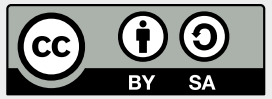
SMO Algorithm

Milan Straka

December 02, 2019

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



Kernel Linear Regression

When dimensionality of input is D, one step of SGD takes $\mathcal{O}(D^3)$.

Surprisingly, we can do better under some circumstances. We start by noting that we can write the parameters $\boldsymbol{w}$ as a linear combination of the input features $\varphi(\boldsymbol{x}_i)$.

By induction, $\boldsymbol{w} = 0 = \sum_i 0 \cdot \varphi(\boldsymbol{x}_i)$, and assuming $\boldsymbol{w} = \sum_i \beta_i \cdot \varphi(\boldsymbol{x}_i)$, after a SGD update we get

$$\begin{aligned}\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha \sum_i \big(t_i - \boldsymbol{w}^T \varphi(\boldsymbol{x}_i)\big)\varphi(\boldsymbol{x}_i) \\ &= \sum_i \Big(\beta_i + \alpha\big(t_i - \boldsymbol{w}^T \varphi(\boldsymbol{x}_i)\big)\Big)\varphi(\boldsymbol{x}_i).\end{aligned}$$

A individual update is $\beta_i \leftarrow \beta_i + \alpha\Big(t_i - \boldsymbol{w}^T \varphi(\boldsymbol{x}_i)\Big)$, and substituting for $\boldsymbol{w}$ we get

$$\beta_i \leftarrow \beta_i + \alpha\Big(t_i - \sum\nolimits_j \beta_j \varphi(\boldsymbol{x}_j)^T \varphi(\boldsymbol{x}_i)\Big).$$

Kernel Linear Regression

We can formulate the alternative linear regression algorithm (it would be called a *dual formulation*):

Input: Dataset ($\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \mathbb{R}^N$), learning rate $\alpha \in \mathbb{R}^+$.

- Set $\beta_i \leftarrow 0$
- Compute all values $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)$
- Repeat
  - Update the coordinates, either according to a full gradient update:
    - $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\boldsymbol{t} - K\boldsymbol{\beta})$
  - or alternatively use single-batch SGD, arriving at:
    - for i in random permutation of $\{1, \ldots, N\}$:
      - $\beta_i \leftarrow \beta_i + \alpha\Big(t_i - \sum_j \beta_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big)$

    In vector notation, we can write $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha(\boldsymbol{t} - K\boldsymbol{\beta})$.

The predictions are then performed by computing $y(\boldsymbol{x}) = \boldsymbol{w}^T \varphi(\boldsymbol{x}) = \sum_i \beta_i \boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{\varphi}(\boldsymbol{x})$.

Support Vector Machines

Assume we have a dataset $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, 1\}^N$, feature map φ and model

$$y(\boldsymbol{x}) \stackrel{\text{def}}{=} \boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{w} + b.$$

We already know that the distance of a point $\boldsymbol{x}_i$ to the decision boundary is

$$\frac{|y(\boldsymbol{x}_i)|}{||\boldsymbol{w}||} = \frac{t_i y(\boldsymbol{x}_i)}{||\boldsymbol{w}||}.$$

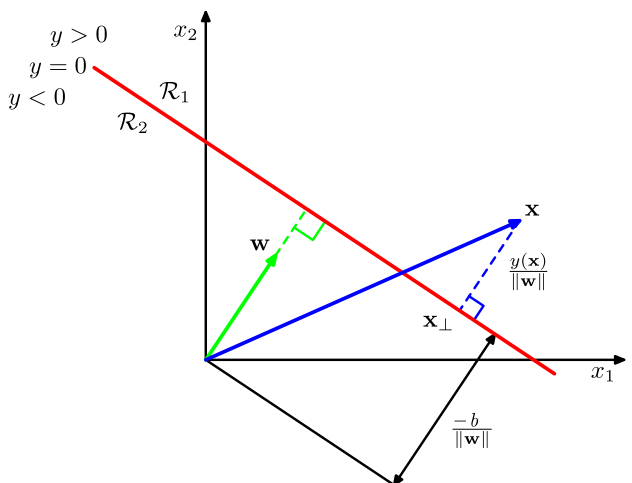


Figure 4.1 of Pattern Recognition and Machine Learning.

