Support Vector Machines (SVM): Theory and Algorithms

Support Vector Classification

Support Vector Machines (SVM) are a classification method capable of dealing with high dimensional non-linear data. SVMs belong to the class of algorithms known as *kernel methods*, since they depend on data only through dot-products. Therefore, they can be replaced by kernel functions, that compute these products in another feature space, different from the input space, and possibly with higher dimension. This presents two main advantages:

1. The ability to generate non-linear decision functions using methods designed for linear classifiers.
2. The classifier can be applied to data that do not have a representation in a fixed dimension space.

Hard-margin SVC

Given a training set $\{\mathbf{x}_i, y_i\}$, i = 1, . . . , n, $y_i \in \{-1, 1\}$, $x_i \in R^d$, let's assume that there is an hyperplane that separates positive samples from the negative ones. The points x that are on the hyperplane satisfy $\mathbf{w}^T\mathbf{x} + b = 0$, where $\mathbf{w}$ is normal to the hyperplane, $|b|/||\mathbf{w}||_2$ is the distance perpendicular from the hyperplane to the origin and $||\mathbf{w}||_2$ is the Euclidean norm of $\mathbf{w}$. Let d_+ (d_-) be the distance from the hyperplane to the closest positive (negative) point. Then, we define the margin of the hyperplane as $d_+ + d_-$. For the linearly separable case, the SVM simply tries to find the maximum-margin separating hyperplane. This can be formulated more formally as follows: assume that the points satisfy the restrictions.

$$\mathbf{x}_i^T\mathbf{w} + b \geq +1 \quad \text{para} \quad y_i = +1 \tag{1}$$

$$\mathbf{x}_i^T\mathbf{w} + b \leq -1 \quad \text{para} \quad y_i = -1 \tag{2}$$

These can be combined in

$$y_i(\mathbf{x}_i^T\mathbf{w} + b) - 1 \geq 0 \quad \forall i, \tag{3}$$

The points that satisfy the equality constraint in Eq. (1) are on the hyperplane $\mathbf{x}_i^T\mathbf{w} + b = 1$ (see Fig 1) and the points that satisfy the equality constraint in Eq. (2) are on the hyperplane $\mathbf{x}_i^T\mathbf{w} + b = -1$.

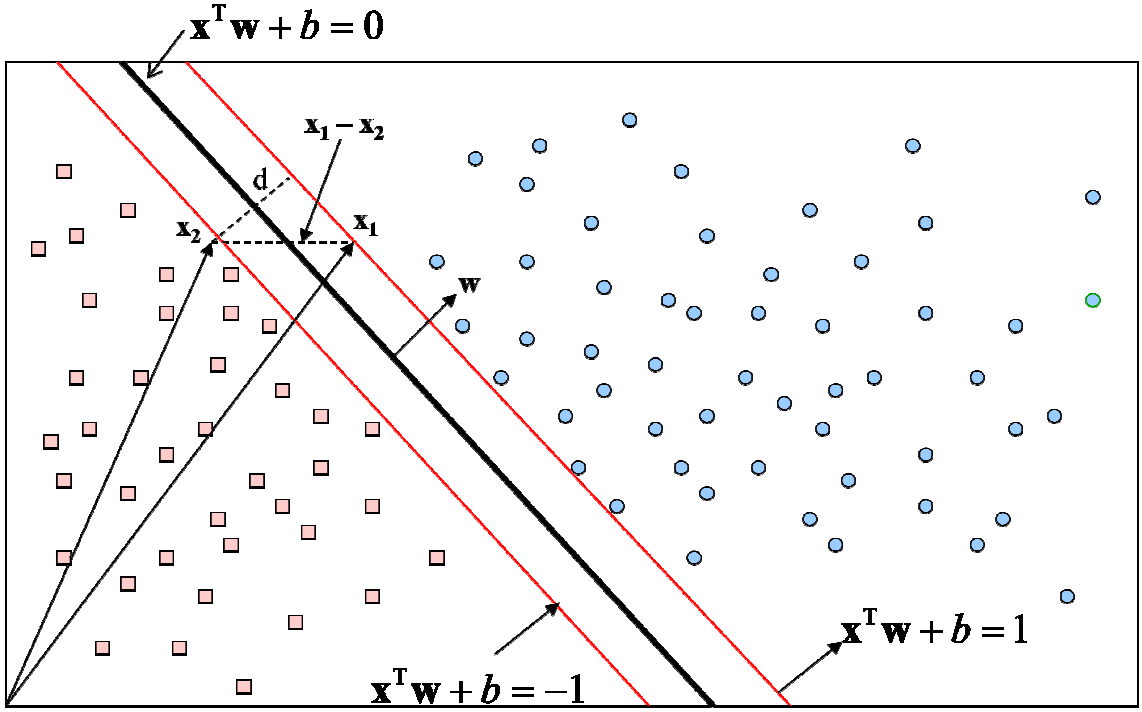

Fig 1.- Decision border and margin of SVM models

If the data is linearly separable, these hyperplanes separate the data perfectly and there are no points between them. The distance from this hyperplanes to $\mathbf{x}_i^T\mathbf{w}+b=0$ is $d_+ = d_- = 1/\|\mathbf{w}\|_2$ and the margin defined above is simply $12/\|\mathbf{w}\|_2$. Thus, the maximum margin hyperplane can be found by minimizing $\|\mathbf{w}\|_2$ subject to the constraints (3)

$$\begin{aligned} &\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 \\ &s.t. \quad y_i(\mathbf{x}_i^T\mathbf{w}+b)-1 \geq 0 \quad \forall i, \end{aligned} \tag{4}$$

where the 1/2 is introduced for convenience.

Definition 1 (Primal optimization problem). Given functions f, g_i, i = 1, . . . , k, and h_i, i = 1, . . . ,m, defined over a domain $\Omega \subseteq \mathbb{R}^n$, the primal problem solves

$$\begin{aligned} &\min \ f(\mathbf{w}), && \mathbf{w} \in \Omega \\ &\text{s.t. } g_i(\mathbf{w}) \leq 0, && i=1,...,k, \\ &\quad h_i(\mathbf{w}) = 0, && i=1,...,m, \end{aligned}$$

where $f(\mathbf{w})$ is known as *objective function*, $g_i(\mathbf{w})$ are the inequality constraints and $h_i(\mathbf{w})$ are the equality constraints. From now on we are going to write $g(\mathbf{w}) \leq 0$ to indicate $g_i(\mathbf{w}) \leq 0$, i = 1, . . . , k, and the same with $h(\mathbf{w})$.

