# SVM原理介绍及其代码实现

# 一、原理介绍

## 1 简介

支持向量机基本上是最好的有监督学习算法了。最开始接触SVM是去年暑假的时候，老师要求交《统计学习理论》的报告，那时去网上下了一份入门教程，里面讲的很通俗，当时只是大致了解了一些相关概念。这次斯坦福提供的学习材料，让我重新学习了一些SVM知识。我看很多正统的讲法都是从VC 维理论和结构风险最小原理出发，然后引出SVM什么的，还有些资料上来就讲分类超平面什么的。这份材料从前几节讲的logistic回归出发，引出了SVM，既揭示了模型间的联系，也让人觉得过渡更自然。

## 2 重新审视logistic回归

Logistic回归目的是从特征学习出一个0/1分类模型，而这个模型是将特性的线性组合作为自变量，由于自变量的取值范围是负无穷到正无穷。因此，使用logistic函数（或称作sigmoid函数）将自变量映射到(0,1)上，映射后的值被认为是属于y=1的概率。

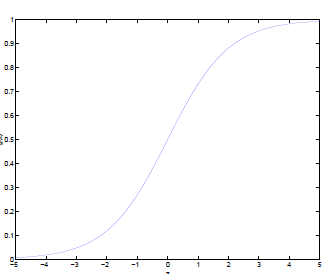
形式化表示就是

假设函数

[clip_image001](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131373510.png)

其中x是n维特征向量，函数g就是logistic函数。

[clip_image002](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113137969.png)

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131375604.png)

可以看到，将无穷映射到了(0,1)。

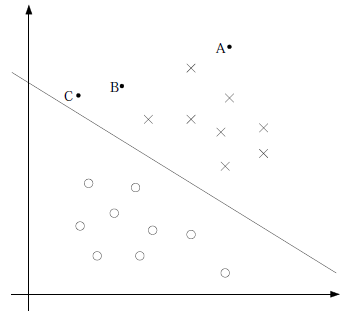
而假设函数就是特征属于y=1的概率。

[clip_image004](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131385015.png)

当我们要判别一个新来的特征属于哪个类时，只需求[clip_image006](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131389409.png)，若大于0.5就是y=1的类，反之属于y=0类。

再审视一下[clip_image006[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131392374.png)，发现[clip_image006[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131403704.png)只和[clip_image008](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131416669.png)有关，[clip_image008[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131416047.png)>0，那么[clip_image010](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113142997.png)，g(z)只不过是用来映射，真实的类别决定权还在[clip_image008[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131425948.png)。还有当[clip_image012](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113143898.png)时，[clip_image006[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131433340.png)=1，反之[clip_image006[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131442402.png)=0。如果我们只从[clip_image008[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131456240.png)出发，希望模型达到的目标无非就是让训练数据中y=1的特征[clip_image012[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131458715.png)，而是y=0的特征[clip_image014](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131455617.png)。Logistic回归就是要学习得到[clip_image016](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131464679.png)，使得正例的特征远大于0，负例的特征远小于0，强调在全部训练实例上达到这个目标。

图形化表示如下：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131479073.png)

中间那条线是[clip_image019](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131479879.png)，logistic回顾强调所有点尽可能地远离中间那条线。学习出的结果也就中间那条线。考虑上面3个点A、B和C。从图中我们可以确定A是×类别的，然而C我们是不太确定的，B还算能够确定。这样我们可以得出结论，我们更应该关心靠近中间分割线的点，让他们尽可能地远离中间线，而不是在所有点上达到最优。因为那样的话，要使得一部分点靠近中间线来换取另外一部分点更加远离中间线。我想这就是支持向量机的思路和logistic回归的不同点，一个考虑局部（不关心已经确定远离的点），一个考虑全局（已经远离的点可能通过调整中间线使其能够更加远离）。这是我的个人直观理解。

## 3 形式化表示

我们这次使用的结果标签是y=-1,y=1，替换在logistic回归中使用的y=0和y=1。同时将[clip_image016[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131482006.png)替换成w和b。以前的[clip_image021](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131483368.png)，其中认为[clip_image023](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131482256.png)。现在我们替换[clip_image025](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113149794.png)为b，后面替换[clip_image027](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131492157.png)为[clip_image029](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131503760.png)（即[clip_image031](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113150662.png)）。这样，我们让[clip_image033](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131511152.png)，进一步[clip_image035](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131513628.png)。也就是说除了y由y=0变为y=-1，只是标记不同外，与logistic回归的形式化表示没区别。再明确下假设函数

[clip_image037](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131518055.png)

上一节提到过我们只需考虑[clip_image008[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131526593.png)的正负问题，而不用关心g(z)，因此我们这里将g(z)做一个简化，将其简单映射到y=-1和y=1上。映射关系如下：

[clip_image039](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131529908.png)

## 4 函数间隔（functional margin）和几何间隔（geometric margin）

给定一个训练样本[clip_image041](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131525971.png)，x是特征，y是结果标签。i表示第i个样本。我们定义函数间隔如下：

[clip_image043](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131534858.png)

可想而知，当[clip_image045](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131539808.png)时，在我们的g(z)定义中，[clip_image047](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131545249.png)，[clip_image049](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113155200.png)的值实际上就是[clip_image051](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131552675.png)。反之亦然。为了使函数间隔最大（更大的信心确定该例是正例还是反例），当[clip_image045[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131559577.png)时，[clip_image053](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131564528.png)应该是个大正数，反之是个大负数。因此函数间隔代表了我们认为特征是正例还是反例的确信度。

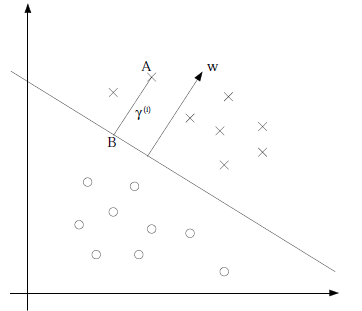
继续考虑w和b，如果同时加大w和b，比如在[clip_image055](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131568639.png)前面乘个系数比如2，那么所有点的函数间隔都会增大二倍，这个对求解问题来说不应该有影响，因为我们要求解的是[clip_image057](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131569478.png)，同时扩大w和b对结果是无影响的。这样，我们为了限制w和b，可能需要加入归一化条件，毕竟求解的目标是确定唯一一个w和b，而不是多组线性相关的向量。这个归一化一会再考虑。

刚刚我们定义的函数间隔是针对某一个样本的，现在我们定义全局样本上的函数间隔

[clip_image058](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131573905.png)

说白了就是在训练样本上分类正例和负例确信度最小那个函数间隔。

接下来定义几何间隔，先看图

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131571364.png)

假设我们有了B点所在的[clip_image057[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113158426.png)分割面。任何其他一点，比如A到该面的距离以[clip_image061](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131584820.png)表示，假设B就是A在分割面上的投影。我们知道向量BA的方向是[clip_image063](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131593359.png)（分割面的梯度），单位向量是[clip_image065](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131131593608.png)。A点是[clip_image041[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132006050.png)，所以B点是x=[clip_image067](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132002114.png)（利用初中的几何知识），带入[clip_image057[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132002637.png)得，

