机器学习实战教程（八）：支持向量机原理篇之手撕线性SVM

摘要

本篇文章参考了诸多大牛的文章写成的，对于什么是SVM做出了生动的阐述，同时也进行了线性SVM的理论推导，以及最后的编程实践，公式较多，还需静下心来一点一点推

一、前言

说来惭愧，断更快半个月了，本打算是一周一篇的。感觉SVM瞬间难了不少，推导耗费了很多时间，同时身边的事情也不少，忙了许久。本篇文章参的文章写成的，对于什么是SVM做出了生动的阐述，同时也进行了线性SVM的理论推导，以及最后的编程实践，公式较多，还需静下心来一点一点推导。

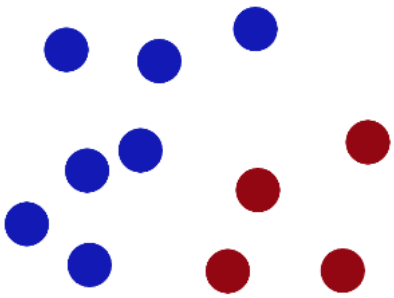
本文出现的所有代码，均可在我的github上下载，欢迎Follow、Star：https://github.com/Jack-Cherish/Machine-Learning

二、什么是SVM？

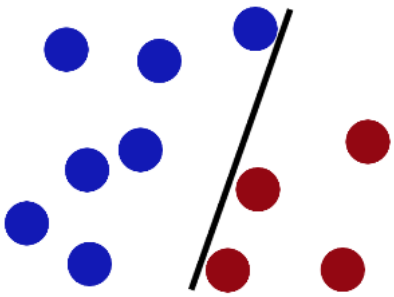
SVM的英文全称是Support Vector Machines，我们叫它支持向量机。支持向量机是我们用于分类的一种算法。让我们以一个小故事的形式，开启我吧。

在很久以前的情人节，一位大侠要去救他的爱人，但天空中的魔鬼和他玩了一个游戏。

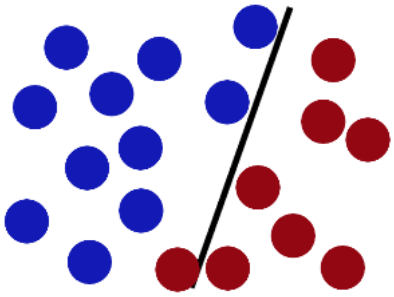
魔鬼在桌子上似乎有规律放了两种颜色的球，说："你用一根棍分开它们？要求：尽量在放更多球之后，仍然适用。"



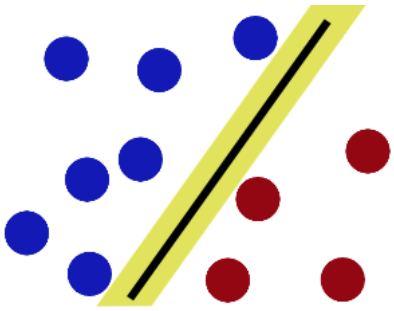
于是大侠这样放，干的不错？



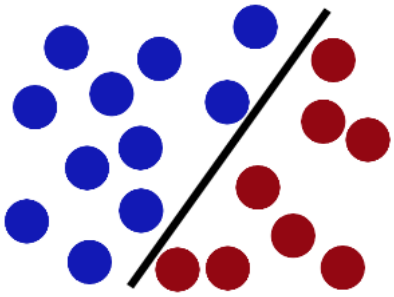
然后魔鬼，又在桌上放了更多的球，似乎有一个球站错了阵营。显然，大侠需要对棍做出调整。



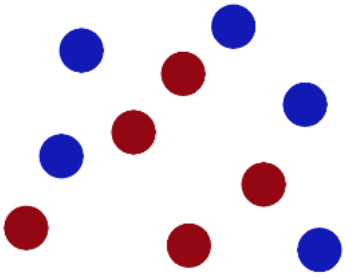
SVM就是试图把棍放在最佳位置，好让在棍的两边有尽可能大的间隙。这个间隙就是球到棍的距离。



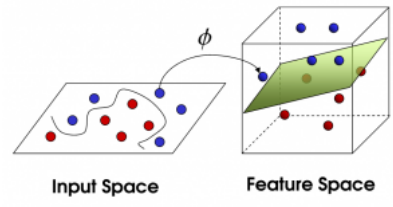
现在好了，即使魔鬼放了更多的球，棍仍然是一个好的分界线。



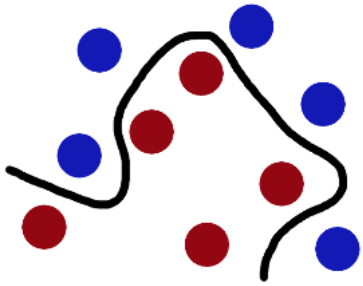
魔鬼看到大侠已经学会了一个trick(方法、招式)，于是魔鬼给了大侠一个新的挑战。



现在，大侠没有棍可以很好帮他分开两种球了，现在怎么办呢？当然像所有武侠片中一样大侠桌子一拍，球飞到空中。然后，凭借大侠的轻功，大侠到了两种球的中间。



现在，从空中的魔鬼的角度看这些球，这些球看起来像是被一条曲线分开了。



再之后，无聊的大人们，把这些球叫做**data**，把棍子叫做**classifier**, 找到最大间隙的**trick**叫做**optimization**，拍桌子叫做**kernelling,** 那张纸叫做**hy**

更为直观地感受一下吧(需要翻墙)：https://www.youtube.com/watch?v=3liCbRZPrZA

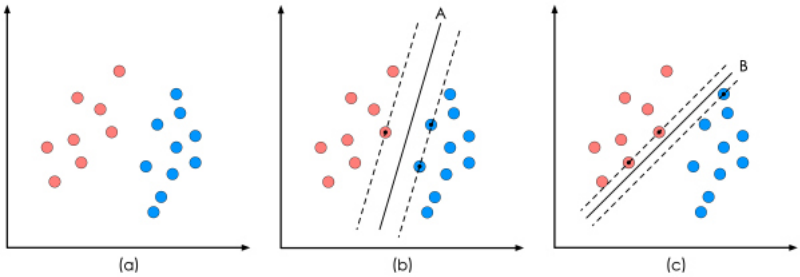
概述一下：

当一个分类问题，数据是线性可分的，也就是用一根棍就可以将两种小球分开的时候，我们只要将棍的位置放在让小球距离棍的距离最大化的位置即可
大间隔的过程，就叫做最优化。但是，现实往往是很残酷的，一般的数据是线性不可分的，也就是找不到一个棍将两种小球很好的分类。这个时候，我们就
样，将小球拍起，用一张纸代替小棍将小球进行分类。想要让数据飞起，我们需要的东西就是核函数(kernel)，用于切分小球的纸，就是超平面。

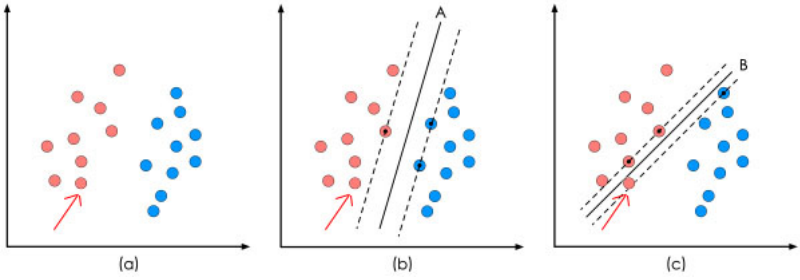
也许这个时候，你还是似懂非懂，没关系。根据刚才的描述，可以看出，问题是从线性可分延伸到线性不可分的。那么，我们就按照这个思路，进行

三、线性SVM

先看下线性可分的二分类问题。



上图中的(a)是已有的数据，红色和蓝色分别代表两个不同的类别。数据显然是线性可分的，但是将两类数据点分开的直线显然不止一条。上图的(b)和
B、C两种不同的分类方案，其中黑色实线为分界线，术语称为“决策面”。每个决策面对应了一个线性分类器。虽然从分类结果上看，分类器A和分类器B
的。但是他们的性能是有差距的，看下图：



