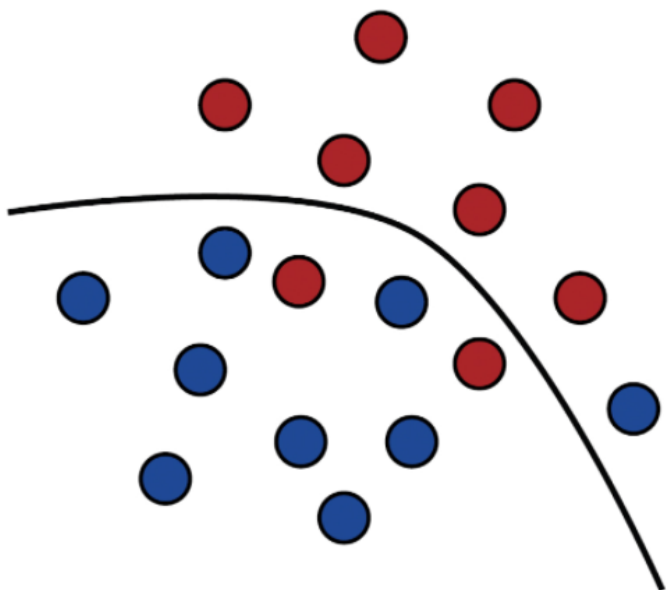
LIBSVM核心代码导读

chilynntse(谢志宁)，*version 1.0*



2016年11月

关于这篇文章

本文主要介绍SVM相关的理论背景，SMO算法以及LIBSVM的一些核心代码，我们会发现，LIBSVM的核心代码大部分都是围绕SMO展开的，如何选取alpha，如何更新alpha，还包括一些优化技巧等。SVM的类型有很多，主要包括：C-SVC，ν-SVC，One-class SVM，ϵ-SVR，ν-SVR，本文主要是基于C-SVC进行阐述，对于一些基本的推导（例如：如何推导出SVM的学习问题）将不在本文再赘述，默认大家对SVM有一定了解

LIBSVM还有很多可以深入的点，例如：框架设计，SVM的多分类问题，缓存，参数选取等等，在这个版本中不能为大家一一介绍，目前只是针对其核心代码为大家导读

本文希望能够将理论与实践相结合，对SVM学习者有一定帮助，同时本文可能存在许多问题，如果对本文有任何评价或建议，欢迎发送邮件到chilynntse@163.com，谢谢

Contents

1　拉格朗日对偶性

1.1　原始问题

$$
\begin{aligned}
\min \quad & f(x) \\
s.t. \quad & g_i(x) \le 0, \quad i = 1, ..., m \\
& h_j(x) = 0, \quad j = 1, ..., p
\end{aligned}
\tag{1}
$$

其中$f(x), g_1(x), ..., g_m(x)$都是凸函数，$h_0(x), h_1(x), ..., h_p(x)$都是仿射函数，具有这样形式的优化问题是一种凸优化问题，此外我们把该问题(1)称为原始问题

可以看到，问题(1)是一个带约束的优化问题，这些约束条件比较烦人，我们可以通过构造拉格朗日函数将约束条件与目标函数结合在一起，我们定义拉格朗日函数为

$$
L(x, \alpha, \beta) = f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) + \sum_{j=1}^{p} \beta_j h_j(x) \qquad \alpha_i \ge 0, \ \beta_j \in R \tag{2}
$$

对于满足约束条件的x，$\max_{\alpha,\beta} L(x, \alpha, \beta)$等价于$f(x)$，下面我们违反的情况

- 当存在一个x使得$g_i(x) > 0$，我们令$a_i \to +\infty$使得$a_i g_i(x) \to +\infty$，其余α_i, β_j均取0，则$\max_{\alpha,\beta} L(x, \alpha, \beta) \to +\infty$
- 当存在一个x使得$h_j(x) \ne 0$，我们令$\beta_j \to \infty$使得$\beta_j h_j(x) \to +\infty$，其余$\alpha_i, \beta_j$均取0，则$\max_{\alpha,\beta} L(x, \alpha, \beta) \to +\infty$

$$
f(x) = \begin{cases} \max\limits_{\alpha,\beta} L(x, \alpha, \beta) & x \ satisfy \ the \ constraint \\ +\infty & others \end{cases}
$$

因此，我们可以定义一个函数

$$
\theta_P(x) = \max_{\alpha,\beta} L(x, \alpha, \beta) \tag{3}
$$

对于满足约束条件的x，$\theta_P(x) \iff f(x)$，然后我们对$f(x)$取极小，也就是对$\theta_P(x)$取极小，即

$$\min_x \theta_P(x) = \min_x \max_{\alpha,\beta} L(x,\alpha,\beta) \tag{4}$$

问题(4)是与原始问题(1)是等价的，区别在于形式不一样，一个“有”约束条件，另一个“没有”约束条件，假设问题(4)的最优解为p^*

1.2 对偶问题

以上我们只是把“带约束形式的原始问题”(1)，改写成了“不带约束形式的原始问题”(4)，接下来我们看下(4)的对偶问题

首先我们先定义一个函数

$$\theta_D(\alpha,\beta) = \min_x L(x,\alpha,\beta) \tag{5}$$

然后对$\theta_D(\alpha,\beta)$取极大，即

$$\max_{\alpha,\beta} \min_x L(x,\alpha,\beta) = \max_{\alpha,\beta} \theta_D(x) \tag{6}$$

问题(6)是原始问题(4)的对偶问题，假设问题(6)的最优解为d^*

1.3 原始问题与对偶问题的关系

容易知道

$$d^* = \max_{\alpha,\beta} \min_x L(x,\alpha,\beta) \le \min_x \max_{\alpha,\beta} L(x,\alpha,\beta) = p^* \tag{7}$$

形象地理解就是，“矮子中最高的$\le$高个中最矮的”

1.3.1 弱对偶

弱对偶成立的条件

$$d^* \le p^* \tag{8}$$

弱对偶对所有的优化问题都成立，对偶问题的最优解d^*就是原始问题最优解p^*的下界，当我们对原始问题的解不清楚时，我们通过对偶问题来估计或者近似原始问题的解，当然我们希望这个下界d^*越逼近p^*越好，最理想的情况就是二者相等，即强对偶

1.3.2 强对偶

强对偶成立的条件

$$d^* = p^* \tag{9}$$

强对偶需要满足一定的条件，但我们在学习SVM时直接假定强对偶条件成立

强对偶成立的条件是：$f(x), g_1(x), ..., g_m(x)$都是凸函数，$h_0(x), h_1(x), ..., h_p(x)$都是仿射函数，并且$g_1(x), ..., g_m(x)$是严格可行的，即存在x，对所有的$g_i(x) < 0$, $i = 1, ..., m$都成立，这个就是Slater条件

这里稍微解释下"$g_1(x), ..., g_m(x)$是严格可行的，即存在x，对所有的$g_i(x) < 0$, $i = 1, ..., m$都成立"。这句话等价于该问题是线性可分的，这里的x其实是SVM中的参数(w, b)，$g_i(x) = 1 - y_i(w\phi(x_i) + b)$，替换下就是，即存在一个超平面(w, b)，使得对所有的样本x_i，都有$1 - y_i(w\phi(x_i) + b) < 0$, $i = 1, ..., m$，下面给出简单证明