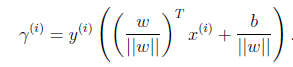
[clip_image069](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132011524.png)

进一步得到

[clip_image070](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132017588.png)

[clip_image061[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132018394.png)实际上就是点到平面距离。

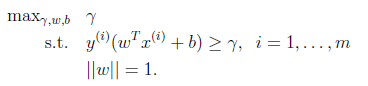
再换种更加优雅的写法：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113202836.png)

当[clip_image073](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132024914.png)时，不就是函数间隔吗？是的，前面提到的函数间隔归一化结果就是几何间隔。他们为什么会一样呢？因为函数间隔是我们定义的，在定义的时候就有几何间隔的色彩。同样，同时扩大w和b，w扩大几倍，[clip_image075](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132037880.png)就扩大几倍，结果无影响。同样定义全局的几何间隔[clip_image076](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132042274.png)

## 5 最优间隔分类器（optimal margin classifier）

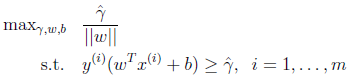
回想前面我们提到我们的目标是寻找一个超平面，使得离超平面比较近的点能有更大的间距。也就是我们不考虑所有的点都必须远离超平面，我们关心求得的超平面能够让所有点中离它最近的点具有最大间距。形象的说，我们将上面的图看作是一张纸，我们要找一条折线，按照这条折线折叠后，离折线最近的点的间距比其他折线都要大。形式化表示为：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132042241.png)

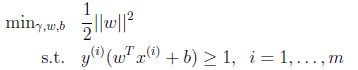
这里用[clip_image075[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132053047.png)=1规约w，使得[clip_image079](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132059634.png)是几何间隔。

到此，我们已经将模型定义出来了。如果求得了w和b，那么来一个特征x，我们就能够分类了，称为最优间隔分类器。接下的问题就是如何求解w和b的问题了。

由于[clip_image081](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132067059.png)不是凸函数，我们想先处理转化一下，考虑几何间隔和函数间隔的关系，[clip_image083](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/2011031311320658.png)，我们改写一下上面的式子：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132079992.png)

这时候其实我们求的最大值仍然是几何间隔，只不过此时的w不受[clip_image081[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113207482.png)的约束了。然而这个时候目标函数仍然不是凸函数，没法直接代入优化软件里计算。我们还要改写。前面说到同时扩大w和b对结果没有影响，但我们最后要求的仍然是w和b的确定值，不是他们的一组倍数值，因此，我们需要对[clip_image086](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132088257.png)做一些限制，以保证我们解是唯一的。这里为了简便我们取[clip_image088](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132099270.png)。这样的意义是将全局的函数间隔定义为1，也即是将离超平面最近的点的距离定义为[clip_image090](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132093108.png)。由于求[clip_image090[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313113210534.png)的最大值相当于求[clip_image092](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132107120.png)的最小值，因此改写后结果为：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131132112071.png)

这下好了，只有线性约束了，而且是个典型的二次规划问题（目标函数是自变量的二次函数）。代入优化软件可解。

到这里发现，这个讲义虽然没有像其他讲义一样先画好图，画好分类超平面，在图上标示出间隔那么直观，但每一步推导有理有据，依靠思路的流畅性来推导出目标函数和约束。

接下来介绍的是手工求解的方法了，一种更优的求解方法。

## 6 拉格朗日对偶（Lagrange duality）

先抛开上面的二次规划问题，先来看看存在等式约束的极值问题求法，比如下面的最优化问题：

clip_image001[9]

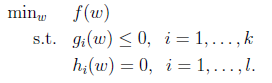
    目标函数是f(w)，下面是等式约束。通常解法是引入拉格朗日算子，这里使用[clip_image003[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234472404.png)来表示算子，得到拉格朗日公式为

clip_image004[6]

    L是等式约束的个数。

    然后分别对w和[clip_image003[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234494190.png)求偏导，使得偏导数等于0，然后解出w和[clip_image006[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234506043.png)。至于为什么引入拉格朗日算子可以求出极值，原因是f(w)的dw变化方向受其他不等式的约束，dw的变化方向与f(w)的梯度垂直时才能获得极值，而且在极值处，f(w)的梯度与其他等式梯度的线性组合平行，因此他们之间存在线性关系。（参考《最优化与KKT条件》）

然后我们探讨有不等式约束的极值问题求法，问题如下：



    我们定义一般化的拉格朗日公式

[clip_image008[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234515322.png)

    这里的[clip_image010[50]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234527731.png)和[clip_image012[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234533205.png)都是拉格朗日算子。如果按这个公式求解，会出现问题，因为我们求解的是最小值，而这里的[clip_image014[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234548679.png)已经不是0了，我们可以将[clip_image010[51]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234555548.png)调整成很大的正值，来使最后的函数结果是负无穷。因此我们需要排除这种情况，我们定义下面的函数：

clip_image015[6]

    这里的P代表primal。假设[clip_image017[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234562941.png)或者[clip_image019[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234562351.png)，那么我们总是可以调整[clip_image010[52]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234577269.png)和[clip_image012[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234574694.png)来使得[clip_image021[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234582120.png)有最大值为正无穷。而只有g和h满足约束时，[clip_image021[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234596755.png)为f(w)。这个函数的精妙之处在于[clip_image023[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131234598641.png)，而且求极大值。

    因此我们可以写作

clip_image024[6]

    这样我们原来要求的min f(w)可以转换成求[clip_image026[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235013492.png)了。

clip_image027[6]

    我们使用[clip_image029[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235021931.png)来表示[clip_image026[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235035769.png)。如果直接求解，首先面对的是两个参数，而[clip_image010[53]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235037099.png)也是不等式约束，然后再在w上求最小值。这个过程不容易做，那么怎么办呢？

    我们先考虑另外一个问题[clip_image030[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235042016.png)

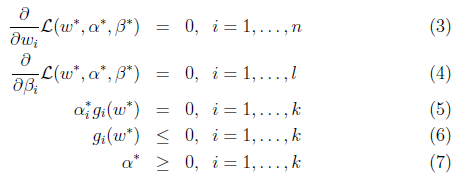
    D的意思是对偶，[clip_image031[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235055538.png)将问题转化为先求拉格朗日关于w的最小值，将[clip_image033[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235066867.png)和[clip_image003[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123507356.png)看作是固定值。之后在[clip_image031[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123507814.png)求最大值的话：

[clip_image034[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235084651.png)

    这个问题是原问题的对偶问题，相对于原问题只是更换了min和max的顺序，而一般更换顺序的结果是Max Min(X) <= MinMax(X)。然而在这里两者相等。用[clip_image036[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235092600.png)来表示对偶问题如下：

clip_image037[6]

    下面解释在什么条件下两者会等价。假设f和g都是凸函数，h是仿射的（affine，[clip_image038[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235116023.png)）。并且存在w使得对于所有的i，[clip_image040[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235119338.png)。在这种假设下，一定存在[clip_image042[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235128748.png)使得[clip_image044[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235129795.png)是原问题的解，[clip_image046[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235134679.png)是对偶问题的解。还有[clip_image047[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123514677.png)另外，[clip_image042[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235149531.png)满足库恩-塔克条件（Karush-Kuhn-Tucker, KKT condition），该条件如下：

