CS 229, Autumn 2009
The Simplified SMO Algorithm

1 Overview of SMO

This document describes a simplified version of the Sequential Minimal Optimization (SMO) algorithm for training support vector machines that you will implement for problem set #2. The full algorithm is described in John Platt's paper[1] [1], and much of this document is based on this source. However, the full SMO algorithm contains many optimizations designed to speed up the algorithm on large datasets and ensure that the algorithm converges even under degenerate conditions. For the data sets you will consider in problem set #2, a much simpler version of the algorithm will suffice, and hopefully give you a better intuition about how the optimization is done. However, it is important to note that the method we describe here is *not* guaranteed to converge for all data sets, so if you want to use SVMs on a real-world application, you should either implement the full SMO algorithm, or find a software package for SVMs. After implementing the algorithm described here, it should be fairly easy to implement the full SMO algorithm described in Platt's paper.

2 Recap of the SVM Optimization Problem

Recall from the lecture notes that a support vector machine computes a linear classifier of the form

$$f(x) = w^T x + b. \tag{1}$$

Since we want to apply this to a binary classification problem, we will ultimately predict $y = 1$ if $f(x) \geq 0$ and $y = -1$ if $f(x) < 0$, but for now we simply consider the function $f(x)$. By looking at the dual problem as we did in Section 6 of the notes, we see that this can also be expressed using inner products as

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \tag{2}$$

where we can substitute a kernel $K(x^{(i)}, x)$ in place of the inner product if we so desire.

The SMO algorithm gives an efficient way of solving the dual problem of the (regularized) support vector machine optimization problem, given in Section 8 of the notes. Specifically, we wish to solve:

$$\max_\alpha \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \tag{3}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \tag{4}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{5}$$

[1] This paper is available at http://research.microsoft.com/~jplatt/smo.html. One important difference is that Platt's paper uses the convention that the linear classifier is of the form $f(x) = w^T x - b$ rather than the convention we use in class (and in this document), that $f(x) = w^T x + b$.

The KKT conditions can be used to check for convergence to the optimal point. For this problem the KKT conditions are

$$\begin{aligned}
\alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 && (6)\\
\alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 && (7)\\
0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1. && (8)\\
& && (9)
\end{aligned}$$

In other words, any α_i's that satisfy these properties for all i will be an optimal solution to the optimization problem given above. The SMO algorithm iterates until all these conditions are satisfied (to within a certain tolerance) thereby ensuring convergence.

3 The Simplified SMO Algorithm

As described in Section 9 of the class notes, the SMO algorithm selects two α parameters, α_i and α_j and optimizes the objective value jointly for both these α's. Finally it adjusts the b parameter based on the new α's. This process is repeated until the α's converge. We now describe these three steps in greater detail.

3.1 Selecting α Parameters

Much of the full SMO algorithm is dedicated to heuristics for choosing which α_i and α_j to optimize so as to maximize the objective function as much as possible. For large data sets, this is critical for the speed of the algorithm, since there are $m(m-1)$ possible choices for α_i and α_j, and some will result in much less improvement than others.

However, for our simplified version of SMO, we employ a much simpler heuristic. We simply iterate over all $\alpha_i, i = 1, \ldots m$. If α_i does not fulfill the KKT conditions to within some numerical tolerance, we select α_j at random from the remaining $m-1$ α's and attempt to jointly optimize α_i and α_j. If none of the α's are changed after a few iteration over all the α_i's, then the algorithm terminates. It is important to realize that by employing this simplification, the algorithm is *no longer* guaranteed to converge to the global optimum (since we are not attempting to optimize all possible α_i, α_j pairs, there exists the possibility that some pair could be optimized which we do not consider). However, for the data sets used in problem set #2, this method achieves the same result as the selection method of the full SMO algorithm.

3.2 Optimizing α_i and α_j

Having chosen the Lagrange multipliers α_i and α_j to optimize, we first compute constraints on the values of these parameters, then we solve the constrained maximization problem. Section 9 of the class notes explains the intuition behind the steps given here.

First we want to find bounds L and H such that $L \leq \alpha_j \leq H$ must hold in order for α_j to satisfy the constraint that $0 \leq \alpha_j \leq C$. It can be shown that these are given by the following:

- If $y^{(i)} \neq y^{(j)}$, $\quad L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i)$ (10)
- If $y^{(i)} = y^{(j)}$, $\quad L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j)$ (11)

Now we want to find α_j so as to maximize the objective function. If this value ends up lying outside the bounds L and H, we simply clip the value of α_j to lie within this range. It can be shown (try to derive this yourself using the material in the class notes, or see [1]) that the optimal α_j is given by:

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta} \tag{12}$$

where

$$\begin{aligned} E_k &= f(x^{(k)}) - y^{(k)} \qquad (13)\\ \eta &= 2\langle x^{(i)}, x^{(j)}\rangle - \langle x^{(i)}, x^{(i)}\rangle - \langle x^{(j)}, x^{(j)}\rangle. \qquad (14) \end{aligned}$$