We therefore want to maximize

$$\arg\max_{w,b} \frac{1}{||\boldsymbol{w}||} \min_i \left[t_i (\boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{w} + b) \right].$$

However, this problem is difficult to optimize directly.

Because the model is invariant to multiplying $\boldsymbol{w}$ and b by a constant, we can say that for the points closest to the decision boundary, it will hold that

$$t_i y(\boldsymbol{x}_i) = 1.$$

Then for all the points we will have $t_i y(\boldsymbol{x}_i) \geq 1$ and we can simplify

$$\arg\max_{w,b} \frac{1}{||\boldsymbol{w}||} \min_i \left[t_i (\boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{w} + b) \right]$$

to

$$\arg\min_{w,b} \frac{1}{2} ||\boldsymbol{w}||^2 \text{ given that } t_i y(\boldsymbol{x}_i) \geq 1.$$

In order to solve the constrained problem of

$$\arg\min_{w,b} \frac{1}{2}||\boldsymbol{w}||^2 \text{ given that } t_i y(\boldsymbol{x}_i) \geq 1,$$

we write the Lagrangian with multipliers $\boldsymbol{a} = (a_1, \ldots, a_N)$ as

$$L = \frac{1}{2}||\boldsymbol{w}||^2 - \sum_i a_i \left[t_i y(\boldsymbol{x}_i) - 1\right].$$

Setting the derivatives with respect to $\boldsymbol{w}$ and b to zero, we get

$$\boldsymbol{w} = \sum_i a_i t_i \varphi(\boldsymbol{x}_i)$$
$$0 = \sum_i a_i t_i$$

Substituting these to the Lagrangian, we get

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

with respect to the constraints $\forall_i : a_i \geq 0$, $\sum_i a_i t_i = 0$ and kernel $K(\boldsymbol{x}, \boldsymbol{z}) = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{z})$.

The solution of this Lagrangian will fulfil the KKT conditions, meaning that

$$\begin{aligned} a_i &\geq 0 \\ t_i y(\boldsymbol{x}_i) - 1 &\geq 0 \\ a_i \big(t_i y(\boldsymbol{x}_i) - 1\big) &= 0. \end{aligned}$$

Therefore, either a point is on a boundary, or $a_i = 0$. Given that the predictions for point $\boldsymbol{x}$ are given by $y(\boldsymbol{x}) = \sum a_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b$, we need to keep only the points on the boundary, the so-called **support vectors**.

Support Vector Machines

The dual formulation allows us to use non-linear kernels.

Figure 7.2 Example of synthetic data from two classes in two dimensions showing contours of constant $y(\mathbf{x})$ obtained from a support vector machine having a Gaussian kernel function. Also shown are the decision boundary, the margin boundaries, and the support vectors.