Definition 2 (Lagrangian function). Given the optimization problem before we define the generalized Lagrangian function as

$$L(\mathbf{w},\boldsymbol{\alpha},\boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1} \beta_i h_i(\mathbf{w}) =$$
$$= f(\mathbf{w}) + \boldsymbol{\alpha}^{\mathbf{T}} g(\mathbf{w}) + \boldsymbol{\beta}^{\mathbf{T}} h(\mathbf{w})$$

We can now define the Lagrangian dual problem

Definition 3 (Dual optimization problem). The Lagrangian dual problem of the primal problem of Definition 1 is the following problem:

$$\begin{aligned} &\max \quad \Theta(\boldsymbol{\alpha},\boldsymbol{\beta}) \\ &s.t. \quad \alpha_i \geq 0 \ \ \forall \mathrm{i} \end{aligned}$$

where $\Theta(\boldsymbol{\alpha},\boldsymbol{\beta}) = \inf_w L(\mathbf{w},\boldsymbol{\alpha},\boldsymbol{\beta})$

Theorem 1. (Weak duality theorem) Let $\mathbf{w} \in \Omega \subset \mathbb{R}^n$ be a feasible solution of the primal problem (2.5) and (α, β) a feasible solution of the dual problem. Then $f(\mathbf{w}) \geq \Theta(\boldsymbol{\alpha},\boldsymbol{\beta})$

Now let's transform the previous problem into the dual equivalent. The objective function (4) is clearly convex and the restrictions are affine, therefore the Lagrangian of the problem is

$$L_P = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{n} \alpha_i y_i (\mathbf{x}_i^{\mathrm{T}} \mathbf{w} + b) + \sum_{i=1}^{n} \alpha_i \tag{5}$$

with $\alpha_i \geq 0, \ \ \forall \mathrm{i}$. The dual problem is thus defined as

$$\max_{\boldsymbol{\alpha}} \left\{ \min_{\mathbf{w},b} L_P \right\} \quad \text{s.t. } \boldsymbol{\alpha} \geq \mathbf{0} \tag{6}$$

Next we have to minimize L_P with respect to **w** and *b*, requiring also that the derivatives of L_P with respect to α_i are 0 and everything under the constraints $\alpha_i \geq 0$ The derivatives of L_P with respect to **w** and *b* are

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{7}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i \tag{8}$$

As we can see in the following Theorem 2

Theorem 2 (Karum-Khun-Tucker, KKT) Given an optimization problem in the convex domain $\Omega \subseteq \mathbb{R}^n$

$$\begin{aligned} &\min\ f(\mathbf{w}), && \mathbf{w} \in \Omega \\ &\text{s.t. } g_i(\mathbf{w}) \le 0, && i=1,...,k, \\ &\quad h_i(\mathbf{w}) = 0, && i=1,...,m, \end{aligned}$$

with convex f, affine g_i, h_i, (that is, allowing for or preserving parallel relationships), the necessary and sufficient conditions for a normal point $\mathbf{w}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*$ and $\boldsymbol{\beta}^*$ such that

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} = 0,$$

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} = 0,$$

$$\begin{aligned} &\alpha_i^* g_i(\mathbf{w}^*) = 0, && i=1,...,k, \\ &g_i(\mathbf{w}^*) \le 0, && i=1,...,k, \\ &\alpha_i^* \ge 0, && i=1,...,k, \end{aligned}$$

Thus, the optimum ($\mathbf{w}^*$, b^*) must satisfy the following KKT conditions:

$$\mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \mathbf{0} \tag{9}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10}$$

$$y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1 \ge 0 \quad \forall i, \tag{11}$$

$$\alpha_i \ge 0 \qquad \forall i, \tag{12}$$

$$\alpha_i (y_i(\mathbf{x}_i^T \mathbf{w} + b) - 1) = 0 \quad \forall i, \tag{13}$$

Equation (13) is called the Karum, Khun Ticher, **KKT** dual complementary condition, and it will be important later to show that the optimal solution of the SVC problem depends only on the coefficients $\alpha_i^* > 0$, which are called the **support vectors**. Substituting Eqs. (7) and (8) in Eq. (5) we get.

$$L_P = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{14}$$

Therefore the new dual problem is defined as

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\alpha_i\alpha_j \, \mathrm{y_i y_j} \, \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{x}_j \\ &s.t. \ \ \alpha_i \geq 0 \quad \forall \mathrm{i}, \\ &\sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{15}$$

The dual problem is much easier to solve since the constraints are simpler. Note that Eq. (9) relates the solutions of the primal and dual problem. Therefore, if we were to solve the dual formulation we could recover the optimal primal weights as

$$\mathbf{w}^* = \sum_{i=1}^{n}\alpha_i^* y_i \mathbf{x}_i$$

where $\boldsymbol{\alpha}^* \in R^n$ is the solution of problem (15). Similarly, Eq. (13) implies that,

$$y_i(\mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{w} + b^*) - 1 = 0$$

for any *i* such that $\alpha_i^* > 0$. From that we can obtain the optimal value of the bias as

$$b^* = \frac{1}{y_i} - \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{w}^* = y_i - \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{w}^*,$$

since $y_i \in \{-1, 1\}$. Although the previous equation should yield the same value for every support vector *i*, in practice it is recommended for numerical stability to compute the bias as the average among all of them,

$$b^* = \frac{1}{|S|}\sum_{i\in S} y_i - \mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{w}^*,$$

where $S = \{\hat{\imath} \mid \alpha_i^* > 0\}$, that is, the set of indices of the support vectors.

Now, we introduce the most common case in practice, that is, when data is not linearly separable.

Soft-margin SVC

In order to allow classification errors when data is not linearly separable, we replace the constraints (3) by

$$y_i(\mathbf{x}_\mathrm{i}^\mathrm{T}\mathbf{w} + b) \geq 1 - \xi_i \quad \forall \mathrm{i}, \tag{16}$$

where $\xi_i \geq 0$ are slack variables that make possible for a point to be inside the margin $0 \leq \xi_i \leq 1$ or to be misclassified ($\xi_i > 1$). Since a data point is wrongly classified if the value of its slack variable is larger than 1, $\sum_{i=1}^{n}\xi_i$ is an upper bound on the number of

classification errors. The new goal is to maximize the margin i.e. minimize $\|\mathbf{w}\|_2^2$ but penalizing classification errors with the term $C\sum_{i=1}^{n}\xi_i$. The parameter $C > 0$ controls the tradeoff between margin maximization and error minimization. The optimization problem is now

$$\begin{aligned} &\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \\ &s.t. \quad y_i(\mathbf{x}_i^{\mathrm{T}}\mathbf{w}+b) \geq 1-\xi_i \quad \forall \mathrm{i}, \\ &\qquad \xi_i \geq 0 \quad \forall \mathrm{i}, \end{aligned} \tag{17}$$

The dual formulation of this problem is obtained similarly to the linearly separable case. The Langrange dual function is

$$L_P = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i(y_i(\mathbf{x}_i^{\mathrm{T}}\mathbf{w}+b)-1+\xi_i) - \sum_{i=1}^{n}\beta_i\xi_i \tag{18}$$

where $\boldsymbol{\alpha} \geq 0 \in R^n$ and $\boldsymbol{\beta} \geq 0 \in R^n$ are the Lagrange dual variables. The KKT conditions are,

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n}\alpha_i y_i \mathbf{x}_i = 0 \tag{19}$$

$$\frac{\partial L_P}{\partial b} = -\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{20}$$

$$\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \tag{21}$$

Substituting the previous equations into the dual function yields the equivalent dual problem.

