SMO 方法的实现及证明

1. 问题的阐述

SVM 是从线性可分情形下的最优分类面发展而来。基本思想可以用图(2-16)的二维情况说明。

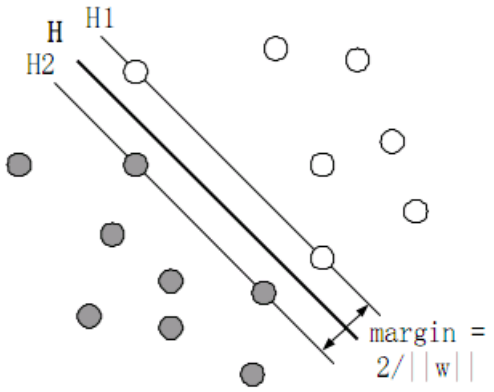


图 2-16 线性可分情况下的最优分类线

图中实心点和空心点代表两类样本，H 为分类线 H1,H2 分别为过各类中离分界线最近的样本且平行于分类线的直线，它们之间的距离叫做分类间隔(margin)。所谓最优分类线就是要求分类线不仅能将两类正确分开(训练错误率为 0)，而且使分类间隔最大。推广到一般线性可分情形，假设分类方程为 $<\mathbf{x},\boldsymbol{\omega}>+b=0$，对其进行归一化，样本集 $(\mathbf{x}_i, y_i), i=1,2,\ldots,n, \mathbf{x}\in R^d, y\in\{+1,-1\}$，满足

$$y_i(<\mathbf{x}_i,\boldsymbol{\omega}>+b)-1\geq 0 \tag{1}$$

构造损失函数作为目标函数及约束条件，即：

$$\text{minimize}: W(\boldsymbol{\alpha})=\frac{\|w\|^2}{2}+C\sum_i \xi_i \tag{2-a}$$

$$\text{subject to } y_i\left(w^T x_i+b\right)\geq 1-\xi_i, \forall i \tag{2-b}$$

$$\xi_i\geq 0, \forall i \tag{2-c}$$

经过拉格朗日变换以及 KKT 定理推导，式子变为：

$$\begin{aligned}&\text{minimize: } W(\alpha)=\sum_i \alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i x_j\\&\text{subject to } 0\leq\alpha_i\leq C \quad \sum_i \alpha_i y_i=0\end{aligned} \tag{3}$$

引入核函数，最后的目标函数变为：

$$\begin{aligned}&\text{maximize}: W(\alpha)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K\left(\mathbf{x_i},\mathbf{x_j}\right)\\&\text{subject to } \sum_{i=1}^{n}\alpha_i y_i=0 \quad 0\le\alpha_i\le C,\forall i\end{aligned} \tag{4}$$

改写为矩阵相乘的格式，得到：

$$\begin{aligned}&\min \quad f(\boldsymbol{\alpha})=\frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{Q}\boldsymbol{\alpha}-\mathbf{e}^{T}\boldsymbol{\alpha}\\&\text{subject to } \mathbf{y}^{\mathrm{T}}\boldsymbol{\alpha}=0 \quad 0\le\alpha_i\le C, i=1,....,l\end{aligned} \tag{5}$$

其中 e 为全 1 向量，C为所有变量的上界，Q为$l\times l$的半正定矩阵。训练向量$\mathbf{x_i}$通过ϕ函数被映射到更高维的空间(可能为无穷维)，$Q_{ij}\equiv y_i y_j K(\mathbf{x}_i,\mathbf{x}_j)$，其中$K(\mathbf{x},\mathbf{y})=\phi(\mathbf{x})^T\phi(\mathbf{y})$为核函数。简言之，即通过训练样本寻找$\boldsymbol{\alpha}$，使得$\frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{Q}\boldsymbol{\alpha}-\mathbf{e}^{T}\boldsymbol{\alpha}$最小。

2. 最小贯序列方法(SMO：Sequential Minimal Optimization)

SMO 是改进的 SVM 训练算法。同其他 SVM 训练 算法一样，SMO 将一个大的 QP 问题分解为一系列小的 QP 问题。与其他算法不一样的是，SMO 处理的是规模最小的 QP 问题，因此能够快速解决并获得解析解，能够有效的改善空间和时间复杂度。

SMO 的优点：

1. 即使很多拉格朗日乘子在界上，SMO 仍然能够较好地处理；
2. SMO 训练线性核 SVM 时性能优异，因为 SMO 的计算时间主要集中在 SVM 的计算上，而线性核 SVM 的计算可以表示为简单的内积，而非线性核的和；
3. SMO 能够很好地处理输入为稀疏向量的情形，因为核计算时间的减少使 SVM 训练过程加速。因为块分解法花费的时间主要在跑 QP 代码上，因此无法有效利用 SVM 的线性特性或者输入向量的稀疏性。
4. SMO 最大的优点在于能够有效处理大规模 QP 问题，因为它减小训练集规模优于块分解方法。