假设问题是线性可分的，即存在(w, b)，使得

$$y_i(w\phi(x_i) + b) > 0$$

假设$y_i(w\phi(x_i) + b)$的最小值为γ，即

$$y_i(w\phi(x_i) + b) \ge \gamma > \frac{1}{2}\gamma$$

整理下得到

$$y_i(w\phi(x_i)+b) > \frac{1}{2}\gamma$$

等式两边同时除以$\frac{1}{2}\gamma$，得到

$$y_i(w'\phi(x_i)+b') > 1$$

其中$w' = \frac{2}{\gamma}w$，$b' = \frac{2}{\gamma}b$，以下两个超平面是等价的

$$w\phi(x_i)+b=0 \iff \frac{2}{\gamma}w\phi(x_i)+\frac{2}{\gamma}b=0$$

总结一下就是，当问题是线性可分时，即存在(w', b')使得$1-y_i(w'\phi(x_i)+b') < 0,\ i=1,...,m$，也就是严格可行的

当强对偶成立时，我们可以得到一些好的性质，即KKT条件，下面我们进行推导

首先我们假设x^*和(α^*, β^*)分别是原始问题和对偶问题的极值点，即

$$d^* = \max_{\alpha,\beta}\min_x L(x,\alpha,\beta) = \min_x L(x,\alpha^*,\beta^*) = \theta_D(\alpha^*,\beta^*)$$

$$p^* = \min_x \max_{\alpha,\beta} L(x,\alpha,\beta) = \max_{\alpha,\beta} L(x^*,\alpha,\beta) = \theta_P(x^*) = f(x^*)$$

$$d^* = p^*$$

根据以上3个等式，我们可以得到

$$\begin{aligned}
f(x^*) &= \theta_D(\alpha^*,\beta^*) \\
&= \min_x L(x,\alpha^*,\beta^*) \\
&= \min_x (f(x) + \sum_{i=1}^{m}\alpha_i^* g_i(x) + \sum_{j=1}^{p}\beta_j^* h_j(x)) \\
&\le f(x^*) + \sum_{i=1}^{m}\alpha_i^* g_i(x^*) + \sum_{j=1}^{p}\beta_j^* h_j(x^*) \\
&\le f(x^*)
\end{aligned} \tag{10}$$

因为对偶问题的最优解等于原始问题的最优解，即$\theta_D(\alpha^*, \beta^*) = f(x^*)$，我们需要将最后两个不等号变成等号

- 第1个不等号变成等号：$L(x, \alpha^*, \beta^*)$在x^*处取得极值，即$L(x, \alpha^*, \beta^*)$在x^*处梯度为0，即$\nabla L(x^*, \alpha^*, \beta^*) = \nabla f(x^*) + \sum_{i=1}^{m} \alpha_i^* \nabla g_i(x^*) + \sum_{j=1}^{p} \beta_j^* \nabla h_j(x^*) = 0$
- 第2个不等号变成等号：$h_j(x^*)$原本就等于0，我们只需要让$\alpha_i^* g_i(x^*) \leq 0 \rightarrow \alpha_i^* g_i(x^*) = 0$，这个是KKT的对偶互补条件
  - 当$a_i^* > 0$时，$g_i(x^*) = 0$，即$y_i(w\phi(x_i) + b) = 1$，x_i就是支持向量
  - 当$g_i(x^*) < 0$时，即$y_i(w\phi(x_i) + b) > 1$，得到$a_i^* = 0$

整理一下，我们得到KKT条件

$$\begin{aligned} g_i(x^*) &\leq 0 \\ h_j(x^*) &= 0 \\ \alpha_i^* &\geq 0 \\ \alpha_i^* g_i(x^*) &= 0 \\ \nabla f(x^*) + \sum_{i=1}^{m} \alpha_i^* \nabla g_i(x^*) + \sum_{j=1}^{p} \beta_j^* \nabla h_j(x^*) &= 0 \end{aligned} \tag{11}$$

前3个条件是我们定义(1)(2)带的约束条件，后2个条件我们由强对偶推KKT(10)产生的

这里再补充一下引入松弛变量ξ_i时的KKT的对偶互补条件，下面是引入松弛变量ξ_i的原始问题

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L} \xi_i \\ s.t. \quad & y_i(w\phi(x_i) + b) \geq 1 - \xi_i \quad i = 1, ..., L \\ & \xi_i \geq 0 \quad i = 1, ..., L \end{aligned} \tag{12}$$

原始问题(12)的拉格朗日函数是

$$L(w,b,\xi,\alpha,\beta)=\frac{1}{2}\|w\|^2+C\sum_{i=1}^{L}\xi_i-\sum_{i=1}^{L}\alpha_i(y_i(w\phi(x_i)+b)-1+\xi_i)-\sum_{i=1}^{L}\beta_i\xi_i \tag{13}$$

其中α_i和β_i是关于两个约束条件的拉格朗日乘子，对偶问题是极大极小问题，在求极小问题时，需要$L(w,b,\xi,\alpha,\beta)$对ξ求导，我们可以得到$C-\alpha_i-\beta_i=0$

根据KKT的对偶互补条件$\alpha_i g_i(x)=0$，即$\alpha_i(y_i(w\phi(x_i)+b)-1+\xi_i)=0$和$\beta_i\xi_i=0$，我们可以得到

$$\begin{gathered}\alpha_i=0\Rightarrow\beta_i=C\Rightarrow\xi_i=0\Rightarrow y_i(w\phi(x_i)+b)\geq 1\\ 0<\alpha_i<C\Rightarrow 0<\beta_i<C\Rightarrow\xi_i=0\Rightarrow y_i(w\phi(x_i)+b)=1\\ \alpha_i=C\Rightarrow\beta_i=0\Rightarrow\xi_i\geq 0\Rightarrow y_i(w\phi(x_i)+b)\leq 1\end{gathered} \tag{14}$$

也就是我们可以根据α_i的取值来判断该点x_i在什么位置

1.3.3 为什么要转换为对偶问题

- 自然地引入核函数
- 方便求解，特别是当x的维数较高时（可能几千几万维），与之对应的w的维数也会很高，在计算w^*时计算量大，通过对偶问题，我们只需要求解出α^*，α^*的数量只与支持向量机的数量有关，最终通过α^*来计算w^*，可以减少计算量
- 目标函数$f(x)$不一定是凸函数，但转换为对偶问题后，对偶函数总是凸函数，具有较好地性质。如下图所示(来自《Convex Optimization》)，左图黑实线是目标函数$f(x)$，显然不是凸函数，最下面的虚线是约束函数$g(x)\leq 0$，x被约束在了$[-0.46,0.46]$的区间中，最优解在$x^*=0.46$（黑点）处取得，最小值为$p^*=1.54$，围绕在黑实线上下的许多虚曲线是拉格朗日函数$L(x,\lambda)$ $\lambda=0.1,0.2,...,1$，我们在右图绘制$d(\lambda)=minL(x,\lambda)$，$\lambda\in[0,1]$，例如：当$\lambda=0$时，然后我们对$L(x,0)$取极小，得到一个值为$d(0)=minL(x,0)$，然后在右图画上(0, d(0))。右图是左图的对偶函数，显然是凸的，我们对它求极大，即$max\,d(\lambda)$，极大值在峰顶处取得，为d^*，右图中与坐标轴平行的虚线是p^*，显然$d^*\leq p^*$