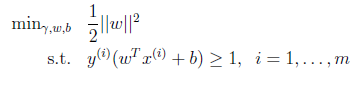


    所以如果[clip_image042[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235189724.png)满足了库恩-塔克条件，那么他们就是原问题和对偶问题的解。让我们再次审视公式（5），这个条件称作是KKT dual complementarity条件。这个条件隐含了如果[clip_image050[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235189102.png)，那么[clip_image052[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235198479.png)。也就是说，[clip_image052[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235209493.png)时，w处于可行域的边界上，这时才是起作用的约束。而其他位于可行域内部（[clip_image054[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235205315.png)的）点都是不起作用的约束，其[clip_image056[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235219917.png)。这个KKT双重补足条件会用来解释支持向量和SMO的收敛测试。

这部分内容思路比较凌乱，还需要先研究下《非线性规划》中的约束极值问题，再回头看看。KKT的总体思想是将极值会在可行域边界上取得，也就是不等式为0或等式约束里取得，而最优下降方向一般是这些等式的线性组合，其中每个元素要么是不等式为0的约束，要么是等式约束。对于在可行域边界内的点，对最优解不起作用，因此前面的系数为0。

## 7 最优间隔分类器（optimal margin classifier）

    重新回到SVM的优化问题：

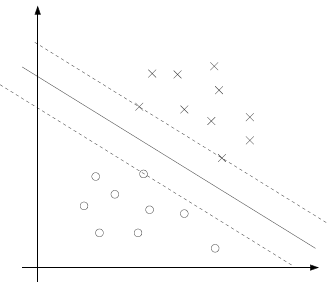


    我们将约束条件改写为：

clip_image058[6]

    从KKT条件得知只有函数间隔是1（离超平面最近的点）的线性约束式前面的系数[clip_image060[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235237560.png)，也就是说这些约束式[clip_image062[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235236621.png)，对于其他的不在线上的点([clip_image064[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123524492.png))，极值不会在他们所在的范围内取得，因此前面的系数[clip_image066[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235259870.png).注意每一个约束式实际就是一个训练样本。

    看下面的图：



    实线是最大间隔超平面，假设×号的是正例，圆圈的是负例。在虚线上的点就是函数间隔是1的点，那么他们前面的系数[clip_image060[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235264688.png)，其他点都是[clip_image066[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235271765.png)。这三个点称作支持向量。构造拉格朗日函数如下：

clip_image068[6]

    注意到这里只有[clip_image010[54]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235294947.png)没有[clip_image012[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235306484.png)是因为原问题中没有等式约束，只有不等式约束。

    下面我们按照对偶问题的求解步骤来一步步进行，

clip_image069[10]

    首先求解[clip_image070[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235317224.png)的最小值，对于固定的[clip_image010[55]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235316635.png)，[clip_image070[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235329285.png)的最小值只与w和b有关。对w和b分别求偏导数。

clip_image071[6]

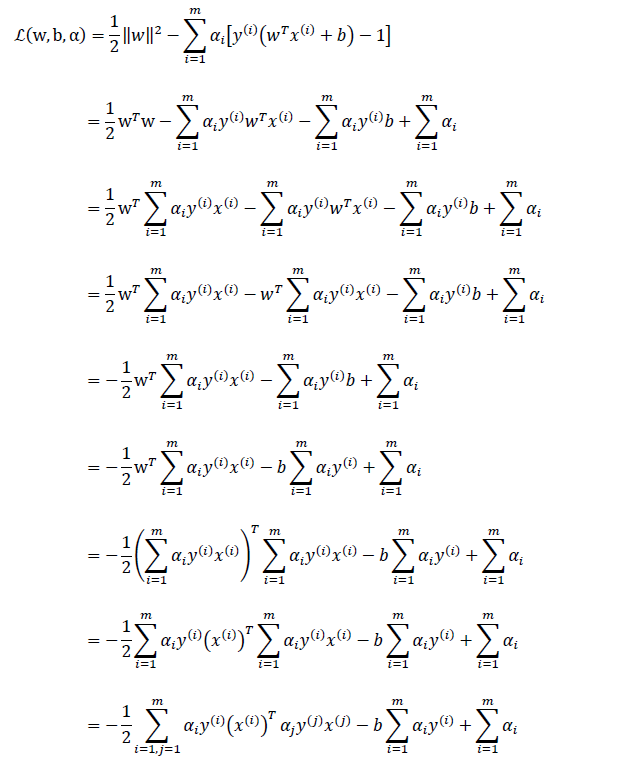
clip_image072[6]

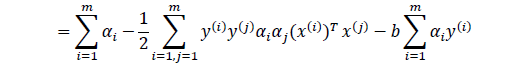
    并得到

clip_image073[6]

    将上式带回到拉格朗日函数中得到，此时得到的是该函数的最小值（目标函数是凸函数）

    代入后，化简过程如下：





　　最后得到

clip_image074[6]

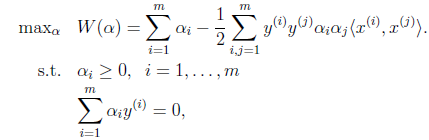
     由于最后一项是0，因此简化为

clip_image075[6]

    这里我们将向量内积[clip_image076[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235368365.png)表示为[clip_image077[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235365268.png)

    此时的拉格朗日函数只包含了变量[clip_image010[56]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235374645.png)。然而我们求出了[clip_image010[57]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235382594.png)才能得到w和b。

    接着是极大化的过程[clip_image069[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/2011031312353920.png)，

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235392986.png)

    前面提到过对偶问题和原问题满足的几个条件，首先由于目标函数和线性约束都是凸函数，而且这里不存在等式约束h。存在w使得对于所有的i，[clip_image040[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235409888.png)。因此，一定存在[clip_image080[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235411217.png)使得[clip_image044[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123541803.png)是原问题的解，[clip_image082[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235429067.png)是对偶问题的解。在这里，求[clip_image010[58]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123543920.png)就是求[clip_image082[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235448869.png)了。

    如果求出了[clip_image010[59]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235454867.png)，根据[clip_image083[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235468704.jpg)即可求出w（也是[clip_image044[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235476130.png)，原问题的解）。然后

clip_image084[6]

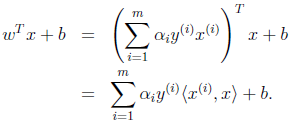
    即可求出b。即离超平面最近的正的函数间隔要等于离超平面最近的负的函数间隔。

    关于上面的对偶问题如何求解，将留给下一篇中的SMO算法来阐明。

    这里考虑另外一个问题，由于前面求解中得到

clip_image086[6]

    我们通篇考虑问题的出发点是[clip_image088[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235489553.png)，根据求解得到的[clip_image010[60]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235499487.png)，我们代入前式得到



    也就是说，以前新来的要分类的样本首先根据w和b做一次线性运算，然后看求的结果是大于0还是小于0,来判断正例还是负例。现在有了[clip_image010[61]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235501830.png)，我们不需要求出w，只需将新来的样本和训练数据中的所有样本做内积和即可。那有人会说，与前面所有的样本都做运算是不是太耗时了？其实不然，我们从KKT条件中得到，只有支持向量的[clip_image060[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103131235519779.png)，其他情况[clip_image066[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110313123552552.png)。因此，我们只需求新来的样本和支持向量的内积，然后运算即可。这种写法为下面要提到的核函数（kernel）做了很好的铺垫。这是上篇，先写这么多了。

