C1-7

Machine Learning
by Andrew Ng, Stanford Engineering

Xiaojie Zhou

szxjzhou@163.com

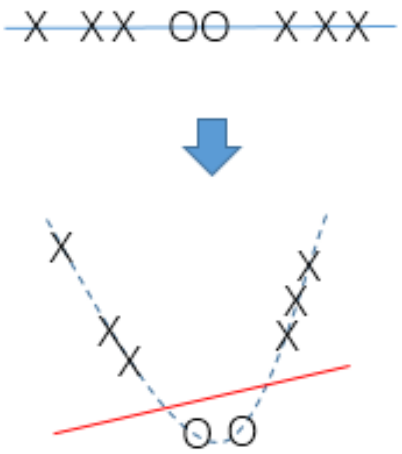
2016.8.29

第八集：顺序最小优化算法

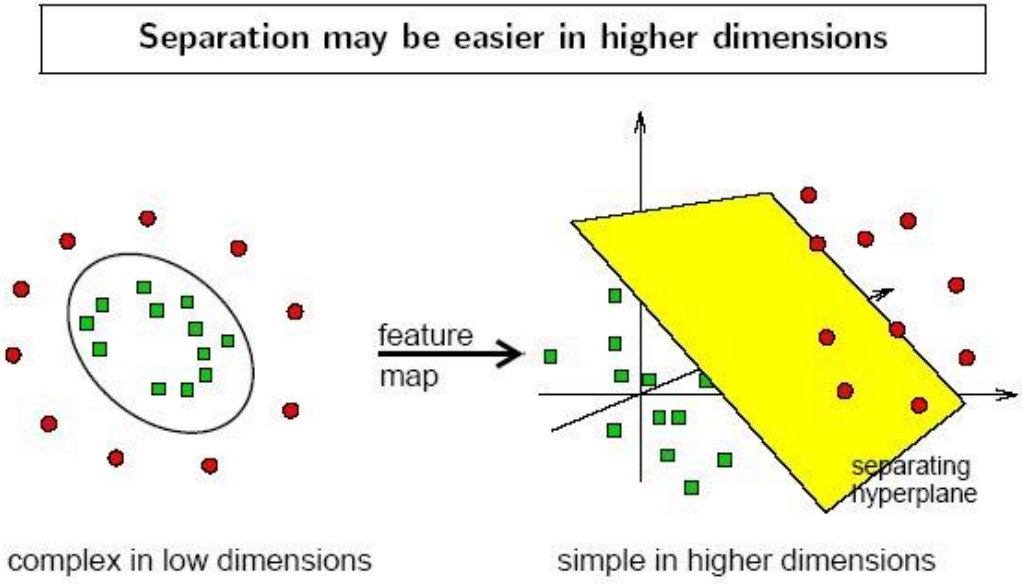
- 核(Kernels)
- 软间隔(Soft Margin)
- 顺序最小优化算法(Sequential Minimal Optimization(SMO) Algorithm)

第八集：顺序最小优化算法

- 核
  - 之前对于SVM的推导建立在数据样本线性可分的基础之上，但对于线性不可分的数据样本就无能为力
    - 既然我们对于线性不可分的样本无能为力，那么我们就要考虑有没有把原来线性不可分的样本变成线性可分的



其实样本无非就是空间上的一个点。此时我们发现，用一条曲线将原数据样本映射到更高维的空间上，就有可能能找到超平面进行划分。这样的映射方式有很多，对于不同特征的数据来说有着不同的映射办法



第八集：顺序最小优化算法

- 核
  - 由此可见将线性不可分的样本变成线性可分关键在于进行从低维到高维的特征映射(feature mapping)的方法
    - 我们将特征x进行特征映射后的结果记为ϕ(x)，这样的方法有很多，针对不同的数据有不同的方法，下面就展示了一种简单的特征映射方法：