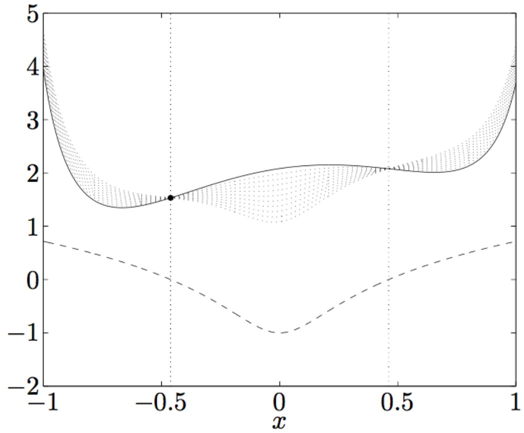


Figure 1: 目标函数f(x)

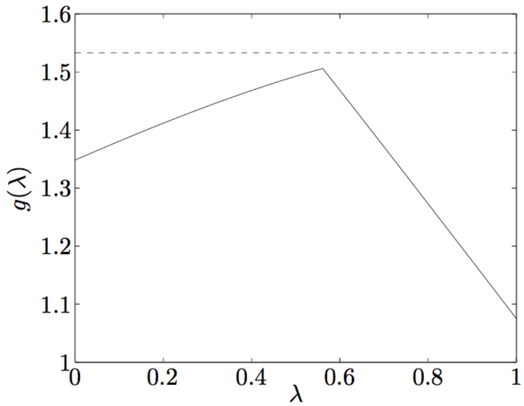


Figure 2: 对偶函数d(λ)

2 SMO

SMO是SVM的求解利器，可以看到LIBSVM中大部分地实现是围绕SMO来展开的下面我们定义一般的原始问题，即含有松弛因子ξ的原始问题

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\xi_i \\ s.t. \quad & y_i(w\phi(x_i)+b) \geq 1-\xi_i \quad i=1,...,L \\ & \xi_i \geq 0 \quad i=1,...,L \end{aligned} \tag{15}$$

问题(15)的对偶问题

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \frac{1}{2}\sum_{i=1}^{L}\sum_{j=1}^{L}\alpha_i\alpha_j y_i y_j K(x_i,x_j) - \sum_{i=1}^{L}\alpha_i \\ s.t. \quad & 0 \leq \alpha_i \leq C \quad i=1,...,L \\ & \sum_{i=1}^{L} y_i\alpha_i = 0 \end{aligned} \tag{16}$$

对偶问题(16)的矩阵形式

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \frac{1}{2}\alpha^T Q\alpha - e^T\alpha \\ s.t. \quad & 0 \leq \alpha_i \leq C \quad i=1,...,L \\ & y^T\alpha = 0 \end{aligned} \tag{17}$$

- L是所有样本的数量
- e是一个全为1的单位向量
- C就是惩罚参数，也是α的upper bound
- Q是一个L乘L的对称矩阵，$Q_{ij} = y_i y_j K(x_i, x_j)$，其中$K(x_i, x_j)$是核函数

原始问题中的最优解w^*可以根据对偶问题的最优解α^*求解得到，即

$$w^* = \sum_{i=1}^{L} y_i \alpha_i^* \phi(x_i) \tag{18}$$

原始问题中的最优解b^*可以根据KKT条件求得，根据KKT条件(14)，我们有

$$0 < \alpha_i < C \Rightarrow y_i(w^* \phi(x_i) + b^*) = 1$$

然后进一步求得

$$\begin{aligned} b^* &= y_i - w^* \phi(x_i) \\ &= y_i - \sum_{j=1}^{L} y_j \alpha_j^* \phi(x_j) \phi(x_i) \\ &= y_i - \sum_{j=1}^{L} y_j \alpha_j^* K(x_j, x_i) \end{aligned} \tag{19}$$

接下来我们需要利用SMO算法进行求解对偶问题(17)的最优解α^*

SMO的核心思想是，在一轮迭代中，不是一次更新α的所有分量，而是每次选中α两个分量i和j，固定其他的分量，在子问题上进行优化

SMO算法涉及到两个核心的问题

1. 如何选择i和j
2. 如何更新α，即选择了i和j后，如何更新α_i和α_j使得(17)中的目标函数$f(\alpha)$变小

下面就这两个问题分别进行说明

2.1 如何选择i和j

这个问题主要参考《Working Set Selection Using Second Order Information for Training Support Vector Machines》这篇论文

2.1.1 基于First Order的方法（一个普遍的做法）

首先解释下什么是First Order，相当于泰勒一阶展开，即梯度

对偶问题(17)的拉格朗日函数

$$L(\alpha, \lambda, \mu) = f(\alpha) - \sum_{i=1}^{L} \lambda_i \alpha_i + \sum_{i=1}^{L} \mu_i(\alpha_i - C) + \sum_{i=1}^{L} \eta y^T \alpha \tag{20}$$

其中$\lambda_i > 0$和$\mu_i > 0$，若α为极值点的话，我们可以得到$\nabla L(\alpha, \lambda, \mu) = 0$，即$\nabla f(\alpha) - \lambda + \mu + \eta y = 0$，整理后得到

$$\nabla f(\alpha) + \eta y = \lambda - \mu \tag{21}$$

其中

$$\begin{aligned} \nabla f(\alpha) &= Q\alpha - e \\ \nabla f(\alpha)_i &= \sum_{j=1}^{L} \alpha_j y_i y_j K(x_i, x_j) - 1 \end{aligned} \tag{22}$$

根据KKT的对偶互补条件，我们可以得到

$$\lambda_i \alpha_i = 0 \tag{23}$$

$$\mu_i (C - \alpha_i) = 0 \tag{24}$$

根据(21)(23)(24)，我们可以得到

$$\begin{aligned} \nabla f(\alpha)_i + \eta y_i \geq 0 \qquad if\ \alpha_i < C \\ \nabla f(\alpha)_i + \eta y_i \leq 0 \qquad if\ \alpha_i > 0 \end{aligned} \tag{25}$$

(25)的解释如下

当$\alpha < C$时，$C-\alpha > 0$，根据(24)得到$\mu = 0$，又因$\lambda \geq 0$，所以$\lambda - \mu \geq 0$，根据(21)得到$\nabla f(\alpha) + \eta y \geq 0$

当$\alpha > 0$时，根据(23)得到$\lambda = 0$，又因$\mu \geq 0$，所以$\lambda - \mu \leq 0$，根据(21)得到$\nabla f(\alpha) + \eta y \leq 0$