在"决策面"不变的情况下，我又添加了一个红点。可以看到，分类器B依然能很好的分类结果，而分类器C则出现了分类错误。显然分类器B的"决策面
于分类器C的"决策面"放置的位置，SVM算法也是这么认为的，它的依据就是分类器B的分类间隔比分类器C的分类间隔大。这里涉及到第一个SVM独有的
隔"。在保证决策面方向不变且不会出现错分样本的情况下移动决策面，会在原来的决策面两侧找到两个极限位置（越过该位置就会产生错分现象），如虚
的位置由决策面的方向和距离原决策面最近的几个样本的位置决定。而这两条平行虚线正中间的分界线就是在保持当前决策面方向不变的前提下的最优决
之间的垂直距离就是这个最优决策面对应的分类间隔。显然每一个可能把数据集正确分开的方向都有一个最优决策面（有些方向无论如何移动决策面的位
类样本完全分开），而不同方向的最优决策面的分类间隔通常是不同的，那个具有“最大间隔”的决策面就是SVM要寻找的最优解。而这个真正的最优解
线所穿过的样本点，就是SVM中的支持样本点，称为"支持向量"。

1、数学建模

求解这个"决策面"的过程，就是最优化。一个最优化问题通常有两个基本的因素：1）目标函数，也就是你希望什么东西的什么指标达到最好；2）优
通过改变哪些因素来使你的目标函数达到最优。在线性SVM算法中，目标函数显然就是那个"分类间隔"，而优化对象则是决策面。所以要对SVM问题进行
先要对上述两个对象（"分类间隔"和"决策面"）进行数学描述。按照一般的思维习惯，我们先描述决策面。

数学建模的时候，先在二维空间建模，然后再推广到多维。

（1）"决策面"方程

我们都知道二维空间下一条直线的方式如下所示：

$$y = ax + b$$

现在我们做个小小的改变，让原来的x轴变成x1，y轴变成x2

$$x_2 = ax_1 + b$$

移项得：

$$ax_1 - x_2 + b = 0$$

将公式向量化得：

$$[a \quad -1]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + b = 0$$

进一步向量化，~~用~~w列向量和x列向量和标量y进一步向量化：

$$\boldsymbol{\omega}^T \boldsymbol{x} + \gamma = 0$$

其中，向量w和x分别为：

$$\boldsymbol{\omega} = [\omega_1, \omega_2]^T, \boldsymbol{x} = [x_1, x_2]^T$$

这里w1=a，w2=-1。我们都知道，最初的那个直线方程a和b的几何意义，a表示直线的斜率，b表示截距，a决定了直线与x轴正方向的夹角，b决定了
点位置。那么向量化后的直线的w和r的几何意义是什么呢？

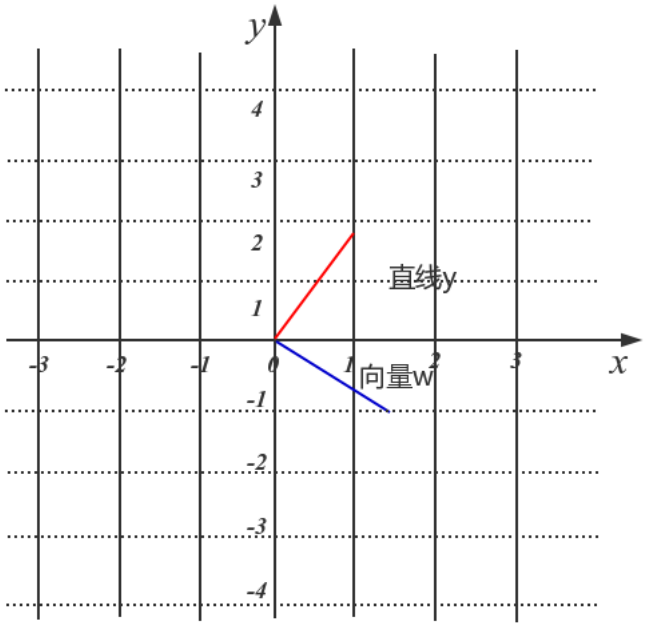
现在假设：

$$a = \sqrt{3}, b = 0$$

可得：

$$\boldsymbol{\omega} = [\sqrt{3}, -1]^T$$

在坐标轴上画出直线和向量w：



蓝色的线代表向量w，红色的线代表直线y。我们可以看到向量w和直线的关系为垂直关系。这说明了向量w也控制这直线的方向，只不过是与这个直线的。标量γ的作用也没有变，依然决定了直线的截距。此时，我们称w为直线的法向量。

二维空间的直线方程已经推导完成，将其推广到n维空间，就变成了超平面方程。(一个超平面，在二维空间的例子就是一个直线)但是它的公式没变，

$$\boldsymbol{\omega}^T\boldsymbol{x}+\gamma=0$$

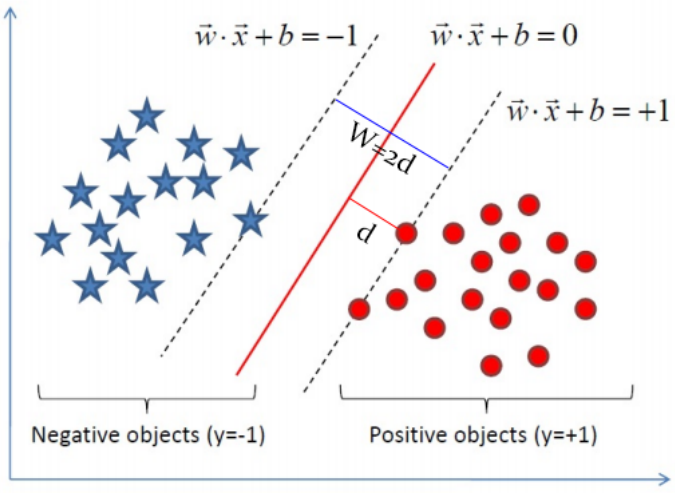
不同之处在于：

$$\boldsymbol{\omega}=[\omega_1,\omega_2,\cdots,\omega_n]^T$$
$$\boldsymbol{x}=[x_1,x_2,\cdots,x_n]^T$$

我们已经顺利推导出了"决策面"方程，它就是我们的超平面方程，之后，我们统称其为超平面方程。

（2）"分类间隔"方程

现在，我们依然对于一个二维平面的简单例子进行推导。



我们已经知道间隔的大小实际上就是支持向量对应的样本点到决策面的距离的二倍。那么图中的距离d我们怎么求？我们高中都学过，点到直线的距离下：

$$d=\left|\frac{Ax_0+By_0+C}{\sqrt{A^2+B^2}}\right|$$

公式中的直线方程为Ax0+By0+C=0，点P的坐标为(x0,y0)。

现在，将直线方程扩展到多维，求得我们现在的超平面方程，对公式进行如下变形：

$$d=\frac{|\boldsymbol{\omega}^T\boldsymbol{x}+\gamma|}{||\boldsymbol{\omega}||}$$

这个d就是"分类间隔"。其中||w||表示w的二范数，求所有元素的平方和，然后再开方。比如对于二维平面：

$$\boldsymbol{\omega}=[\omega_1,\omega_2]^T$$

那么，

$$||\boldsymbol{\omega}||=\sqrt[2]{{\omega_1}^2+{\omega_2}^2}$$

我们目的是为了找出一个分类效果好的超平面作为分类器。分类器的好坏的评定依据是分类间隔W=2d的大小，即分类间隔w越大，我们认为这个超平越好。此时，求解超平面的问题就变成了求解分类间隔W最大化的为题。W的最大化也就是d最大化的。