## 8 核函数（Kernels）

考虑我们最初在“线性回归”中提出的问题，特征是房子的面积x，这里的x是实数，结果y是房子的价格。假设我们从样本点的分布中看到x和y符合3次曲线，那么我们希望使用x的三次多项式来逼近这些样本点。那么首先需要将特征x扩展到三维[clip_image002[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034289942.png)，然后寻找特征和结果之间的模型。我们将这种特征变换称作特征映射（feature mapping）。映射函数称作[clip_image004[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034282417.png)，在这个例子中

[clip_image006[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034285731.png)

我们希望将得到的特征映射后的特征应用于SVM分类，而不是最初的特征。这样，我们需要将前面[clip_image008[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034298207.png)公式中的内积从[clip_image010[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034294270.png)，映射到[clip_image012[42]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203429333.png)。

至于为什么需要映射后的特征而不是最初的特征来参与计算，上面提到的（为了更好地拟合）是其中一个原因，另外的一个重要原因是样例可能存在线性不可分的情况，而将特征映射到高维空间后，往往就可分了。（在《数据挖掘导论》Pang-Ning Tan等人著的《支持向量机》那一章有个很好的例子说明）

将核函数形式化定义，如果原始特征内积是[clip_image014[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034291172.png)，映射后为[clip_image016[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034309711.png)，那么定义核函数（Kernel）为

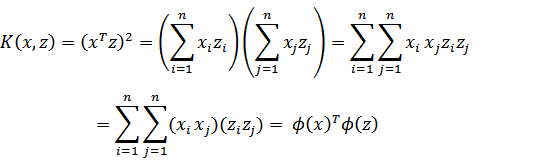
[clip_image018[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034302186.png)

到这里，我们可以得出结论，如果要实现该节开头的效果，只需先计算[clip_image020[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034315500.png)，然后计算[clip_image022[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034315991.png)即可，然而这种计算方式是非常低效的。比如最初的特征是n维的，我们将其映射到[clip_image024[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034326481.png)维，然后再计算，这样需要[clip_image026[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034321398.png)的时间。那么我们能不能想办法减少计算时间呢？

先看一个例子，假设x和z都是n维的，

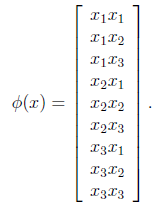
[clip_image028[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034331889.png)

展开后，得

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034337396.png)

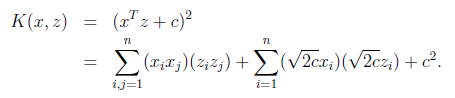
这个时候发现我们可以只计算原始特征x和z内积的平方（时间复杂度是O(n)），就等价与计算映射后特征的内积。也就是说我们不需要花[clip_image026[7]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034341266.png)时间了。

现在看一下映射函数（n=3时），根据上面的公式，得到

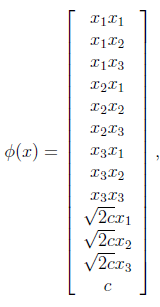
[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034345660.png)

也就是说核函数[clip_image033[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034359215.png)只能在选择这样的[clip_image004[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034356118.png)作为映射函数时才能够等价于映射后特征的内积。

再看一个核函数

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034362464.png)

对应的映射函数（n=3时）是

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034367414.png)

更一般地，核函数[clip_image037[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034372921.png)对应的映射后特征维度为[clip_image039[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034378984.png)。（求解方法参见<http://zhidao.baidu.com/question/16706714.html>）。

由于计算的是内积，我们可以想到IR中的余弦相似度，如果x和z向量夹角越小，那么核函数值越大，反之，越小。因此，核函数值是[clip_image020[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034377838.png)和[clip_image041[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203438804.png)的相似度。

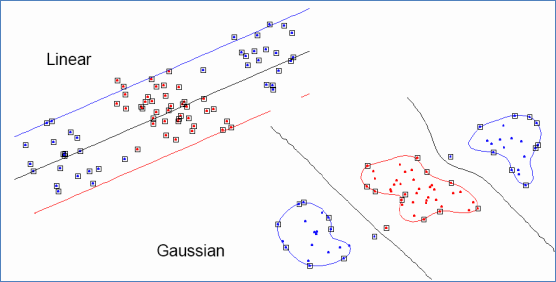
再看另外一个核函数

[clip_image042[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034396834.png)

这时，如果x和z很相近（[clip_image044[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034396245.png)），那么核函数值为1，如果x和z相差很大（[clip_image046[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034398720.png)），那么核函数值约等于0。由于这个函数类似于高斯分布，因此称为高斯核函数，也叫做径向基函数(Radial Basis Function 简称RBF)。它能够把原始特征映射到无穷维。

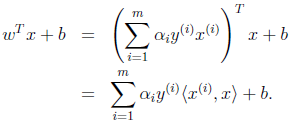
既然高斯核函数能够比较x和z的相似度，并映射到0到1，回想logistic回归，sigmoid函数可以，因此还有sigmoid核函数等等。

下面有张图说明在低维线性不可分时，映射到高维后就可分了，使用高斯核函数。

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203440606.png)

(来自Eric Xing的slides)

注意，使用核函数后，怎么分类新来的样本呢？线性的时候我们使用SVM学习出w和b，新来样本x的话，我们使用[clip_image050[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034414967.png)来判断，如果值大于等于1，那么是正类，小于等于是负类。在两者之间，认为无法确定。如果使用了核函数后，[clip_image050[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034419394.png)就变成了[clip_image052[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034419045.png)，是否先要找到[clip_image054[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034424312.png)，然后再预测？答案肯定不是了，找[clip_image054[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034436721.png)很麻烦，回想我们之前说过的

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034435783.png)

只需将[clip_image057[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034447701.png)替换成[clip_image059[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034443765.png)，然后值的判断同上。

## 9 核函数有效性判定

问题：给定一个函数K，我们能否使用K来替代计算[clip_image022[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034446240.png)，也就说，是否能够找出一个[clip_image061[12]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034459554.png)，使得对于所有的x和z，都有[clip_image018[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034452553.png)？

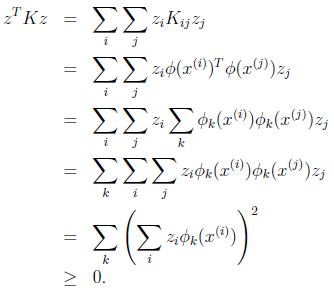
比如给出了[clip_image063[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034451124.png)，是否能够认为K是一个有效的核函数。

下面来解决这个问题，给定m个训练样本[clip_image065[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034464472.png)，每一个[clip_image067[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034463882.png)对应一个特征向量。那么，我们可以将任意两个[clip_image067[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034472388.png)和[clip_image069[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203448337.png)带入K中，计算得到[clip_image071[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034489191.png)。I可以从1到m，j可以从1到m，这样可以计算出m\*m的核函数矩阵（Kernel Matrix）。为了方便，我们将核函数矩阵和[clip_image073[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034491110.png)都使用K来表示。