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \ \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\alpha_i\alpha_j \, \mathrm{y_i y_j} \, \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j \\ &s.t. \ 0 \leq \alpha_i \leq C \quad \forall \mathrm{i}, \\ &\sum_{i=1}^{n}\alpha_i y_i = 0 \end{aligned} \tag{22}$$

We can also write the previous problem more compactly in vector form

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2}\boldsymbol{\alpha}^{\mathbf{T}}\mathbf{Q}\boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathbf{T}}\mathbf{1} \right\}$$
$$\text{s.t. } \boldsymbol{\alpha}^{\mathbf{T}}\mathbf{y} = 0 \qquad (23)$$
$$0 \le \alpha_i \le C, \quad \forall \text{i}$$

where $\mathbf{Q} \in R^{n \times n}$ is a matrix whose entries are $Q_{ij} = y_i y_j \mathbf{x_i^T x_j}$ and $\mathbf{1} \in \text{R}^n$ is a vector of ones. Note how we transformed the maximization problem into a minimization problem just by flipping the sign of the objective function.

Kernel trick

Despite extending SVMs to deal with non-separable data, the decision function is still linear, that is, the surface that separates both classes is an hyperplane. Most problems in practice are in fact nonlinear and thus it is useful to extend the SVM formulation for this type of problems. The nonlinear SVM is based on the kernel trick explained next.

The kernel trick comes from the idea of transforming the input variables into another set of features in a higher dimensional space, where the data points are linearly separable. More formally, let $\phi(\mathbf{x}_i)$ be a function that takes a point in a *d*-dimensional space and returns another point in a *D*-dimensional space, $D >> d$. If ϕ is choosen correctly then we obtain another problem that is linearly separable in this new feature space.

However, if we try to compute explicitly the value of $\phi(\mathbf{x}_i)$, we run into two main problems, one of them practical and another one theoretical:

1. The feature space can have a very large dimension, even infinite.
2. It can be very computationally expensive to compute the mapping values every time they are needed, or even storing them in memory.

The kernel trick comes from the observation that the dual function (22) only depends on **x** through dot products, that is, they always appear in pairs. If we define the kernel function as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \qquad (24)$$

the dual problem can be rewritten as

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \alpha_i \alpha_j \, y_i y_j \, k(\mathbf{x}_i^T \mathbf{x}_j)$$
$$s.t. \;\; 0 \le \alpha_i \le C \qquad \forall i,$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{25}$$

Note that the compact formulation (23) does not change, only the definition of the matrix **Q**, $Q_{ij} = y_i y_j k(\mathbf{x_i^T x_j})$. We will refer to the matrix **Q** as **kernel matrix**.

Therefore it is not necessary to know the mapping function ϕ but only the kernel function $k(\cdot,\cdot)$. The only condition is that the kernel function must be decomposed into a dot product of two functions. Mercer's theorem states which types of kernels can be used, although in practice there are a few known kernels that met Mercer's condition and they are the most used. An example is the RBF kernel

$$k(\mathbf{x}_{i,} \mathbf{x}_j) = \exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2^2) \tag{26}$$

where γ is a hyper-parameter that represents the width of the kernel.

Last, the value of **w** in the transformed space is

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \phi(\mathbf{x}_i)$$

and usually it cannot be computed explicitly. However, a new point x can be classified using again the kernel trick

$$\mathbf{w}^T \phi(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) \tag{27}$$

Note also that only patterns with α_i>0 (**support vectors**) are needed for the classification of a new point, which is a nice property.

The **selection of the hyper-parameters** is a big problem, since the SVM performance depends greatly on their value. For the kernel SVC formulation we need to select both the complexity parameter *C* and the parameters of the kernel function. In the case of the RBF kernel we only have one additional hyper-parameter, namely, the width of the kernel .

ν-Support Vector Classification

As we mentioned before, given a particular problem, it is not straightforward to select the best value for the hyper-parameter *C*. In general, *C* controls the trade-off between

minimizing the errors and maximizing the margin. Thus, as *C* grows, the complexity of the hypothesis space also grows and the SVC becomes more intolerant to errors.

Relating this to Section 2.1.4, if *C* is too large the model could exhibit over-fitting. On the other hand, if *C* is very small the model could suffer from under-fitting. In the extreme case where $C = 0$ the soft-margin SVC reverts to the hard-margin case, and the model only cares about maximizing the margin.

In practice, the most common method to set the ***C*** **hyper-parameter** is to perform crossvalidation over a grid of values and select the one with the lowest error. However this computation is quite expensive since, for a given dataset we would have to train many models just to find a good value for *C*, with no guarantees of optimality. To alleviate this problem, Schölkopf et al. (2000) suggest a new SVC formulation which replaces the *C* parameter by $\nu \in (0, 1]$ which is in principle easier to tune. The primal optimization problem of the ν-SVC is

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\rho} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \nu\rho + \frac{1}{n}\sum_{i=1}^{n}\xi_i \\ s.t. \quad & y_i(\mathbf{w}^T\phi(\mathbf{x_i}) + b) \geq \rho - \xi_i \quad \forall \text{i}, \\ & \xi_i \geq 0 \quad \forall \text{i, i=1,...,n} \qquad \rho \geq 0 \end{aligned} \tag{28}$$

And the corresponding dual problem is (Schölkopf et al., 2000)

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^{\mathbf{T}}\mathbf{Q}\boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{\alpha}^{\mathbf{T}}\mathbf{y} = 0 \\ & \boldsymbol{\alpha}^{\mathbf{T}}\mathbf{1} \geq \nu, \\ & 0 \leq \alpha_i \leq 1/n, \quad \forall \text{i}, \quad \text{i=1,...,n} \end{aligned} \tag{29}$$

Compared to the original SVC dual (23), there are two differences. First, there is an additional constraint, involving the new hyper-parameter ν. Second, the linear term $\boldsymbol{\alpha}^{\mathbf{T}}\mathbf{1}$ has disappeared from the objective function.

Regarding the new hyper-parameter ν, Schölkopf et al. (2000) prove that it is an upper-bound on the fraction of margin errors and a lower bound on the fraction of SVs. Thus, ν has an intrinsic meaning and it is more easy to interpret its value compared to *C*. Chang and Lin (2011) also give a tighter range for the feasible values of ν,

$$\nu \leq \frac{2}{n}\min\left(\left|\{\text{i} \mid y_i = 1\}\right|, \left|\{\text{i} \mid y_i = -1\}\right|\right) \leq 1$$

so the usable range of ν in practice is actually smaller than (0, 1].

One-class Support Vector Machine

The One-class SVM was proposed by Schölkopf et al. (2001) for estimating the support of a high-dimensional distribution. Another application of this formulation is the problem of novelty detection, which is often reduced to estimating the density of the probability of the data.

Therefore the SVM classifies examples as "novel" if the density function has high probability and viceversa. The primal problem is

$$\begin{aligned} \min_{\mathbf{w},\xi,\rho} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \\ s.t. \quad & \mathbf{w}^T\phi(\mathbf{x}_i) \geq \rho - \xi_i \quad \forall \text{i}, \\ & \xi_i \geq 0 \quad \forall \text{i, i=1,...,n} \qquad \rho \geq 0 \end{aligned} \tag{30}$$

and the corresponding dual problem

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\boldsymbol{\alpha}^{\mathbf{T}}\mathbf{Q}\boldsymbol{\alpha} \\ \text{s.t.} \quad & \boldsymbol{\alpha}^{\mathbf{T}}\mathbf{1} = 1, \\ & 0 \leq \alpha_i \leq \frac{1}{\nu n}, \qquad \text{i=1,...,n} \end{aligned} \tag{31}$$

Support Vector Regression

The full kernel SVM problem (25) is a binary classfication problem, since the possible values for the targets are either −1 or +1. However, the SVM formulation can be extended in order to solve regression problems, and we are going to do so in this section.