下面我们证明下，这里的η其实就是截距b

当$0 < \alpha_i < C$时，根据(25)，我们有

$$\nabla f(\alpha)_i + \eta y_i = 0 \qquad if\ 0 < \alpha_i < C$$

变形后可以得到

$$\eta = -y_i \nabla f(\alpha)_i \qquad if\ 0 < \alpha_i < C \tag{26}$$

根据(14)中的

$$0 < \alpha_i < C \Rightarrow y_i(w\phi(x_i) + b) = 1$$

以及(18)和(22)，我们可以得到

$$\begin{aligned} b &= y_i - w\phi(x_i) \\ &= y_i - \sum_{j=1}^{L} \alpha_j y_j K(x_j, x_i) \\ &= -y_i \nabla f(\alpha)_i \qquad if\ 0 < \alpha_i < C \end{aligned} \tag{27}$$

根据以及(26)和以及(27)，我们发现当$0 < \alpha_i < C$时

$$\eta = b$$

同理可证（其他情况$\alpha_i = 0$和$\alpha_i = C$），接下来，我们就用b去替换η

接下来我们定义两个集合，这两个集合是关于α_t下标t的集合

$$\begin{aligned} I_{up}(\alpha) &= \{t | \alpha_t < C, y_t = 1\ or\ \alpha_t > 0, y_t = -1\} \\ I_{low}(\alpha) &= \{t | \alpha_t < C, y_t = -1\ or\ \alpha_t > 0, y_t = 1\} \end{aligned} \tag{28}$$

因为$y_i=\pm 1$，我们可以结合以及(28)，将以及(25)改写一下（左右两边同乘y_i）

$$
\begin{aligned}
-y_i \nabla f(\alpha)_i \le b, \forall_i \in I_{up}(\alpha) \\
-y_j \nabla f(\alpha)_j \ge b, \forall_j \in I_{low}(\alpha)
\end{aligned} \tag{29}
$$

将以及(29)合并一下得到

$$
-y_i \nabla f(\alpha)_i \le b \le -y_j \nabla f(\alpha)_j \qquad i \in I_{up}\ and\ j \in I_{low} \tag{30}
$$

我们令

$$
\begin{aligned}
m(\alpha) &= \max_{i \in I_{up}(\alpha)} -y_i \nabla f(\alpha)_i \\
M(\alpha) &= \min_{j \in I_{low}(\alpha)} -y_j \nabla f(\alpha)_j
\end{aligned} \tag{31}
$$

一个合理驻点α需要满足

$$
m(\alpha) \le M(\alpha) \tag{32}
$$

下面我们给出“违反对(violating piar)”的定义

如果$i \in I_{up}(\alpha), j \in I_{low}(\alpha),\ and\ -y_i \nabla f(\alpha)_i > -y_j \nabla f(\alpha)_j$，那么$\{i,j\}$就是一个违反对（$\alpha_i$和$\alpha_j$的下标）

一个普遍的做法就是每次迭代挑选“最大违反对(maximal violating pair)”，即

$$
\begin{aligned}
i \in arg\ \max_t \{-y_t \nabla f(\alpha^k)_t \mid t \in I_{up}(\alpha^k)\} \\
j \in arg\ \max_t \{-y_t \nabla f(\alpha^k)_t \mid t \in I_{low}(\alpha^k)\}
\end{aligned} \tag{33}
$$

因为最优的α需要满足$m(\alpha) \le M(\alpha)$，如果违反了，它就是不是最优的，所以每次找出这个“最大违反对”，然后更新它们的值

最后我们再设置一个停止条件，即$m(\alpha^k) - M(\alpha^k) < \epsilon$，这里$\epsilon$称为容忍值，如果在第k轮的时候，$m(\alpha^k)$没有比$M(\alpha^k)$大太多的话，就停止

可以证明，这种挑选方法是利用了$f(\alpha)$的一阶信息，具体证明请见论文

2.1.2 基于Second Order的方法（LIBSVM）

LIBSVM的working set selection是根据second order information来选择的，其中i的选择与first order一样，与first order不一样的是j的选择，j除了是违反对之外，还需要它使目标函数减少最多，second order自然要比first order要好，这就好比牛顿法（second order）要比梯度下降（first order）快

接下来的问题就是我们要找到一个j

根据泰勒公式在α^k处地二阶展开，我们得到

$$\begin{aligned} f(\alpha^k + d) &= f(\alpha^k) + \nabla f(\alpha^k)^T d + \frac{1}{2} d^T \nabla^2 f(\alpha^k) d \\ &= f(\alpha^k) + \nabla f(\alpha^k)_B^T d_B + \frac{1}{2} d_B^T \nabla^2 f(\alpha^k)_{BB} d_B \end{aligned} \tag{34}$$

其中$B = \{i, j\}$，注意这里的变量只有α_i和α_j，所以对其他$\alpha_t, t \neq i\ and\ t \neq j$的导数为0，直接舍去，我们只取含有B的分量

我们定义一个子问题，这个子问题只跟B有关，即α_i和α_j有关

$$\begin{aligned} Sub(B) &= f(\alpha^k + d) - f(\alpha^k) \\ &= \nabla f(\alpha^k)_B^T d_B + \frac{1}{2} d_B^T \nabla^2 f(\alpha^k)_{BB} d_B \end{aligned} \tag{35}$$

我们的目标就是让Sub(B)越小越好，即希望α^k加上偏移量d，$f(\alpha^k + d)$的值能够比更新之前$f(\alpha^k)$要小，也就是$f(\alpha^k + d) << f(\alpha^k)$，整个问题就变成

$$\begin{aligned} \min_{d_B} \quad & \nabla f(\alpha^k)_B^T d_B + \frac{1}{2} d_B^T \nabla^2 f(\alpha^k)_{BB} d_B \\ s.t. \quad & y_B^T d_B = 0 \\ & d_t \geq 0,\ if\ a_t^k = 0, t \in B \\ & d_t \leq 0,\ if\ a_t^k = C, t \in B \end{aligned} \tag{36}$$

这里解释下以及(36)的约束条件，因为

$$\begin{cases} y_i \alpha_i + y_j \alpha_j = Constant \\ y_i(\alpha_i + d_i) + y_j(\alpha_j + d_j) = Constant \end{cases}$$

所以

$$y_i d_i + y_j d_j = 0 \Rightarrow y_B^T d_B = 0$$

又因为

$$0 \le \alpha_i \le C$$

$$0 \le \alpha_i + d_i \le C$$

所以

$$-\alpha_i \le d_i \le C - \alpha_i$$

即当$\alpha_i = 0$时，$d_i \ge 0$，当$\alpha_i = C$时，$d_i \le 0$

定理：如果$B = \{i, j\}$是一个违反对且$K_{ii} + K_{jj} - 2K_{ij} > 0$，上述问题能够取得的最小值是

$$-\frac{(-y_i \nabla f(\alpha^k)_i + y_i \nabla f(\alpha^k)_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})} \tag{37}$$

