Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines

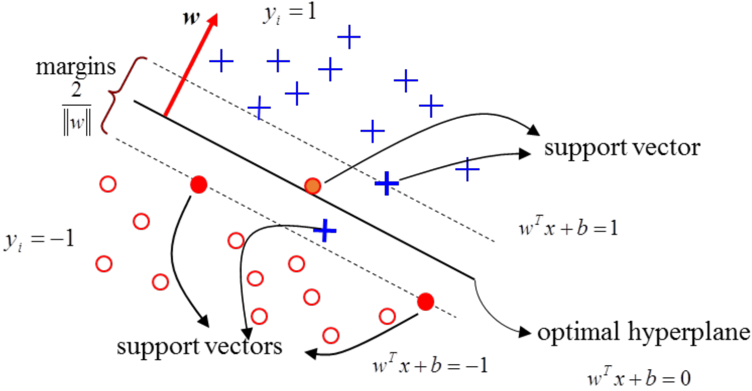
John C. Platt, 1998, Microsoft Research

Sheng-Yen Chou

Department of Computer Science
National Tsing Hua University

December 16, 2021

Support Vector Machine(SVM)



Support Vector Machine(SVM)

In classification problem, we aim to find out a pair of parallel decision planes $w^\top \mathbf{x}_+ + b_+ \geq 1$ and $w^\top \mathbf{x}_- + b_- \leq -1$ such that we can separate two groups of data points $\mathbf{x}_+, \mathbf{x}_-$ perfectly and enlarge the gap $\frac{2}{||w||}$ between the nearest decision planes.
Then we can write down the optimization problem as

$$\max_{w,b} \frac{2}{||w||} \text{ subject to } \mathbf{y}^\top \odot (w^\top \mathbf{x} + b) \geq 1$$

In terms of minimization

$$\min_{w,b} \frac{||w||}{2} \text{ subject to } \mathbf{y}^\top \odot (w^\top \mathbf{x} + b) \geq 1$$

Where the operator $\odot$ means the element-wise product and the norm $||\cdot||$ is the Euclidean(L2) norm.

Lagrange Function

Let $\alpha \in \mathbb{R}^N, \alpha > 0$ be the Lagrange multiplier, the Lagrangian function is

$$L(w, b, \alpha) = \frac{||w||}{2} - (\mathbf{y}^\top \odot (w^\top \mathbf{x} + b) - 1)\alpha$$

$$= \frac{1}{2} w^\top w - \sum_{i=1}^{N} \alpha_i (y_i (w^\top x_i + b) - 1)$$

We aim to minimize the Lagrangian function to solve the constraint optimization problem

$$\min_{w,b,\alpha} L(w, b, \alpha), \ \alpha > 0$$

Derive Dual Problem

A common way to obtain the minimum of the function is letting the gradient of the function be 0.
As for variable w, we can derive

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \quad \rightarrow \quad w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

As for variable b, we can derive

$$\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_{i=1}^{N} -\alpha_i y_i = 0 \quad \rightarrow \quad \sum_{i=1}^{N} \alpha_i y_i = 0$$

Derive Dual Problem

Then, we can substitute w and b of the Lagrangian function $L(w, b, \alpha)$ as

$$L(w, b, \alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^\top x_j$$

Consider kernel trick, we can replace the inner product as kernel function $k(x_i, x_j) = \phi(x_i)\phi(x_j)$ with embedding function $\phi(\cdot)$

$$L(w, b, \alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

Now, we can rewrite the optimization problem of hard-margin SVM as

$$\sup_\alpha \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$
$$\text{subject to } 0 \leq \alpha_i, \sum_{i=1}^{N} \alpha_i y_i = 0$$

Quadratic Programming

General Form of Quadratic Programming

$$\min_x g(x) = x^\top Gx + x^\top c$$

$$\text{s.t. } \mathrm{a}_i^\top x = b_i, i \in \mathcal{E}$$
$$a_i^\top x \geq b_i, i \in \mathcal{I}$$

The optimization problem of soft-margin SVM is

$$\sup_\alpha \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$
$$\text{subject to } 0 \leq \alpha_i \leq C, \sum_{i=1}^N \alpha_i y_i = 0$$

Which satisfies quadratic programming absolutely and C is the hyperparameter of soft-margin.

