Lecturer: Matthias Seeger
Assistants: Abdallah Elguindy, Mathias Humbert, Young Jun Ko, Xinchao Wang

Miniproject: Documentation for Support Vector Machines

October 24, 2013

1 Sequential Minimal Optimization

In this note, we describe the sequential minimal optimization (SMO) algorithm to solve the soft margin support vector machine (SVM) binary classification problem, which is to be implemented as part of the miniproject. In order to understand our arguments here, the reader is advised to study the SVM chapter of the course notes, in particular the "Solving the Support Vector Machine" section (the section "Lagrange Multipliers and Lagrangian Duality" is not required). The SMO algorithm is due to Platt [**?**], but we make use of algorithmic improvements proposed by Keerthi *et.al.* [**?**]. The interested reader is encouraged to study these references as well.
The notation we employ here is the same as in the SVM chapter of the course notes. SMO solves the dual of the soft margin SVM problem:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \left\{ \Phi(\boldsymbol{\alpha}) = \frac{1}{2} \sum\nolimits_{i,j=1}^{n} \alpha_i \alpha_j t_i t_j K_{ij} - \sum\nolimits_{i=1}^{n} \alpha_i \right\}, \\ &\text{subj. to } \alpha_i \in [0, C],\ i = 1, \ldots, n, \quad \sum\nolimits_{i=1}^{n} t_i \alpha_i = 0. \end{aligned} \tag{1}$$

Here, $\boldsymbol{K} = [K_{ij}] = [K(\boldsymbol{x}_i, \boldsymbol{x}_j)]$ is the kernel matrix for the training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, t_i) \,|\, i = 1, \ldots, n\}$, where $t_i \in \{-1, +1\}$. SMO is essentially a constrained coordinate descent algorithm. In each iteration, Φ is minimized w.r.t. α_i and α_j for some $i \neq j$. Note that due to the equality constraint, we have to update at least two variables in order to stay in the feasible set, so SMO is "minimal" in that sense. There are two questions we have to deal with:

- How to pick $\{i, j\}$ for the next update? When are we done?
- How to minimize Φ w.r.t. α_i, α_j, keeping all other $\boldsymbol{\alpha}$ coordinates fixed?

The SVM discriminant function is given by the kernel expansion

$$y(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i t_i K(\boldsymbol{x}, \boldsymbol{x}_i) - b,$$

where b is a bias term, which can be determined from the final $\boldsymbol{\alpha}$. **Attention**: In the course notes, the bias parameter has a different sign: it is added to the kernel expansion to get $y(\boldsymbol{x})$, while here, we subtract b. We do this in order to keep our notation the same as in [**?**]. This does not matter much, since b is determined after convergence of SMO only. But take care to rewrite the formula for b given in the course notes in your code.
Recall from the course notes that a solution of the dual (1) is determined by the KKT conditions:

$$\begin{aligned} \alpha_i = 0: &\quad t_i y(\boldsymbol{x}_i) - 1 \geq 0 \\ \alpha_i \in (0, C): &\quad t_i y(\boldsymbol{x}_i) - 1 = 0 \\ \alpha_i = C: &\quad t_i y(\boldsymbol{x}_i) - 1 \leq 0 \end{aligned}$$

If we define

$$f_i = \sum_{j=1}^{n} \alpha_j t_j K_{ij} - t_i, \tag{2}$$

then

$$(f_i - b)t_i = (y(\boldsymbol{x}_i) - t_i)\, t_i = t_i y(\boldsymbol{x}_i) - 1,$$

since $t_i^2 = 1$. This means we can rewrite the KKT conditions as follows:

$$\begin{aligned} \alpha_i = 0: &\quad (f_i - b)t_i \geq 0 \\ \alpha_i \in (0, C): &\quad (f_i - b)t_i = 0 \\ \alpha_i = C: &\quad (f_i - b)t_i \leq 0 \end{aligned}$$

Let us partition $\{1, \ldots, n\}$ into I_0, I_+, I_-, where

$$\begin{aligned} &I_0 = \{i \mid 0 < \alpha_i < C\}, \quad I_+ = \{i \mid t_i = +1, \alpha_i = 0 \text{ or } t_i = -1, \alpha_i = C\}, \\ &I_- = \{i \mid t_i = -1, \alpha_i = 0 \text{ or } t_i = +1, \alpha_i = C\}. \end{aligned}$$