证明如下，首先定义

$$\begin{cases} \hat{d}_i = y_i d_i \\ \hat{d}_j = y_j d_j \end{cases}$$

因为以及(36)

$$y_B^T d_B = 0$$

所以

$$\hat{d}_i = -\hat{d}_j$$

我们再定义几个记号，根据(22)我们知道

$$\nabla f(\alpha)_i = \sum_{j=1}^{L} \alpha_j y_i y_j K(x_i, x_j) - 1$$

定义

$$Q_{ii} = \frac{\nabla f(\alpha)_i}{\nabla \alpha_i} = y_i y_i K(x_i, x_i)$$

$$Q_{ij} = \frac{\nabla f(\alpha)_i}{\nabla \alpha_i} = y_i y_j K(x_i, x_j)$$

$$Q_{ji} = \frac{\nabla f(\alpha)_j}{\nabla \alpha_i} = y_j y_i K(x_j, x_i)$$

$$Q_{jj} = \frac{\nabla f(\alpha)_j}{\nabla \alpha_j} = y_j y_j K(x_j, x_j)$$

接下来，我们将(36)的目标函数改写一下

$$\begin{aligned}
sub(B) &= \nabla f(\alpha^k)_B^T d_B + \frac{1}{2} d_B^T \nabla^2 f(\alpha^k)_{BB} d_B \\
&= \frac{1}{2} \begin{bmatrix} d_i & d_j \end{bmatrix} \begin{bmatrix} \frac{\nabla f(\alpha)_i}{\nabla \alpha_i} & \frac{\nabla f(\alpha)_i}{\nabla \alpha_i j} \\ \frac{\nabla f(\alpha)_j}{\nabla \alpha_i} & \frac{\nabla f(\alpha)_j}{\nabla \alpha_j} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\alpha^k)_i & \nabla f(\alpha^k)_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \\
&= \frac{1}{2} \begin{bmatrix} d_i & d_j \end{bmatrix} \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\alpha^k)_i & \nabla f(\alpha^k)_j \end{bmatrix} \begin{bmatrix} d_i \\ d_j \end{bmatrix} \\
&= \frac{1}{2}(d_i^2 y_i^2 K_{ii} + d_i d_j y_i y_j K_{ij} + d_j d_i y_j y_i K_{ji} + d_j^2 y_j^2 K_{jj}) + (\nabla f(\alpha)_i d_i + \nabla f(\alpha)_j d_j) \\
&= \frac{1}{2}[(d_i y_i)^2 K_{ii} - 2(d_j y_j)^2 K_{ij} + (d_j y_j)^2 K_{jj}] + (\nabla f(\alpha)_i d_i y_i^2 + \nabla f(\alpha)_j d_j y_j^2) \\
&= \frac{1}{2}(K_{ii} + K_{jj} - 2K_{ij}) \hat{d}_j^2 + (-y_i \nabla f(\alpha^k)_i + y_j \nabla f(\alpha^k)_j) \hat{d}_j
\end{aligned} \tag{38}$$

因为我们假设$K_{ii} + K_{jj} - 2K_{ij} > 0$，且B是违反对，即$-y_i \nabla f(\alpha)_i > -y_j \nabla f(\alpha)_j$，我们定义

$$\begin{aligned}
a_{ij} &= K_{ij} + K_{jj} - 2K_{ij} > 0 \\
b_{ij} &= -y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j > 0
\end{aligned} \tag{39}$$

将以及(39)带入以及(38)，我们可以得到

$$\begin{aligned}
&\frac{1}{2}(K_{ii} + K_{jj} - 2K_{ij}) \hat{d}_j^2 + (-y_i \nabla f(\alpha^k)_i + y_j \nabla f(\alpha^k)_j) \hat{d}_j \\
=&\frac{1}{2} a_{ij} \hat{d}_j^2 + b_{ij} \hat{d}_j
\end{aligned} \tag{40}$$

标准的二次函数$f(x) = ax^2 + bx + c$，极值点在$x = -\frac{b}{2a}$处取得，极值为$y = \frac{4ac-b^2}{4a}$

在该问题中，极值点在$\hat{d}_j = -\frac{b_{ij}}{2 \cdot \frac{1}{2} a_{ij}} = -\frac{b_{ij}}{a_{ij}}$处取得，极值为$\frac{0-b_{ij}^2}{4 \cdot \frac{1}{2} a_{ij}} = -\frac{b_{ij}^2}{2a_{ij}}$，即$-\frac{(-y_i \nabla f(\alpha^k)_i + y_i \nabla f(\alpha^k)_j)^2}{2(K_{ii} + K_{jj} - 2K_{ij})}$

整理下，我们可以得到

$$
\begin{aligned}
d_j &= y_j \hat{d}_j = -y_j \frac{b_{ij}}{a_{ij}} \\
d_i &= y_i \hat{d}_i = -y_i \hat{d}_j = -y_i(-\frac{b_{ij}}{a_{ij}}) = y_i \frac{b_{ij}}{a_{ij}}
\end{aligned} \tag{41}
$$

因此j的选择就是

$$
j \in arg \min_t \{-\frac{b_{it}^2}{a_{it}} \mid t \in I_{low}(\alpha^k), -y_t \nabla f(\alpha)_t < -y_i \nabla f(\alpha)_i\} \tag{42}
$$

以上就是原理部分，下面我们具体来看下LIBSVM源码中的具体实现，对应的部分是select_working_set这个函数

Part1. 定义相关变量

```
double Gmax = -INF;
double Gmax2 = -INF;
int Gmax_idx = -1;
int Gmin_idx = -1;
double obj_diff_min = INF;
```

- Gmax $\iff$ $m(\alpha) = \max_{i \in I_{up}(\alpha)} -y_i \nabla f(\alpha)_i$
- Gmax2 $\iff$ $-M(\alpha) = \max_{j \in I_{low}(\alpha)} y_j \nabla f(\alpha)_j$
- Gmax_idx $\iff$ i
- Gmin_idx $\iff$ j
- obj_diff_min $\iff$ $f(\alpha)$的极小值，我们需要记录取得极小值时的j

Part2. 选择i

```
for (int t = 0; t < active_size; t++) {
    if (y[t] == +1) {
        if(!is_upper_bound(t)) {
            // y_t = 1 and a_t < C
            if (-G[t] >= Gmax) {
                Gmax = -G[t];
                Gmax_idx = t;
            }
        }
    } else {
        if (!is_lower_bound(t)) {
            // y_t = -1 and a_t > 0
            if (G[t] >= Gmax) {
                Gmax = G[t];
                Gmax_idx = t;
            }
        }
    }
}
```

- i的选择只能来自于$I_{up}(\alpha) = \{t|\alpha_t < C, y_t = 1 \ or \ \alpha_t > 0, y_t = -1\}$，对应两个注释的地方
- G[t] $\iff$ $\nabla f(\alpha)_i$
  - 当$y_i = 1$时，-G[t] $= -\nabla f(\alpha)_i = -y_i \nabla f(\alpha)_i$
  - 当$y_i = -1$时，G[t] $= \nabla f(\alpha)_i = -y_i \nabla f(\alpha)_i$
- Gmax_idx = t $\iff$ (33)中i的选择