如果假设K是有效地核函数，那么根据核函数定义

[clip_image075[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034496617.png)

可见，矩阵K应该是个对称阵。让我们得出一个更强的结论，首先使用符号[clip_image077[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034492124.png)来表示映射函数[clip_image020[12]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203450629.png)的第k维属性值。那么对于任意向量z，得

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034519417.png)

最后一步和前面计算[clip_image063[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203452954.png)时类似。从这个公式我们可以看出，如果K是个有效的核函数（即[clip_image073[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034529285.png)和[clip_image080[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034533679.png)等价），那么，在训练集上得到的核函数矩阵K应该是半正定的（[clip_image082[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034546121.png)）

这样我们得到一个核函数的必要条件：

K是有效的核函数 ==> 核函数矩阵K是对称半正定的。

可幸的是，这个条件也是充分的，由Mercer定理来表达。

|  |
| --- |
| **Mercer定理：**  如果函数K是[clip_image084[26]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034544104.png)上的映射（也就是从两个n维向量映射到实数域）。那么如果K是一个有效核函数（也称为Mercer核函数），那么当且仅当对于训练样例[clip_image065[7]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034545707.png)，其相应的核函数矩阵是对称半正定的。 |

Mercer定理表明为了证明K是有效的核函数，那么我们不用去寻找[clip_image061[13]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034555674.png)，而只需要在训练集上求出各个[clip_image086[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034565575.png)，然后判断矩阵K是否是半正定（使用左上角主子式大于等于零等方法）即可。

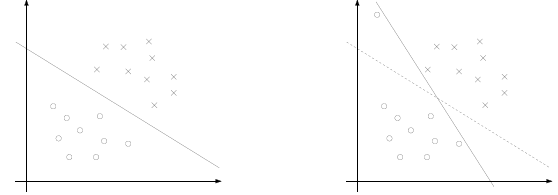
许多其他的教科书在Mercer定理证明过程中使用了[clip_image088[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034569969.png)范数和再生希尔伯特空间等概念，但在特征是n维的情况下，这里给出的证明是等价的。

核函数不仅仅用在SVM上，但凡在一个模型后算法中出现了[clip_image090[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034574330.png)，我们都可以常使用[clip_image073[12]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034584264.png)去替换，这可能能够很好地改善我们的算法。

## 10 规则化和不可分情况处理（Regularization and the non-separable case）

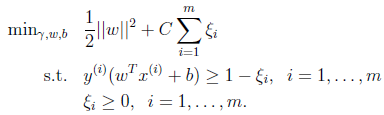
我们之前讨论的情况都是建立在样例线性可分的假设上，当样例线性不可分时，我们可以尝试使用核函数来将特征映射到高维，这样很可能就可分了。然而，映射后我们也不能100%保证可分。那怎么办呢，我们需要将模型进行调整，以保证在不可分的情况下，也能够尽可能地找出分隔超平面。

看下面两张图：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037075228.png)

可以看到一个离群点（可能是噪声）可以造成超平面的移动，间隔缩小，可见以前的模型对噪声非常敏感。再有甚者，如果离群点在另外一个类中，那么这时候就是线性不可分了。

这时候我们应该允许一些点游离并在在模型中违背限制条件（函数间隔大于1）。我们设计得到新的模型如下（也称软间隔）：

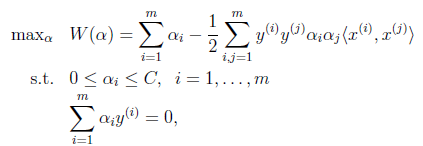
[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037085162.png)

引入非负参数[clip_image004](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037095620.png)后（称为松弛变量），就允许某些样本点的函数间隔小于1，即在最大间隔区间里面，或者函数间隔是负数，即样本点在对方的区域中。而放松限制条件后，我们需要重新调整目标函数，以对离群点进行处罚，目标函数后面加上的[clip_image006](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037095554.png)就表示离群点越多，目标函数值越大，而我们要求的是尽可能小的目标函数值。这里的C是离群点的权重，C越大表明离群点对目标函数影响越大，也就是越不希望看到离群点。我们看到，目标函数控制了离群点的数目和程度，使大部分样本点仍然遵守限制条件。

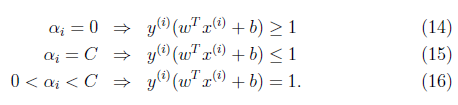
模型修改后，拉格朗日公式也要修改如下：

[clip_image008](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203710504.jpg)

这里的[clip_image010](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037111834.png)和[clip_image012](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037126195.png)都是拉格朗日乘子，回想我们在拉格朗日对偶中提到的求法，先写出拉格朗日公式（如上），然后将其看作是变量w和b的函数，分别对其求偏导，得到w和b的表达式。然后代入公式中，求带入后公式的极大值。整个推导过程类似以前的模型，这里只写出最后结果如下：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203713207.png)

此时，我们发现没有了参数[clip_image004[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037171173.png)，与之前模型唯一不同在于[clip_image010[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037185534.png)又多了[clip_image015](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037194563.png)的限制条件。需要提醒的是，b的求值公式也发生了改变，改变结果在SMO算法里面介绍。先看看KKT条件的变化：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037208924.png)

第一个式子表明在两条间隔线外的样本点前面的系数为0，离群样本点前面的系数为C，而支持向量（也就是在超平面两边的最大间隔线上）的样本点前面系数在(0,C)上。通过KKT条件可知，某些在最大间隔线上的样本点也不是支持向量，相反也可能是离群点。

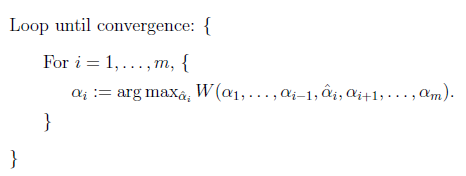
## 11 坐标上升法（Coordinate ascent）

在最后讨论[clip_image018](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037203875.png)的求解之前，我们先看看坐标上升法的基本原理。假设要求解下面的优化问题：

[clip_image019](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037211333.png)

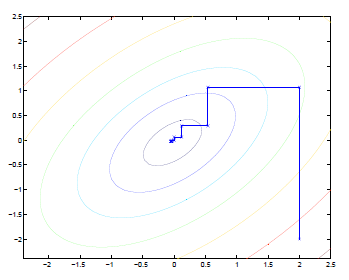
这里W是[clip_image021](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037215171.png)向量的函数。之前我们在回归中提到过两种求最优解的方法，一种是梯度下降法，另外一种是牛顿法。现在我们再讲一种方法称为坐标上升法（求解最小值问题时，称作坐标下降法，原理一样）。

方法过程：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037227265.png)

最里面语句的意思是固定除[clip_image010[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037232498.png)之外的所有[clip_image024](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037244068.png)，这时W可看作只是关于[clip_image010[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037242050.png)的函数，那么直接对[clip_image010[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203725871.png)求导优化即可。这里我们进行最大化求导的顺序i是从1到m，可以通过更改优化顺序来使W能够更快地增加并收敛。如果W在内循环中能够很快地达到最优，那么坐标上升法会是一个很高效的求极值方法。