Moreover, let $I_{\text{up}} = I_0 \cup I_+$, $I_{\text{low}} = I_0 \cup I_-$. Then, the KKT conditions can be written as

$$b \leq f_i \; \forall i \in I_{\text{up}} \quad \text{and} \quad b \geq f_i \; \forall i \in I_{\text{low}}.$$

If we define

$$b_{\text{up}} = \min\{f_i \mid i \in I_{\text{up}}\}, \quad b_{\text{low}} = \max\{f_i \mid i \in I_{\text{low}}\},$$

the KKT conditions are fulfilled at $\boldsymbol{\alpha}$ iff $b_{\text{low}} \leq b_{\text{up}}$ (as we can squeeze b in between if and only if this inequality holds). For numerical reasons, and also to attain convergence in a reasonable number of iterations, we will be satisfied with approximately fulfilled conditions:

$$b_{\text{low}} \leq b_{\text{up}} + 2\tau, \quad \tau > 0. \tag{3}$$

Here, τ is a small positive number, such as 10^{-8}. This is the termination condition for the SMO algorithm. As long as it does not hold, there will always be a violated pair,

$$f_i > f_j + 2\tau, \quad i \in I_{\text{low}},\ j \in I_{\text{up}} \tag{4}$$

on which an update has to be done. As a heuristic to minimize the number of updates, we may select the most violated pair ($f_i = b_{\text{low}}$, $f_j = b_{\text{up}}$). Before discussing the pair selection in detail, we derive the minimization for a chosen pair.

1.1 Minimization w.r.t. α_i, α_j

Suppose that (i, j) is a violated pair. The fact which drives SMO is that we can minimize Φ analytically w.r.t. α_i, α_j, keeping all other coordinates of $\boldsymbol{\alpha}$ fixed. The feasible set of this bivariate minimization is given by the box constraints and the requirement to keep the equality constraint valid. We summarize results at the end of this section.
Denote the old values by $\tilde{\alpha}_i, \tilde{\alpha}_j$, the minimizers by α_i, α_j, and $\Delta_i = \alpha_i - \tilde{\alpha}_i$, $\Delta_j = \alpha_j - \tilde{\alpha}_j$. Moreover, let $\sigma = t_i t_j \in \{-1, +1\}$. The feasible set is given by

$$\alpha_i, \alpha_j \in [0, C], \quad t_i \Delta_i + t_j \Delta_j = 0.$$

The last equation is obtained by subtracting the equality constraint for the new from that for the old values. Multiplying it by t_i, we have

$$\Delta_i + \sigma \Delta_j = 0 \quad \Rightarrow \quad \alpha_i = \tilde{\alpha}_i - \sigma(\alpha_j - \tilde{\alpha}_j) = w - \sigma \alpha_j, \quad w := \tilde{\alpha}_i + \sigma \tilde{\alpha}_j.$$

Therefore, the problem boils down to a *univariate* quadratic minimization for α_j only, subject to $\alpha_j \in [L, H]$. In order to work out L and H, note that $\alpha_j \in [0, C]$ and

$$\alpha_i = w - \sigma \alpha_j \in [0, C] \quad \Rightarrow \quad w - C \leq \sigma \alpha_j \leq w.$$

Distinguishing between $\sigma = +1$ and $\sigma = -1$, we obtain

$$L = \max\left\{0, \sigma w - \mathrm{I}_{\{\sigma=+1\}} C\right\}, \quad H = \min\left\{C, \sigma w + \mathrm{I}_{\{\sigma=-1\}} C\right\}, \quad \sigma w = \tilde{\alpha}_j + \sigma \tilde{\alpha}_i. \tag{5}$$

We solve for α_j in two steps. First, we determine the unconstrained minimum point, then clip it according to $[L, H]$. Details for the first step can be found in [**?**] (exercise!). If we plug $\alpha_i = w - \sigma \alpha_j$ into Φ, then

$$\frac{\partial \Phi}{\partial \alpha_j} = (K_{ii} + K_{jj} - 2K_{ij})\alpha_j - \sigma(K_{ii} - K_{ij})w - t_j(v_i - v_j) + \sigma - 1.$$

Here,

$$v_i = \sum_{p \neq i,j} \alpha_p t_p K_{ip} = f_i + t_i - \tilde{\alpha}_i t_i K_{ii} - \tilde{\alpha}_j t_j K_{ij}, \quad v_j = f_j + t_j - \tilde{\alpha}_i t_i K_{ij} - \tilde{\alpha}_j t_j K_{jj}.$$