Part3. 选择j

```
for (int j = 0; j < active_size; j++) {
    if (y[j] == +1) {
        if (!is_lower_bound(j)) {
            // y_j = 1 and a_j > 0
            double grad_diff = Gmax + G[j];
            if (G[j] >= Gmax2)
                Gmax2 = G[j];
            if (grad_diff > 0) {
                double obj_diff;
                double quad_coef = QD[i]+QD[j]-2.0*y[i]*Q_i[j];
                if (quad_coef > 0)
                    obj_diff=-(grad_diff*grad_diff)/quad_coef;
                else
                    obj_diff=-(grad_diff*grad_diff)/TAU;
                if (obj_diff <= obj_diff_min) {
                    Gmin_idx=j;
                    obj_diff_min = obj_diff;
                }
            }
        }
    } else {
        if (!is_upper_bound(j)) {
            // y_j = -1 and a_j > 0
            double grad_diff= Gmax-G[j];
            if (-G[j] >= Gmax2)
                Gmax2 = -G[j];
            if (grad_diff > 0) {
                double obj_diff;
                double quad_coef = QD[i]+QD[j]+2.0*y[i]*Q_i[j];
                if (quad_coef > 0)
```

```
                obj_diff=-(grad_diff*grad_diff)/quad_coef;
            else
                obj_diff=-(grad_diff*grad_diff)/TAU;
            if (obj_diff <= obj_diff_min) {
                Gmin_idx=j;
                obj_diff_min = obj_diff;
            }
        }
    }
  }
}
```

- j的选择只能来自于$I_{low}(\alpha) = \{t|\alpha_t < C, y_t = -1 \ or \ \alpha_t > 0, y_t = 1\}$，对应两个注释的地方
- grad_diff = Gmax $\pm G[j] \quad \Longleftrightarrow \quad b_{ij} = -y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j$
- grad_diff $> 0 \quad \Longleftrightarrow \quad -y_i \nabla f(\alpha)_i > -y_j \nabla f(\alpha)_j$
- quad_coef $\quad \Longleftrightarrow \quad a_{ij} = K_{ij} + K_{jj} - 2K_{ij}$，字面意思应该是二次项的系数
- if (obj_diff $\leq$ obj_diff_min) $\quad \Longleftrightarrow \quad j \in arg\ \min_t \{-\frac{b_{it}^2}{a_{it}} \mid t \in I_{low}(\alpha^k), -y_t \nabla f(\alpha)_t < -y_i \nabla f(\alpha)_i\}$
- obj_diff_min $\quad \Longleftrightarrow \quad -\frac{b_{ij}^2}{2a_{ij}} = -\frac{(-y_i \nabla f(\alpha^k)_i + y_i \nabla f(\alpha^k)_j)^2}{2(K_{ii}+K_{jj}-2K_{ij})}$，忽略掉系数2

Part4. 判断停止以及返回(i, j)

```
if(Gmax+Gmax2 < eps || Gmin_idx == -1)
    return 1;
out_i = Gmax_idx;
out_j = Gmin_idx;
return 0;
```

- Gmax + Gmax2 < eps 对应于$m(\alpha^k) - M(\alpha^k) < \epsilon$

2.2 如何更新α

接下来的内容主要参考《LIBSVM: A Library for Support Vector Machines》原文的内容

2.1节我们阐述了如何选择违反对，但是我们最终的目的是要找到一个最优的$\alpha^T = [\alpha_1, \alpha_2, ..., \alpha_L]$，现在我们只是找到了$\alpha_i$和$\alpha_j$，所以我们需要在每轮迭代中更新这两个变量，使得目标函数变小，在程序实现中，还需要更新其他辅助参数，例如在LIBSVM中，我们还需要更新变量G和G_bar

2.2.1 更新α

我们将问题(36)重新改写成下面形式

$$\begin{aligned}
\min_{d_i,d_j} \quad & \frac{1}{2}\begin{bmatrix} d_i & d_j \end{bmatrix}\begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ji} & Q_{jj} \end{bmatrix}\begin{bmatrix} d_i \\ d_j \end{bmatrix} + \begin{bmatrix} \nabla f(\alpha^k)_i & \nabla f(\alpha^k)_j \end{bmatrix}\begin{bmatrix} d_i \\ d_j \end{bmatrix} \\
s.t. \quad & y_i d_i + y_j d_j = 0 \\
& -\alpha_i^k \le d_i \le C_i - \alpha_i^k \\
& -\alpha_j^k \le d_j \le C_j - \alpha_j^k
\end{aligned} \tag{43}$$

根据以及(41)，我们得到

$$\begin{aligned}
a_i^{k+1} = a_i^k + d_i = a_i^k + y_i\frac{b_{ij}}{a_{ij}} \\
a_j^{k+1} = a_j^k + d_j = a_j^k - y_j\frac{b_{ij}}{a_{ij}}
\end{aligned} \tag{44}$$

这里上标k代表第k轮，下面分两种情况进行讨论

（1）第一种情况$y_i \neq y_j$

$$\begin{aligned}
\alpha_i^{k+1} &= \alpha_i^k + y_i\frac{b_{ij}}{a_{ij}} \\
&= \alpha_i^k + y_i\frac{(-y_i\nabla f(\alpha)_i + y_j\nabla f(\alpha)_j)}{a_{ij}} \\
&= \alpha_i^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}
\end{aligned} \tag{45}$$

$$\begin{aligned}
\alpha_j^{k+1} &= \alpha_j^k - y_j \frac{b_{ij}}{a_{ij}} \\
&= \alpha_j^k - y_j \frac{(-y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j)}{a_{ij}} \\
&= \alpha_j^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}
\end{aligned} \tag{46}$$

我们定义

$$delta_{y_i \neq y_j} = \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \tag{47}$$

（2）第二种情况$y_i = y_j$

$$\begin{aligned}
\alpha_i^{k+1} &= \alpha_i^k + y_i \frac{b_{ij}}{a_{ij}} \\
&= \alpha_i^k + y_i \frac{(-y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j)}{a_{ij}} \\
&= \alpha_i^k + \frac{-\nabla f(\alpha)_i + \nabla f(\alpha)_j}{a_{ij}} \\
&= \alpha_i^k - \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}
\end{aligned} \tag{48}$$

$$\begin{aligned}
\alpha_j^{k+1} &= \alpha_j^k - y_j \frac{b_{ij}}{a_{ij}} \\
&= \alpha_j^k - y_j \frac{(-y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j)}{a_{ij}} \\
&= \alpha_j^k + \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}
\end{aligned} \tag{49}$$

我们定义

$$delta_{y_i = y_j} = \frac{\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}} \tag{50}$$

下面我们还是继续讨论（1）$y_i \neq y_j$的情况，$y_i = y_j$同理

因为$\sum_{i=1}^{L} \alpha_i y_i = 0$的约束，所以

$$y_i\alpha_i^k + y_j\alpha_j^k = y_i\alpha_i^{k+1} + y_j\alpha_j^{k+1} = constant$$

又因为$y_i \neq y_j$，我们有

$$\begin{aligned} y_iy_i\alpha_i^k + y_iy_j\alpha_j^k = \alpha_i^k - \alpha_j^k = constant' \\ y_iy_i\alpha_i^{k+1} + y_iy_j\alpha_j^{k+1} = \alpha_i^{k+1} - \alpha_j^{k+1} = constant' \end{aligned} \tag{51}$$