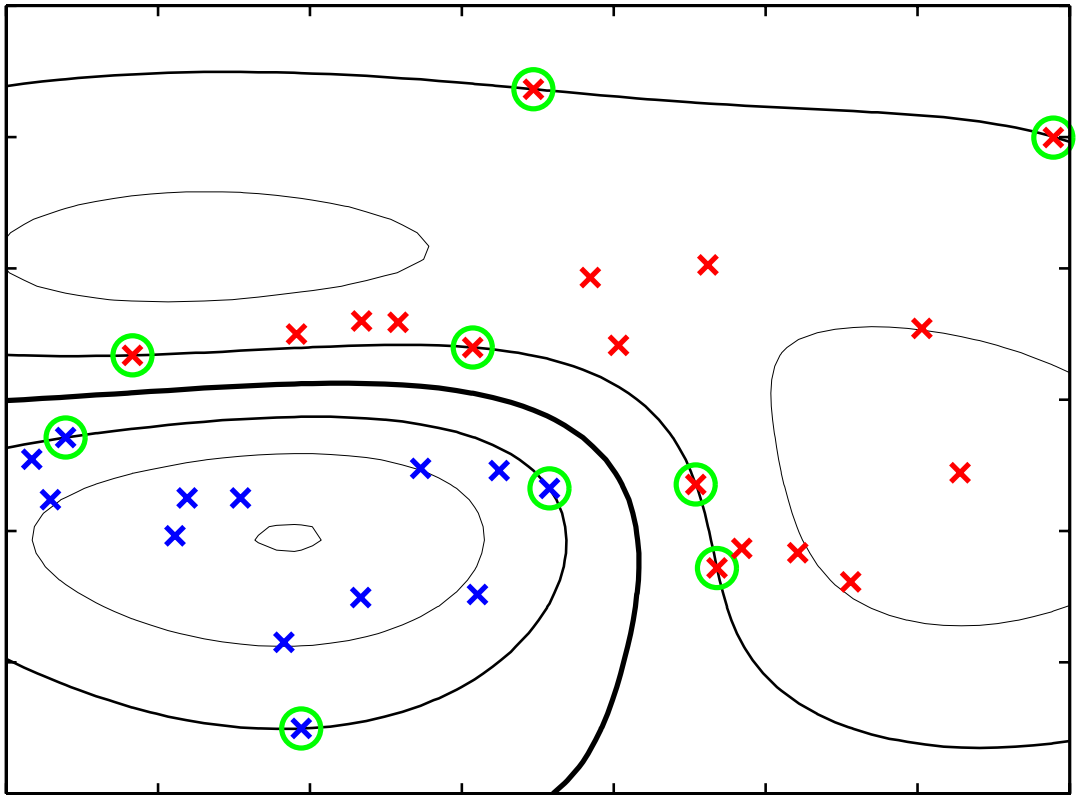


Figure 7.2 of Pattern Recognition and Machine Learning.

Support Vector Machines for Non-linearly Separable Data

Until now, we assumed the data to be linearly separable – the **hard-margin SVM** variant. We now relax this condition to arrive at **soft-margin SVM**. The idea is to allow points to be in the margin or even on the *wrong side* of the decision boundary. We introduce *slack variables* $\xi_i \geq 0$, one for each training instance, defined as

$$\xi_i = \begin{cases} 0 & \text{for points fulfilling } t_i y(\boldsymbol{x}_i) \geq 1, \\ |t_i - y(\boldsymbol{x}_i)| & \text{otherwise.} \end{cases}$$

Therefore, $\xi_i = 0$ signifies a point outside of margin, $0 < \xi_i < 1$ denotes a point inside the margin, $\xi_i = 1$ is a point on the decision boundary and $\xi_i > 1$ indicates the point is on the opposite side of the separating hyperplane.

Therefore, we want to optimize

$$\arg\min_{w,b} C \sum_i \xi_i + \frac{1}{2}||\boldsymbol{w}||^2 \text{ given that } t_i y(\boldsymbol{x}_i) \geq 1 - \xi_i \text{ and } \xi_i \geq 0.$$

Support Vector Machines for Non-linearly Separable Data

We again create a Lagrangian, this time with multipliers $\boldsymbol{a} = (a_1, \ldots, a_N)$ and also $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N)$:

$$L = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_i \xi_i - \sum_i a_i \left[t_i y(\boldsymbol{x}_i) - 1 + \xi_i\right] - \sum_i \mu_i \xi_i .$$

Solving for the critical points and substituting for $\boldsymbol{w}$, b and $\boldsymbol{\xi}$ (obtaining an additional constraint $\mu_i = C - a_i$ compared to the previous case), we obtain the Lagrangian in the form

$$L = \sum_i a_i - \frac{1}{2}\sum_i \sum_j a_i a_j t_i t_j K(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

which is identical to the previous case, but the constraints are a bit different:

$$\forall_i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$

Support Vector Machines for Non-linearly Separable Data

Using KKT conditions, we can see that the support vectors (examples with $a_i > 0$) are the ones with $t_i y(\boldsymbol{x}_i) = 1 - \xi_i$, i.e., the examples on the margin boundary, inside the margin and on the opposite side of the decision boundary.