You can think of E_k as the error between the SVM output on the kth example and the true label $y^{(k)}$. This can be calculated using equation (2). When calculating the η parameter you can use a kernel function K in place of the inner product if desired. Next we clip α_j to lie within the range $[L, H]$

$$\alpha_j := \begin{cases} H & \text{if } \alpha_j > H \\ \alpha_j & \text{if } L \le \alpha_j \le H \\ L & \text{if } \alpha_j < L. \end{cases} \tag{15}$$

Finally, having solved for α_j we want to find the value for α_i. This is given by

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(\text{old})} - \alpha_j) \tag{16}$$

where $\alpha_j^{(\text{old})}$ is the value of α_j before optimization by (12) and (15).

The full SMO algorithm can also handle the rare case that $\eta = 0$. For our purposes, if $\eta = 0$, you can treat this as a case where we cannot make progress on this pair of α's.

3.3 Computing the b threshold

After optimizing α_i and α_j, we select the threshold b such that the KKT conditions are satisfied for the ith and jth examples. If, after optimization, α_i is not at the bounds (i.e., $0 < \alpha_i < C$), then the following threshold b_1 is valid, since it forces the SVM to output $y^{(i)}$ when the input is $x^{(i)}$

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(\text{old})})\langle x^{(i)}, x^{(i)}\rangle - y^{(j)}(\alpha_j - \alpha_j^{(\text{old})})\langle x^{(i)}, x^{(j)}\rangle. \tag{17}$$

Similarly, the following threshold b_2 is valid if $0 < \alpha_j < C$

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(\text{old})})\langle x^{(i)}, x^{(j)}\rangle - y^{(j)}(\alpha_j - \alpha_j^{(\text{old})})\langle x^{(j)}, x^{(j)}\rangle. \tag{18}$$

If both $0 < \alpha_i < C$ and $0 < \alpha_j < C$ then both these thresholds are valid, and they will be equal. If both new α's are at the bounds (i.e., $\alpha_i = 0$ or $\alpha_i = C$ and $\alpha_j = 0$ or $\alpha_j = C$) then all the thresholds between b_1 and b_2 satisfy the KKT conditions, we we let $b := (b_1 + b_2)/2$. This gives the complete equation for b,

$$b := \begin{cases} b_1 & \text{if } 0 < \alpha_i < C \\ b_2 & \text{if } 0 < \alpha_j < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases} \tag{19}$$

4 Pseudo-Code for Simplified SMO

In this section we present pseudo-code for the simplified SMO algorithm. This should help you get started with your implementation of the algorithm for problem set #2.

Algorithm: Simplified SMO

Input:

C: regularization parameter
tol: numerical tolerance
max_passes: max # of times to iterate over α's without changing
$(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$: training data

Output:

$\alpha \in \mathbb{R}^m$: Lagrange multipliers for solution
$b \in \mathbb{R}$: threshold for solution

- Initialize $\alpha_i = 0, \forall i, \quad b = 0$.
- Initialize $passes = 0$.
- **while** $(passes < max\_passes)$
    - $num\_changed\_alphas = 0$.
    - **for** $i = 1, \ldots m$,
        - Calculate $E_i = f(x^{(i)}) - y^{(i)}$ using (2).
        - **if** $((y^{(i)} E_i < -tol \quad \&\& \quad \alpha_i < C) \mid\mid (y^{(i)} E_i > tol \quad \&\& \quad \alpha_i > 0))$
            - Select $j \neq i$ randomly.
            - Calculate $E_j = f(x^{(j)}) - y^{(j)}$ using (2).
            - Save old α's: $\alpha_i^{(\text{old})} = \alpha_i$, $\alpha_j^{(\text{old})} = \alpha_j$.
            - Compute L and H by (10) or (11).
            - **if** $(L == H)$
              continue to next i.
            - Compute η by (14).
            - **if** $(\eta >= 0)$
              continue to next i.
            - Compute and clip new value for α_j using (12) and (15).
            - **if** $(|\alpha_j - \alpha_j^{(\text{old})}| < 10^{-5})$
              continue to next i.
            - Determine value for α_i using (16).
            - Compute b_1 and b_2 using (17) and (18) respectively.
            - Compute b by (19).
            - $num\_changed\_alphas := num\_changed\_alphas + 1$.
        - **end if**
    - **end for**
    - **if** $(num\_changed\_alphas == 0)$
      $passes := passes + 1$
    - **else**
      $passes := 0$
- **end while**

References

[1] Platt, John. *Fast Training of Support Vector Machines using Sequential Minimal Optimization*, in Advances in Kernel Methods – Support Vector Learning, B. Scholkopf, C. Burges, A. Smola, eds., MIT Press (1998).