使用最小贯序列方法（SMO）每次仅选择两个元素共同优化，在其他参数固定的前提下，找到这两个参数的最优值，并更新相应的α向量，而这两个点乘子的优化可以获得解析解。尽管需要更多的迭代才收敛，每次迭代需要很少的操作，因此算法在整体上的速度上有数量级的提高。包括

收敛时间在内，算法其他特征没有矩阵操作，它不需要存储核矩阵，所需要的内存大小和训练数据集的大小线性增长，因此也有效降低了空间复杂度。

通常来说，获得精确的最优值是不太现实的，因此需要定义近似最优条件，因此后面提及优化一词时，将代表近似最优解。实现技术主要考虑两个步骤：如何选择工作集；如何更新两个点乘子。

2.1 工作集选择

因为 SMO 方法每一步选择和更新两个拉格朗日乘子，并且至少有一个拉格朗日乘子在更新前违背了 KKT 条件，因此根据 Osuna 原理每一步能够减小目标函数值。因此迭代收敛能够得到保证。为了能够快速收敛，SMO 启发式地选择两个拉格朗日乘子进行优化。

2.1.1 Platt 方法[1]

根据 KKT 条件，可以得出：

$$\alpha_i = 0 \Leftrightarrow y_i f(x_i) \geq 1 \qquad (6a)$$

$$0 < \alpha_i < C \Leftrightarrow y_i f(x_i) = 1 \qquad (6b)$$

$$\alpha_i = C \Leftrightarrow y_i f(x_i) \leq 1 \qquad (6c)$$

其中，

$$f(x_i) = \sum_{j=1}^{l} y_j \alpha_j K(\mathbf{x}_i \cdot \mathbf{x}_j) + b \qquad (7)$$

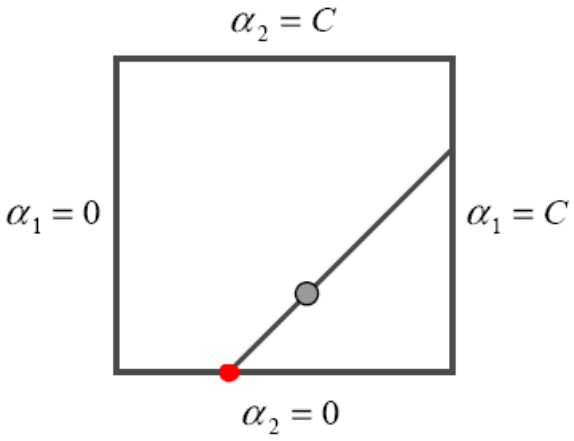
Platt 方法通过式（6）检测 α_i 是否满足 KKT 条件。它将启发式选择第一个和第二个拉格朗日乘子视为独立的过程。选择第一个乘子作为 SMO 算法的外层循环。任何违反 KKT 条件(6)的乘子（或样本）都是合法的第一个拉格朗日乘子。外层循环中分两种情况选择第一个乘子：

A. 在所有乘子中选择（第 1 次或者 $\alpha_i \in (0,C)$ 没有违反 KKT 条件乘子的情形）；

B. 在界内 $\alpha_i \in (0,C)$ 中选择（其它情形，这是常规情况）。

外层循环分两种情况选择第一个乘子的理由：当 SMO 算法有进展时，界上的乘子（即等于 0 或者 C 的乘子）更新后很有可能仍然停留在界上，界内的乘子才发生改变，因此首先在界内乘子中选择，可以加快算法的运行时间。

界上乘子更新后很有可能仍然停留在界上的解释：



$$y_1 \neq y_2 \Rightarrow \alpha_1 - \alpha_2 = k$$

上图红点表示待更新的两个乘子的对应坐标，其中 α_2 在界上 $\alpha_2 = 0$，假设核矩阵正定，根据

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

$\eta > 0$,因此若 α_2 更新后值发生变化，必须 $y_2(E_1 - E_2) > 0$，而在后面的工作集选择并不能够确保该式成立，若 $y_2(E_1 - E_2) \leq 0$，则 α_2 的值保持不变，因此更新失败，仍然留在界上。其他情形依次类推。

总言之，外循环首先对整个训练集进行迭代，确定是否每一个实例都违背 KKT 条件。如果一个实例违背了 KKT 条件，它就作为待优化的候选乘子。**遍历一次整个训练集**后（最多在整个训练集寻找一次违背 KKT 条件的 α_2），外循环对所有拉格朗日乘子既非 0 也非 C(non-bound,注意 non-bound 是针对 $0 \leq \alpha_i \leq C$ 这一约束条件而言的)的例子进行迭代，重复进行，每个例子检查是否满足 KKT 条件，违背该条件的实例作为优化的候选实例。外循环**不断重复对这些非边界实例的遍历**直到所有的实例在 ε 内满足 KKT 条件。外循环返回并对整个训练集进行迭代。