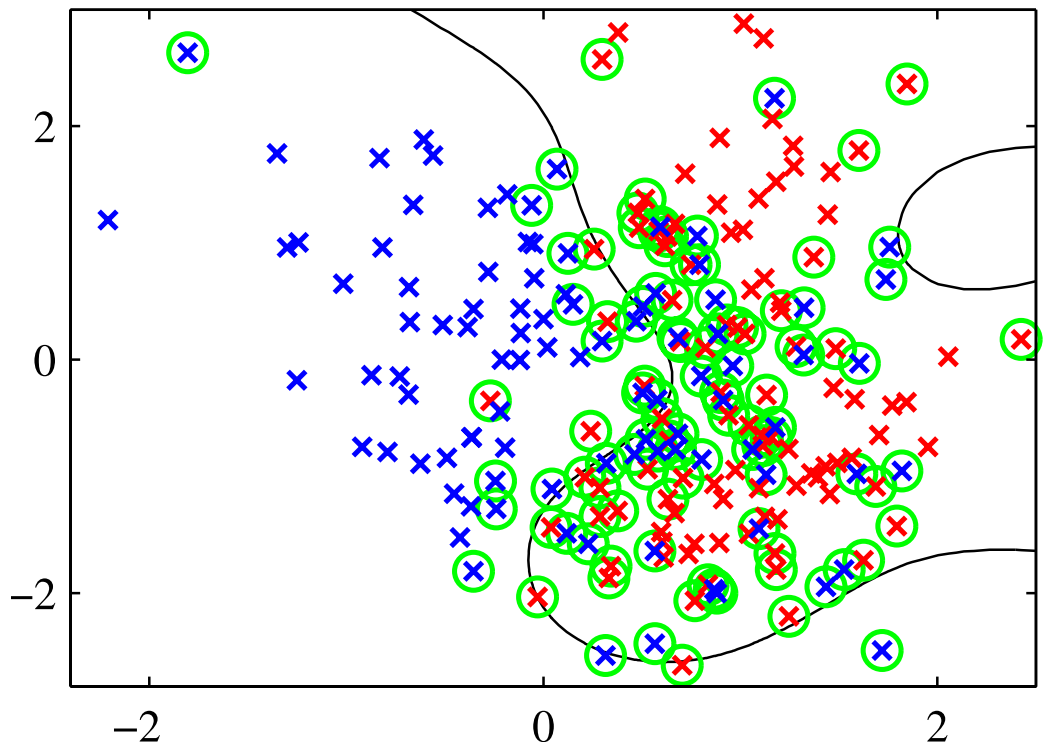


Figure 7.4 of Pattern Recognition and Machine Learning.

Sequential Minimal Optimization Algorithm

To solve the dual formulation of a SVM, usually Sequential Minimal Optimization (SMO; John Platt, 1998) algorithm is used.

Before we introduce it, we start by introducing **coordinate descent** optimization algorithm.

Consider solving unconstrained optimization problem

$$\arg\min_{\boldsymbol{w}} L(w_1, w_2, \ldots, w_D).$$

Instead of the usual SGD approach, we could optimize the weights one by one, using the following algorithm