Issues of traditional ways to solve quadratic programming

- Most of methods like Gradient projection method, Newton's method... solve the whole quadratic programming in one time. However, SVM has a large matrix to solve(about N^2) and it would be very time consuming.
- How about **divide-and-conquer**?

Divide-and-Conquer

Osuna's Theorem

Divide the variables $\alpha_1, \alpha_2, ..., \alpha_N$ into 2 groups as α_B and α_H. Fix the values of α_H and only update the variables of α_B. Moving a variable that violates the optimality conditions(KKT conditions in SVM) from α_N to α_B gives a strict improvement in the cost function when the sub-problem is re-optimized.

- According If we can optimize a variable that violating the KKT conditions each time, then the target function(cost function) would converge.
- A series of works are proposed based on Osuna's Theorem, like Chunking, Osuna's algorithm... But still, the sub-problem isn't small enough.
- SMO only solves the minimal size of sub-problem while considering KKT conditions, that is 2-variable quadratic programming.

Sequential Minimal Optimization(SMO)

Sequential Minimal Optimization(SMO)

The SMO(Sequential Minimal Optimization) algorithm is proposed from the paper Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines in 1998 by J. Platt.

There are 3 steps

- Step 1. Select and Update
  Select 2 variables α_i, α_j and update
- Step 2. Bosk Constraint
  Clip the value of α_j with complementary slackness
  Derive the new values α_i^*, α_j^*
- Step 3. Update Bias
  Derive new bias b^* from α_i^*, α_j^*

Step 1. Select and Update

Denote x_i, y_i as i-th data point and label. Let $K_{i,j} = k(x_i, x_j)$, where $k(a, b)$ is the kernel function and $f_\phi(x_i)$ is the prediction function.

- In order to optimize the target function $L(w, b, \alpha)$, we can derive the gradient of it and let the gradient be 0.

$$\frac{\partial L(w, b\alpha)}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0 \rightarrow w = \sum_{i=1}^{N} \alpha_i y_i x_i$$

- In order to hold the constraint, both new variables and old variables should satisfy the constraint

$$\sum_{i=1}^{N} \alpha_i y_i = \alpha_1 y_1 + \alpha_2 y_2 + \sum_{i=3}^{N} \alpha_i y_i = 0$$

$$\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1 y_1 + \alpha_2 y_2 = -\sum_{i=3}^{N} \alpha_i y_i = \zeta$$

Step 1. Select and Update

Plug the two equations into the target function and we can derive

$$E_i = f(x_i) - y_i,\ E_j = f(x_j) - y_j$$

$$\eta = K_{i,i} + K_{j,j} - 2K_{i,j}$$

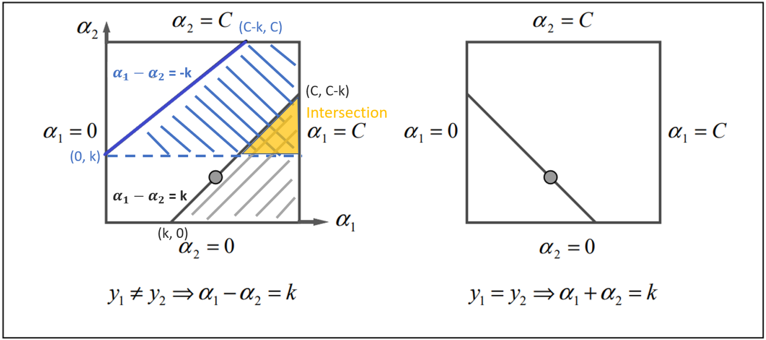
Then, we get a new value of α_j

$$\alpha_j^{new} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta}$$

To understand intuitively, you can see η as **Learning Rate** and $y_j(E_i - E_j)$ as a kind of **Loss**.

Step 2. Bosk Constraint

To satisfy the complementary slackness $\alpha_1 y_1 + \alpha_2 y_2 = \zeta,\ 0 \leq \alpha_i \leq C$, we need to **clip** the α_j^{new} under blue and grey area.