需要注意的事 KKT 条件是在精度 ε 内检查的。一般来说，ε 设为 10^{-3}。识别系统一般不要求在较高精确度上检查 KKT 条件：例如正界输出 0.999 到 1.001 之间的数是可接受的。如果要求很高的输出精度，SMO 算法将无法快速收敛。

一旦第一个乘子被选中，SMO 以最大化优化时的步长为目标来选择第二个乘子。现在计算核函数 K 需要一定的时间花销，根据

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

其中

$$E_i = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x_i) - y_i$$

$$F_i = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x_i) + b - y_i$$

$$= f(x)_i - y_i = E_i + b$$

第二个乘子的迭代步长大致正比于 $E_1 - E_2$，因此应选择第二个乘子最大化 $|E_1 - E_2|$，即当 E_1 为正时最小化 E_2， E_1 当为负时最大化 E_2。

使用上述第二个乘子的启发式选择在一些特殊的情况，SMO 无法有正的进展（positive progress）。例如，第一个和第二个乘子指向同一个输入向量。在这种情况，SMO 对第二个乘子的启发式选择使用分层的方法指导它寻找到一个拉格朗日乘子对能够实现正的进展，所谓正的进展通过乘子优化后两个乘子的增量非零。分层的启发式选择第二个乘子的方法描述如下，如果上述的启发式选择无法实现正的进展，SMO 开始在非边界实例中迭代，寻找能够实现正的进展的实例，如果没有非边界实例能够实现正的进展，SMO 开始在整个训练集中迭代指导寻找到能够实现正的进展的实例。为了避免 SMO 倾向于训练集中位置靠前的训练集，在非边界实例中和在整个训练集中寻找都从一个随机位置开始。在一些极其特殊的情形，没有实例能够满足要求，此时，第一个实例将被忽略，SMO 继续寻找第一个实例。

$b, f(x_i), F$ 的更新：

（1） b 的更新

为使乘子 α_1 或者 α_2 的 KKT 条件成立。若 α_1^{new} 在界内，则有 $\mathrm{E}_1^{new} = 0$ 根据

$$\mathrm{F}_1^{new} = \sum_{j=1}^{n} \alpha_j y_j K_{1j} + b_1^{new} - y_i = 0$$

$$\mathrm{F}_1^{\mathrm{old}} = \sum_{j=1}^{n} \alpha_j y_j K_{1j} + b - y_i$$

可以得到：

$$b_1^{new} = b - \mathrm{F}_1^{\mathrm{old}} - y_1(\alpha_1^{new} - \alpha_1^{old})K(x_1, x_1) - y_2(\alpha_2^{new} - \alpha_2^{old})K(x_2, x_1)$$

同理，若 α_2^{new} 在界内，则有

$$b_2^{new} = b - \mathrm{F}_2^{\mathrm{old}} - y_1(\alpha_1^{new} - \alpha_1^{old})K(x_2, x_1) - y_2(\alpha_2^{new} - \alpha_2^{old})K(x_2, x_2)$$

如果 α_1^{new} 和 α_2^{new} 都在界内，根据 $\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(F_1 - F_2)}{\kappa} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\kappa}$ 不难得出

$b_1^{new}=b_2^{new}$，则有

A. 若α_1^{new}和α_2^{new}有一个在界内，b取对应b_i^{new}（i=1,2）的，如α_1^{new}在界内，b 取b_1^{new}；

B. 若α_1^{new}和α_2^{new}都在界上，那么b_1^{new}和b_2^{new}之间的任意数都满足 KKT 条件，都可作为b的更新值，一般取$b^{new}=\frac{b_1^{new}+b_2^{new}}{2}$。

下面给出$b^{new}=tb_1^{new}+(1-t)b_2^{new},t\in(0,1)$满足 KKT 条件的证明。由于两个乘子都在界上，因此α_2^{new}经过了裁剪，令

$$\alpha_2^{new}=\alpha_2^{old}+\lambda\frac{y_2(F_1-F_2)}{\eta},\lambda\in[0,1) \qquad (*)$$

化简得

$$\begin{aligned}F_1^{new}&=(1-\lambda)(1-t)(F_1-F_2)\\F_2^{new}&=-t(1-\lambda)(F_1-F_2)\end{aligned}\quad\lambda\in[0,1),t\in(0,1)$$

i. 若$\alpha_1^{new}=\alpha_2^{new}$（同为 C 或者 0）

1) $y_1\neq y_2$。不妨令$\alpha_1^{new}=\alpha_2^{new}=C$，由于（*）式，所以$y_2(F_1-F_2)>0$，可得$y_2F_2^{new}<0$，亦可得到$y_1F_1^{new}<0$，因此更新的两个乘子都满足 KKT 条件，同理可证明$\alpha_1^{new}=\alpha_2^{new}=0$的情形。