（3）约束条件

看起来，我们已经顺利获得了目标函数的数学形式。但是为了求解w的最大值。我们不得不面对如下问题：

- 我们如何判断超平面是否将样本点正确分类？
- 我们知道要求距离d的最大值，我们首先需要找到支持向量上的点，怎么在众多的点中选出支持向量上的点呢？

上述我们需要面对的问题就是约束条件，也就是说我们优化的变量d的取值范围受到了限制和约束。事实上约束条件一直是最优化问题里最让人头疼的我们已经知道了这些约束条件确实存在，就不得不用数学语言对他们进行描述。但SVM算法通过一些巧妙的小技巧，将这些约束条件融合到一个不等式里

这个二维平面上有两种点，我们分别对它们进行标记：

- 红颜色的圆点标记为1，我们人为规定其为正样本；
- 蓝颜色的五角星标记为-1，我们人为规定其为负样本。

对每个样本点xi加上一个类别标签yi：

$$y_i = \begin{cases} +1 & \text{红色点} \\ -1 & \text{蓝色点} \end{cases}$$

如果我们的超平面方程能够完全正确地对上图的样本点进行分类，就会满足下面的方程：

$$\begin{cases} \boldsymbol{\omega}^T x_i + \gamma > 0 & y_i = 1 \\ \boldsymbol{\omega}^T x_i + \gamma < 0 & y_i = -1 \end{cases}$$

如果我们要求再高一点，假设决策面正好处于间隔区域的中轴线上，并且相应的支持向量对应的样本点到决策面的距离为d，那么公式进一步写成：

$$\begin{cases} \dfrac{\boldsymbol{\omega}^T x_i + \gamma}{||\boldsymbol{\omega}||} \geq \mathrm{d} & \forall y_i = 1 \\ \dfrac{\boldsymbol{\omega}^T x_i + \gamma}{||\boldsymbol{\omega}||} \leq -\mathrm{d} & \forall y_i = -1 \end{cases}$$

上述公式的解释就是，对于所有分类标签为1和-1样本点，它们到直线的距离都大于等于d(支持向量上的样本点到超平面的距离)。公式两边都除以d，

$$\begin{cases} \boldsymbol{\omega_d}^T x_i + \gamma_d \geq 1 & \forall y_i = 1 \\ \boldsymbol{\omega_d}^T x_i + \gamma_d \leq -1 & \forall y_i = -1 \end{cases}$$

其中，

$$\boldsymbol{\omega}_d = \frac{\boldsymbol{\omega}}{||\boldsymbol{\omega}||d}, \quad \gamma_d = \frac{\gamma}{||\boldsymbol{\omega}||d}$$

因为||w||和d都是标量。所以上述公式的两个矢量，依然描述一条直线的法向量和截距。

$$\boldsymbol{\omega}_d^T \boldsymbol{x} + \gamma_d = 0$$
$$\boldsymbol{\omega}^T \boldsymbol{x} + \gamma = 0$$

上述两个公式，都是描述一条直线，数学模型代表的意义是一样的。现在，让我们对wd和γd重新起个名字，就叫它们w和γ。因此，我们就可以说：'间隔的两类样本点，我们一定可以找到一些超平面，使其对于所有的样本点均满足下面的条件："

$$\begin{cases} \boldsymbol{\omega}^T x_i + \gamma \geq 1 & \forall y_i = 1 \\ \boldsymbol{\omega}^T x_i + \gamma \leq -1 & \forall y_i = -1 \end{cases}$$

上述方程即给出了SVM最优化问题的约束条件。这时候，可能有人会问了，为什么标记为1和-1呢？因为这样标记方便我们将上述方程变成如下形式

$$y_i\left(\boldsymbol{\omega}^T x_i + \gamma\right) \geq 1 \quad \forall x_i$$

正是因为标签为1和-1，才方便我们将约束条件变成一个约束方程，从而方便我们的计算。

（4）线性SVM优化问题基本描述

现在整合一下思路，我们已经得到我们的目标函数：

$$d = \frac{\left|\boldsymbol{\omega}^T \boldsymbol{x} + \gamma\right|}{||\boldsymbol{\omega}||}$$

我们的优化目标是是d最大化。我们已经说过，我们是用支持向量上的样本点求解d的最大化的问题的。那么支持向量上的样本点有什么特点呢？

$$\left|\boldsymbol{\omega}^T x_i + \gamma\right| = 1 \quad \forall \text{支持向量上的样本点} x_i$$

你赞同这个观点吗？所有支持向量上的样本点，都满足如上公式。如果不赞同，请重看"分类间隔"方程推导过程。

现在我们就可以将我们的目标函数进一步化简：

$$\mathrm{d} = \frac{1}{||\boldsymbol{\omega}||}$$

因为，我们只关心支持向量上的点。随后我们求解d的最大化问题变成了||w||的最小化问题。进而||w||的最小化问题等效于

$$\min \frac{1}{2}||\omega||^2$$

为什么要做这样的等效呢？这是为了在进行最优化的过程中对目标函数求导时比较方便，但这绝对不影响最优化问题最后的求解。我们将最终的目标函
放在一起进行描述：

$$\min \frac{1}{2}||\omega||^2$$
$$\text{s.t.} \quad y_i\left(\boldsymbol{\omega}^T x_i + b\right) \geq 1, i = 1,2,\ldots,n$$

这里n是样本点的总个数，缩写s.t.表示"Subject to"，是"服从某某条件"的意思。上述公式描述的是一个典型的不等式约束条件下的二次型函数优化问
支持向量机的基本数学模型。

（5）求解准备

我们已经得到支持向量机的基本数学模型，接下来的问题就是如何根据数学模型，求得我们想要的最优解。在学习求解方法之前，我们得知道一点，
的求解方法有一个前提，就是我们的目标函数必须是凸函数。理解凸函数，我们还要先明确另一个概念，凸集。在凸几何中，凸集(convex set)是在)凸组
空间的子集。看一幅图可能更容易理解：

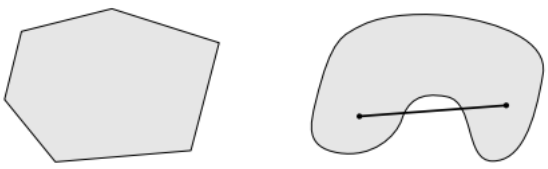


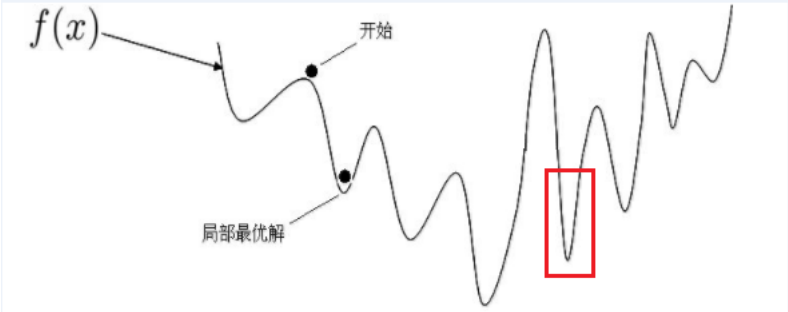
Figure 1: Examples of a convex set (a) and a non-convex set (b).