SVMs applied to a regression problem perform a linear regression in the feature space using the **ε-insensitive loss** as cost function and, at the same time, regularizing the weights so large values are penalized. It is important to note that the feature space is different from the input space, and thus depending on the kernel the problem could be nonlinear on the input variables. The ε-insensitive loss is defined as

$$L_\varepsilon(y, f(\mathbf{x},\mathbf{w})) = \begin{cases} 0 & \text{if} \quad |y - f(\mathbf{x},\mathbf{w})| \leq \varepsilon \\ |y - f(\mathbf{x},\mathbf{w})| - \varepsilon & \text{otherwise} \end{cases} \tag{32}$$

where $f(\mathbf{x},\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}) + \mathrm{b}$ denotes the linear model in the feature space. The previous function is plotted in Fig. 1 (right)

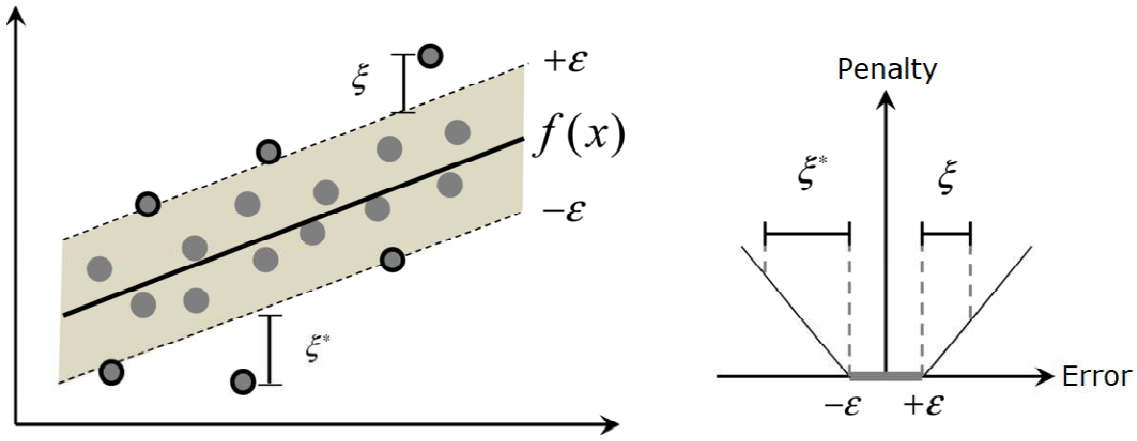

Figure 1: Linear Support Vector Regression (left) and ε-insensitive loss (right). Support vectors are drawn as with a black outline (Yu et al., 2014).

Let $\mathbf{y} \in \mathrm{R}^n$ be now continuous target values. The regression SVM solves the following optimization problem

$$\begin{aligned} \min_{\mathbf{w},b,\xi,\xi^*} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + \mathrm{C}\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ s.t. \quad & y_i - f(\mathbf{x}_i\mathbf{w}) - b \le \varepsilon + \xi_i^* \\ & f(\mathbf{x}_i\mathbf{w}) + b - y_i \le \varepsilon + \xi_i \\ & \xi_i, \xi_i^* \ge 0 \quad \forall \mathrm{i},\ \mathrm{i}=1,...,\mathrm{n} \end{aligned} \tag{33}$$

where the parameter ε controls the width of the ε-*insensitive* tube and the parameter C determines the tradeoff between model complexity and the amount of deviations larger than ε allowed.

The objective function of the ε-SVR is illustrated in Fig. 1 (left). Only the points outside the ε-insensitive tube, depicted with a black outline, are penalized and contribute to the solution of problem (33) (support vectors). This penalty is proportional to the distance to the ε-tube. On the other hand, errors in the interval $[-\varepsilon,\varepsilon]$ are ignored. It can be shown that a good value for ε has to be proportional to the input noise level, i.e. $\varepsilon \propto \sigma$ (Cherkassky and Ma, 2004).

The dual of problem (33) can be computed similarly to the classification case, arriving at

$$\begin{aligned}
&\min_{\boldsymbol{\alpha},\alpha^*} \ \frac{1}{2}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^{\mathbf{T}}\mathbf{Q}(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)+\ \varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}\mathrm{y}_i(\alpha_i-\alpha_i^*)\\
&\text{s.t.}\quad (\boldsymbol{\alpha}-\boldsymbol{\alpha}^*)^{\mathrm{T}}\mathbf{1}=0,\\
&\qquad 0\le\alpha_i,\alpha_i^*\le C, \qquad \text{i=1,...,n}
\end{aligned} \tag{34}$$

where $Q_{ij}=k(\mathbf{x}_i,\mathbf{x}_\mathbf{j})\cong\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_\mathbf{j})$ It will be useful in the following sections to rewrite the previous problem into a more compact form. Let us define

$$\tilde{\boldsymbol{\alpha}}=\begin{bmatrix}\boldsymbol{\alpha}'\\ \boldsymbol{\alpha}\end{bmatrix}\in R^{2n}\ ,\ \tilde{\mathbf{Q}}=\begin{bmatrix}\mathbf{Q} & \mathbf{-Q}\\ \mathbf{-Q} & \mathbf{Q}\end{bmatrix}\in R^{2n\times 2n}\ ,\ \mathbf{p}=\begin{bmatrix}\boldsymbol{\varepsilon}\mathbf{1_n}-\mathbf{y}\\ \boldsymbol{\varepsilon}\mathbf{1_n}+\mathbf{y}\end{bmatrix}\in R^{2n}$$

$$\tilde{\mathbf{y}}=[1,...,1,-1,...,-1]^T\in R^{2n}$$

Then it can be shown that problem (35) is equivalent to the following:

$$\begin{aligned}
&\min_{\boldsymbol{\alpha},\alpha^*} \ \frac{1}{2}\tilde{\boldsymbol{\alpha}}^{\mathbf{T}}\tilde{\mathbf{Q}}\tilde{\boldsymbol{\alpha}}+\ \mathbf{p}^{\mathrm{T}}\tilde{\boldsymbol{\alpha}}\\
&\text{s.t.}\quad \tilde{\boldsymbol{\alpha}}^{\mathbf{T}}\tilde{\mathbf{y}}=0,\\
&\qquad 0\le\alpha_i\le C, \qquad \text{i=1,...,2n}
\end{aligned} \tag{35}$$

Note how formulations (23) and (35) share a lot of similarities, simply by assigning in the first case $\mathbf{p} = \mathbf{1} \in \mathrm{R}^n$. Thus SVR can be seen as a classification problem with carefully chosen artificial labels and in an enlarged space.

Again, the selection of the hyper-parameters is even a bigger problem, since now we have three hyper-parameters to tune instead of two. One possible option is to use cross-validation over a 3D grid of (C,σ,ε) values, selecting the tuple with the lowest generalization error.