$$x \longrightarrow \phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

    - 当我们建立好某种特征映射ϕ后，只需要用ϕ(x)替换原特征x，即可成功将低维数据映射到高维数据，而这样映射后的高维数据是有可能线性可分的

第八集：顺序最小优化算法

- 核
  - 对于低维线性不可分的问题，将数据特征x进行高维特征映射是可行的方案，但这样做带来的直接问题是映射后向量维数的增加
    - 向量维数的增加带来的直接问题就是内存空间的不足（ϕ(x)比x更加占用内存空间）和运算时间（ϕ(x)比x运算量更大）的迅速增大，甚至使得ϕ(x)无法进行表示，这也就是维数灾难(curse of dimension)问题
    - 而核(kernels)方法提供了一种ϕ(x)内积（点乘）计算的高效解决方法
      - 给定某种特征映射ϕ，定义该特征映射的核为任意两个向量x, z在特征映射后的点乘

$$K(x,z) = \phi(x)^T \phi(z)$$

      - 因此当我们建立好某种特征映射ϕ后，对于任意两个向量x, z的点乘<x, z>都可以用K(x, z)进行代替

第八集：顺序最小优化算法

- 核
  - 对于核来说，甚至可以在不显式表示出ϕ(x)的情况下进行对于任意两个向量x, z在特征映射后的点乘进行运算
    - 由于核具有这样的优势，因此我们更加倾向于显式构造核函数，对于特征映射ϕ进行隐式的表示（实际上特征映射的主要目的就是将低维向量映射到高维而已），从而达到节省内存空间，加快运算速度的目的

$$K(x,z) = (x^T z)^2$$

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) \end{aligned}$$

实际上左边这个非常简单的核函数就隐含着一种特征映射的方法。这种特征映射能将一个n维的向量映射成一个n^2维的向量（右图中展示了将一个3维向量对应映射成9维向量的例子）。从中可以看到核函数在进行向量特征映射后的点乘计算的优势（原本单纯计算出φ(x)时间复杂度已经是$O(n^2)$了，现在计算内积只要O(n)）

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

第八集：顺序最小优化算法

- 核
  - 下面来看一些经典的核函数
    - 1、线性核函数：K(x, z)=<x, z>（核函数的最基本形式）
    - 2、多项式核函数： K(x, z)=(k<x, z>+c)^d，其中k, c, d均为系数

假设 x, z 均为 n 维的向量：

$$K(x,z)=(k<x,z>+c)^d=C_d^0(k<x,z>)^d+C_d^1(k<x,z>)^{d-1}c+\cdots+C_d^dc^d$$

$$=\begin{bmatrix}(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x1^d\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x1^{d-1}x2\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x1^{d-1}xn\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x2^d\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x2^{d-1}x1\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}x2^{d-1}xn\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}xn^d\\(C_d^1)^{\frac{1}{2}}k^{d-\frac{1}{2}}x1^{d-1}c^{\frac{1}{2}}\\\vdots\\(C_d^1)^{\frac{1}{2}}k^{d-\frac{1}{2}}xn^{d-1}c^{\frac{1}{2}}\\\vdots\\(C_d^d)^{\frac{1}{2}}c^{\frac{d}{2}}\end{bmatrix}\cdot\begin{bmatrix}(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z1^d\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z1^{d-1}z2\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z1^{d-1}zn\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z2^d\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z2^{d-1}z1\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}z2^{d-1}zn\\\vdots\\(C_d^0)^{\frac{1}{2}}k^{\frac{d}{2}}zn^d\\(C_d^1)^{\frac{1}{2}}k^{d-\frac{1}{2}}z1^{d-1}c^{\frac{1}{2}}\\\vdots\\(C_d^1)^{\frac{1}{2}}k^{d-\frac{1}{2}}zn^{d-1}c^{\frac{1}{2}}\\\vdots\\(C_d^d)^{\frac{1}{2}}c^{\frac{d}{2}}\end{bmatrix}=<\phi(x),\phi(z)>$$

$$\begin{aligned}K(x,z) &= (x^Tz+c)^2\\ &= \sum_{i,j=1}^{n}(x_ix_j)(z_iz_j)+\sum_{i=1}^{n}(\sqrt{2c}x_i)(\sqrt{2c}z_i)+c^2.\end{aligned}$$

$$\phi(x)=\begin{bmatrix}x_1x_1\\x_1x_2\\x_1x_3\\x_2x_1\\x_2x_2\\x_2x_3\\x_3x_1\\x_3x_2\\x_3x_3\\\sqrt{2c}x_1\\\sqrt{2c}x_2\\\sqrt{2c}x_3\\c\end{bmatrix}$$