左右量图都是一个集合。**如果集合中任意2个元素连线上的点也在集合中，那么这个集合就是凸集。**显然，上图中的左图是一个凸集，上图中的右图是

凸函数的定义也是如此，其几何意义表示为函数任意两点连线上的值大于对应自变量处的函数值。若这里凸集C即某个区间L，那么，设函数f为定义在
数，若对L上的任意两点x1，x2和任意的实数λ，λ属于(0,1)，总有：

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

则函数f称为L上的凸函数，当且仅当其上镜图（在函数图像上方的点集）为一个凸集。再看一幅图，也许更容易理解：



像上图这样的函数，它整体就是一个非凸函数，我们无法获得全局最优解的，只能获得局部最优解。比如红框内的部分，如果单独拿出来，它就是一个凸函数。

我们的目标函数：

$$\min \frac{1}{2}||\omega||^2$$

很显然，它是一个凸函数。所以，可以使用我接下来讲述的方法求取最优解。

通常我们需要求解的最优化问题有如下几类：

- 无约束优化问题，可以写为：

$$min\ f\ (x)$$

- 有等式约束的优化问题，可以写为：

$$min\ f\ (x)$$
$$s.t.\ h_{i(x)} = 0, \quad i = 1,2,\dots,n$$

- 有不等式约束的优化问题，可以写为：

$$min\ f\ (x)$$
$$s.t.\ g_{i(x)} \leq 0, \quad i = 1,2,\dots,n$$
$$h_{j(x)} = 0, \quad j = 1,2,\dots,m$$

对于第(a)类的优化问题，尝试使用的方法就是费马大定理(Fermat)，即使用求取函数f(x)的导数，然后令其为零，可以求得候选最优值，再在这些候选值中验证；如果是凸函数，可以保证是最优解。这也就是我们高中经常使用的求函数的极值的方法。

对于第(b)类的优化问题，常常使用的方法就是拉格朗日乘子法（Lagrange Multiplier），即把等式约束h_i(x)用一个系数与f(x)写为一个式子，称为拉格朗日函数，而系数称为拉格朗日乘子。通过拉格朗日函数对各个变量求导，令其为零，可以求得候选值集合，然后验证求得最优值。

对于第(c)类的优化问题，常常使用的方法就是KKT条件。同样地，我们把所有的等式、不等式约束与f(x)写为一个式子，也叫拉格朗日函数，系数也称为拉格朗日乘子，通过一些条件，可以求出最优值的**必要条件**，这个条件称为KKT条件。

必要条件和充要条件如果不理解，可以看下面这句话：

- A的**必要条件**就是A可以推出的**结论**
- A的**充分条件**就是可以推出A的**前提**

了解到这些，现在让我们再看一下我们的最优化问题：

$$\min \frac{1}{2}||\omega||^2$$
$$s.t.\ \ y_i(\boldsymbol{\omega}^T x_i + b) \geq 1, i = 1,2,\dots,n$$

现在，我们的这个对优化问题属于哪一类？很显然，它属于第(c)类问题。在学习求解最优化问题之前，我们还要学习两个东西：拉格朗日函数和KKT条

（6）拉格朗日函数

首先，我们先要从宏观的视野上了解一下**拉格朗日对偶问题出现的原因和背景。**

我们知道我们要求解的是最小化问题，所以一个直观的想法是如果我能够构造一个函数，使得该函数在可行解区域内与原目标函数完全一致，而在可行解区域外的数值非常大，甚至是无穷大，那么这个**没有约束条件的新目标函数的优化问题**就与原来**有约束条件的原始目标函数的优化问题**是等价的问题。这就是使用拉格朗日方法所想达到的，它将约束条件放到目标函数中，**从而将有约束优化问题转换为无约束优化问题。**

随后，人们又发现，使用拉格朗日获得的函数，使用求导的方法求解依然困难。进而，需要对问题再进行一次转换，即使用一个数学技巧：**拉格朗日对**

所以，显而易见的是，我们在拉格朗日优化我们的问题这个道路上，**需要进行下面二个步骤：**

- 将有约束的原始目标函数转换为无约束的新构造的拉格朗日目标函数
- 使用拉格朗日对偶性，将不易求解的优化问题转化为易求解的优化

下面，进行第一步：**将有约束的原始目标函数转换为无约束的新构造的拉格朗日目标函数**

公式变形如下：

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) - 1\right)$$

其中αi是拉格朗日乘子，αi大于等于0，是我们构造新目标函数时引入的系数变量(我们自己设置)。现在我们令：

$$\theta(w) = \max_{\alpha_i \geq 0} \mathcal{L}(w, b, \alpha)$$

当样本点不满足约束条件时，即**在可行解区域外：**

$$y_i(\boldsymbol{\omega}^T x_i + b) < 1$$

此时，我们将αi设置为正无穷，此时θ(w)显然也是正无穷。

当样本点满足约束条件时，即**在可行解区域内：**

$$y_i(\boldsymbol{\omega}^T x_i + b) \geq 1$$

此时，显然θ(w)为原目标函数本身。我们将上述两种情况结合一下，就得到了新的目标函数：

$$\theta(\boldsymbol{\omega}) = \begin{cases} \frac{1}{2}||\boldsymbol{\omega}||^2 & x \in \text{可区域} \\ +\infty & x \in \text{非可行区域} \end{cases}$$

此时，再看我们的初衷，就是为了建立一个在可行解区域内与原目标函数相同，在可行解区域外函数值趋近于无穷大的新函数，现在我们做到了。

现在，我们的问题变成了求新目标函数的最小值，即：

$$\min_{w,b} \theta(w) = \min_{w,b} \max_{\alpha_i \geq 0} \mathcal{L}(w, b, \alpha) = p^*$$

这里用p*表示这个问题的最优值，且和最初的问题是等价的。

接下来，我们进行第二步：**将不易求解的优化问题转化为易求解的优化**

我们看一下我们的新目标函数，先求最大值，再求最小值。这样的话，我们首先就要面对带有需要求解的参数w和b的方程，而αi又是不等式约束，这样的问题不好做。所以，我们需要使用拉格朗日函数对偶性，将最小和最大的位置交换一下，这样就变成了：

$$\max_{\alpha_i \geq 0} \min_{w,b} \mathcal{L}(w, b, \alpha) = d^*$$

交换以后的新问题是原始问题的对偶问题，这个新问题的最优值用d*来表示。而且d*<=p*。我们关心的是d=p的时候，这才是我们要的解。需要什么样的条件才能得到d=p呢？

- 首先必须满足这个优化问题是凸优化问题。
- 其次，需要满足KKT条件。

凸优化问题的定义是：**求取最小值的目标函数为凸函数的一类优化问题。**目标函数是凸函数我们已经知道，这个优化问题又是求最小值。所以我们的是凸优化问题。

接下里，就是探讨是否满足KKT条件了。

（7）KKT条件

我们已经使用拉格朗日函数对我们的目标函数进行了处理，生成了一个新的目标函数。通过一些条件，可以求出最优值的必要条件，这个条件就是接下条件。一个最优化模型能够表示成下列标准形式：

$$\begin{aligned} &\min f(\boldsymbol{x}) \\ &s.t. \quad h_j(\boldsymbol{x}) = 0, j = 1, 2, \cdots, p \\ &\qquad g_k(\boldsymbol{x}) \leq 0, k = 1, 2, \cdots, q \\ &\qquad \boldsymbol{x} \in X \subset \mathbb{R}^n \end{aligned}$$

KKT条件的全称是Karush-Kuhn-Tucker条件，KKT条件是说最优值条件必须满足以下条件：

- 条件一：经过拉格朗日函数处理之后的新目标函数L(w,b,α)对x求导为零：
- 条件二：h(x) = 0；
- 条件三：α*g(x) = 0；

对于我们的优化问题：

$$\begin{aligned} &\min \frac{1}{2}||\omega||^2 \\ &\text{s.t.} \quad y_i\left(\boldsymbol{\omega}^T x_i + b\right) \geq 1, i = 1,2, \ldots, n \end{aligned}$$

显然，条件二已经满足了。另外两个条件为啥也满足呢？

这里原谅我省略一系列证明步骤，感兴趣的可以移步这里：点我查看

这里已经给出了很好的解释。现在，凸优化问题和KKT都满足了，问题转换成了对偶问题。而求解这个对偶学习问题，可以分为三个步骤：首先要让L和b最小化，然后求对α的极大，最后利用SMO算法求解对偶问题中的拉格朗日乘子。现在，我们继续推导。

（8）对偶问题求解

第一步：

根据上述推导已知：

$$\max_{\alpha_i \geq 0} \min_{w,b} \mathcal{L}(w, b, \alpha) = d^*$$

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i + b) - 1\right)$$

首先固定α，要让L(w,b,α)关于w和b最小化，我们分别对w和b偏导数，令其等于0，即：

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

将上述结果带回L(w,b,α)得到：

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i + b\right) - 1\right]$$

从上面的最后一个式子，我们可以看出，此时的L(w,b,α)函数只含有一个变量，即αi。

第二步：

现在内侧的最小值求解完成，我们求解外侧的最大值，从上面的式子得到

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$$

$$s.t.\ \alpha_i \geq 0, i = 1, 2, \cdots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