Algorithms

In this section we are going to review the main ideas to solve the SVM problem. In what follows we will usually write SVM to refer to the *C*-SVC problem, which is the most common and well known formulation. Nonetheless, many of these solvers can also be extended to the regression setting (ε-SVR) but not to other formulations like the ν-**SVM, which are only considered in LIBSVM** (Chang and Lin, 2011).

SVM solvers can be broadly classified into primal solvers and dual solvers. Historically, SVM solvers usually tackle the dual problem since the constrains are simpler and they can be extended seamlessly to the kernel setting, but they usually scale poorly with the sample size *n*. Popular dual solvers are SVM-Light (Joachims, 1998), SVM-Perf

(Joachims, 2006; Joachims and Yu, 2009), LIBLINEAR (Fan et al., 2008) and LIBSVM (Chang and Lin, 2011). If we wanted to solve the primal problem directly we could in principle absorb the constraints into the objective function, similarly to the Lasso problem. However, this only works for the linear SVM, since the primal problem of kernel SVMs involves the non-linear functions ϕ_i, which may not even be available for some common kernels like the RBF kernel.

Although the primal of the kernel SVM can not be solved in its usual formulation (36), the *representer theorem* (Kimeldorf and Wahba, 1971) allows us to re-parametrize the weights **w** and tackle the primal objective directly (Chapelle, 2007). This formulation does not depend directly on the functions ϕ_i, has no constraints and can handle kernels. Another advantage of the primal formulation is that is easy to safely stop the algorithm early to obtain an approximate solution.

However, when working with the dual there are no guarantees that an ε-optimal solution of the dual problem corresponds to an ε-optimal solution of the primal problem, and thus more sophisticated stopping conditions are needed. Some examples of primal solvers include NORMA (Kivinen et al., 2004), SGD-SVM (Bousquet and Bottou, 2008; Bottou, 2010) and Pegasos (Shalev-Shwartz et al., 2007).

In the following sections we are going present what we believe are the main algorithms to solve the SVM problem, not only in terms of popularity but also in quality of the software packages implementing them and reported efficiency. One of them is a primal solver, that works for both the linear and non-linear SVMs (Pegasos), and the other two are dual solvers, both from the same authors, one solving only the linear SVM (LIBLINEAR) and the other one solving both linear and nonlinear SVMs (LIBSVM). The former implements stochastic dual coordinate descent while the latter uses greedy dual block coordinate descent with the smallest block size possible (two coefficients). Note that, in general, even though LIBSVM solves both formulations it works especially well for the nonlinear one. Therefore, in general, either LIBLINEAR or Pegasos should be used to solve the linear SVM .

Primal gradient methods

The primal formulation of the SVM, (17) can be rewritten as an unconstrained minimization problem

$$\min_{\mathbf{w}} \quad \sum_{i=1}^{n} [1 - y_i f(x_i)]_+ \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \tag{36}$$

where $f(\mathbf{x}) = \phi(\mathbf{x})^T w + \mathrm{b}$ and the subscript "+" indicates the positive part. This has the form loss + penalty , where a regularization term is usually added to enforce some desirable properties on the solution, such us smoothness, sparsity, low-rank, and so on, changing the criterion function to

$$J(\mathbf{w}) = \underset{error}{E_D(\mathbf{w})} + \underset{regularization}{\omega(\mathbf{w})}$$

Regularization also helps to control the complexity of the problem and avoid overfitting. It can be shown that the solution to (36), with $\lambda = \frac{1}{C}$, is the same as that for (17) (Hastie et al., 2003).

The loss function L(y,f(**x**)) = $[1-yf(\mathbf{x})]_+$ is known as the **"hinge" loss** and it is a particular case of the ε-insensitive loss (32) used for SVR with $\varepsilon = 0$. Figure 4.2 shows that it is reasonable for two-class classification, when compared with the logistic regression loss function.

As we mentioned before, the previous unconstrained problem depends on the non-linear functions $\phi_i(\cdot)$, which sometimes can not be computed explicitly.

Thus, in order to **extend the primal formulation to handle kernels**, Chapelle (2007) considers first the equivalent problem

$$\min_{f \in \mathrm{H}} \quad \frac{\lambda}{2}\|f\|_{\mathrm{H}}^2 + \frac{1}{n}\sum_{i=1}^{n} L(y_i f(\mathbf{x}_i)) \tag{37}$$

where H is a reproducing kernel Hilbert space (RKHS) with associated kernel k and L is any loss function, for instance the hinge-loss. Then, using the **representer theorem** (Kimeldorf and Wahba, 1971), the previous problem can also be written as

$$\min_{\boldsymbol{\alpha}} \ \lambda \boldsymbol{\alpha}^{\mathbf{T}} \mathbf{K} \boldsymbol{\alpha} + \frac{1}{n}\sum_{i=1}^{n} L(y_i, \mathbf{K}_{\mathbf{i}}^{\mathbf{T}} \boldsymbol{\alpha}) \tag{38}$$

where ***K*** is the kernel matrix and $\mathrm{K_i}$ the *i*th column of **K**. Problem (38) is quadratic, unconstrained and no longer depends on the non-linear functions ϕ_i . However, note that the bias term *b* is dropped from the calculations for simplicity, but they can be easily modified to take it into account (Chapelle, 2007).

The previous problem can be solved by standard techniques like gradient descent, conjugate gradient or Newton's method, as long as *L* is differentiable. Since the hinge-loss is not differentiable, Chapelle (2007) considers first the squared hinge-loss

$$L(y_i f(\mathbf{x}_i)) = \max(0,1 - y_i f(\mathbf{x}_i))^p \quad (39)$$

with p = 2. This is sometimes referred as L_2-SVM, in contrast to the L_1-SVM, where p = 1

For linear SVMs the Pegasos algorithm (Shalev-Shwartz et al., 2007) is a straight application of stochastic gradient descent, which we also considered to solve the Lasso problem. Lasso (Least Absolute Shrinkage and Selection Operator) is a technique that reduces (shrinks) some coefficients and sets others to 0; therefore it tries to maintain the advantages of both subset selection and Ridge Regression. Let's assume, without loss of generality, that the x_{ij} are normalized to have zero mean and unit variance, and the output variables y_i have zero mean. Let $\hat{\mathbf{w}} = (\hat{w}_1, \hat{w}_2, ..., \hat{w}_d)^T$ be the Lasso estimate, defined by

$$\hat{\mathbf{w}} = \arg\min \left\{ \sum_{i=1}^{n} \left(y_i - \sum_j w_j x_{ij} \right)^2 \right\} \quad \text{s.t.} \sum_j |w_j| \le t,$$

where t is a tuning parameter. This parameter controls the amount of shrinking that is applied to the estimates. The Lasso is the sum of a differentiable loss function and a convex regularizer, while the SVM is the other way around, that is, we have a differentiable regularizer (ℓ_2-norm) but a non-differentiable loss (**hinge-loss**).