2) $y_1=y_2$，$\alpha_1=\alpha_2=\alpha_1^{new}=\alpha_2^{new}$（此时，$\lambda=0$），此时最多只有一个乘子满足 KKT 条件，根据（16-b）计算 L 和 H，可得到 L=H，因此遇到该情形时，将不会进行后面的更新。

ii. 若$\alpha_1^{new}\neq\alpha_2^{new}$（一个为 0，另一个为 C）

1) $y_1=y_2$不妨令$\alpha_2^{new}=C,\alpha_1^{new}=0$，由于（*）式，所以$y_2(F_1-F_2)>0$，则有$y_2F_2^{new}<0$,亦可得到$y_1F_1^{new}>0$，因此更新的两个乘子都满足 KKT 条件，同理可证明$\alpha_2^{new}=0,\alpha_1^{new}=C$的情形。

2) $y_1\neq y_2$，$\alpha_2^{new}=\alpha_2$，$\alpha_1^{new}=\alpha_1$，$y_1F_1^{new}$与$y_2F_2^{new}$同号，不可能同时满足 KKT 条件。根据（16-a）计算 L 和 H，可得到 L=H，因此遇到该情形时，将不会进行后面的更新。

（2） $f(\mathbf{x})_i$的更新

$$f(\mathbf{x})_i^{new} = \sum_{j=1}^{n} \alpha_j^{new} y_j K(x_j, x_i) + b^{new}$$

（3） $F(\mathbf{x})_i$的更新

$$F(\mathbf{x})_i^{new} = f(\mathbf{x})_i^{new} - y_i$$

```
procedure examineExample(i2)
   y2 = target[i2]
   alph2 = Lagrange multiplier for i2
   F2 = SVM output on point[i2] - y2 (check in error cache)
   r2 = F2*y2
   if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
   {
     if (number of non-zero & non-C alpha > 1)
     {
       i1 = result of second choice heuristic (section 2.2)
       if takeStep(i1,i2)
         return 1
     }


     loop over all non-zero and non-C alpha, starting at a random
point
     {
        i1 = identity of current alpha
        if takeStep(i1,i2)
          return 1
     }
     loop over all possible i1, starting at a random point
     {
        i1 = loop variable
        if (takeStep(i1,i2)
          return 1
```

```
      }
    }
    return 0
endprocedure
main routine:
     numChanged = 0;
     examineAll = 1;
     while (numChanged > 0 | examineAll)
     {
         numChanged = 0;
         if (examineAll)
             loop I over all training examples
                 numChanged += examineExample(I)
         else
             loop I over examples where alpha is not 0 & not C
                 numChanged += examineExample(I)
         if (examineAll == 1)
             examineAll = 0
         else if (numChanged == 0)
             examineAll = 1
    }
```

该工作集选择方法存在一些不足，导致算法效率较低。因为它计算和使用一个单一阈值(b)的方式，可能会带来一些不必要的低效。在任何情况，SMO 根据两个已经优化的乘子来修改 b。但是，当算法检查剩下的实例是否违背 KKT 条件时，很有可能存在一个不同的 b 值能够工作得更好。所以在 SMO 算法中，很有可能存在这种情况，尽管 α 已经迭代得到了一个满足 $b_{up} \geq b_{low}$ 的值，SMO 却因为当前的 b 值不合适而无法检测到该情况。在这种情况下，SMO 为了进行更新步骤不得不付出大量且不必要的资源以搜索第二个乘子。

2.1.2 Keerthi 方法[2]

(1)步骤

1） 把训练样例分为五个集合：

$$\begin{aligned} &I_0=\{i:0<\alpha_i<C\};\\ &I_1=\{i:y_i=+1,\alpha_i=0\};\\ &I_2=\{i:y_i=-1,\alpha_i=C\};\\ &I_3=\{i:y_i=+1,\alpha_i=C\};\\ &I_4=\{i:y_i=-1,\alpha_i=0\}. \end{aligned} \tag{8}$$

2）计算 b_{up} 和 b_{low}，α_1 和 α_2 就是 b_{up} 和 b_{low} 对应的乘子：

$$\begin{aligned} &b_{up}=\min\{E_i:i\in I_0\cup I_1\cup I_2\}\\ &b_{low}=\max\{E_i:i\in I_0\cup I_3\cup I_4\}\\ &\text{其中 } E_i=\sum_{j=1}^{n}\alpha_j y_j K(x_j,x_i)-y_i \end{aligned} \tag{9}$$