这里展示了多项式核函数的一般展开情况，并举了一个三维向量下运用多项式核函数进行对应特征映射后向量内积运算的问题，从中可以更进一步看到核函数在进行向量特征映射后的点乘计算的优势

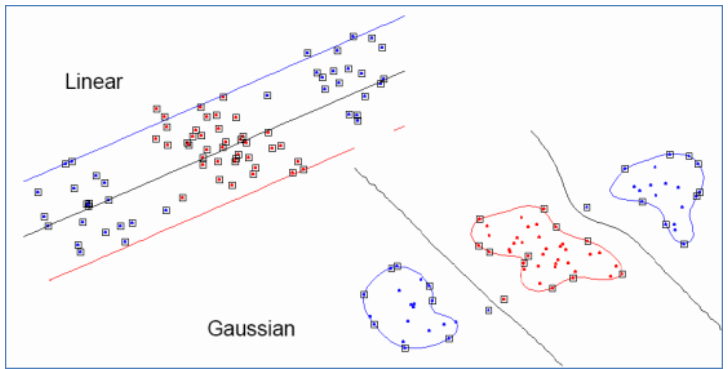
第八集：顺序最小优化算法

- 核
  - 下面来看一些经典的核函数
    - 3、径向基函数：一个取值仅仅依赖于离某一参考点距离的实值函数，如果两个样本点距离小，那么这两个点特征映射后的点乘小；如果两个样本点距离大，那么这两个点特征映射后的点乘大，其中比较经典的是高斯核函数(Gaussian kernel)

$$K(x,z)=\exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$$

这里面σ的值根据实际情况进行定义，方法是选取一个小的训练样本集试出一个还不错的σ的值

- 而这样的核函数被广泛用于机器学习的问题中，原因在于这样的函数能将相近的样本聚在一起，从而大大提升样本在升维后的线性可分性



第八集：顺序最小优化算法

- 核
  - 但是对于径向基函数来说往往我们无法很容易地像多项式核函数一样显式地将特征映射ϕ表示出来
    - 因此对于一个核函数来说，我们怎么知道这个核存在着对应的特征映射ϕ（即使这个特征映射是不能被显式表示的）
      - 在探究这个问题前，先引入一个概念“核矩阵”(Kernel matrix)：对于有n个样本的训练集X来说（X={x1, x2, ..., xn}），其核矩阵为一个n阶方阵，其每个元素Kij为对应样本核函数运算的值（即Kij=K(xi, xj)=< ϕ(xi), ϕ(xj) >）
        - 从中可以很明显看出，核矩阵是对称阵
        - 与此同时，核矩阵应是半正定矩阵（即对于任意向量z，z^TKz的结果非负）

$$
\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j &&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j &&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j &&\geq 0.
\end{aligned}
$$

第八集：顺序最小优化算法

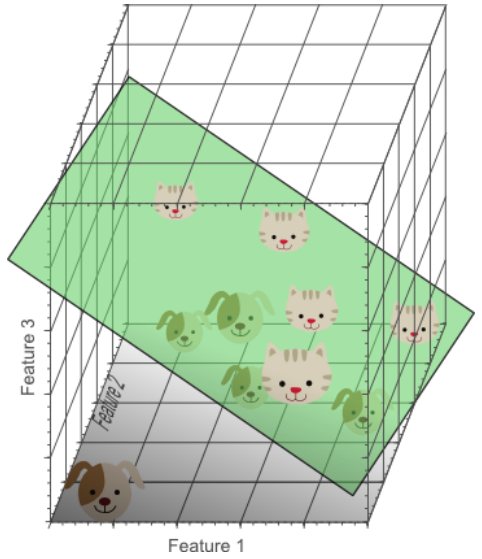
- 核
  - 实际上，这也是一个存在着对应的特征映射ϕ的核的充要条件
    - Mercer定理：一个存在着对应的特征映射ϕ的核函数当且仅当对于任意有限个样本的集合X（ X={x1, x2, …, xn}, n<∞），其核矩阵是一个对称半正定矩阵
      - 这个定理的重要意义在于给出了检验核函数正确性的方法，因此核也被称为Mercer核
  - 核方法作为特征升维映射后的高效点乘计算法，在众多领域有着普遍的应用
    - 其中一个就是应用于SVM中，直接将SVM中的点乘结果做替换就可以将原本线性不可分的样本变成线性可分

