	0.1	0.262468	0.705219	-4.814520	67	67
bcqp	poly	10	0.1	0.023814	0.639114	-36.677747	67	67
qp	poly	10	0.1	0.022628	0.646376	-11.830936	67	67
bcqp	rbf	10	0.1	0.060908	0.733892	-3.633330	67	67
qp	rbf	10	0.1	0.111514	0.683448	-5.253019	67	67
bcqp	poly	100	0.1	0.021451	0.639114	-36.677747	67	67
qp	poly	100	0.1	0.019278	0.646376	-11.830936	67	67
bcqp	rbf	100	0.1	0.049753	0.733892	-3.633330	67	67
qp	rbf	100	0.1	0.113938	0.683448	-5.253019	67	67
bcqp	poly	1	0.2	0.028676	0.617963	-26.867830	66	66
qp	poly	1	0.2	0.060558	0.646709	-11.845840	67	67
bcqp	rbf	1	0.2	0.158712	0.644671	-4.372948	67	67
qp	rbf	1	0.2	0.317687	0.697766	-4.973421	67	67
bcqp	poly	10	0.2	0.023270	0.617963	-26.867830	66	66
qp	poly	10	0.2	0.048967	0.646709	-11.845840	67	67
bcqp	rbf	10	0.2	0.155017	0.644671	-4.372948	67	67
qp	rbf	10	0.2	0.167689	0.664793	-5.391712	67	67
bcqp	poly	100	0.2	0.022938	0.617963	-26.867830	66	66
qp	poly	100	0.2	0.052707	0.646709	-11.845840	67	67
bcqp	rbf	100	0.2	0.127478	0.644671	-4.372948	67	67
qp	rbf	100	0.2	0.154184	0.664793	-5.391712	67	67
bcqp	poly	1	0.3	0.060247	0.591564	-26.749052	66	66
qp	poly	1	0.3	0.059887	0.623022	-11.794656	67	67
bcqp	rbf	1	0.3	0.252759	0.549688	-5.236443	67	67
qp	rbf	1	0.3	0.456361	0.683198	-5.011083	67	67
bcqp	poly	10	0.3	0.049147	0.591564	-26.749052	66	66
qp	poly	10	0.3	0.056914	0.623022	-11.794656	67	67
bcqp	rbf	10	0.3	0.231289	0.549688	-5.236443	67	67
qp	rbf	10	0.3	0.221825	0.672122	-5.178610	67	67
bcqp	poly	100	0.3	0.043412	0.591564	-26.749052	66	66
qp	poly	100	0.3	0.079256	0.623022	-11.794656	67	67
bcqp	rbf	100	0.3	0.180342	0.549688	-5.236443	67	67
qp	rbf	100	0.3	0.157599	0.672122	-5.178610	67	67

9.2.2 Squared Epsilon-insensitive loss

Primal formulation etc

solver	C	epsilon	fit_time	train_r2	val_r2	n_iter	nr_train_sv	nr_val_sv
liblinear	1	0.1	0.001047	0.978134	0.974000	88	67	32
sgd	1	0.1	0.431650	0.978126	0.973976	351	66	32
liblinear	10	0.1	0.003412	0.978183	0.973971	768	66	33
sgd	10	0.1	0.059331	0.978184	0.973958	47	66	33
liblinear	100	0.1	0.005186	0.976943	0.973781	1000	66	33
sgd	100	0.1	0.007187	-18.813057	-18.646387	5	67	33
liblinear	1	0.2	0.001002	0.978132	0.974006	85	66	32
sgd	1	0.2	0.502981	0.978125	0.973973	348	66	32
liblinear	10	0.2	0.003714	0.978183	0.973965	776	66	33
sgd	10	0.2	0.049581	0.978184	0.973957	45	66	33
liblinear	100	0.2	0.004780	0.978119	0.973252	1000	66	33
sgd	100	0.2	0.011757	-18.899196	-18.743996	5	67	33
liblinear	1	0.3	0.000923	0.978130	0.974015	87	66	32
sgd	1	0.3	0.490674	0.978125	0.973972	345	66	32
liblinear	10	0.3	0.003632	0.978183	0.973975	758	66	32
sgd	10	0.3	0.062641	0.978183	0.973955	44	66	33
liblinear	100	0.3	0.004555	0.977854	0.974064	1000	65	32
sgd	100	0.3	0.009181	-18.782077	-18.637810	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.