现在我们的优化问题变成了如上的形式。对于这个问题，我们有更高效的优化算法，即序列最小优化（SMO）算法。我们通过这个优化算法能得到α们就可以求解出w和b，进而求得我们最初的目的：找到超平面，即"决策平面"。

总结一句话：我们为啥使出吃奶的劲儿进行推导？因为我们要将最初的原始问题，转换到可以使用SMO算法求解的问题，这是一种最流行的求解方法求解方法？因为它牛逼啊！

对于上述问题，我们就可以使用SMO算法进行求解了，但是，SMO算法又是什么呢？

2、SMO算法

现在，我们已经得到了可以用SMO算法求解的目标函数，但是对于怎么编程实现SMO算法还是感觉无从下手。那么现在就聊聊如何使用SMO算法进

（1）Platt的SMO算法

1996年，John Platt发布了一个称为SMO的强大算法，用于训练SVM。SM表示序列最小化(Sequential Minimal Optimizaion)。Platt的SMO算法是分解为多个小优化问题来求解的。这些小优化问题往往很容易求解，并且对它们进行顺序求解的结果与将它们作为整体来求解的结果完全一致的。在结果时，SMO算法的求解时间短很多。

SMO算法的目标是求出一系列alpha和b，一旦求出了这些alpha，就很容易计算出权重向量w并得到分隔超平面。

SMO算法的工作原理是：每次循环中选择两个alpha进行优化处理。一旦找到了一对合适的alpha，那么就增大其中一个同时减小另一个。这里所谓的两个alpha必须符合以下两个条件，条件之一就是两个alpha必须要在间隔边界之外，而且第二个条件则是这两个alpha还没有进行过区间化处理或者不在边

（2）SMO算法的解法

先来定义特征到结果的输出函数为：

$$u = \boldsymbol{\omega}^T \boldsymbol{x} + b$$

接着，我们回忆一下原始优化问题，如下：

$$\min \frac{1}{2}||\omega||^2$$
$$\text{s.t.} \quad y_i\left(\boldsymbol{\omega}^T x_i + b\right) \geq 1, i = 1,2,\ldots,n$$

求导得：

$$\boldsymbol{\omega} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i}$$

将上述公式带入输出函数中：

$$u = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i}^T \boldsymbol{x} + b$$

与此同时，拉格朗日对偶后得到最终的目标化函数：

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$$
$$s.t.\ \alpha_i \geq 0, i = 1,2,\cdots,n$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

将目标函数变形，在前面增加一个符号，将最大值问题转换成最小值问题：

$$\min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i\, \alpha_j y_i y_j \boldsymbol{x_i}^T \boldsymbol{x_j} - \sum_{i=1}^{n} \alpha_i$$
$$s.t. \quad \alpha_i \geq 0, \quad i = 1,2,\cdots,n$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

实际上，对于上述目标函数，是存在一个假设的，即数据100%线性可分。但是，目前为止，我们知道几乎所有数据都不那么"干净"。这时我们就可以的**松弛变量**(slack variable)，来允许有些数据点可以处于超平面的错误的一侧。这样我们的优化目标就能保持仍然不变，但是此时我们的约束条件有所改变

$$s.t. \quad C \geq \alpha_i \geq 0, \quad i = 1,2,\cdots,n$$
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

根据KKT条件可以得出其中αi取值的意义为：

$$\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1$$

$$0 < \alpha_i < C \Leftrightarrow y_i u_i = 1$$

$$\alpha_i = C \Leftrightarrow y_i u_i \leq 1$$

- 对于第1种情况，表明αi是正常分类，在边界内部；
- 对于第2种情况，表明αi是支持向量，在边界上；
- 对于第3种情况，表明αi是在两条边界之间。

而最优解需要满足KKT条件，即上述3个条件都得满足，以下几种情况出现将会不满足：

$$y_i u_i \leq 1 \quad \alpha_i < C$$

$$y_i u_i \geq 1 \quad \alpha_i > 0$$

$$y_i u_i = 1 \quad \alpha_i = 0 \text{ 或者 } \alpha_i = C$$

也就是说，如果存在不能满足KKT条件的αi，那么需要更新这些αi，这是第一个约束条件。此外，更新的同时还要受到第二个约束条件的限制，即：

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

因为这个条件，我们同时更新两个α值，因为只有成对更新，才能保证更新之后的值仍然满足和为0的约束，假设我们选择的两个乘子为α1和α2：

$$\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta$$

其中， ksi为常数。因为两个因子不好同时求解，所以可以先求第二个乘子α2的解（α2 new），得到α2的解（α2 new）之后，再用α2的解（α2 new）解（α1 new ）。为了求解α2 new ，得先确定α2 new的取值范围。假设它的上下边界分别为H和L，那么有：

$$L \leq \alpha_2^{new} \leq H$$

接下来，综合下面两个条件：

$$C \geq \alpha_i \geq 0, \quad i = 1,2,\cdots,n$$

$$\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta$$

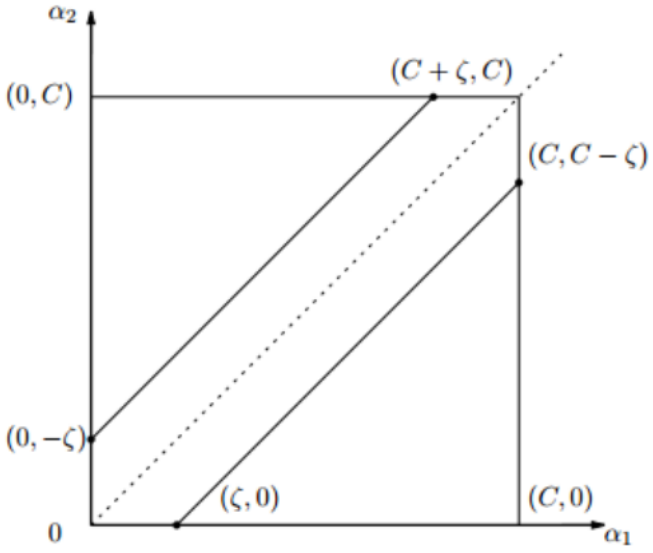
当y1不等于y2时，即一个为正1，一个为负1的时候，可以得到：

$$\alpha_1^{old} - \alpha_2^{old} = \zeta$$

所以有：

$$L = \max(0, -\zeta), \quad H = \min(C, C - \zeta)$$

此时，取值范围如下图所示：



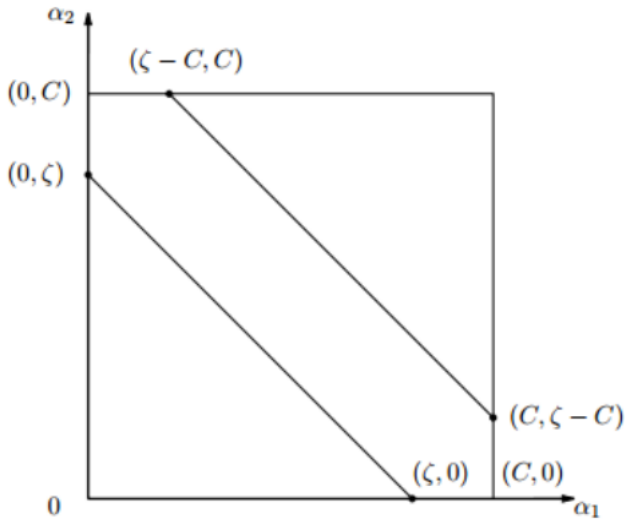
当y1等于y2时，即两个都为正1或者都为负1，可以得到：

$$\alpha_1^{old} + \alpha_2^{old} = \zeta$$

所以有：

$$L = \max(0, \zeta - C), \quad H = \min(C, \zeta)$$

此时，取值范围如下图所示：



如此，根据y1和y2异号或同号，可以得出α2 new的上下界分别为：

$$\begin{cases} L = \max(0, \alpha_2^{old} - \alpha_1^{old}), H = \min(C, C + \alpha_2^{old} - \alpha_1^{old}) & \text{if } y_1 \neq y_2 \\ L = \max(0, \alpha_2^{old} + \alpha_1^{old} - C), H = \min(C, \alpha_2^{old} + \alpha_1^{old}) & \text{if } y_1 = y_2 \end{cases}$$