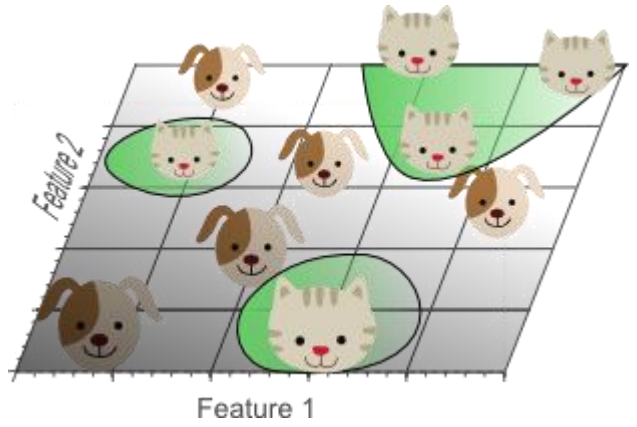
$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

$$\begin{aligned} w^T x + b &= \left(\sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\ &= \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b. \end{aligned}$$

第八集：顺序最小优化算法

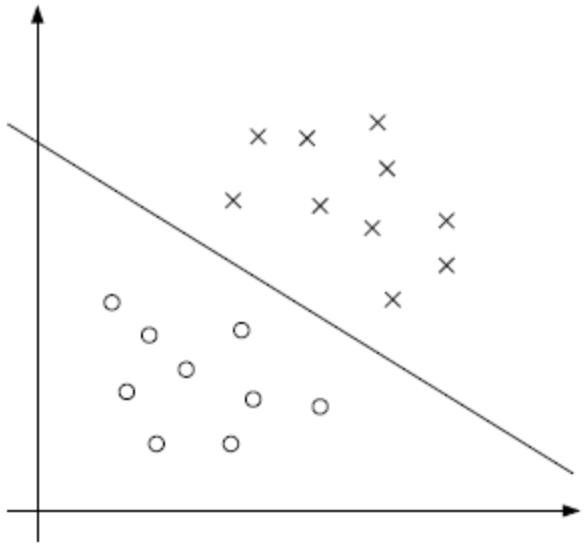
- 核
  - 但是对于将低维特征升到高维来说，固然极大地增大了原来线性不可分的样本变成线性可分的可能性
    - 但是数据永远不可能是完美的，将低维特征升到高维并不能保证数据一定变得线性可分，而且一味地变成线性可分将会导致严重的过拟合问题，失去了泛化的能力（这也是维数灾难的另一大问题所在）

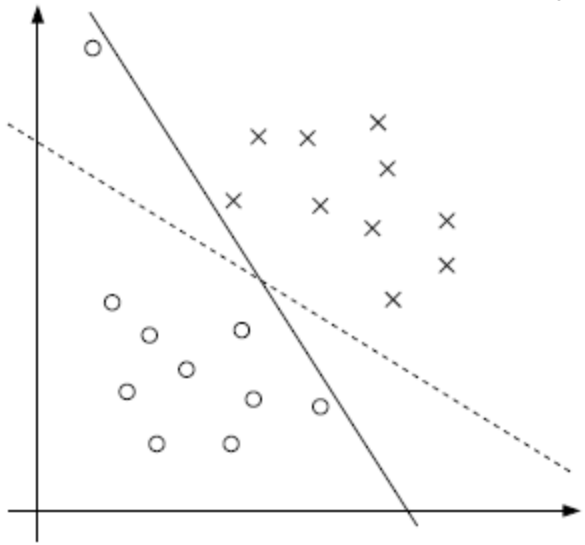




第八集：顺序最小优化算法

- 软间隔
  - 由于将低维特征升到高维并不能保证数据一定变得线性可分，而且为了避免出现严重的过拟合问题，我们考虑将严格线性可分放松限制





第八集：顺序最小优化算法

- 软间隔
  - 因此我们对于原优化问题做出如下的修改

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

$$\min_{\gamma,w,b} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{m} \xi_i$$
$$\text{s.t.} \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m$$
$$\xi_i \geq 0, \quad i = 1, \ldots, m.$$