Algorithm 10: Pegasos for the linear SVM

Input : $\mathbf{w} = 0 \in R^d$ and λ
for $k = 1, 2, \ldots$ **do**
  Sample *i* uniformly at random from {1, . . . , *n*}
  $\eta_k = 1/(\lambda k)$
  if $y_i \mathbf{x}_i^T \mathbf{w}^k < 1$ **then**
    $\mathbf{w}^{k+1} \leftarrow (1 - \eta_k \lambda)\mathbf{w}^k + \eta_k y_i \mathbf{x}_i$
  else
    $\mathbf{w}^{k+1} \leftarrow (1 - \eta_k \lambda)\mathbf{w}^k$
  end
end

Every iteration Pegasos selects an index *i* uniformly at random and considers the following approximation of the SVC objective function:

$$f(\mathbf{x_i}, y_i) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2 + L(y_i, \mathbf{x_i^T w})$$

where $L(y_i, \mathrm{t}) = \max(0, 1 - \mathrm{y}_i \mathrm{t})$ The subgradient of the hinge-loss is

$$\partial_k L(y_i, \mathrm{t}) = \begin{cases} -y_i \mathbf{x_i} & \text{if} \quad \mathrm{y}_i t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{40}$$

and thus the subgradient of the approximate objective is

$$\partial_k f(\mathbf{x_i}, y_i) = \lambda \mathbf{w} - \mathbf{s}$$

where $\mathbf{s} \in \partial_k L$. Let $\eta_k = 1/(\lambda k)$ be a step-size; the weight updates of stochastic subgradient descent can be written as

$$\begin{aligned} \mathbf{w}^{\mathrm{k+1}} &= \mathbf{w^k} - \eta_k \partial_k f(\mathbf{x_i}, y_i) \\ &= \mathbf{w^k} - \eta_k (\lambda \mathbf{w^k} + \mathbf{s}) \\ &= (1 - \eta_k \lambda)\mathbf{w^k} - \eta_k \mathbf{s} \end{aligned}$$

The pseudo-code of Pegasos is shown in Algorithm 10.

Pegasos can also be extended to the nonlinear case using the representer theorem. This theorem let us write **w** as a linear combination of the training examples

$$\mathbf{w} = \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j) \tag{41}$$

For each k, let $\boldsymbol{\alpha}^k \in R^n$ be the vector such that α_j^k counts how many times example j has been selected so far and it had a non-zero loss, namely

$$\alpha_j^k = \left|\left\{k' \leq \mathrm{k} / \mathrm{i} = \mathrm{j} \ \text{ and } \mathrm{y}_j \phi(\mathbf{x_j})^T \mathbf{w}^{k'} < 1\right\}\right|$$

Algorithm 11: Pegasos for the nonlinear SVM

Input :_ $\boldsymbol{\alpha} = \mathbf{0} \in R^n$ and λ
for *k* = 1, 2, . . . **do**

Sample *i* uniformly at random from {1, . . . , *n*}
forall $j \neq i$ **do**
$\alpha_j^{k+1} = \alpha_j^k$
end
if $y_i \frac{1}{\lambda k} \sum_{j=1}^{n} \alpha_j^k y_i k(\mathbf{x}_i, \mathbf{x}_j) < 1$ **then**
$\alpha_i^{k+1} = \alpha_i^k + 1$
else
$\alpha_i^{k+1} = \alpha_i^k$
end
end

Although we cannot compute the feature map ϕ directly, note that in Algorithm 10 we only perform two operations on **w**: scaling by a constant and dot-product. This last operation can be written **only** in terms of kernel evaluations,

$$\mathbf{w} = \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j) \quad ; \quad \mathbf{w}^T \phi(\mathbf{x_i}) = \sum_{j=1}^{n} \alpha_j y_j \phi(\mathbf{x}_j)^T \phi(\mathbf{x}_i) \quad ; \quad \mathbf{w}^T \phi(\mathbf{x_i}) = \sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i)$$

Thus, instead of keeping in memory the vector **w**, nonlinear Pegasos maintains the vector **α**. The pseudocode of the implementation of Pegasos for nonlinear SVMs is shown in Algorithm 11.

Dual coordinate methods

The ideas behind the previous algorithms could be extended to the dual SVM. If our aim is to solve very large linear SVM problems and **we ignore the bias term b**, problem (23) becomes

$$\min_{\boldsymbol{\alpha}} \left\{ \frac{1}{2} \boldsymbol{\alpha}^{\mathbf{T}} \mathbf{Q} \boldsymbol{\alpha} - \boldsymbol{\alpha}^{\mathbf{T}} \mathbf{1} \right\} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \forall \text{i}, \quad \text{i=1,...,n} \tag{42}$$

where $Q_{ij} = y_i y_j \mathbf{x_i^T x_j}$. Note that when removing the bias the constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ disappears. Hsieh et al. (2008) consider coordinate descent methods to solve the previous problem. Coordinate descent picks an index *i* and optimizes the objective function with respect to the coefficient α_i only, keeping the rest fixed. Let $f(\boldsymbol{\alpha})$ be the objective function of the dual SVM (42), to update the coefficient α_i we solve the **following one-variable sub-problem**:

$$\min_{d} \left\{ f(\boldsymbol{\alpha} + d\mathbf{e_i}) \right\} \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \forall \text{i}, \quad \text{i=1,...,n} \tag{43}$$

Ignoring terms that do not depend on α_i, we get

$$f(\boldsymbol{\alpha} + d\mathbf{e_i}) = \frac{1}{2}Q_{ii}d^2 + \nabla_i f(\boldsymbol{\alpha})d + Const. = \mathrm{d}(\frac{1}{2}Q_{ii}d + \nabla_i f(\boldsymbol{\alpha})) + Const.$$

Problem (4.3.9) has an optimum at d = 0 if and only **if the projected gradient**,

$$\nabla_i^p f(\boldsymbol{\alpha})_i = \begin{cases} \nabla_i f(\boldsymbol{\alpha}) & \text{if } 0<\alpha_i<\mathrm{C} \\ \min(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i=0 \\ \max(0, \nabla_i f(\boldsymbol{\alpha})) & \text{if } \alpha_i=\mathrm{C}, \end{cases}$$

vanishes (Hsieh et al., 2008), and in that case α_i does not need to be updated. Otherwise, the unconstrained optimal d value is

$$d^* = \frac{-\nabla_i f(\boldsymbol{\alpha})}{Q_{ii}}$$

and the update on α_i is

$$\alpha_i^{k+1} = \min\left\{\max\left\{\alpha_i^k - \frac{\nabla_i f(\alpha)}{Q_{ii}}, 0\right\}, C\right\},$$

where. Thus, $Q_{ii} = \mathbf{x_i^T x}_i$ can be precomputed and stored in memory. The gradient of the objective function with respect to α_i is

$$\nabla_i f(\boldsymbol{\alpha}) = [\mathbf{Q}\boldsymbol{\alpha}]_i - 1 = \sum_{j=1}^{n} Q_{ij}\alpha_j - 1,$$

The matrix **Q** may be too large to be precomputed and stored in memory, so we need to compute *n* dot products to evaluate the gradient on the fly, for a total cost of $O(n\bar{d})$ where $\bar{d}$ is the average of nonzero elements per instance. Such operation is expensive, with a worst case of O(*nd*) operations when the instances are dense.