Note that f_i and f_j in these equations denote the values for the old values $\tilde{\alpha}_i$, $\tilde{\alpha}_j$. Also, $\eta := K_{ii} + K_{jj} - 2K_{ij}$ must be positive, since the kernel matrix $\boldsymbol{K}$ is positive definite:

$$\eta = (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j)^T \boldsymbol{K} (\boldsymbol{\delta}_i - \boldsymbol{\delta}_j) > 0.$$

Plugging $w = \tilde{\alpha}_i + \sigma \tilde{\alpha}_j$ into $(\partial \Phi)/(\partial \alpha_j) = 0$, some algebra gives

$$\alpha_j^{\mathrm{unc}} = \tilde{\alpha}_j + \frac{t_j(f_i - f_j)}{\eta} \tag{6}$$

for the unconstrained minimizer. Finally, if α_j^{unc} does not lie in $[L, H]$, we clip it accordingly:

$$\alpha_j = \left\{ \begin{array}{ll} \alpha_j^{\mathrm{unc}} & \mid \ \alpha_j^{\mathrm{unc}} \in [L, H] \\ L & \mid \ \alpha_j^{\mathrm{unc}} < L \\ H & \mid \ \alpha_j^{\mathrm{unc}} > H \end{array} \right\}. \tag{7}$$

Finally, in rare cases it can happen that η is zero or very close to zero. In these cases, we evaluate Φ (up to a constant) at both ends of the segment $[L, H]$ and update $\alpha_j = L$ or $\alpha_j = H$, depending on which value is lower. Since the relevant part of Φ is given by

$$\Phi = \frac{1}{2}\left(K_{ii}\alpha_i^2 + K_{jj}\alpha_j^2\right) + \sigma K_{ij}\alpha_i\alpha_j + t_i\alpha_i v_i + t_j\alpha_j v_j - \alpha_i - \alpha_j + \mathrm{const},$$

we can compute the value Φ_L for $\alpha_j = L$ as

$$\Phi_L = \frac{1}{2}\left(K_{ii}(L_i)^2 + K_{jj}L^2\right) + \sigma K_{ij}L_i L + t_i L_i v_i + t_j L v_j - L_i - L, \quad L_i = w - \sigma L, \quad (8)$$

where w, v_i and v_j are defined above. The value Φ_H for $\alpha_j = H$ is obtained in the same way, with $L \to H$ and $L_i \to H_i = w - \sigma H$.
To summarize this section, the minimization w.r.t. α_i, α_j works as follows. First, we determine the feasible range $[L, H]$ for α_j (5), and $\eta = K_{ii} + K_{jj} - 2K_{ij}$. If $\eta > 10^{-15}$, we determine the minimizer α_j via (6) and (7). Otherwise, we determine Φ_L and Φ_H, setting $\alpha_j = L$ if $\Phi_L < \Phi_H$, otherwise $\alpha_j = H$. Finally, we set $\alpha_i = w - \sigma\alpha_j$. Importantly, all these computations are $O(1)$ and require a single kernel evaluation (K_{ij}; we assume that the diagonal of the kernel matrix is precomputed), given that we maintain the values $[f_p]$ up to date.

1.2 Heuristics for choosing violated pair (i, j)

Based on [**?**], we discuss how to select the most violated pair (i, j). As mentioned above, we will use the optimality conditions (3) and (4) to check for optimality at $\boldsymbol{\alpha}$. For efficiency reasons we maintain a vector $\boldsymbol{f}$ with entries as defined in (2). We define new indices i_{low} and i_{up} such that

$$f_{i_{\text{up}}} = b_{\text{up}} = \min\left\{f_i \mid i \in I_{\text{up}}\right\}, \quad f_{i_{\text{low}}} = b_{\text{low}} = \max\left\{f_i \mid i \in I_{\text{low}}\right\}.$$

Obviously, the pair $(i_{\text{low}}, i_{\text{up}})$ is the most violated one. We now use (3) to check for optimality. If (3) holds, then we have converged to the optimal solution and we exit the main loop. Otherwise, we return the pair $(i_{\text{low}}, i_{\text{up}})$.

2 Pseudocode

Now we are in position to provide the complete pseudocode for the SVM optimization algorithm. The reader should be able to write down a complete program from this pseudocode.