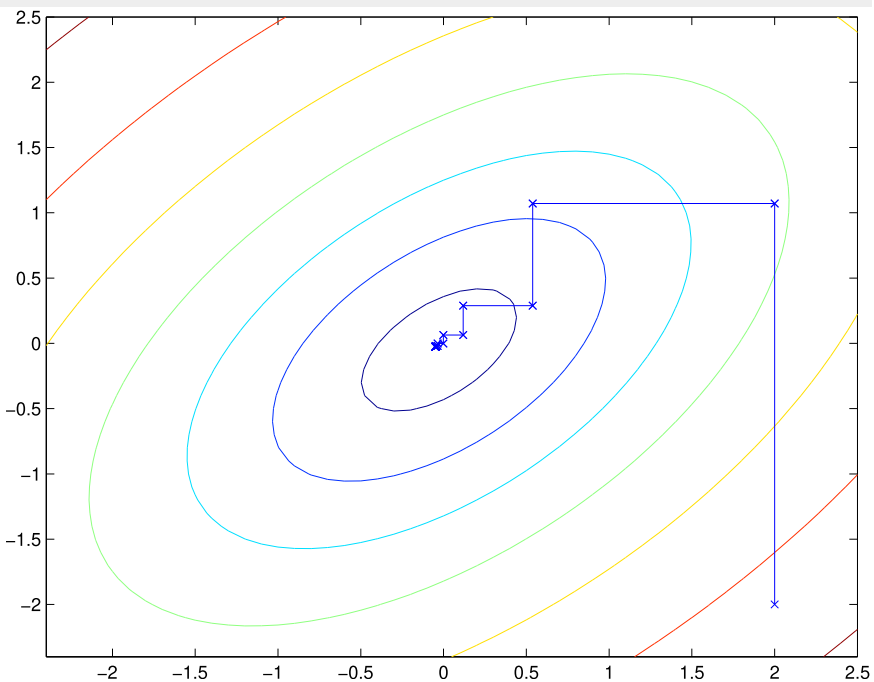
- loop until convergence
  - for i in $\{1, 2, \ldots, D\}$:
    - $w_i \leftarrow \arg\min_{w_i} L(w_1, w_2, \ldots, w_D)$

Sequential Minimal Optimization Algorithm

- loop until convergence
- for i in $\{1, 2, \ldots, D\}$:
  - $w_i \leftarrow \arg\min_{w_i} L(w_1, w_2, \ldots, w_D)$

If the inner $\arg\min$ can be performed efficiently, the coordinate descent can be fairly efficient.

Note that we might want to choose w_i in different order, for example by trying to choose w_i providing the largest decrease of L.



CS229 Lecture 3 Notes, http://cs229.stanford.edu/notes/cs229-notes3.pdf

Sequential Minimal Optimization Algorithm

In soft-margin SVM, we try to minimize

$$L = \sum_i a_i - \frac{1}{2} \sum_i \sum_j a_i a_j t_i t_j K(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

such that

$$\forall_i : C \geq a_i \geq 0 \text{ and } \sum_i a_i t_i = 0.$$

The KKT conditions for the solution can be reformulated (while staying equivalent) as

$$\begin{aligned}
a_i > 0 &\Rightarrow t_i y(\boldsymbol{x}_i) \leq 1, \quad \text{because } a_i > 0 \Rightarrow t_i y(\boldsymbol{x}_i) = 1 - \xi_i \text{ and we have } \xi_i \geq 0, \\
a_i < C &\Rightarrow t_i y(\boldsymbol{x}_i) \geq 1, \quad \text{because } a_i < C \Rightarrow \mu_i > 0 \Rightarrow \xi_i = 0 \text{ and } t_i y(\boldsymbol{x}_i) \geq 1 - \xi_i, \\
0 < a_i < C &\Rightarrow t_i y(\boldsymbol{x}_i) = 1, \quad \text{a combination of both.}
\end{aligned}$$

Sequential Minimal Optimization Algorithm

At its core, the SMO algorithm is just a coordinate descent.

It tries to find such α_i fulfilling the KKT conditions – for soft-margin SVM, KKT conditions are sufficient conditions for optimality (the loss is convex and inequality constraints affine).

However, note that because of the $\sum a_i t_i = 0$ constraint we cannot optimize just one a_i, because a single a_i is determined from the others.

Therefore, in each step we pick two a_i, a_j coefficients and try to minimize the loss while fulfilling the constraints.

- loop until convergence (until $\forall i : a_i < C \Rightarrow t_i y(\boldsymbol{x}_i) \geq 1$ and $a_i > 0 \Rightarrow t_i y(\boldsymbol{x}_i) \leq 1$)
  - for i in $\{1, 2, \ldots, D\}$, for $j \neq i$ in ${1, 2, ..., D}:
    - $a_i, a_j \leftarrow \arg\min_{a_i, a_j} L(a_1, a_2, \ldots, a_D)$ such that $C \geq a_i \geq 0$, $\sum_i a_i t_i = 0$

Sequential Minimal Optimization Algorithm

The SMO is an efficient algorithm, because we can compute the update to a_i, a_j efficiently, because there exists an closed form solution.