3）计算 $b_{up}\geq b_{low}$，则终止迭代，否则进行两个乘子的更新操作。

(2)证明

根据 KKT 定理有：

$$\nabla f(\boldsymbol{\alpha})+b\mathbf{y}=\boldsymbol{\lambda}-\boldsymbol{\mu} \tag{10-a}$$

$$\lambda_i\alpha_i>0,\mu_i(C-\alpha_i)=0,\lambda_i\geq 0,\mu_i\geq 0 \tag{10-b}$$

注意：$\nabla f(\boldsymbol{\alpha})=\mathbf{Q}\boldsymbol{\alpha}-\mathbf{e}$，即为 $f(\boldsymbol{\alpha})$ 的梯度，结合（10-a）和（10-b）可以得到：

当 $\alpha_i>0$ 时， $\lambda_i=0,\mu_i\geq 0$，则有

$$\nabla f(\boldsymbol{\alpha})_i+by_i\leq 0 \tag{11-a}$$

当 $\alpha_i<C$ 时， $\lambda_i\geq 0,\mu_i=0$，则有

$$\nabla f(\boldsymbol{\alpha})_i+by_i\geq 0 \tag{11-b}$$

通过（9-a）和（9-b）可知，当 $\alpha_i>0,y_i=-1$ 或者 $\alpha_i<C,y_i=1$ 时，将对应的索引集合用 I_{up} 表示，有

$$y_i\nabla f(\boldsymbol{\alpha})_i\geq -b \tag{12-a}$$

同理，当 $\alpha_i>0,y_i=1$ 或者 $\alpha_i<C,y_i=-1$ 时,将对应的索引集合用 I_{low} 表示，有

$$y_i \nabla f(\boldsymbol{\alpha})_i \le -b \qquad (12\text{-b})$$

因此，若 $\boldsymbol{\alpha}^*$ 为可行解，设 $m(\boldsymbol{\alpha}^*) = \min(\nabla f(\boldsymbol{\alpha}^*)_i), i \in I_{up}$ 以及 $M(\boldsymbol{\alpha}^*) = \max(\nabla f(\boldsymbol{\alpha}^*)_i), i \in I_{low}$，则有 $m(\boldsymbol{\alpha}^*) \ge M(\boldsymbol{\alpha}^*)$，实际上，已经证明 $\boldsymbol{\alpha}^*$ 为可行驻点当且仅当 $m(\boldsymbol{\alpha}^*) \ge M(\boldsymbol{\alpha}^*)$,因此可作为终止条件。

不难得到

$$\nabla f(\boldsymbol{\alpha})_i = y_i \sum_{j=1}^{l} \alpha_j y_j K(\mathbf{x_i}, \mathbf{x_j}) - 1 \qquad (13)$$

由（10-a）及（10-b）可以得到，当 $0 < \alpha_i < C$ 时，则有

$$\nabla f(\boldsymbol{\alpha})_i + b y_i = 0, \qquad (14)$$

将式（13）代入式子（14），化简得到

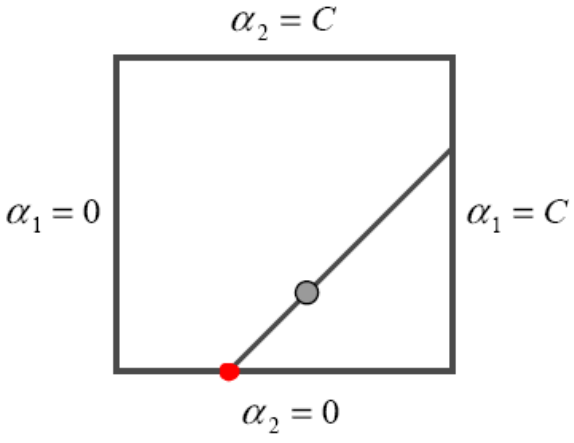
$$y_i \left(\sum_{j=1}^{l} \alpha_j y_j K(\mathbf{x_i}, \mathbf{x_j}) + b \right) = 1 \qquad (15)$$

因此式（12-a）中的 b 即为式（2-b）中的 b。

注意该方法较 Platt 方法而言，不会出现界上乘子（即等于 0 或者 C 的乘子）更新后很有可能仍然停留在界上的情况，以下图为例，红点表示更新前乘子对的对应值，（$\alpha_2 = 0$，在界上），α_2 对应的 E 值为 b_{low}，因此 $y_2 = -1$，根据