这个界限就是编程的时候需要用到的。已经确定了边界，接下来，就是推导迭代式，用于更新 α值。

我们已经知道，更新α的边界，接下来就是讨论如何更新α值。我们依然假设选择的两个乘子为α1和α2。固定这两个乘子，进行推导。于是目标函数变

$$
\begin{aligned}
W(a_2) &= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \boldsymbol{x_i}^T \boldsymbol{x_j} \alpha_i \alpha_j \\
&= \alpha_1 + \alpha_2 + \sum_{i=3}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{2} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} + \sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j}\right) \\
&= \alpha_1 + \alpha_2 + \sum_{i=3}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{2}\left(\sum_{j=1}^{2} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} + \sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j}\right) \\
&\quad - \frac{1}{2}\sum_{i=3}^{n}\left(\sum_{j=1}^{2} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} + \sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j}\right) \\
&= \alpha_1 + \alpha_2 + \sum_{i=3}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{2}\sum_{j=1}^{2} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} - \sum_{i=1}^{2}\sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} \\
&\quad - \frac{1}{2}\sum_{i=3}^{n}\sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j} \\
&= \alpha_1 + \alpha_2 - \frac{1}{2}\alpha_1{}^2 \boldsymbol{x_1}^T \boldsymbol{x_1} - \frac{1}{2}\alpha_2{}^2 \boldsymbol{x_2}^T \boldsymbol{x_2} - y_1 y_2 \alpha_1 \alpha_2 \boldsymbol{x_1}^T \boldsymbol{x_2} - y_1 \alpha_1 \sum_{j=3}^{n} \alpha_j y_j \boldsymbol{x_1}^T \boldsymbol{x_j} \\
&\quad - y_2 \alpha_2 \sum_{j=3}^{n} \alpha_j y_j \boldsymbol{x_2}^T \boldsymbol{x_j} + \sum_{i=3}^{n}\alpha_i - \frac{1}{2}\sum_{i=3}^{n}\sum_{j=3}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x_i}^T \boldsymbol{x_j}
\end{aligned}
$$

为了描述方便，我们定义如下符号：

$$
\begin{aligned}
f(\boldsymbol{x_i}) &= \sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x_i}^T \boldsymbol{x_j} + b \\
v_i &= \sum_{j=3}^{n} \alpha_j y_j \boldsymbol{x_i}^T \boldsymbol{x_j} = f(\boldsymbol{x_i}) - \sum_{j=1}^{2} \alpha_j y_j \boldsymbol{x_i}^T \boldsymbol{x_j} - b
\end{aligned}
$$

最终目标函数变为：

$$
\begin{aligned}
W(a_2) = \alpha_1 + \alpha_2 - \frac{1}{2}\alpha_1{}^2 \boldsymbol{x_1}^T \boldsymbol{x_1} - \frac{1}{2}\alpha_2{}^2 \boldsymbol{x_2}^T \boldsymbol{x_2} - y_1 y_2 \alpha_1 \alpha_2 \boldsymbol{x_1}^T \boldsymbol{x_2} - y_1 \alpha_1 v_1 - y_2 \alpha_2 v_2 \\
+ constant
\end{aligned}
$$

我们不关心constant的部分，因为对于α1和α2来说，它们都是常数项，在求导的时候，直接变为0。对于这个目标函数，如果对其求导，还有个未知推导出α1和α2的关系，然后用α2代替α1，这样目标函数就剩一个未知数了，我们就可以求导了，推导出迭代公式。所以现在继续推导α1和α2的关系。注条件：

$$
\sum_{i=1}^{n} \alpha_i y_i = 0
$$

我们在求α1和α2的时候，可以将α3,α4,...,αn和y3,y4,...,yn看作常数项。因此有：

$$
\alpha_1 y_1 + \alpha_2 y_2 = B
$$

我们不必关心常数B的大小，现在将上述等式两边同时乘以y1，得到(y1y1=1)：

$$
\alpha_1 = \gamma - s\alpha_2
$$

其中γ为常数By1，我们不关心这个值，s=y1y2。接下来，我们将得到的α1带入W(α2)公式得：

$$
\begin{aligned}
W(a_2) = \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}(\gamma - s\alpha_2)^2 \boldsymbol{x_1}^T \boldsymbol{x_1} - \frac{1}{2}\alpha_2{}^2 \boldsymbol{x_2}^T \boldsymbol{x_2} - s(\gamma - s\alpha_2)\alpha_2 \boldsymbol{x_1}^T \boldsymbol{x_2} \\
- y_1(\gamma - s\alpha_2)v_1 - y_2 \alpha_2 v_2 + constant
\end{aligned}
$$

这样目标函数中就只剩下α2了，我们对其求偏导（注意：s=y1y2，所以s的平方为1，y1的平方和y2的平方均为1）：

$$
\begin{aligned}
\frac{\partial W(a_2)}{\partial a_2} &= -\mathrm{s} + 1 + \gamma s \boldsymbol{x_1}^T \boldsymbol{x_1} - \alpha_2 \boldsymbol{x_1}^T \boldsymbol{x_1} - \alpha_2 \boldsymbol{x_2}^T \boldsymbol{x_2} \\
&\quad - \gamma s \boldsymbol{x_1}^T \boldsymbol{x_2} + 2\alpha_2 \boldsymbol{x_1}^T \boldsymbol{x_2} + y_2 v_1 - y_2 v_2 = 0
\end{aligned}
$$

继续化简，将s=y1y2带入方程。

$$
\alpha_2^{new} = \frac{y_2\left(y_2 - y_1 + y_1 \gamma(\boldsymbol{x_1}^T \boldsymbol{x_1} - \boldsymbol{x_1}^T \boldsymbol{x_2}) + v_1 - v_2\right)}{\boldsymbol{x_1}^T \boldsymbol{x_1} + \boldsymbol{x_2}^T \boldsymbol{x_2} - 2\boldsymbol{x_1}^T \boldsymbol{x_2}}
$$

我们令：

$$E_i = f(x_i) - y_i$$
$$\eta = \boldsymbol{x_1}^T\boldsymbol{x_1} + \boldsymbol{x_2}^T\boldsymbol{x_2} - 2\boldsymbol{x_1}^T\boldsymbol{x_2}$$

Ei为误差项，η为学习速率。

再根据我们已知的公式：

$$\gamma = {\alpha_1}^{\text{old}} + s{\alpha_2}^{old}$$
$$v_j = \sum_{i=3}^{n} \alpha_i y_i \boldsymbol{x_j}^T\boldsymbol{x_i} = f(\boldsymbol{x_j}) - \sum_{i=1}^{2} \alpha_i y_i \boldsymbol{x_j}^T\boldsymbol{x_i} - b$$

将α2 new继续化简得：

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

这样，我们就得到了最终需要的迭代公式。这个是没有经过剪辑是的解，需要考虑约束：

$$0 < \alpha_i < C$$

根据之前推导的α取值范围，我们得到最终的解析解为：

$$\alpha_2^{new,clipped} = \begin{cases} H & \alpha_2^{new} > H \\ \alpha_2^{new} & L \le \alpha_2^{new} \le H \\ L & \alpha_2^{new} < L \end{cases}$$

又因为：

$$\alpha_1^{old} = \gamma - s\alpha_2^{old}$$
$$\alpha_1^{new} = \gamma - s\alpha_2^{new,clipped}$$

