# 序列最小优化算法(SMO)

SMO是SVM的求解算法，SVM的对偶形式为：

$$
\begin{aligned}
&\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j x_i^T x_j \\
&\alpha_i \geq 0 \\
&\sum_{i=1}^{N} \alpha_i y_i = 0
\end{aligned}
$$

其中，$x$是数据点，$y$是分类标签，$\alpha$是我们要求的参数。在求出$\alpha$之后，超平面可以表示为：

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b$$

SMO算法就是要找出最优的$\alpha$值，具体做法为随机选取$\alpha_i$
、$\alpha_j$，将其他的$\alpha$固定，比方说我们选择$\alpha_1$、$\alpha_2$进行优化，则$\alpha_3$、$\alpha_4$、...、$\alpha_N$都被看作常数，因此SVM的对偶形式可以改写为：

$$
\begin{aligned}
\max_{\alpha_1,\alpha_2} A(\alpha_1,\alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2}\Bigg( & K_{1,1} y_1 y_1 \alpha_1^2 + 2K_{1,2} y_1 y_2 \alpha_1 \alpha_2 + K_{2,2} y_2^2 \alpha_2^2 + \\
& 2 y_1 \alpha_1 \sum_{i=3}^{N} \alpha_i y_i K_{i,1} + 2 y_2 \alpha_2 \sum_{i=3}^{N} \alpha_i y_i K_{i,2} \Bigg) + C \\
\text{其中：} K_{i,j} = x_i^T x_j &
\end{aligned}
$$

根据$\sum_{i=1}^{N} \alpha_i y_i = 0$可以得出：

$$
\begin{aligned}
\alpha_1 y_1 + \alpha_2 y_2 = k \\
\alpha_1 = k y_1 - \alpha_2 y_1 y_2
\end{aligned}
$$

将$\alpha_1$带入$A(\alpha_1,\alpha_2)$中，并对$\alpha_2$求导可得$\frac{\partial A}{\partial \alpha_2}$的表达式，令$\frac{\partial A}{\partial \alpha_2} = 0$，即可求出让$A(\alpha_1,\alpha_2)$最大的$\alpha_2$。但此时求的$\alpha_2$含有$\alpha_i$，我们可以利用$\alpha_1 y_1 + \alpha_2 y_2 = k$的条件，消去$\alpha_i$，最终求得：

$$
\begin{aligned}
&\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta} \\
&E_i = f(x_i) - y_i \qquad\qquad (\text{预测值与真实值的误差}) \\
&\eta = K_{1,1} + K_{2,2} - 2K_{1,2}
\end{aligned}
$$

# 软间隔与条件约束

软间隔就是允许一些样本分类出错，此时最优化目标就变成了，在满足最大化间隔的同时，出错的样本分类尽可能的少：

$$\min_{\omega,b} \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{m}\left(max\left\{0, y_i\left(\omega^T x_i + b\right) - 1\right\}\right)$$

通过构造拉格朗日函数，求偏导可以得到$\alpha$的一个约束范围：

$$0 \leq \alpha_i \leq C$$

然而，我们每次随机选择两个$\alpha$进行优化，这两个$\alpha$存在一个线性关系，我们可以得到$\alpha_1$和$\alpha_2$的上下界:

$$
\begin{aligned}
&\text{当 } y_i \neq y_j: \\
&L = \max\{0, \alpha_2^{old} - \alpha_1^{old}\} \\
&H = \min\{C, C + \alpha_2^{old} - \alpha_1^{old}\} \\
&\text{当 } y_i = y_j: \\
&L = \max\{0, \alpha_2^{old} + \alpha_1^{old} - C\} \\
&H = \min\{C, \alpha_2^{old} + \alpha_1^{old}\}
\end{aligned}
$$

因此当我们求出$\alpha$后，我们将$\alpha$约束到$[L, H]$这个范围内。

最后给出用$\alpha$求出其他参数的公式

$$
\begin{aligned}
\alpha_1^{new} &= \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha_2^{new}) \\
b_1^{new} &= -E_1 - y_1 K_{1,1} (\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{1,2} (\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\
b_2^{new} &= -E_2 - y_1 K_{1,2} (\alpha_1^{new} - \alpha_1^{old}) - y_2 K_{2,2} (\alpha_2^{new} - \alpha_2^{old}) + b^{old} \\
\omega &= \sum_{i=1}^{N} \alpha_i y_i x_i^T
\end{aligned}
$$

$$
b = \begin{cases} b_1^{new} & 0 < \alpha_1^{new} < C \\ b_2^{new} & 0 < \alpha_2^{new} < C \\ \dfrac{b_1^{new} + b_2^{new}}{2} & \text{others} \end{cases}
$$

# 实现

定义一个SMO类，`__init__`函数负责初始化$\alpha$，$\omega$，$b$等参数

```
class SMO():
    def __init__(self, data, label, C, iterations=40):
        """
        :data      :数据
        :label     :标签
        :C         :软间隔常数
        :iterations:最大迭代次数
        """
        self.data       = data
        self.label      = label
        self.C          = C
        self.iterations = iterations
        self.m,_        = self.data.shape
        self.alpha      = np.zeros(self.m)        # 一维向量
        self.b          = 0
        self.cnt        = 0
```