Algorithm 12: Dual coordinate descent for the Linear SVM

Input : $\boldsymbol{\alpha} = \mathbf{0} \in \mathbb{R}^n$, $\mathbf{w} = \mathbf{0} \in \mathbb{R}^d$ and C

while $\boldsymbol{\alpha}$ is not optimal **do**

Pick an index i from {1, . . . , n}

$g = y_i \mathbf{w}^\mathbf{T} \mathbf{x}_i - 1$

$$p = \begin{cases} \min(g, 0) & \text{if } \alpha_i = 0 \\ \max(g, 0) & \text{if } \alpha_i = C \\ g & \text{if } 0<\alpha_i < C \end{cases}$$

if $|p| \neq 0$ **then**

$\alpha'_i \leftarrow \alpha_i$

$\alpha_i \leftarrow \min(\max(\alpha_i - \frac{g}{Q_{ii}}, 0), C)$

$\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha'_i) y_i \mathbf{x_i}$

end

end

However, since we are only solving the linear formulation, we have an explicit representation of the primal weights,

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{44}$$

Thus, we can compute the gradient as

$$\nabla_i f(\boldsymbol{\alpha}) = y_i \mathbf{w}^{\mathbf{T}} \mathbf{x_i} - 1,$$

in $O(\bar{d})$ operations, by maintaining both primal and dual weights. Again, the cost of computing (44) is $O(n\bar{d})$, but we can update it in only $O(\bar{d})$ as

$$\mathbf{w}^{k+1} = \mathbf{w}^{\mathbf{k}} + (\alpha_i^{k+1} - \alpha_i^k) y_i \mathbf{x_i}$$

This algorithm is implemented in the well-know software package LIBLINEAR (Fan et al., 2008).The pseudo-code is shown in Algorithm 12.

To compare the update of **w** with Stochastic Gradient Descent and Stochastic Coordinate Descent, we have to discuss first how to select the index *i*. As it was the case for the Lasso problem, there are several options: we can cycle through all the indices, cycle through a random permutation of the indices or pick an index uniformly at random. In the latter, the algorithm would be a stochastic dual coordinate descent method. The main difference is that now we need to project the gradient and clip the coefficients in order to satisfy the box constraints.

Decomposition methods

The main difficulty of solving the dual problem (23) of the non-linear SVM is that computing the whole kernel matrix **Q** is very expensive, hence we do not want to do it at every iteration.

Although **Q** only depends on the training data {**X, y**}, pre-computing the whole matrix may not be a very good idea (Bottou and Lin, 2007) for at least two reasons:

• It's wasteful: The expression of the gradient only depends on kernel values k($\mathbf{x}_i$, $\mathbf{x}_j$) that involve at least one support vector, and we can also recognize the optima using those kernel values only. In general, no more than 15% to 50% kernel values are actually needed to train the whole model.
• It may not fit into memory: Even if we were willing to spend time pre-computing the full kernel matrix, as n grows it may become prohibitively large and cannot be stored in memory.

Decomposition methods were designed to handle such difficulties. Some earlier works on decomposition methods for SVM include, for example, Osuna et al. (1997), Joachims (1998), Platt (1998), Keerthi et al. (2001), and Hsu and Lin (2002). Subsequent developments include, for example, Fan et al. (2005b), Palagi and Sciandrone (2005), and Glasmachers and Igel (2006). A good review can be found in Chang and Lin (2011). Popular software to train the SVM dual using decomposition methods include LIBSVM (Chang and Lin, 2011) and SVM-Light (Joachims, 1998).

Similarly to block coordinate descent, in decomposition methods only a subset of the coefficients $\boldsymbol{\alpha}$ are updated at every iteration, namely, the working set B. This subset leads to a small sub-problem being optimized at every iteration. More formally, let us define the complement **of the working set**, N = {1, . . . , n} \ B. Then, in every iteration decomposition methods fix all the coefficients $\alpha_i \in N$ and optimize only the ones in the working set B,

$$\min_{\boldsymbol{\alpha}_i,\, \mathrm{i}\in\mathrm{B}} \ \frac{1}{2}\sum_{i\in B}\sum_{j\in B}\alpha_i\alpha_j\, \mathrm{y_i y_j}\, k(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i\in B}\sum_{j\in N}\alpha_i\alpha_j\, \mathrm{y_i y_j}\, k(\mathbf{x}_i,\mathbf{x}_j) - \sum_{i\in B}\alpha_i$$

$$s.t.\ \ 0 \le \alpha_i \le C \qquad \mathrm{i}\in \mathrm{B},$$

$$\sum_{i\in B}\alpha_i y_i + \sum_{i\in N}\alpha_i y_i = 0$$

where we have ignored some constant factors that only depend on the coefficients in the set N.

An extreme case is the Sequential Minimal Optimization (SMO) algorithm (Platt, 1998), which updates only two coefficients per iteration. Then, for the SVM dual problem it can be shown that this sub-problem can be solved analytically and thus no actual optimization is needed. SMO working set has only two elements B = {i, j}, so the previous problem is equivalent to

$$\begin{aligned}
&\min_{\alpha_i,\,\alpha_j} \ \frac{1}{2}(\alpha_i^2 Q_{ii} + 2\alpha_i\alpha_j Q_{ij} + \alpha_j^2 Q_{jj}) + \alpha_i \sum_{k\in N} Q_{ik}\alpha_k + \alpha_j \sum_{k\in N} Q_{jk}\alpha_k - \alpha_i - \alpha_j \\
&s.t. \quad \mathrm{y}_i\alpha_i + \mathrm{y}_j\alpha_j = -\sum_{k\in N} y_k\alpha_k \\
&0 \le \alpha_i, \alpha_j \le C
\end{aligned} \tag{45}$$

Note that, in a given iteration, problem (45) only depends on the kernel rows $\mathbf{Q}_i$ and $\mathbf{Q}_j$. Those rows can be computed on-the-fly and thus there is no need to keep the full matrix in memory.

Suppose we are at iteration *k* and our current estimate of the coefficients is $\boldsymbol{\alpha}^k$. Then, SMO's update can be written as

$$\begin{aligned}
\alpha_i^{k+1} &= \alpha_i^k + \delta_i, \\
\alpha_j^{k+1} &= \alpha_j^k + \delta_j, \\
\alpha_l^{k+1} &= \alpha_l^k, \qquad \forall \mathrm{l} \neq \mathrm{i,j}
\end{aligned}$$

Since the linear constraint must hold for the updated coefficients $\boldsymbol{\alpha}^{k+1}$ and assuming $\boldsymbol{\alpha}^k$ is feasible we have

$$\begin{aligned}
&\mathbf{y}^{\mathbf{T}}\boldsymbol{\alpha}^{k+1} = 0 \\
&\mathbf{y}^{\mathbf{T}}\boldsymbol{\alpha}^{k} + y_i\delta_i + y_j\delta_j = 0 \\
&y_i\delta_i + y_j\delta_j = 0
\end{aligned}$$

Thus, for an update δ_i, δ_j to be feasible it must hold that

$$y_i\delta_i = -y_j\delta_j = \rho$$

Since $y \in \{-1, 1\}$, we can parameterized both steps as a function of ρ,

$$\delta_i = y_i\rho \quad \text{and} \quad \delta_j = -y_j\rho$$

arriving at the updates

$$\begin{aligned}
\alpha_i^{k+1} &= \alpha_i^k + y_i\rho, \\
\alpha_j^{k+1} &= \alpha_j^k - y_j\rho, \\
\alpha_l^{k+1} &= \alpha_l^k, \qquad \forall \mathrm{l} \neq \mathrm{i,j}
\end{aligned}$$

or, in vector form,

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \rho(y_i\mathbf{e_i} - y_j\mathbf{e_j}) = \boldsymbol{\alpha}^k + \rho\mathbf{d} \tag{46}$$

where $\mathbf{d} = y_i \mathbf{e_i} - y_j \mathbf{e_j}$ and $\mathbf{e}_k$ is the vector with all 0 but a 1 in the kth entry. The vector **d** is actually a descent direction and thus, in this context, ρ can be interpreted as a positive stepsize that gets us closer to the optimum by taking a step in the direction **d**.