消去γ得：

$$\alpha_1^{new} = \alpha_1^{old} + y_1y_2(\alpha_2^{old} - \alpha_2^{new,clipped})$$

这样，我们就知道了怎样计算α1和α2了，也就是如何对选择的α进行更新。

当我们更新了α1和α2之后，需要重新计算阈值b，因为b关系到了我们f(x)的计算，也就关系到了误差Ei的计算。

我们要根据α的取值范围，去更正b的值，使间隔最大化。当α1 new在0和C之间的时候，根据KKT条件可知，这个点是支持向量上的点。因此，满足

$$y_1(\boldsymbol{\omega}^T\boldsymbol{x_1} + b) = 1$$

公式两边同时乘以y1得(y1y1=1)：

$$\sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x_i}\,\boldsymbol{x_1} + b = y_1$$

因为我们是根据α1和α2的值去更新b，所以单独提出i=1和i=2的时候，整理可得：

$$b_1^{new} = y_1 - \sum_{i=3}^{n} \alpha_i y_i \boldsymbol{x_i}^T\boldsymbol{x_1} - {\alpha_1}^{new} y_1 \boldsymbol{x_1}^T\boldsymbol{x_1} - {\alpha_2}^{new} y_2 \boldsymbol{x_2}^T\boldsymbol{x_1}$$

其中前两项为：

$$y_1 - \sum_{i=3}^{n} \alpha_i y_i \boldsymbol{x_i}^T\boldsymbol{x_1} = -E_1 + {\alpha_1}^{old} y_1 \boldsymbol{x_1}^T\boldsymbol{x_1} + {\alpha_2}^{old} y_2 \boldsymbol{x_2}^T\boldsymbol{x_1} + b^{old}$$

将上述两个公式，整理得：

$$b_1^{new} = b^{old} - E_1 - y_1({\alpha_1}^{new} - {\alpha_1}^{old})\boldsymbol{x_1}^T\boldsymbol{x_1} - y_2({\alpha_2}^{new} - {\alpha_2}^{old})\boldsymbol{x_2}^T\boldsymbol{x_1}$$

同理可得b2 new为：

$$b_2^{new} = b^{old} - E_2 - y_1(\alpha_1{}^{new} - \alpha_1{}^{old})\boldsymbol{x_1}^T\boldsymbol{x_2} - y_2(\alpha_2{}^{new} - \alpha_2{}^{old})\boldsymbol{x_2}^T\boldsymbol{x_2}$$

当b1和b2都有效的时候，它们是相等的，即：

$$b^{new} = b_1^{new} = b_2^{new}$$

当两个乘子都在边界上，则b阈值和KKT条件一致。当不满足的时候，SMO算法选择他们的中点作为新的阈值：

$$b = \begin{cases} b_1, & 0 < \alpha_1^{new} < C \\ b_2 & 0 < \alpha_2^{new} < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases}$$

最后，更新所有的α和b，这样模型就出来了，从而即可求出我们的分类函数。

现在，让我们梳理下SMO算法的步骤：

- 步骤1：计算误差：

$$E_i = f(\boldsymbol{x_i}) - \mathrm{y}_i = \sum_{j=1}^{n} \alpha_j y_j \boldsymbol{x_i}^T \boldsymbol{x_j} + b - \mathrm{y}_i$$

- 步骤2：计算上下界L和H：

$$\begin{cases} L = \max(0, \alpha_j{}^{old} - \alpha_i{}^{old}), H = \min(C, C + \alpha_j{}^{old} - \alpha_i{}^{old}) & if\ \mathrm{y}_i \neq \mathrm{y}_j \\ L = \max(0, \alpha_j{}^{old} + \alpha_i{}^{old} - C), H = \min(C, \alpha_j{}^{old} + \alpha_i{}^{old}) & if\ \mathrm{y}_i \neq \mathrm{y}_j \end{cases}$$

- 步骤3：计算η：

$$\eta = \boldsymbol{x_i}^T\boldsymbol{x_i} + \boldsymbol{x_j}^T\boldsymbol{x_j} - 2\boldsymbol{x_i}^T\boldsymbol{x_j}$$

- 步骤4：更新αj：

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

- 步骤5：根据取值范围修剪αj：

$$\alpha^{new,clipped} = \begin{cases} H & if\ \alpha_2^{new} \geq H \\ \alpha_2^{new} & if\ L \leq \alpha_2^{new} \leq H \\ L & if\ \alpha_2^{new} \leq L \end{cases}$$

- 步骤6：更新αi：

$$\alpha_i^{new} = \alpha_i^{old} + y_i y_j (\alpha_j^{old} - \alpha_j^{new,clipped})$$

- 步骤7：更新b1和b2：

$$b_1^{new} = b^{old} - E_i - y_i(\alpha_i{}^{new} - \alpha_i{}^{old})\boldsymbol{x_i}^T\boldsymbol{x_i} - y_j(\alpha_j{}^{new} - \alpha_j{}^{old})\boldsymbol{x_j}^T\boldsymbol{x_i}$$

$$b_2^{new} = b^{old} - E_j - y_i(\alpha_i{}^{new} - \alpha_i{}^{old})\boldsymbol{x_i}^T\boldsymbol{x_j} - y_j(\alpha_j{}^{new} - \alpha_j{}^{old})\boldsymbol{x_j}^T\boldsymbol{x_j}$$

- 步骤8：根据b1和b2更新b：

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

四、编程求解线性SVM

已经梳理完了SMO算法实现步骤，接下来按照这个思路编写代码，进行实战练习。

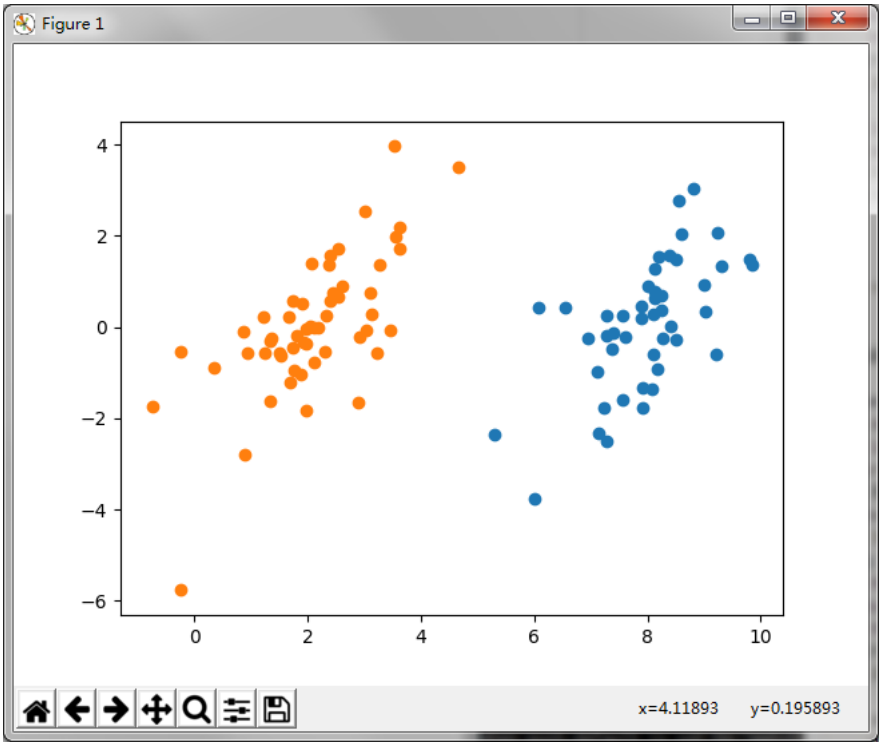
1、可视化数据集

我们先使用简单的数据集进行测试，数据集下载地址：数据集下载

编写程序可视化数据集，看下它是长什么样的：

```
# -*- coding:UTF-8 -*-
import matplotlib.pyplot as plt
import numpy as np

"""
函数说明:读取数据

Parameters:
    fileName - 文件名
Returns:
    dataMat - 数据矩阵
    labelMat - 数据标签
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-21
"""
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():                                    #逐行读取，滤除空格等
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])      #添加数据
        labelMat.append(float(lineArr[2]))                          #添加标签
    return dataMat,labelMat

"""
函数说明:数据可视化

Parameters:
    dataMat - 数据矩阵
    labelMat - 数据标签
Returns:
    无
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-21
"""
def showDataSet(dataMat, labelMat):
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1])   #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1]) #负样本散点图
    plt.show()

if __name__ == '__main__':
    dataMat, labelMat = loadDataSet('testSet.txt')
    showDataSet(dataMat, labelMat)
```