定义超平面方程$f(x) = \sum_{i=1}^{m} \alpha_i y_i x_i^T x + b$

```python
    def f(self, x):
        """
        预测值
        f = w.T*x+b
          = sum(alpha_i*y_i*x_i.T*x)+b
        """
        fx = (self.alpha*self.label).dot(x.dot(self.data.T)) + self.b
        return fx
```

smo求解

当前迭代次数小于最大迭代次数：

```python
def solution(self):
        while self.cnt < self.iterations:
```

1. $\alpha_i$依次取$\alpha_1, \alpha_2, \ldots, \alpha_N$，$\alpha_j$随机取一个于$\alpha_i$不同的$\alpha$值。

   ```python
   for i in range(self.m):
        jlist = [x for x in range(i)] + [x for x in range(i+1,self.m)]
        j = random.choice(jlist)
   ```

2. 根据公式求出$E_i$，$E_j$，$K_{i,i}$，$K_{j,j}$，$K_{i,j}$，$\eta$

   ```python
           fx_i = self.f(self.data[i])
           E_i = fx_i - self.label[i]
           fx_j = self.f(self.data[j])
           E_j = fx_j - self.label[j]
           K_ii = self.data[i].T.dot(self.data[i])
           K_jj = self.data[j].T.dot(self.data[j])
           K_ij = self.data[i].T.dot(self.data[j])
           eta = K_ii + K_jj - 2*K_ij
   ```

3. 求出新的$\alpha_j$值：$\alpha_j^{new} = \alpha_j^{old} + \frac{y_j(E_i - E_j)}{\eta}$

   ```python
           alpha_i_old, alpha_j_old = self.alpha[i], self.alpha[j]
           alpha_j_new = alpha_j_old + self.label[j]*(E_i - E_j)/eta
   ```

4. 将$\alpha_j$约束到$[L, H]$范围内

   ```python
           if self.label[i] != self.label[j]:
               L = max(0, alpha_j_old-alpha_i_old)
               H = min(self.C, self.C+alpha_j_old-alpha_i_old)
           else:
               L = max(0, alpha_j_old+alpha_i_old-self.C)
               H = min(self.C, alpha_j_old+alpha_i_old)

           # 将alpha值约束到(L,H)
           if alpha_j_new < L:
               alpha_j_new = L
           elif alpha_j_new > H:
               alpha_j_new = H
   ```

5. 求出新的$\alpha_i$，并将类中的$\alpha_i$，$\alpha_j$更新

```
    alpha_i_new = alpha_i_old + self.label[i]*(self.label[j])*(alpha_j_old-alpha_j_new)
    self.alpha[i], self.alpha[j] = alpha_i_new, alpha_j_new
```

6. 更新$b$

```
    b_i = -E_i - self.label[i]*K_ii*(alpha_i_new-alpha_i_old) - self.label[j]*K_ij*(alpha_j_new-alpha_j_old) + self.b
    b_j = -E_j - self.label[i]*K_ij*(alpha_i_new-alpha_i_old) - self.label[j]*K_jj*(alpha_j_new-alpha_j_old) + self.b

    if 0 < alpha_i_new < self.C:
        self.b = b_i
    elif 0 < alpha_j_new < self.C:
        self.b = b_j
    else:
        self.b = (b_i + b_j)/2
```