Step 2. Bosk Constraint

if $y_i = y_j$ **then**

$B_U = \min(C, \alpha_j + \alpha_i)$

$B_L = \max(0, \alpha_j + \alpha_i - C)$

else

$B_U = \min(C, C + \alpha_j - \alpha_i)$

$B_L = \max(0, \alpha_j - \alpha_i)$

end if

$\alpha_j^* = CLIP(\alpha_j^{new}, B_L, B_U)$

$\alpha_i^* = \alpha_i + y_i y_j(\alpha_j - \alpha_j^*)$

Step 3. Update Bias

We denote the prediction of the SVM as $f_\phi(x)$

$$f_\phi(x_p) = w^\top \phi(x_p) + b = b + \sum_{i=1}^{N} \alpha_i y_i k(x_i, x_p)$$

While $f_\phi^*(x)$ is the prediction of the SVM with new variables α_1^*, α_2^*, and b^*

$$f_\phi^*(x_p) = \sum_{i=3}^{N} \alpha_i y_i K_{i,p} + \alpha_1^* y_1 K_{1,p} + \alpha_2^* y_2 K_{2,p} + b^* = y_p$$

Step 3. Update Bias

Thus, we can derive 3 cases

- CASE 1. If $0 < \alpha_1^* < C$, the data point x_1 should right on the margin and $f_\phi^*(x_1) = y_1$. The bias derived from α_1.

$$b_1^* = y_1 - \sum_{i=3}^{N} \alpha_i y_i K_{i,1} - \alpha_1^* y_1 K_{1,1} - \alpha_2^* y_2 K_{2,1}$$

$$= -E_1 - y_1 K_{1,1}(\alpha_1^* - \alpha_1) - y_2 K_{2,1}(\alpha_2^* - \alpha_2) + b$$

- CASE 2. If $0 < \alpha_2^* < C$, the data point x_2 should right on the margin and $f_\phi^*(x_2) = y_2$. The bias derived from α_2.

$$b_2^* = y_2 - \sum_{i=3}^{N} \alpha_i y_i K_{i,2} - \alpha_1^* y_1 K_{1,2} - \alpha_2^* y_2 K_{2,2}$$

$$= -E_2 - y_1 K_{1,2}(\alpha_1^* - \alpha_1) - y_2 K_{2,2}(\alpha_2^* - \alpha_2) + b$$

Step 3. Update Bias

- CASE 3. When the data point x_i, x_j are both not on the margin, we choose the average of b_1^*, b_2^* as the updated value.

$$b^* = \frac{b_1^* + b_2^*}{2}$$

Step 3. Update Bias

When $0\alpha_i^* C$, the data point x_i is right on the margin such that $f_\phi(x) = y_i$.

$b_i^* = -E_i - y_i K_{i,i}(\alpha_i^* - \alpha_i) - y_j K_{j,i}(\alpha_j^* - \alpha_j) + b$
$b_j^* = -E_j - y_i K_{i,j}(\alpha_i^* - \alpha_i) - y_j K_{j,j}(\alpha_j^* - \alpha_j) + b$
if $0 \leq \alpha_i \leq C$ **then**
  $b^* = b_i^*$
else if **then**$0 \leq \alpha_j \leq C$
  $b^* = b_j^*$
else
  $b^* = \frac{b_i^* + b_j^*}{2}$

Benchmark

The timing performance of the SMO algorithm versus the chunking algorithm for the linear SVM on the adult data set is shown in the table below:

Training Set Size	SMO time	Chunking time	Number of Non-Bound Support Vectors	Number of Bound Support Vectors
1605	0.4	37.1	42	633
2265	0.9	228.3	47	930
3185	1.8	596.2	57	1210
4781	3.6	1954.2	63	1791
6414	5.5	3684.6	61	2370
11221	17.0	20711.3	79	4079
16101	35.3	N/A	67	5854
22697	85.7	N/A	88	8209
32562	163.6	N/A	149	11558

The timing performance of SMO and chunking using a Gaussian SVM is shown below:

Training Set Size	SMO time	Chunking time	Number of Non-Bound Support Vectors	Number of Bound Support Vectors
1605	15.8	34.8	106	585
2265	32.1	144.7	165	845
3185	66.2	380.5	181	1115
4781	146.6	1137.2	238	1650
6414	258.8	2530.6	298	2181
11221	781.4	11910.6	460	3746
16101	1784.4	N/A	567	5371
22697	4126.4	N/A	813	7526
32562	7749.6	N/A	1011	10663

Thanks For Listening