As a result, due to the equality constraint, we can write both updates in α_i and α_j as a function of a single parameter ρ. Therefore, we have effectively transformed the decomposition problem associated with the working set $B = \{i, j\}$ (45) into a new problem depending only on the stepsize ρ,

$$\begin{aligned} \min_{\rho} \ & \frac{1}{2}(\boldsymbol{\alpha}^{\mathbf{k}} + \rho\mathbf{d})^T \mathbf{Q}(\boldsymbol{\alpha}^{\mathbf{k}} + \rho\mathbf{d}) - (\boldsymbol{\alpha}^{\mathbf{k}} + \rho\mathbf{d})^T \mathbf{1} \\ s.t. \quad & 0 \le \alpha_i + \mathrm{y}_i \rho \le C, \\ & 0 \le \alpha_j - \mathrm{y}_j \rho \le C, \end{aligned}$$

The previous problem is one-dimensional, quadratic, convex and only has two inequality constraints, thanks to the fact that every α_l, $\forall \mathrm{l} \neq \mathrm{i,j}$ is not updated. Minimizing the previous quadratic function with respect to ρ we get,

$$0 = \frac{1}{2}\mathbf{d^T Q d}\rho + \mathbf{d^T Q}\boldsymbol{\alpha}^{\mathbf{k}} - \mathbf{d}^T \mathbf{1}$$

and the optimum is attained at

$$\rho^* = - \frac{\mathbf{d^T Q}\boldsymbol{\alpha}^{\mathbf{k}} - \mathbf{d}^T \mathbf{1}}{\mathbf{d^T Q d}} \tag{47}$$

To compute the previous optimum value we have ignored the inequality constraints so it may need to be modified. However, it turns out that we can simply check if it belongs to the feasible region and clip the stepsize accordingly

$$\hat{\rho}^* = \max\left\{ y_i \alpha_i, -y_j \alpha_j, \min\left\{ -y_i (C - \alpha_i), y_j (C - \alpha_j), \rho^* \right\} \right\} \tag{48}$$

The final SMO updates are

$$\begin{aligned} \alpha_i^{k+1} &= \alpha_i^k + y_i \hat{\rho}^*, \\ \alpha_j^{k+1} &= \alpha_j^k - y_j \hat{\rho}^*, \end{aligned}$$

So far we have not said anything about how to select the two coefficients *i* and *j* to be updated at each iteration.

Algorithm 13: Sequential Minimal Optimization (SMO)

Input: $\boldsymbol{\alpha} = \mathbf{0} \in \mathbb{R}^d$, and C

while *stopping condition not met* **do**

Select working set (*i, j*)

Compute unconstrained stepsize ρ as in Eq. (47)

Clip the stepsize if necessary as in Eq. (48)

$\alpha_i \leftarrow \alpha_i + y_i \hat{\rho}^*$

$\alpha_j \leftarrow \alpha_j - y_j \hat{\rho}^*$

end

BIBLIOGRAPHY

Bottou, L. (2010). "Large-scale machine learning with stochastic gradient descent". In: Proceedings of COMPSTAT'2010. Springer, pp. 177–186

Bottou, L. and Lin, C.-J. (2007). "Support vector machine solvers". In: Large scale kernel machines, pp. 301–320.

Bousquet, O. and Bottou, L. (2008). "The tradeoffs of large scale learning". In: Advances in neural information processing systems, pp. 161–168.

Chang, C.-C. and Lin, C.-J. (2011). "LIBSVM: a Library for Support Vector Machines". In: ACM Trans. Intell. Syst. Technol. 2.3, 27:1–27:27. issn: 2157-6904.

Chapelle, O. (2007). "Training a support vector machine in the primal". In: Neural computation 19.5, pp. 1155–1178.

Cherkassky, V. and Ma, Y. (2004). "Practical selection of SVM parameters and noise estimation for SVM regression". In: Neural Networks 17, pp. 113–126.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005b). "Working set selection using second order information for training support vector machines". In: Journal of machine learning research 6.Dec, pp. 1889–1918.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). "LIBLINEAR: A library for large linear classification". In: Journal of machine learning research 9.Aug, pp. 1871–1874.

Glasmachers, T. and Igel, C. (2006). "Maximum-gain working set selection for SVMs". In: Journal of Machine Learning Research 7. Jul, pp. 1437–1466.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. (2008). "A dual coordinate descent method for large-scale linear SVM". In: Proceedings of the 25th international conference on Machine learning. ACM, pp. 408–415.

Hsu, C.-W. and Lin, C.-J. (2002). "A simple decomposition method for support vector machines". In: Machine Learning 46.1-3, pp. 291–314.

Joachims, T. (1998). Making large-scale SVM learning practical. Tech. rep. Technical Report, SFB 475: Komplexitätsreduktion in Multivariaten Datenstrukturen, Universität Dortmund.

Joachims, T. (2006). "Training linear SVMs in linear time". In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 217-226.

Joachims, T. and Yu, C.-N. J. (2009). "Sparse kernel SVMs via cutting-plane training". In: Machine Learning 76.2-3, pp. 179–193.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., and Murthy, K. R. K. (2001). “Improvements to Platt’s SMO algorithm for SVM classifier design”. In: Neural computation 13.3, pp. 637–649.

Kimeldorf, G. and Wahba, G. (1971). “Some results on Tchebycheffian spline functions”. In: Journal of mathematical analysis and applications 33.1, pp. 82–95.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). “Online learning with kernels”. In: IEEE transactions on signal processing 52.8, pp. 2165–2176.

Osuna, E., Freund, R., and Girosi, F. (1997). Support vector machines: Training and applications. Tech. rep.

Palagi, L. and Sciandrone, M. (2005). “On the convergence of a modified version of SVM light algorithm”. In: Optimization methods and Software 20.2-3, pp. 317–334.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Tech. rep.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). “New support vector algorithms”. In: Neural computation 12.5, pp. 1207–1245.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. (2001). “Estimating the support of a high-dimensional distribution”. In: Neural computation 13.7, pp. 1443–1471.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). “Pegasos: Primal estimated sub-gradient solver for svm”. In: Proceedings of the 24th international conference on machine learning. ACM, pp. 807–814.

Yu, K., Leufen, G., Hunsche, M., Noga, G., Chen, X., and Bareth, G. (2014). “Investigation of Leaf Diseases and Estimation of Chlorophyll Concentration in Seven Barley Varieties Using Fluorescence and Hyperspectral Indices”. In: Remote Sensing 6.1, pp. 64–86. issn: 2072-4292.