下面通过一张图来展示：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182037262167.png)

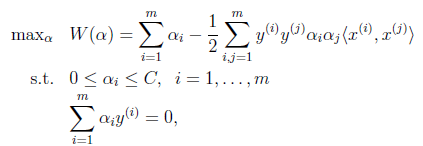
椭圆代表了二次函数的各个等高线，变量数为2，起始坐标是(2,-2)。图中的直线式迭代优化的路径，可以看到每一步都会向最优值前进一步，而且前进路线是平行于坐标轴的，因为每一步只优化一个变量。

## 12 SMO优化算法（Sequential minimal optimization）

SMO算法由Microsoft Research的John C. Platt在1998年提出，并成为最快的二次规划优化算法，特别针对线性SVM和数据稀疏时性能更优。关于SMO最好的资料就是他本人写的《Sequential Minimal Optimization A Fast Algorithm for Training Support Vector Machines》了。

我拜读了一下，下面先说讲义上对此方法的总结。

首先回到我们前面一直悬而未解的问题，对偶函数最后的优化问题：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042433212.png)

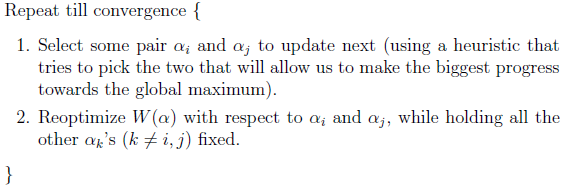
要解决的是在参数[clip_image003](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042431751.png)上求最大值W的问题，至于[clip_image005](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042448097.png)和[clip_image007](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042454094.png)都是已知数。C由我们预先设定，也是已知数。

按照坐标上升的思路，我们首先固定除[clip_image009](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042464867.png)以外的所有参数，然后在[clip_image009[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042479228.png)上求极值。等一下，这个思路有问题，因为如果固定[clip_image009[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042489129.png)以外的所有参数，那么[clip_image009[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042481538.png)将不再是变量（可以由其他值推出），因为问题中规定了

[clip_image010](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042498997.png)

因此，我们需要一次选取两个参数做优化，比如[clip_image009[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042506423.png)和[clip_image012](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042509945.png)，此时[clip_image012[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042519846.png)可以由[clip_image009[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042524207.png)和其他参数表示出来。这样回带到W中，W就只是关于[clip_image009[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042548469.png)的函数了，可解。

这样，SMO的主要步骤如下：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204256430.png)

意思是，第一步选取一对[clip_image015](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042575630.png)和[clip_image017](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042585531.png)，选取方法使用启发式方法（后面讲）。第二步，固定除[clip_image015[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182042591528.png)和[clip_image017[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043003381.png)之外的其他参数，确定W极值条件下的[clip_image015[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043017742.png)，[clip_image017[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043017676.png)由[clip_image015[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043021722.png)表示。

SMO之所以高效就是因为在固定其他参数后，对一个参数优化过程很高效。

下面讨论具体方法：

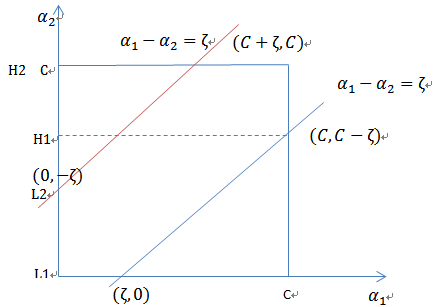
假设我们选取了初始值[clip_image003[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043031655.png)满足了问题中的约束条件。接下来，我们固定[clip_image019](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043031622.png)，这样W就是[clip_image009[7]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043049048.png)和[clip_image012[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043045394.png)的函数。并且[clip_image009[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043059755.png)和[clip_image012[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043061292.png)满足条件：

[clip_image020](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043078718.png)

由于[clip_image019[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043075620.png)都是已知固定值，因此为了方面，可将等式右边标记成实数值[clip_image022](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043089175.png)。

[clip_image023](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043089425.png)

当[clip_image025](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043096535.png)和[clip_image027](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204310896.png)异号时，也就是一个为1，一个为-1时，他们可以表示成一条直线，斜率为1。如下图：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043119717.png)

横轴是[clip_image009[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043122126.png)，纵轴是[clip_image012[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043136695.png)，[clip_image009[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043166299.png)和[clip_image012[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043179024.png)既要在矩形方框内，也要在直线上，因此

[clip_image044](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043182546.png)，[clip_image046](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043184465.png)

同理，当[clip_image025[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043199938.png)和[clip_image027[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043207888.png)同号时，

[clip_image048](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043205870.png)，[clip_image050](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043214200.png)

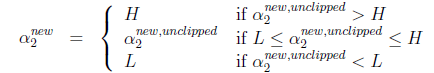
然后我们打算将[clip_image009[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043216086.png)用[clip_image012[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043225672.png)表示：

[clip_image051](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043233446.png)

然后反代入W中，得

[clip_image052](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043236968.png)

展开后W可以表示成[clip_image054](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043245506.png)。其中a,b,c是固定值。这样，通过对W进行求导可以得到[clip_image012[7]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043258264.png)，然而要保证[clip_image012[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043257850.png)满足[clip_image056](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043264196.png)，我们使用[clip_image058](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043269702.png)表示求导求出来的[clip_image012[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043275525.png)，然而最后的[clip_image012[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043274587.png)，要根据下面情况得到：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043289504.png)

这样得到[clip_image061](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043296407.png)后，我们可以得到[clip_image009[12]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043309896.png)的新值[clip_image063](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043319797.png)。

下面进入Platt的文章，来找到启发式搜索的方法和求b值的公式。

这边文章使用的符号表示有点不太一样，不过实质是一样的，先来熟悉一下文章中符号的表示。

文章中定义特征到结果的输出函数为

[clip_image064](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043317222.png)

与我们之前的[clip_image066](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043328585.png)实质是一致的。

原始的优化问题为：

[clip_image067](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043329075.png)

求导得到：

[clip_image068](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043337373.png)

经过对偶后为：

[clip_image069](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043347307.png)

s.t. [clip_image070](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043346925.png)

[clip_image071](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043353827.png)

这里与W函数是一样的，只是符号求反后，变成求最小值了。[clip_image073](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043357142.png)和[clip_image075](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043365091.png)是一样的，都表示第i个样本的输出结果（1或-1）。

经过加入松弛变量[clip_image077](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043373040.png)后，模型修改为：

[clip_image078](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043389909.png)

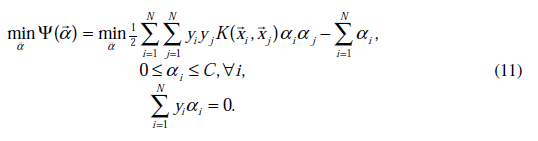
[clip_image079](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043384303.png)

由公式（7）代入（1）中可知，

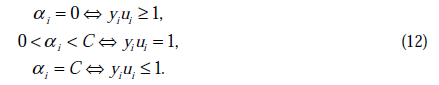
[clip_image080](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043396430.png)

这个过程和之前对偶过程一样。

重新整理我们要求的问题为：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043405218.png)

与之对应的KKT条件为：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043412611.png)