  - 这种方法也被称为l1-正则化(l1-regularization)
    - 首先我们在不等式约束条件中加上一个参数ξ，该参数放松了对于划分准确性的限制（因为当$y(\omega^T x+b)\geq 0$时划分是正确的，当$y(\omega^T x+b)<0$时划分是错误的），因此现在这约束表达的内容从“划分正确且函数距离至少为1”变成了“划分正确且函数距离至少为1-ξ”，甚至再到“划分错误但错误的函数距离至多为ξ-1”
    - 但与此同时我们也不鼓励划分错误的行为的发生，因此在目标函数处加上了惩罚项，其中惩罚项的系数C控制了惩罚的程度，如果C值大那么划分错误的代价大。反之，C值小那么划分错误的代价小。由于在目标函数处加上了惩罚项，因此我们还需要限制ξ的值大于等于0，从而令惩罚项至少为0
    - 修改后仍然是个凸优化问题

第八集：顺序最小优化算法

- 软间隔
  - 下面将开始求解修改后的凸优化问题
    - 求解第一步：先得到对应的广义Lagrange算子的表达形式

$$\mathcal{L}(w,b,\xi,\alpha,r)=\frac{1}{2}w^Tw+C\sum_{i=1}^{m}\xi_i-\sum_{i=1}^{m}\alpha_i\left[y^{(i)}(x^Tw+b)-1+\xi_i\right]-\sum_{i=1}^{m}r_i\xi_i$$

    - 求解第二步对于参数ω进行微分
      - 在这里面参数ω由超平面斜率ω，截距项b和正则化项ξ构成，因此需要对这些参数分别求偏微分，结果为：

$$w=\sum_{i=1}^{m}\alpha_i y^{(i)}x^{(i)}\qquad\sum_{i=1}^{m}\alpha_i y^{(i)}=0\qquad C-a_i-r_i=0$$

    - 求解第三步，对于参数β进行微分（由于这里没有参数β因此跳过这一步）

第八集：顺序最小优化算法

- 软间隔
  - 下面将开始求解修改后的凸优化问题
    - 求解第四步：用参数α替换原式中参数ω和β，并进行化简，最后得到：

$$\mathcal{L}(w,b,\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j(x^{(i)})^Tx^{(j)}$$

    - 求解第五步：写出原问题等价的对偶优化问题

$$\begin{aligned}\max_\alpha\quad & W(\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j\langle x^{(i)},x^{(j)}\rangle\\ \text{s.t.}\quad & 0\le\alpha_i\le C,\quad i=1,\dots,m\\ & \sum_{i=1}^{m}\alpha_iy^{(i)}=0,\end{aligned}$$

实际上第一个约束条件0≤αi≤C由三个约束条件合并而来，这三个条件分别是：0≤αi，C-αi-ri=0，0≤ri

第八集：顺序最小优化算法

- 软间隔
  - 与此同时根据KKT条件不难发现存在如下的结论：

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

由于KKT条件中说明αi^*和$gi(\omega^*)$中至少一项为0，对应到这个例子中αi与$y(\omega^T x+b)-1+\xi i$中有一项为0，ri与ξi中有一项为0。对于第一条结论来说，$\alpha i=0$说明$y(\omega^T x+b)-1+\xi i \leqslant 0$，由于$C-\alpha i-ri=0$，因此$ri \neq 0$，$\xi i=0$，$y(\omega^T x+b) \geqslant 1$；对于第二条结论来说，$\alpha i=C$说明$y(\omega^T x+b)-1+\xi i=0$，由于$C-\alpha i-ri=0$，因此$ri=0$，$\xi i \leqslant 0$，$y(\omega^T x+b) \leqslant 1$）；对于第三条结论来说，$0<\alpha i<C$说明$\alpha i$和ri均不为0，这也就使得$y(\omega^T x+b)-1+\xi i=0$，$\xi i=0$，$y(\omega^T x+b)=1$

通过$0<\alpha i<C \rightarrow y(\omega^T x+b)=1$可以发现支持向量的特点为$0<\alpha i<C$，从而将这些支持向量找出来即可通过之前的办法求出超平面截距b

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 现在对于SVM最后一个问题进行求解，如何求解如下的对偶优化问题