运行程序，查看结果：



这就是我们使用的二维数据集，显然线性可分。现在我们使用简化版的SMO算法进行求解。

2、简化版SMO算法

按照上述已经推导的步骤编写代码：

```
# -*- coding:UTF-8 -*-
from time import sleep
import matplotlib.pyplot as plt
import numpy as np
import random
import types

"""
函数说明:读取数据

Parameters:
    fileName - 文件名
Returns:
    dataMat - 数据矩阵
    labelMat - 数据标签
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-21
"""
def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():                                    #逐行读取，滤除空格等
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])      #添加数据
        labelMat.append(float(lineArr[2]))                          #添加标签
    return dataMat,labelMat


"""
函数说明:随机选择alpha

Parameters:
    i - alpha
    m - alpha参数个数
Returns:
    j -
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-21
"""
def selectJrand(i, m):
    j = i                                 #选择一个不等于i的j
    while (j == i):
        j = int(random.uniform(0, m))
    return j

"""
```

```
函数说明:修剪alpha

Parameters:
    aj - alpha值
    H - alpha上限
    L - alpha下限
Returns:
    aj - alpah值
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-21
"""
def clipAlpha(aj,H,L):
    if aj > H:
        aj = H
    if L > aj:
        aj = L
    return aj

"""
函数说明:简化版SMO算法

Parameters:
    dataMatIn - 数据矩阵
    classLabels - 数据标签
    C - 松弛变量
    toler - 容错率
    maxIter - 最大迭代次数
Returns:
    无
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-23
"""
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    #转换为numpy的mat存储
    dataMatrix = np.mat(dataMatIn); labelMat = np.mat(classLabels).transpose()
    #初始化b参数，统计dataMatrix的维度
    b = 0; m,n = np.shape(dataMatrix)
    #初始化alpha参数，设为0
    alphas = np.mat(np.zeros((m,1)))
    #初始化迭代次数
    iter_num = 0
    #最多迭代matIter次
    while (iter_num < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            #步骤1：计算误差Ei
            fXi = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])
            #优化alpha，更设定一定的容错率。
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and (alphas[i] > 0)):
                #随机选择另一个与alpha_i成对优化的alpha_j
                j = selectJrand(i,m)
                #步骤1：计算误差Ej
                fXj = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                #保存更新前的aplpha值，使用深拷贝
                alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
                #步骤2：计算上下界L和H
                if (labelMat[i] != labelMat[j]):
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
                else:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                if L==H: print("L==H"); continue
                #步骤3：计算eta
                eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - dataMatrix[i,:]*dataMatrix[i,:].T - dataMatrix[j,:]*dataMatrix[j,:].T
                if eta >= 0: print("eta>=0"); continue
                #步骤4：更新alpha_j
                alphas[j] -= labelMat[j]*(Ei - Ej)/eta
                #步骤5：修剪alpha_j
                alphas[j] = clipAlpha(alphas[j],H,L)
                if (abs(alphas[j] - alphaJold) < 0.00001): print("alpha_j变化太小"); continue
                #步骤6：更新alpha_i
                alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])
                #步骤7：更新b_1和b_2
                b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T - labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[
                b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T - labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[
                #步骤8：根据b_1和b_2更新b
                if (0 < alphas[i]) and (C > alphas[i]): b = b1
                elif (0 < alphas[j]) and (C > alphas[j]): b = b2
                else: b = (b1 + b2)/2.0
                #统计优化次数
                alphaPairsChanged += 1
                #打印统计信息
                print("第%d次迭代 样本:%d, alpha优化次数:%d" % (iter_num,i,alphaPairsChanged))
        #更新迭代次数
        if (alphaPairsChanged == 0): iter_num += 1
        else: iter_num = 0
        print("迭代次数: %d" % iter_num)
    return b,alphas

```

```
"""
函数说明:分类结果可视化

Parameters:
    dataMat - 数据矩阵
    w - 直线法向量
    b - 直线解决
Returns:
    无
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-23
"""
def showClassifer(dataMat, w, b):
    #绘制样本点
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1], s=30, alpha=0.7)   #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1], s=30, alpha=0.7) #负样本散点图
    #绘制直线
    x1 = max(dataMat)[0]
    x2 = min(dataMat)[0]
    a1, a2 = w
    b = float(b)
    a1 = float(a1[0])
    a2 = float(a2[0])
    y1, y2 = (-b- a1*x1)/a2, (-b - a1*x2)/a2
    plt.plot([x1, x2], [y1, y2])
    #找出支持向量点
    for i, alpha in enumerate(alphas):
        if alpha > 0:
            x, y = dataMat[i]
            plt.scatter([x], [y], s=150, c='none', alpha=0.7, linewidth=1.5, edgecolor='red')
    plt.show()


"""
函数说明:计算w

Parameters:
    dataMat - 数据矩阵
    labelMat - 数据标签
    alphas - alphas值
Returns:
    无
Author:
    Jack Cui
Blog:
    http://blog.csdn.net/c406495762
Zhihu:
    https://www.zhihu.com/people/Jack--Cui/
Modify:
    2017-09-23
"""
def get_w(dataMat, labelMat, alphas):
    alphas, dataMat, labelMat = np.array(alphas), np.array(dataMat), np.array(labelMat)
    w = np.dot((np.tile(labelMat.reshape(1, -1).T, (1, 2)) * dataMat).T, alphas)
    return w.tolist()


if __name__ == '__main__':
    dataMat, labelMat = loadDataSet('testSet.txt')
    b,alphas = smoSimple(dataMat, labelMat, 0.6, 0.001, 40)
    w = get_w(dataMat, labelMat, alphas)
    showClassifer(dataMat, w, b)
```

程序运行结果：

其中，中间的蓝线为求出来的分类器，用红圈圈出的点为支持向量点。

五、总结

- 本文主要进行了线性SVM的推导，并通过编程实现一个简化版SMO算法；
- 本文的简化版SMO算法在选取α的时候，没有选择启发式的选择方法，并且两个乘子的计算没有进行优化，所以算法比较耗时，下一篇文章会讲解
  化方法；
- 本文讨论的是线性SVM，没有使用核函数，下一篇文章将会讲解如何应用核函数，将SVM应用于非线性数据集；
- 如有问题，请留言。如有错误，还望指正，谢谢！

PS： 如果觉得本篇本章对您有所帮助，欢迎关注、评论、赞！

参考资料：

- [1] 五岁小孩也能看懂的SVM：https://www.zhihu.com/question/21094489/answer/86273196
- [2] 五岁小孩也能看懂的SVM ：
  https://www.reddit.com/r/MachineLearning/comments/15zrpp/please_explain_support_vector_machines_svm_like_i/
- [3] pluskid大牛博客：http://blog.pluskid.org/?page_id=683
- [4] 陈东岳老师文章：https://zhuanlan.zhihu.com/p/24638007
- [5] 深入理解拉格朗日乘子法和KKT条件：http://blog.csdn.net/xianlingmao/article/details/7919597
- [6] 充分条件和必要条件：https://www.zhihu.com/question/30469121
- [7] 凸函数：https://zh.wikipedia.org/wiki/%E5%87%B8%E5%87%BD%E6%95%B0
- [8]《机器学习实战》第6章内容。
- [9] SVM之SMO算法：http://www.cnblogs.com/zangrunqiang/p/5515872.html

JackCui

关注人工智能及互联网的个人博客