到这里smo就已经实现了。根据$\alpha$值，我们可以求出$\omega$，再结合求出的$b$，就可以求出超平面了。

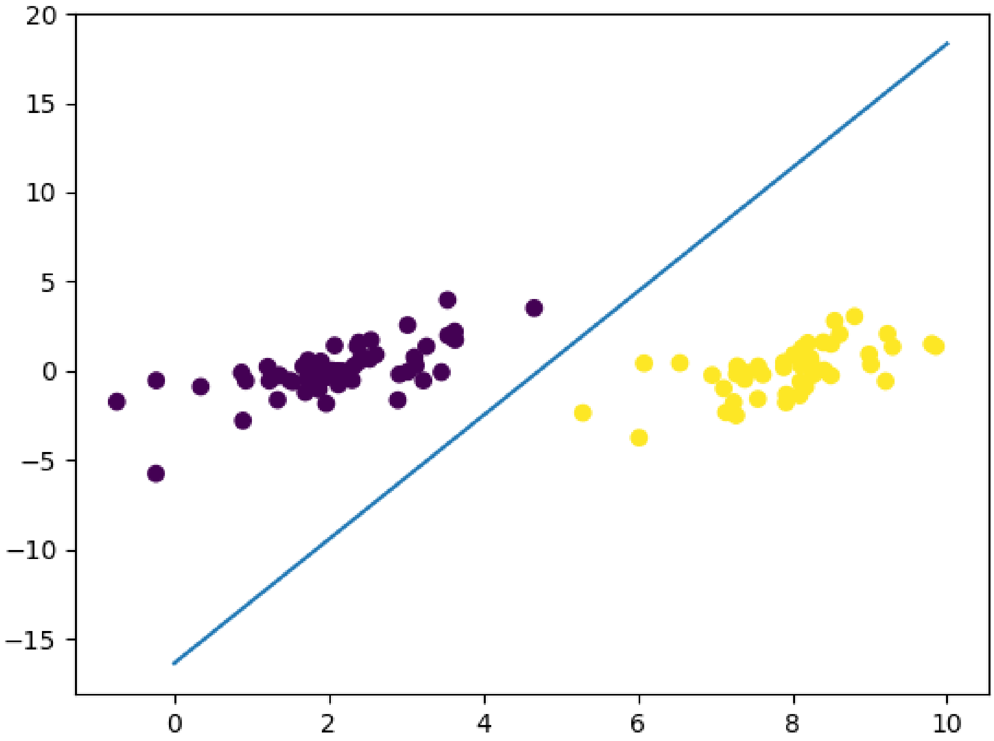
# 画出超平面

设我们求得的$\omega = (\omega_1, \omega_2)$，对于超平面上一点$x = (x_1, x_2)$，则

$$y = \omega_1 x_1 + \omega_2 x_2 + b = 0$$
$$x_2 = \frac{-b - \omega_1 x_1}{\omega_2}$$

```
if __name__ == '__main__':
    data,labels =load_data('./testSet.txt')
    smo = SMO(data,labels,0.6,100)
    alphas,b = smo.solution()
    w = (smo.alpha*smo.label).reshape(1,-1).dot(smo.data)
    y1 = (-smo.b - w[0,0]*0)/w[0,1]
    y2 = (-smo.b - w[0,0]*10)/w[0,1]

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(data[:,0],data[:,1],c=labels)
    ax.plot([0,10], [y1,y2])
```




```python
def solution(self):
    while self.cnt < self.iterations:
        for i in range(self.m):
            jlist = [x for x in range(i)] + [x for x in range(i+1,self.m)]
            j = random.choice(jlist)
            fx_i = self.f(self.data[i])
            E_i = fx_i - self.label[i]
            fx_j = self.f(self.data[j])
            E_j = fx_j - self.label[j]
            K_ii = self.data[i].T.dot(self.data[i])
            K_jj = self.data[j].T.dot(self.data[j])
            K_ij = self.data[i].T.dot(self.data[j])
            eta = K_ii + K_jj - 2*K_ij

            alpha_i_old, alpha_j_old = self.alpha[i], self.alpha[j]
            alpha_j_new = alpha_j_old + self.label[j]*(E_i - E_j)/eta

            if self.label[i] != self.label[j]:
                L = max(0, alpha_j_old-alpha_i_old)
                H = min(self.C, self.C+alpha_j_old-alpha_i_old)
            else:
                L = max(0, alpha_j_old+alpha_i_old-self.C)
                H = min(self.C, alpha_j_old+alpha_i_old)

            # 将alpha值约束到(L,H)
            if alpha_j_new < L:
                alpha_j_new = L
            elif alpha_j_new > H:
                alpha_j_new = H
```

```
                alpha_i_new = alpha_i_old + self.label[i]*(self.label[j])*
(alpha_j_old-alpha_j_new)

                if abs(alpha_j_new-alpha_j_old) < 0.00001:
                    continue

                self.alpha[i], self.alpha[j] = alpha_i_new, alpha_j_new

                b_i = -E_i - self.label[i]*K_ii*(alpha_i_new-alpha_i_old) -
self.label[j]*K_ij*(alpha_j_new-alpha_j_old) + self.b
                b_j = -E_j - self.label[i]*K_ij*(alpha_i_new-alpha_i_old) -
self.label[j]*K_jj*(alpha_j_new-alpha_j_old) + self.b

                if 0 < alpha_i_new < self.C:
                    self.b = b_i
                elif 0 < alpha_j_new < self.C:
                    self.b = b_j
                else:
                    self.b = (b_i + b_j)/2

            self.cnt += 1

        return self.alpha, self.b


if __name__ == '__main__':
    data,labels =load_data('./testSet.txt')
    smo = SMO(data,labels,0.6,100)

    alphas,b = smo.solution()
    print(alphas,b)
    w = (smo.alpha*smo.label).reshape(1,-1).dot(smo.data)
    y1 = (-smo.b - w[0,0]*np.array([0]))/w[0,1]
    y2 = (-smo.b - w[0,0]*np.array([10]))/w[0,1]

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(data[:,0],data[:,1],c=labels)
    ax.plot([0,10], [y1,y2])

    # 绘制支持向量
    for i, alpha in enumerate(alphas):
        if abs(alpha) > 1e-3:
            x, y = data[i]
            ax.scatter([x], [y], s=150, c='none', alpha=0.7,
                       linewidth=1.5, edgecolor='#AB3319')
    plt.show()
```