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2 (E_1 - E_2)}{\eta}$$

$\eta > 0$，易有 $y_2 (E_1 - E_2) > 0$，因此 α_2 的值必定能够更新成功，依次类推，所以该方法不需要像 Platt 方法那样先考虑更新非边界值。



$$y_1 \ne y_2 \Rightarrow \alpha_1 - \alpha_2 = k$$

该方法一些优化技巧：

1） 假定在任何情况下，E_i对所有i可用。假设i_low是b_{low}对应的索引，i_up是b_{up}对应的索引，此时，寻找特定的用于优化的 i 非常方便。例如，假设$i \in I_1 \cup I_2$，我们只需要检查$E_i < E_{i_low} - 2\varepsilon$是否成立，如果成立，则认为出现了违反对$\{i_low, i\}$。因此，寻找违反对是同时进行的，这与原始的 SMO 算法是不同的。正如下面所看到的，我们通过高效的更新方法计算和使用（i_low，b_{low}）和(i_up, b_{up})。

2) 为了达到高效，我们将在更新$\alpha_i, i \in I_0$做出大量努力，缓存E_i，$i \in I_0$被保持和更新。对所有的$i \in I_0$都满足优化条件时，将检查所有的索引值进行优化。

3) 一些额外的步骤需要添加到 TakeStep 过程中。**当使用（i_2, i_1）作为索引对成功更新后，令$\bar{I} = I_0 \cup \{i_1, i_2\}$。我们只计算不完全的（$i\_low$，$b_{low}$）和$(i_up, b_{up})$，也就是仅使用$\bar{I}$（$\bar{I}$表示被更新过的元素，因此索引值 i 对应的 Ei 有可能为 0 或者 C）**。注意这些额外的步骤只需要少量的花费，因为缓存E_i，$i \in I_0$是可用的，并且能很快地更新E_{i1}和E_{i2}。因为（i_low，b_{low}）和(i_up, b_{up})可能从$\{i_2, i_1\}$中产生并且在后续的步骤中作为i_1的选择，因此我们将E_{i1}和E_{i2}的值存储在缓存中。

改进：实验表明直接从活动集中寻找(i_2, i_1)比上述算法更加高效，作者考虑使用上述步骤可能是受 Platt 的影响，认为界上乘子很有可能更新失败，实际上，使用该方法更新边界乘子一定成功（已证明），因此很有可能得到更大的步长加速收敛。

4）当仅作用在$\alpha_i, i \in I_0$时，也即一个 ExamineAll=0 的循环。注意当$b_{up} \le b_{low} - 2\tau$在某个点成立，说明对应的乘子为最大违反对(maximal violating pair)，否则跳出 ExamineAll=0 的循环。

5) Keerthi 提出了两种实现在I_0集合中选择最大违反对的方法（ExamineAll=0）

方法 1：遍历$i_2 \in I_0$，对每个i_2，检验是否违反 KKT 条件，如果违反，选择合适的i_1，例如，检验是否满足$E_{i2} \le b_{low} - 2\tau$,如果满足，可选择$i_1 = i_low$。

方法 2：总是选择最坏的违反对；选择$i_2 = i_low, i_1 = i_up$。

需要注意的是，方法 1 和 Platt 方法除了判断是否满足优化条件外其余都是相同的。方法 2 可以认为是方法 1 的进一步优化，因为在方法 2 有效利用了缓存寻找违反对。

6）当所有$i \in I_0$对应的乘子都满足最优条件时，即I_0中找不到违反对，算法返回到(ExamineAll=1)，在整个训练集中检查最有条件。我们依次遍历所有索引。因为通过I_0我们计算出部分的（i_low，b_{low}）和(i_up, b_{up})，检查每一个索引i并对这些值进行更新。对于一个给定的i，先计算E_i，然后通过当前的（i_low，b_{low}）和(i_up, b_{up})检查最优条件，如果不违反，E_i用于更新这些值。例如，若$i \in I_1 \cup I_2$并且$E_i \le b_{low} - 2\tau$,表明(i, i_low)是一个违反对，对应两个乘子进行更新操作，否则E_i用于更新(i_up, b_{up})，也即$E_i < b_{up}$时，我们进行下列操作：$i\_up := i$，$b_{up} := E_i$。

7） 当 ExamineAll=1 并且无违反对出现时，所有索引对应的乘子满足最优条件，优化结束。

2．2 更新两个乘子α_1和α_2 [3]

对条件$\sum_{j=1}^{l} \alpha_j y_j = 0$需要在迭代中实现，意味着每步能优化的乘子的最小个数为 2；无论何时一个乘子被更新，至少需要调整另一个乘子来保持条件成立。因此首先需要对首先被更新的乘子进行上下界约束，使得两个乘子同时满足$0 < \alpha_i < C$的约束。

2.2.1 步骤

1）计算上下界 L 和 H, 若 L=H 时返回。

$$L = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right), H = \min\left(C, C + \alpha_2^{old} - \alpha_1^{old}\right), \text{ if } y_1 \neq y_2 \quad (16\text{-a})$$

$$L = \max\left(0, \alpha_2^{old} + \alpha_1^{old} - C\right), \ H = \min\left(C, \alpha_2^{old} + \alpha_1^{old}\right), \text{ if } y_1 = y_2 \quad (16\text{-b})$$