Assume that we are updating a_i and a_j. Then from the $\sum_k a_k t_k = 0$ condition we can write $a_i t_i = -\sum_{k \neq i} a_k t_k$. Given that $t_i^2 = 1$ and denoting $\zeta = -\sum_{k \neq i, k \neq j} a_k t_k$, we get

$$a_i = t_i(\zeta - a_j t_j).$$

Minimizing $L(\boldsymbol{a})$ with respect to a_i and a_j then amounts to minimizing a quadratic function of a_j, which has an analytical solution.

Note that the real SMO algorithm has several heuristics for choosing a_i, a_j such that the L can be minimized the most.

Sequential Minimal Optimization Algorithm Sketch

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, 1\}^N$), kernel $\boldsymbol{K}$, regularization parameter C, tolerance tol, $max\_passes\_without\_a\_changing$ value

- Initialize $a_i \leftarrow 0$, $b \leftarrow 0$, $passes \leftarrow 0$
- **while** $passes < max\_passes\_without\_a\_changing$:
  - $changed\_as \leftarrow 0$
  - **for** i in $1, 2, \ldots, N$:
    - $E_i \leftarrow y(\boldsymbol{x}_i) - t_i$
    - **if** ($a_i < C$ **and** $t_i E_i < -tol$) **or** ($a_i > 0$ **and** $t_i E_i > tol$):
      - Choose $j \neq i$ randomly
      - Update a_i, a_j and b
      - $changed\_as \leftarrow changed\_as + 1$
  - **if** $changed\_as = 0$: $passes \leftarrow passes + 1$
  - **else**: $passes \leftarrow 0$

Sequential Minimal Optimization Algorithm Sketch

Input: Dataset ($\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\boldsymbol{t} \in \{-1, 1\}^N$), kernel $\boldsymbol{K}$, regularization parameter C, tolerance tol, $max\_passes\_without\_a\_changing$ value

- Update a_i, a_j, b:
  - Express a_i using a_j
  - Find a_j optimizing the loss L quadratic with respect to a_j
  - Clip a_j so that $0 \leq a_i, a_j \leq C$
  - Compute corresponding a_i
  - Compute b matching to updated a_i, a_j

Sequential Minimal Optimization Update Rules

We already know that $a_i = t_i(\zeta - a_j t_j)$.

To find a_j optimizing the loss L, we use the formula for locating a vertex of a parabola

$$a_j^{\text{new}} \leftarrow a_j - \frac{\partial L / \partial a_j}{\partial^2 L / \partial a_j^2},$$

which is in fact one Newton-Raphson iteration step.

Denoting $E_j \stackrel{\text{def}}{=} y(\boldsymbol{x}_j) - t_j$, we can compute the first derivative as

$$\frac{\partial L}{\partial a_j} = t_j(E_i - E_j)$$

and the second derivative as

$$\frac{\partial^2 L}{\partial a_j^2} = 2K(\boldsymbol{x}_i, \boldsymbol{x}_j) - K(\boldsymbol{x}_i, \boldsymbol{x}_i) - K(\boldsymbol{x}_j, \boldsymbol{x}_j).$$

Sequential Minimal Optimization Update Rules

If the second derivative is positive, we know that the vertex is really a minimum, in which case we get

$$a_j^{\text{new}} \leftarrow a_j - t_j \frac{E_i - E_j}{2K(\boldsymbol{x}_i, \boldsymbol{x}_j) - K(\boldsymbol{x}_i, \boldsymbol{x}_i) - K(\boldsymbol{x}_j, \boldsymbol{x}_j)}.$$

We then clip a_j so that $0 \leq a_i, a_j \leq C$, by clipping a_j to range $[L, H]$ with

$$t_i = t_j \Rightarrow L = \max(0, a_i + a_j - C), H = \min(C, a_i + a_j)$$
$$t_i \neq t_j \Rightarrow L = \max(0, a_j - a_i), H = \min(C, C + a_j - a_i).$$