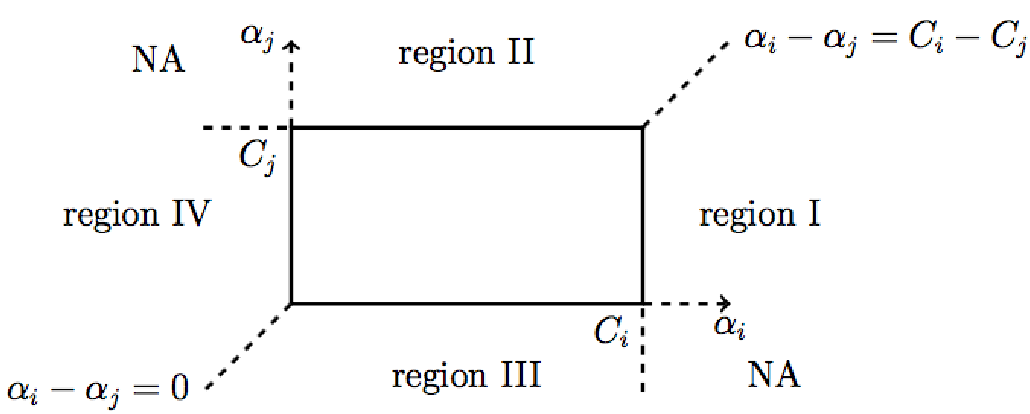
然后我们有

$$\alpha_i^k - \alpha_j^k = \alpha_i^{k+1} - \alpha_j^{k+1} = constant'$$

最终我们有

$$\alpha_i^{k+1} - \alpha_i^k = \alpha_j^{k+1} - \alpha_j^k \tag{52}$$

如下图所示，α_i^{k+1}，α_j^{k+1}的取值也是有约束的，$(\alpha_i^{k+1}, \alpha_j^{k+1})$必须在中间的矩形中，不合理的区域是在下图的4个区域中（$region\ I, region\ II, region\ III, region\ IV$），不可能在NA区域中（即左上角和右下角），所以整个候选区域是在中间矩形＋4个区域，合理的只有矩形，如果落在了4个区域中，我们要做一些clipping的工作



我们对这幅图解释一下，首先我们看看$\alpha_i-\alpha_j$的取值范围

$$\begin{cases} 0 \le \alpha_i^k \le C_i \\ 0 \le \alpha_j^k \le C_j \end{cases} \Rightarrow \begin{cases} 0 \le \alpha_i^k \le C_i \\ -C_j \le -\alpha_j^k \le 0 \end{cases}$$

$$\Rightarrow -C_j \le \alpha_i^k - \alpha_j^k \le C_i \tag{53}$$

这说明$\alpha_i^k-\alpha_j^k$的取值是有范围的（$-C_j \le \alpha_i - \alpha_j \le C_i$），该约束与k无关，即不管k是在第几轮迭代，$\alpha_i^k-\alpha_j^k$必须满足这个约束

（1）左上角NA区域

$$\alpha_i < 0 \; and \; \alpha_j > C_j \quad \Rightarrow \quad \alpha_i - \alpha_j < -C_j$$

与(53)矛盾

（2）右下角NA区域

$$\alpha_i > C_i \; and \; \alpha_j < 0 \quad \Rightarrow \quad \alpha_j - \alpha_i < -C_i \quad \Rightarrow \quad \alpha_i - \alpha_j > C_i$$

与(53)矛盾

（3）region I，在这个区域，我们有

$$\begin{cases} \alpha_i - \alpha_j > C_i - C_j \\ a_i > C_i \end{cases}$$

由于$a_i > C_i$，也就是a_i超出了矩形区域，我们让a_i落回到矩形区域的边缘，即让$a_i^{k+1}=C_i$，因为(52)即$\alpha_i^{k+1}-\alpha_i^k=\alpha_j^{k+1}-\alpha_j^k$，我们将$a_i^{k+1}=C_i$带入$\alpha_i^{k+1}-\alpha_i^k=\alpha_j^{k+1}-\alpha_j^k$，得到

$$\begin{cases} a_i^{k+1} = C_i \\ \alpha_j^{k+1} = C_i - \alpha_i^k + \alpha_j^k = C_i - (\alpha_i^k - \alpha_j^k) \end{cases}$$

（4）region II，在这个区域，我们有

$$\begin{cases} \alpha_i - \alpha_j < C_i - C_j \\ \alpha_j > C_j \end{cases}$$

我们让

$$\begin{cases} \alpha_j^{k+1} = C_j \\ \alpha_i^{k+1} = C_j + \alpha_i^k - \alpha_j^k \end{cases}$$

（5）region III，在这个区域，我们有

$$\begin{cases} \alpha_i - \alpha_j > 0 \\ a_j < 0 \end{cases}$$

我们让

$$\begin{cases} \alpha_j^{k+1} = 0 \\ \alpha_i^{k+1} = \alpha_i^k - \alpha_j^k \end{cases}$$

（6）region IV，在这个区域，我们有

$$\begin{cases} \alpha_i - \alpha_j < 0 \\ \alpha_i < 0 \end{cases}$$

我们让

$$\begin{cases} \alpha_i^{k+1} = 0 \\ \alpha_j^{k+1} = -(\alpha_i^k - \alpha_j^k) \end{cases}$$

以上就是原理部分，下面我们具体来看下LIBSVM源码中的具体实现，对应的部分是Solve这个函数，注意下面给出的是$y_i != y_j$的情况

```
if (y[i]!=y[j]) {
    double quad_coef = QD[i]+QD[j]+2*Q_i[j];
    if (quad_coef <= 0)
        quad_coef = TAU;
    double delta = (-G[i]-G[j])/quad_coef;
    double diff = alpha[i] - alpha[j];

    // rectangle
    alpha[i] += delta;
    alpha[j] += delta;

    // case of 4 region
    if(diff > 0) {
        if(alpha[j] < 0) { // region 3
            alpha[j] = 0;
            alpha[i] = diff;
        }
    } else {
        if (alpha[i] < 0) { // region 4
            alpha[i] = 0;
            alpha[j] = -diff;
        }
    }
    if (diff > C_i - C_j) {
        if (alpha[i] > C_i) { // region 1
            alpha[i] = C_i;
            alpha[j] = C_i - diff;
        }
    } else {
```

```
            if(alpha[j] > C_j) { // region 2
                alpha[j] = C_j;
                alpha[i] = C_j + diff;
            }
        }
}
```

- quad_coef $\iff$ $a_{ij} = K_{ij} + K_{jj} - 2K_{ij}$
- delta $\iff$ $delta_{y_i \neq y_j} = \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}$
- diff $\iff$ $\alpha_i - \alpha_j$，因为我们要判断$\alpha_i - \alpha_j$的取值范围
- alpha[i] += delta; $\iff$ $\alpha_i^{k+1} = \alpha_i^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}$
- alpha[j] += delta; $\iff$ $\alpha_j^{k+1} = \alpha_j^k + \frac{-\nabla f(\alpha)_i - \nabla f(\alpha)_j}{a_{ij}}$

2.3 更新辅助变量

2.3.1 更新G

```
double delta_alpha_i = alpha[i] - old_alpha_i;
double delta_alpha_j = alpha[j] - old_alpha_j;

for (int k = 0; k < active_size; k++) {
    G[k] += Q_i[k]*delta_alpha_i + Q_j[k]*delta_alpha_j;
}
```