2）计算W_s的二阶导数

$$\eta = 2\phi(x_1)^T \phi(x_2) - \phi(x_1)^T \phi(x_1) - \phi(x_2)^T \phi(x_2)$$

3）更新α_2

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

$$E_i = \sum_{j=1}^{n} \alpha_j y_j K(x_j, x_i) - y_i$$

4）计算修剪后的α_2

$$\alpha_2^{new,clip} = \begin{cases} H, & \alpha_2^{new} \ge H \\ \alpha_2^{new}, & L \le \alpha_2^{new} \le H \\ L, & \alpha_2^{new} \le L \end{cases}$$

5）更新 α_1

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new,clip})$$

6）更新所有 E_i

$$E_i^{new} = E_i^{old} + y_1(\alpha_1^{new} - \alpha_1^{old})K(x_1, x_i) + y_2(\alpha_2^{new} - \alpha_2^{old})K(x_2, x_i)$$

在一些特殊的场合，η将不为正。当核函数不满足 Mercer 条件时将会计算得到负的η，会让目标函数变为不定的。即使核函数正确也可能会计算得到一个为 0 的η,假如多个实例具有相同的输入向量。在任何情况下，SMO 都能够起作用，即使η不为正，此时应该在线段两端点处计算目标函数 Ψ 的值。

$f_1 = y_1 E_1 - \alpha_1 K_{11} - s\alpha_2 K_{12};$

$f_2 = y_2 E_2 - s\alpha_1 K_{12} - \alpha_2 K_{22};$

$L_1 = \alpha_1 + s(\alpha_2 - L);$

$H_1 = \alpha_1 + s(\alpha_2 - H);$

$$\psi_L = L_1 f_1 + L f_2 + \frac{1}{2} L_1^2 K_{11} + \frac{1}{2} L^2 K_{22} + sLL_1 K_{12};$$

$$\psi_H = H_1 f_1 + H f_2 + \frac{1}{2} L_1^2 K_{11} + \frac{1}{2} H^2 K_{22} + sHH_1 K_{12};$$

SMO 将拉格朗日乘子移到目标函数最小对应的端点处。如果端点处对应的目标函数相同（在一个小的误差范围ε内），并且核满足 Mercer 条件，此时，更新乘子的步骤将不能够有正的进展，此时将重新选择第二个乘子。

2．3 计算 b

当所有乘子都满足 KKT 条件后，所有的α值确定，也即经过核映射后的超平面法向量确定。只需确定 b 的值，分类平面便确定。根据原始式

$$\text{minimize}: W(\boldsymbol{\alpha}) = \frac{\|w\|^2}{2} + C\sum_i \xi_i$$

$$\text{subject to } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

由于 w随着α值确定而确定，所以转换为

$$\text{minimize}: W'(b) = \sum_i \xi_i$$

$$\text{subject to } y_i\left(w^T x_i + b\right) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

将所有非支持向量舍弃，对于所有支持向量，由 KKT 条件，有

$$y_i\left(w^T x_i + b\right) \leq 1, \forall i$$

$$\xi_i = 1 - y_i\left(w^T x_i + b\right), \forall i$$

所以最后优化问题转化为

$$\text{maximize: } W''(b) = b\sum_i y_i$$

$$\text{subject to} \quad 1 - y_i\left(w^T x_i + b\right) \geq 0, \forall i$$

约束式会得到 b 的上界 upperBound 和下界 lowerBound，即一个取值区间。若 $\sum_i y_i$ 为正，则取上界，若 $\sum_i y_i$ 为负，则取下界，若 $\sum_i y_i$ 为 0，则取上下界的均值。

2.2.2 证明

定义：

$$v_i = \sum_{j=3}^{l} \alpha_j y_j K\left(\mathbf{x_i}, \mathbf{x_j}\right) = f(\mathbf{x_i}) - \sum_{j=1}^{2} \alpha_j y_j K\left(\mathbf{x_i}, \mathbf{x_j}\right) - b, i = 1,2$$

考虑到将 α_1 和 α_2 的函数作为目标：

$$W(\alpha_1, \alpha_2) = (\alpha_1 + \alpha_2) - \frac{1}{2}\alpha_1^2 \|\phi(x_1)\|^2 - \frac{1}{2}\alpha_2^2 \|\phi(x_2)\|^2 - \alpha_1\alpha_2 y_1 y_2 \phi(x_1)^T \phi(x_2) - y_1\alpha_1 v_1 - y_2\alpha_2 v_2 + \text{常数}$$