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

  - 在考虑这样一个问题前先考虑在无约束的情况下如何进行优化

$$\max_{\alpha} W(\alpha_1, \alpha_2, \ldots, \alpha_m).$$

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 对于一个无约束的求一个凸函数的最大值的问题求解的方法很多，比如说之前讲过的梯度下降法、牛顿方法
    - 实际上还有一种高效的方法用于求一个凸函数的最大值的问题，称为坐标上升法(coordinate ascent)，其算法核心在于控制变量优化（每一次只优化一个参数，而其它参数保持不变，然后依次对每个参数进行优化）

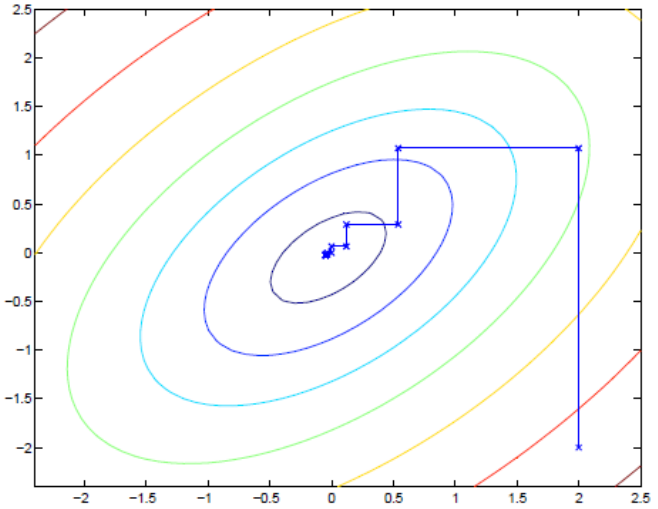
Loop until convergence: {

For $i = 1, \ldots, m$, {

$\alpha_i := \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$.

}

}



第八集：顺序最小优化算法

- 顺序最小优化算法
  - 实际上对于坐标上升法来说存在着一些变种一般会使得求解效率更高
    - 其中一种对算法的改进在于不需要每次从第一个参数顺序优化到最后一个参数，可以根据实际情况选择参数的优化顺序
  - 但是对于之前提到的优化问题来说坐标上升法不能直接应用
    - 其原因在于第二项约束中约束了α的加权和，因此无法在固定其它参数的情况下对某一个参数进行优化
    - 由此得到坐标上升法的另一种改进方法，每次针对两个参数进行优化，这也就得到了顺序最小优化(SMO)算法

$$
\begin{aligned}
\max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
\text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\
& \sum_{i=1}^{m} \alpha_i y^{(i)} = 0,
\end{aligned}
$$

第八集：顺序最小优化算法

- 顺序最小优化算法
  - SMO算法的整体流程如下

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum).
2. Reoptimize $W(\alpha)$ with respect to α_i and α_j, while holding all the other α_k's $(k \neq i, j)$ fixed.

}

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

这里面提到了任选两个参数进行优化的问题。其实就算这两个参数选的不好，带来的后果无非收敛慢一点，也不会影响最终的收敛。一种比较简单的选取方法为选距离收敛条件最远的参数。而对于SVM的优化问题来说，可以拿左下三个式子作为收敛的条件。

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 下面更进一步来讨论如何在固定其它参数的情况下进行两个变量的优化
    - 在这里以α1, α2作为优化的对象进行优化，首先先要表示出对应的约束条件
      - 首先在约束条件中对于α的线性加权和进行了如下的限制：