Finally we set

$$a_i^{\text{new}} \leftarrow a_i - t_i t_j \left(a_j^{\text{new}} - a_j\right).$$

Sequential Minimal Optimization Update Rules

To arrive at the bias update, we consider the KKT condition that for $0 < a_j^{\text{new}} < C$ it must hold that $t_j y(\boldsymbol{x}_j) = 1$. Combining it with with $b = E_j + t_j - \sum_l a_l t_l K(\boldsymbol{x}_j, \boldsymbol{x}_l)$, we get the following value

$$b_j = b - E_j - t_i(a_i^{\text{new}} - a_i)K(\boldsymbol{x}_i, \boldsymbol{x}_j) - t_j(a_j^{\text{new}} - a_j)K(\boldsymbol{x}_j, \boldsymbol{x}_j).$$

Analogously for $0 < a_i^{\text{new}} < C$ we get

$$b_i = b - E_i - t_i(a_i^{\text{new}} - a_i)K(\boldsymbol{x}_i, \boldsymbol{x}_i) - t_j(a_j^{\text{new}} - a_j)K(\boldsymbol{x}_j, \boldsymbol{x}_i).$$

Finally, if $a_j^{\text{new}}, a_i^{\text{new}} \in \{0, C\}$, we know that all values between b_i and b_j fulfil the KKT conditions. We therefore arrive at the following update for bias:

$$b^{\text{new}} = \begin{cases} b_i & \text{if } 0 < a_i^{\text{new}} < C \\ b_j & \text{if } 0 < a_j^{\text{new}} < C \\ \frac{b_i + b_j}{2} & \text{otherwise.} \end{cases}$$

Multiclass SVM

There are two general approach for building a K-class classifier by combining several binary classifiers:

- **one-versus-rest** scheme: K binary classifiers are constructed, the i-th separating instances of class i from all others; during prediction, the one with highest probability is chosen
  - the binary classifiers need to return calibrated probabilities (not SVM)
- **one-versus-one** scheme: $\binom{K}{2}$ binary classifiers are constructed, one for each (i, j) pair of class indices; during prediction, the class with the majority of votes wins (used by SVM)

However, both of the above approaches suffer from serious difficulties, because training the binary classifiers separately creates usually several regions which are ambiguous.

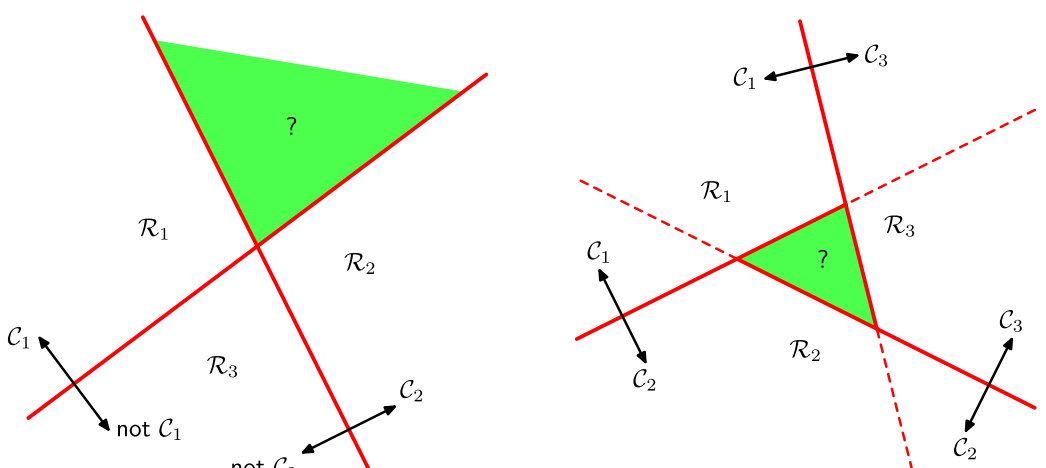


Figure 4.2 of Pattern Recognition and Machine Learning.