注意约束 $\sum_{j=1}^{l} \alpha_j^{old} y_j = \sum_{j=1}^{l} \alpha_j y_j = 0$ 意味着条件：

$$\alpha_1 + s\alpha_2 = \text{常数} = \alpha_1^{old} + s\alpha_2^{old} = \gamma$$

这里 $s = y_1 y_2$，利用这个方程可以用 α_2^{new} 计算 α_1^{new}，此约束下目标函数可写为：

$$W(\alpha_2) = \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}K_{11}(\gamma - s\alpha_2)^2 - \frac{1}{2}\alpha_2^2 K_{22} - sK_{12}(\gamma - s\alpha_2)\alpha_2 - y_1(\gamma - s\alpha_2)v_1 - y_2\alpha_2 v_2 + \text{常数}$$

对 $W(\alpha_2)$ 求偏导，驻点应满足:

$$\frac{\partial W(\alpha_2)}{\partial \alpha_2} = 1 - s + sK_{11}(\gamma - s\alpha_2) - \alpha_2 K_{22} + K_{12}\alpha_2 - sK_{12}(\gamma - s\alpha_2) + y_2 v_1 - y_2 v_2 = 0$$

由此得出：

$$\alpha_2^{new}(K_{11} + K_{12} + 2K_{22}) = 1 - s + \gamma s(K_{11} - K_{12}) + y_2(v_1 - v_2) = y_2(y_2 - y_1 + \gamma y_1(K_{11} - K_{12}) + v_1 - v_2)$$

令 $\kappa = K_{11} + K_{12} + 2K_{22}$，因此

$$\begin{aligned}\alpha_2^{new}\kappa y_2 &= y_2 - y_1 + f(\mathbf{x}_1) - \sum_{j=1}^{2}\alpha_j y_j K(\mathbf{x}_1,\mathbf{x}_j) + \gamma y_1 K_{11} \\ &\quad - f(\mathbf{x}_2) + \sum_{j=1}^{2}\alpha_j y_j K(\mathbf{x}_2,\mathbf{x}_j) - \gamma y_2 K_{12} \\ &= \alpha_2 y_2 \kappa + (f(\mathbf{x}_1) - y_1 - b) - (f(\mathbf{x}_2) - y_2 - b)\end{aligned}$$

给出：

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\kappa}$$

得证。

需要注意的是，尽管有 $\alpha_2^{new} = \alpha_2^{old} - \frac{y_2(E_1 - E_2)}{\kappa}$ 的形式，α_2^{old} 实际上是凑出来的，目的是为了方便计算，在等号右边的式子中，化简后最终将看不到 α_2^{old} 的身影（类似 y=-x+(x-1)）。

这种算法得到了非常广泛的应用，然而 $f(\boldsymbol{\alpha}^k)$ 与 $f(\boldsymbol{\alpha}^{k+1})$ 并无显式的对比，目前尚并无收敛的充分证明，因此带有启发性质。是一种重要的思路。另一种思路是通过 $f(\boldsymbol{\alpha}^k)$ 与 $f(\boldsymbol{\alpha}^{k+1})$ 进行显式的对比，在每一步中 $f(\boldsymbol{\alpha}^{k+1}) < f(\boldsymbol{\alpha}^k)$，[1]已经能够证明，$f(\boldsymbol{\alpha}^k)$ 的收敛能够得到保证。

令 $\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + d$，其中 $d_1 = \alpha_1^{new} - \alpha_1^{old}, d_2 = \alpha_2^{new} - \alpha_2^{old}$，令 $s = y_1 y_2$，由于 $\alpha_1^{old} + s\alpha_2^{old} = \alpha_1^{new} + s\alpha_2^{new}$，可以得到 $d_1 = -sd_2$，则有

$$\begin{aligned}f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^k) &\approx \nabla f(\boldsymbol{\alpha}^k) d \\ &= \nabla f(\boldsymbol{\alpha}^k)_1 d_1 + \nabla f(\boldsymbol{\alpha}^k)_2 d_2 \\ &= d_2(\nabla f(\boldsymbol{\alpha}^k)_2 - s\nabla f(\boldsymbol{\alpha}^k)_1) \\ &= y_2 d_2 (F_2 - F_1)\end{aligned}$$

由于 $d_2 = \lambda \frac{y_2(E_1 - E_2)}{\kappa}, \lambda \in (0,1]$，当且仅当被裁剪时 $\lambda \in (0,1)$；

$$f(\boldsymbol{\alpha}^{k+1}) - f(\boldsymbol{\alpha}^k) = -\lambda \frac{(E_1 - E_2)^2}{\kappa} < 0$$

因此，上述算法计算 $f(\boldsymbol{\alpha}^k)$ 能够保证收敛。

[1] Chih-Jen Lin.***On the Convergence of the Decomposition Method for Support Vector Machines***

[2] S.S. Keerthi ***Improvements to Platt's SMO Algorithm for SVM Classifier Design***
[3] John C. Platt ***Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines***
[4] Osuna, E. ***Improved Training Algorithm for Support Vector Machines***