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

      - 因此对于α1, α2这两个优化对象来说有：

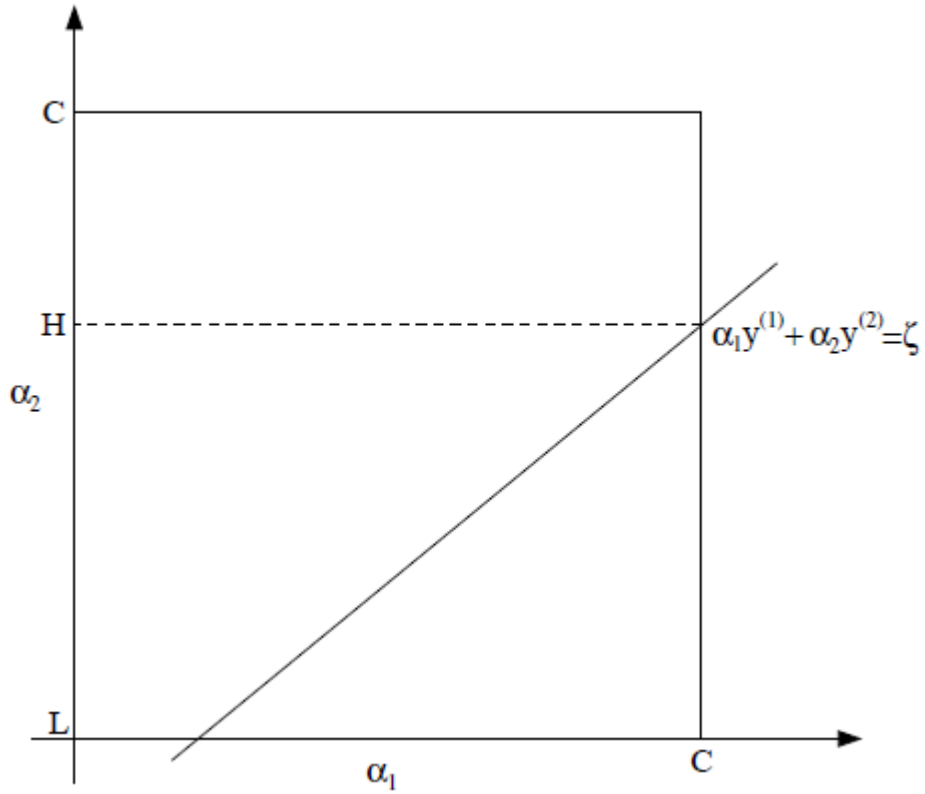
$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{m} \alpha_i y^{(i)}$$

      - 由于其他的参数都是固定的，因此等式的右边是固定的，可以用系数ζ进行表示：

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta$$

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 接下来加上另外一个限制条件考虑在这两个约束条件下的优化问题



在左边这幅图中展示了两个约束条件共同作用下的情况，对于另一个约束条件0≤αi≤C来说，在只考虑两个参数α1, α2的情况下相当于一个矩形框限制了参数α1, α2的取值必须在框内，而另一个约束条件相当于一条直线限制了参数α1, α2的取值必须在线上。因此最终α1, α2的将在一条线段上进行取值

由于参数α1, α2的取值必须在线上，因此我们可以用α2表示出α1从而使得整个问题只有一个变量α2（除参数α1, α2其余变量均被限制。与此同时，在下面这个式子中使用了一个技巧，由于y={-1, +1}，因此$y^2=1$，在下面这个式子中左右两边同时乘上了y）

$$\alpha_1 = (\zeta - \alpha_2 y^{(2)}) y^{(1)}$$

$$W(\alpha_1, \alpha_2, \ldots, \alpha_m) = W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \ldots, \alpha_m)$$

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 这时由于我们已经通过α2表示出α1从而使得整个问题变成了单变量优化问题，此时用α2表示出的α1代入下面所示的原优化目标函数

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

    - 代入后发现，这个式子变成了单变量α2的二次函数优化问题。只不过这里面还限制了二次函数定义域的取值（还有对于α的取值约束在），但是这求解起来已经非常简单了（因为对于某段连续的二次函数来说，其最值要么在边界取得，要么就是原二次函数的全局最值）

$$\alpha_2^{new} = \begin{cases} H & \text{if } \alpha_2^{new,unclipped} > H \\ \alpha_2^{new,unclipped} & \text{if } L \leq \alpha_2^{new,unclipped} \leq H \\ L & \text{if } \alpha_2^{new,unclipped} < L \end{cases}$$

第八集：顺序最小优化算法

- 顺序最小优化算法
  - 最后来总结一下顺序最小优化算法的特点
    - 实际上顺序最小优化算法的收敛步骤要比牛顿方法要来得多（实际上收敛步骤也和参数优化的顺序相关，关于这个问题在Platt的论文中有说明），但这个方法的好处在于进行每一步迭代的成本比较低，而牛顿方法进行每步迭代的成本非常高
      - 论文：" Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines" by John Platt
    - SMO算法的核心实际上还是用类似于坐标上升的方法对每一个参数依次进行优化，每次要挑选两个参数的原因在于存在对于参数的加权和的限制，而这样的优化是有效的，因为每次迭代都在接近最优解