这个KKT条件说明，在两条间隔线外面的点，对应前面的系数[clip_image084](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043427528.png)为0，在两条间隔线里面的对应[clip_image084[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043435477.png)为C，在两条间隔线上的对应的系数[clip_image084[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043438999.png)在0和C之间。

将我们之前得到L和H重新拿过来：

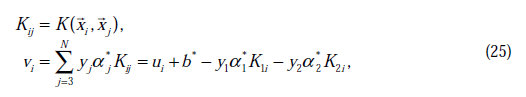
[clip_image085](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043443917.png)

[clip_image086](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043454723.png)

之前我们将问题进行到这里，然后说将[clip_image009[13]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043453785.png)用[clip_image012[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043463718.png)表示后代入W中，这里将代入[clip_image088](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043473935.png)中，得

[clip_image090](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043481012.jpg)

其中

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043493770.png)

这里的[clip_image093](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043494784.png)和[clip_image095](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043502176.png)代表某次迭代前的原始值，因此是常数，而[clip_image009[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043515981.png)和[clip_image012[12]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043527518.png)是变量，待求。公式（24）中的最后一项是常数。

由于[clip_image009[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043535467.png)和[clip_image012[13]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043548192.png)满足以下公式

[clip_image097](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043557254.png)

因为[clip_image099](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204355253.png)的值是固定值，在迭代前后不会变。

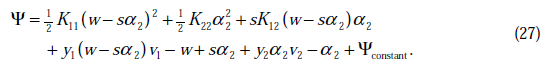
那么用s表示[clip_image101](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204356502.png)，上式两边乘以[clip_image103](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043564024.png)时，变为：

[clip_image104](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043577023.png)

其中

[clip_image106](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043573925.png)

代入（24）中，得

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204358828.png)

这时候只有[clip_image012[14]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204359205.png)是变量了，求导

[clip_image109](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182043596235.jpg)

如果[clip_image088[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044003105.png)的二阶导数大于0（凹函数），那么一阶导数为0时，就是极小值了。

假设其二阶导数为0（一般成立），那么上式化简为：

[clip_image110](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044019658.png)

将w和v代入后，继续化简推导，得（推导了六七行推出来了）

[clip_image112](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044012416.jpg)

我们使用[clip_image114](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044026462.png)来表示：

[clip_image115](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044031412.png)

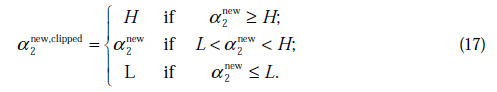
通常情况下目标函数是正定的，也就是说，能够在直线约束方向上求得最小值，并且[clip_image117](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044065476.png)。

那么我们在（30）两边都除以[clip_image114[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044071822.png)可以得到

[clip_image118](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204408327.png)

这里我们使用[clip_image061[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204408261.png)表示优化后的值，[clip_image012[15]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204409195.png)是迭代前的值，[clip_image120](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044103717.png)。

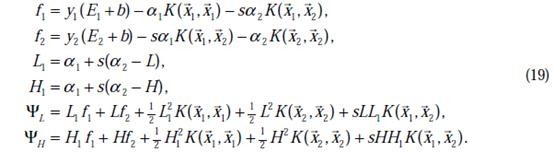
与之前提到的一样[clip_image061[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044104000.png)不是最终迭代后的值，需要进行约束：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044116409.png)

那么

[clip_image122](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044116027.png)

在特殊情况下，[clip_image114[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044121817.png)可能不为正，如果核函数K不满足Mercer定理，那么目标函数可能变得非正定，[clip_image114[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044133354.png)可能出现负值。即使K是有效的核函数，如果训练样本中出现相同的特征x，那么[clip_image114[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044145763.png)仍有可能为0。SMO算法在[clip_image114[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204415124.png)不为正值的情况下仍有效。为保证有效性，我们可以推导出[clip_image114[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/2011031820441558.png)就是[clip_image088[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044163580.png)的二阶导数，[clip_image124](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044173481.png)，[clip_image088[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044183382.png)没有极小值，最小值在边缘处取到（类比[clip_image126](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044196107.png)），[clip_image128](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044205692.png)时更是单调函数了，最小值也在边缘处取得，而[clip_image012[16]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044217545.png)的边缘就是L和H。这样将[clip_image130](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044229398.png)和[clip_image132](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044224905.png)分别代入[clip_image088[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044234839.png)中即可求得[clip_image088[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204423346.png)的最小值，相应的[clip_image130[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044244707.png)还是[clip_image132[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044252656.png)也可以知道了。具体计算公式如下：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044261477.jpg)

至此，迭代关系式出了b的推导式以外，都已经推出。

b每一步都要更新，因为前面的KKT条件指出了[clip_image084[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044274900.png)和[clip_image136](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044283198.png)的关系，而[clip_image138](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044294211.png)和b有关，在每一步计算出[clip_image084[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044294145.png)后，根据KKT条件来调整b。

b的更新有几种情况：

[](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044317045.jpg)

(来自罗林开的ppt)

这里的界内指[clip_image142](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044326323.png)，界上就是等于0或者C了。

前面两个的公式推导可以根据[clip_image144](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044337686.png)

和对于[clip_image142[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044337096.png)有[clip_image146](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044332603.png)的KKT条件推出。

这样全部参数的更新公式都已经介绍完毕，附加一点，如果使用的是线性核函数，我们就可以继续使用w了，这样不用扫描整个样本库来作内积了。

w值的更新方法为：

[clip_image147](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044341109.png)

根据前面的

[clip_image068[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044359406.png)

公式推导出。

## 13 SMO中拉格朗日乘子的启发式选择方法

终于到了最后一个问题了，所谓的启发式选择方法主要思想是每次选择拉格朗日乘子的时候，优先选择样本前面系数[clip_image142[2]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044353485.png)的[clip_image084[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204436911.png)作优化（论文中称为无界样例），因为在界上（[clip_image084[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044378860.png)为0或C）的样例对应的系数[clip_image084[7]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044382697.png)一般不会更改。

这条启发式搜索方法是选择第一个拉格朗日乘子用的，比如前面的[clip_image012[17]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318204439646.png)。那么这样选择的话，是否最后会收敛。可幸的是Osuna定理告诉我们只要选择出来的两个[clip_image084[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044408595.png)中有一个违背了KKT条件，那么目标函数在一步迭代后值会减小。违背KKT条件不代表[clip_image142[3]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044408529.png)，在界上也有可能会违背。是的，因此在给定初始值[clip_image084[9]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044418812.png)=0后，先对所有样例进行循环，循环中碰到违背KKT条件的（不管界上还是界内）都进行迭代更新。等这轮过后，如果没有收敛，第二轮就只针对[clip_image142[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044418746.png)的样例进行迭代更新。

在第一个乘子选择后，第二个乘子也使用启发式方法选择，第二个乘子的迭代步长大致正比于[clip_image149](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044414253.png)，选择第二个乘子能够最大化[clip_image149[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044425823.png)。即当[clip_image151](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044439661.png)为正时选择负的绝对值最大的[clip_image153](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044439595.png)，反之，选择正值最大的[clip_image153[1]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044443465.png)。