Algorithm 1 SMO Implementation

Input:
$\boldsymbol{t}$: Target vector (entries $+1$ or -1).
$\boldsymbol{K}$: Kernel matrix.
// Initialization
$\boldsymbol{\alpha} = \mathbf{0}$;
$\boldsymbol{f} = -\boldsymbol{t}$;
Compute index sets I_{low} and I_{up};
while (1) **do** // Main Loop
  $(i, j) =$ select_pair; // Select most violated pair
  if $j = -1$ **then**
    break;
  end if
  $\sigma = t_i t_j$;
  Compute L, H from (5);
  $\eta = K_{ii} + K_{jj} - 2K_{ij}$;
  if $\eta > 10^{-15}$ **then**
    Compute the minimum along the direction of the constraint from (6);
    Clip unconstrained minimum to the ends of the line segment according to (7);
  else // the second derivative is negative
    Compute Φ_H and Φ_L accoding to (8);
    if $\Phi_L > \Phi_H$ **then**
      $\alpha_j^{new} = H$;
    else
      $\alpha_j^{new} = L$;
    end if
  end if
  $\alpha_i^{new} \leftarrow \alpha_i + \sigma(\alpha_j - \alpha_j^{new})$; // compute new α_i from the new α_j
  Update $\boldsymbol{\alpha}$ vector with the new values α_i^{new}, α_j^{new};
  $\boldsymbol{f} \leftarrow \boldsymbol{f} + t_i(\alpha_i^{new} - \alpha_i)\boldsymbol{K}_{\cdot,i} + t_j(\alpha_j^{new} - \alpha_j)\boldsymbol{K}_{\cdot,j}$;
  // $\boldsymbol{K}_{\cdot,i}$ denotes the i-th column of the kernel matrix $\boldsymbol{K}$
  Update sets I_{low}, I_{up};
end while

procedure select_pair
// Compute i_{up} and i_{low}
$i_{\text{up}} = \text{argmin}_i \{f_i \mid i \in I_{\text{up}}\}$;
$i_{\text{low}} = \text{argmax}_i \{f_i \mid i \in I_{\text{low}}\}$;
// Check for optimality
if $f_{i_{\text{low}}} \leq f_{i_{\text{up}}} + 2 \cdot \tau$ **then**
  $i_{\text{low}} = -1$;
  $i_{\text{up}} = -1$;
end if
// return the most violated pair
return $(i_{\text{low}}, i_{\text{up}})$;
end procedure

2.1 Hints for Implementation

Apart from the main loop, you do not need any loops in order to implement SMO. In particular, make sure to understand why the update of (α_i, α_j) for a violated pair (i, j) requires $O(1)$ operations only.

The update of $\boldsymbol{f} = [f_p]$ requires two columns $\boldsymbol{K}_{\cdot,i}$ and $\boldsymbol{K}_{\cdot,j}$ of the kernel matrix $\boldsymbol{K}$. If your training dataset has a size of up to 5000 or so, it makes sense to compute the entire kernel matrix $\boldsymbol{K}$ up front. Otherwise, you can compute required columns on demand. Advanced implementations maintain a cache of kernel columns. For example, you will find that indices $i \in I_0$ are frequently part of the most violated pair. However, such optimizations are not required for this project.

Whatever you do, make sure not to use any loops for evaluating the kernel matrix $\boldsymbol{K}$ or columns $\boldsymbol{K}_{\cdot,i}$ thereof. Here is how you do it. Whether you use the Gaussian or the polynomial kernel, you first compute a matrix of all pairwise inner products or squared norms, then apply the kernel function elementwise to this matrix (elementwise application of a scalar function is supported in `Matlab` and is very efficient). For the Gaussian kernel, suppose that $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ is your data matrix (n patterns, d attributes). Let $\boldsymbol{d} = \text{diag}(\boldsymbol{X}\boldsymbol{X}^T) = [\|\boldsymbol{x}_i\|^2] \in \mathbb{R}^n$, which in `Matlab` you compute as `d = sum(x.*x,2)`. Then,

$$\boldsymbol{A} = \left[(1/2)\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right] = (1/2)\boldsymbol{d}\boldsymbol{1}_n^T + (1/2)\boldsymbol{1}_n\boldsymbol{d}^T - \boldsymbol{X}\boldsymbol{X}^T.$$

And if $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{\tau}{2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}$, then $\boldsymbol{K} = f(\boldsymbol{A})$, where $f(a) = e^{-\tau a}$. In the same way, you avoid loops when computing $\boldsymbol{K}_{\cdot,i}$.