- G[t] $\iff$ $\nabla f(\alpha)_i = \sum_{j=1}^{L} \alpha_j y_i y_j K(x_i, x_j) - 1$
- Q_i[k] $\iff$ $y_i y_j K(x_i, x_j)$
- G[k]的更新相当于，除了α_i和α_j，其他α_t都没有改变，然后将α_i和α_j的改变量(delta_alpha_i和delta_alpha_j)更新到G[k]

2.3.2 更新G_bar和alpha的状态

G我们已经说完了，就是$\nabla f(\alpha)$，这里需要解释下G_bar是什么东西

首先我们来看看α_i取值在训练过程中的变化

在训练过程中，当α_i到达了边界，即$\alpha_i = 0$或$\alpha_i = C$，α_i的值就不会改变了，它们的状态应该是inactive，而对于$0 < \alpha_i < C$的那些α_i，它们的状态是active的，这些$i \in A$，集合A的大小为active_size，在每一轮中都要检测active size中的α_i, 如果它们到达了边界，要把它们inactive

我们更新G，其实只更新active的α_i梯度，即$\nabla f(\alpha)_i$，对于那些inactive的α_i，我们为了加速计算，我们用$\bar{G}$，也就是G_bar来存储，解释如下

首先定义

$$\bar{G} = C \sum_{j:a_j=C} Q_{ij}, \quad i = 1, ..., L \tag{54}$$

对于$i \notin A$，即那些inactive的α_i，我们需要计算它们的梯度，根据(22)，我们有

$$\begin{aligned}
\nabla f(\alpha)_i &= \sum_{j=1}^{L} Q_{ij}\alpha_j + e_i \\
&= \sum_{\alpha_j=0} Q_{ij} * 0 + \sum_{\alpha_j=C} Q_{ij} * C + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j + e_i \\
&= \sum_{\alpha_j=C} Q_{ij} * C + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j + e_i \\
&= \bar{G}_i + \sum_{0<\alpha_j<C} Q_{ij}\alpha_j + e_i
\end{aligned} \tag{55}$$

提前计算好$\bar{G}$，这样就可以帮助我们加速计算梯度，下面我们来看下代码

```
bool ui = is_upper_bound(i);
bool uj = is_upper_bound(j);
update_alpha_status(i);
update_alpha_status(j);
int k;
if (ui != is_upper_bound(i)) {
```

```
    Q_i = Q.get_Q(i,l);
        if(ui) {
        for(k=0;k<l;k++)
            G_bar[k] -= C_i * Q_i[k];
        } else {
            for(k=0;k<l;k++)
            G_bar[k] += C_i * Q_i[k];
        }
}

if (uj != is_upper_bound(j)) {
    Q_j = Q.get_Q(j,l);
    if (uj) {
        for(k=0;k<l;k++)
            G_bar[k] -= C_j * Q_j[k];
    } else {
        for(k=0;k<l;k++)
            G_bar[k] += C_j * Q_j[k];
    }
}
```

- ui != is_upper_bound(i)成立说明i的状态前后必须有变化
- if(ui)成立说明i以前是upper bound，现在不是upper bound，也就是现在$\alpha_i \neq C$，所以从$k \in \{1, 2, ...L\}$上的G_bar都要减掉C Q_ij，对应代码就是G_bar[k] -= C_i * Q_i[k]，同理，其他对应模块

3 参数w和b

3.1 计算参数w

根据(18)，我们知道

$$w^* = \sum_{i=1}^{L} y_i \alpha_i^* \phi(x_i)$$

但我们这里并不需要显式地存储w，因为决策函数是

$$sign(w^T\phi(x) + b) = sign(\sum_{i=1}^{L} y_i \alpha_i K(x_i, x) + b) \tag{56}$$

我们只需要存储$y_i\alpha_i$和b即可

3.2 计算参数b

C-SVC和ϵ-SVR中的b和one-class SVM中的$-\rho$是等价的，即$b = -\rho$，在LIBSVM中存储的其实是ρ

根据(19)，我们知道

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

这里我们只是用了一个支持向量机$x_i(0 < \alpha_i < C)$，在LIBSVM中是取了所有的支持向量机，最后取平均值，即

$$\begin{aligned} b^* &= \frac{\sum_{i:0<\alpha_i<C}(y_i - \sum_{j=1}^{L} y_j \alpha_j K(x_j, x_i))}{|\{i \mid 0 < \alpha_i < C\}|} \\ &= \frac{\sum_{i:0<\alpha_i<C}(-y_i \nabla f(\alpha)_i)}{|\{i \mid 0 < \alpha_i < C\}|} \end{aligned} \tag{57}$$

根据$b = -\rho$，我们可以得到

$$\rho = \frac{\sum_{i:0<\alpha_i<C}(y_i \nabla f(\alpha)_i)}{|\{i \mid 0 < \alpha_i < C\}|} \tag{58}$$

$$
\begin{aligned}
&-M(\alpha) = max\{y_j \nabla f(\alpha)_j \mid a_j = 0, y_j = -1 \; or \; a_j = C, y_j = 1\} \\
\leq &\rho \qquad (59) \\
\leq &-m(\alpha) = min\{y_i \nabla f(\alpha)_i \mid a_i = 0, y_i = 1 \; or \; a_i = C, y_i = -1\}
\end{aligned}
$$

如果没有满足条件的α_i，即没有$0 < \alpha_i < C$，我们利用(59)，取这个范围的中点，接下来我们看下代码，对应的函数是Solver::calculate_rho

```
double r;
int nr_free = 0;
double ub = INF, lb = -INF, sum_free = 0;
for (int i=0;i<active_size;i++) {
    double yG = y[i]*G[i];
    if (is_upper_bound(i)) {
        if (y[i]==-1)
            ub = min(ub,yG);
        else
            lb = max(lb,yG);
    } else if (is_lower_bound(i)) {
        if (y[i]==+1)
            ub = min(ub,yG);
        else
            lb = max(lb,yG);
    } else {
        ++nr_free;
        sum_free += yG;
    }
}
if (nr_free>0)
    r = sum_free/nr_free;
else
    r = (ub+lb)/2;
```

```
return r;
```

- yG $\Longleftrightarrow$ $y_i \nabla f(\alpha)$
- nr_free $\Longleftrightarrow$ $|\{i \mid 0 < \alpha_i < C\}|$
- if(nr_free > 0) $\Longleftrightarrow$ 判断是否存在$0 < \alpha_i < C$
- lb $\Longleftrightarrow$ $-M(\alpha) = \max_{j \in I_{low}(\alpha)} y_j \nabla f(\alpha_j)$
- ub $\Longleftrightarrow$ $-m(\alpha) = \min_{i \in I_{up}(\alpha)} y_i \nabla f(\alpha)_i$
- r = (ub+lb)/2; $\Longleftrightarrow$ 取(59)的中点

参考论文

[1]《LIBSVM: A Library for Support Vector Machines》

[2]《Working Set Selection Using Second Order Information for Training Support Vector Machines》

[3]《Convex Optimization》

[4]《统计学习方法》

[5] http://blog.pluskid.org/?p=702

[6] http://blog.csdn.net/zhuyue3938199/article/details/7469868

[7] http://blog.csdn.net/zhuyue3938199/article/details/7605977

[8] http://blog.csdn.net/cyningsun/article/details/8705648