最后的收敛条件是在界内（[clip_image142[5]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044446987.png)）的样例都能够遵循KKT条件，且其对应的[clip_image084[10]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182044456049.png)只在极小的范围内变动。

至于如何写具体的程序，请参考John C. Platt在论文中给出的伪代码。

## 14 总结

这份SVM的讲义重点概括了SVM的基本概念和基本推导，中规中矩却又让人醍醐灌顶。起初让我最头疼的是拉格朗日对偶和SMO，后来逐渐明白拉格朗日对偶的重要作用是将w的计算提前并消除w，使得优化函数变为拉格朗日乘子的单一参数优化问题。而SMO里面迭代公式的推导也着实花费了不少时间。

对比这么复杂的推导过程，SVM的思想确实那么简单。它不再像logistic回归一样企图去拟合样本点（中间加了一层sigmoid函数变换），而是就在样本中去找分隔线，为了评判哪条分界线更好，引入了几何间隔最大化的目标。

之后所有的推导都是去解决目标函数的最优化上了。在解决最优化的过程中，发现了w可以由特征向量内积来表示，进而发现了核函数，仅需要调整核函数就可以将特征进行低维到高维的变换，在低维上进行计算，实质结果表现在高维上。由于并不是所有的样本都可分，为了保证SVM的通用性，进行了软间隔的处理，导致的结果就是将优化问题变得更加复杂，然而惊奇的是松弛变量没有出现在最后的目标函数中。最后的优化求解问题，也被拉格朗日对偶和SMO算法化解，使SVM趋向于完美。

另外，其他很多议题如SVM背后的学习理论、参数选择问题、二值分类到多值分类等等还没有涉及到，以后有时间再学吧。其实朴素贝叶斯在分类二值分类问题时，如果使用对数比，那么也算作线性分类器。

# 二、代码实现例子

## 1. SVM的介绍1

代码：

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**from** sklearn.model\_selection **import** train\_test\_split

**import** matplotlib **as** mpl

**import** matplotlib.pyplot **as** plt

**def** *iris\_type*(s):

it = {b'Iris-setosa': 0, b'Iris-versicolor': 1, b'Iris-virginica': 2}

**return** it[s]

*# 'sepal length', 'sepal width', 'petal length', 'petal width'*

iris\_feature = u'花萼长度', u'花萼宽度', u'花瓣长度', u'花瓣宽度'

**def** *show\_accuracy*(a, b, tip):

acc = a.ravel() == b.ravel()

**print**(tip + '正确率：', np.mean(acc))

**if** \_\_name\_\_ == "\_\_main\_\_":

path = '8.iris.data' *# 数据文件路径*

data = np.loadtxt(path, dtype=float, delimiter=',', converters={4: iris\_type})

x, y = np.split(data, (4,), axis=1)

x = x[:, :2]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=1, train\_size=0.6)

*# 分类器*

*# clf = svm.SVC(C=0.1, kernel='linear', decision\_function\_shape='ovr')*

clf = svm.SVC(C=0.8, kernel='rbf', gamma=20, decision\_function\_shape='ovr')

clf.fit(x\_train, y\_train.ravel())

*# 准确率*

**print**(clf.score(x\_train, y\_train)) *# 精度*

y\_hat = clf.predict(x\_train)

show\_accuracy(y\_hat, y\_train, '训练集')

**print**(clf.score(x\_test, y\_test))

y\_hat = clf.predict(x\_test)

show\_accuracy(y\_hat, y\_test, '测试集')

*# 画图*

x1\_min, x1\_max = x[:, 0].min(), x[:, 0].max() *# 第0列的范围*

x2\_min, x2\_max = x[:, 1].min(), x[:, 1].max() *# 第1列的范围*

x1, x2 = np.mgrid[x1\_min:x1\_max:500j, x2\_min:x2\_max:500j] *# 生成网格采样点*

grid\_test = np.stack((x1.flat, x2.flat), axis=1) *# 测试点*

**print**("grid\_test=\n",grid\_test)

Z = clf.decision\_function(grid\_test) *# 样本到决策面的距离*

**print**(Z)

grid\_hat = clf.predict(grid\_test) *# 预测分类值*

**print**(grid\_hat)

grid\_hat = grid\_hat.reshape(x1.shape) *# 使之与输入的形状相同*

mpl.rcParams['font.sans-serif'] = [u'SimHei']

mpl.rcParams['axes.unicode\_minus'] = **False**

cm\_light = mpl.colors.ListedColormap(['#A0FFA0', '#FFA0A0', '#A0A0FF'])

cm\_dark = mpl.colors.ListedColormap(['g', 'r', 'b'])

x1\_min, x1\_max = x[:, 0].min(), x[:, 0].max() *# 第0列的范围*

x2\_min, x2\_max = x[:, 1].min(), x[:, 1].max() *# 第1列的范围*

x1, x2 = np.mgrid[x1\_min:x1\_max:500j, x2\_min:x2\_max:500j] *# 生成网格采样点*

grid\_test = np.stack((x1.flat, x2.flat), axis=1) *# 测试点*

plt.pcolormesh(x1, x2, grid\_hat, cmap=cm\_light)

plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', s=50, cmap=cm\_dark) *# 样本*

plt.scatter(x\_test[:, 0], x\_test[:, 1], s=120, facecolors='none', zorder=10) *# 圈中测试集样本*

plt.xlabel(iris\_feature[0], fontsize=13)

plt.ylabel(iris\_feature[1], fontsize=13)

plt.xlim(x1\_min, x1\_max)

plt.ylim(x2\_min, x2\_max)

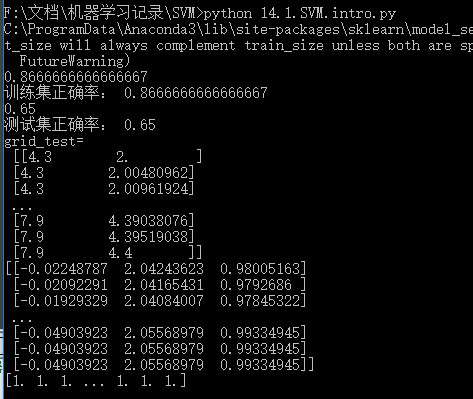
plt.title(u'鸢尾花SVM二特征分类', fontsize=15)

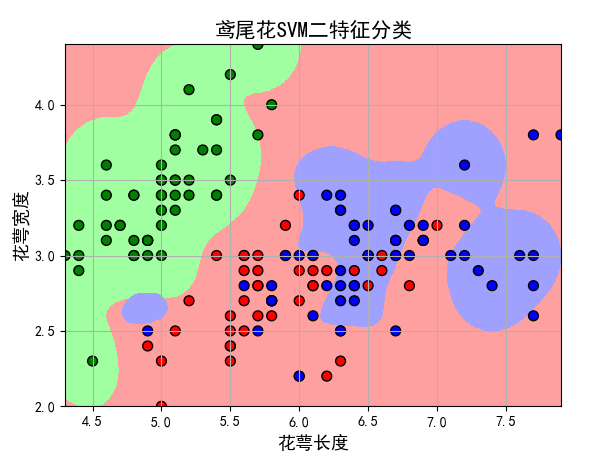
plt.grid()

plt.show()

数据集·：

结果：





## 2. SVM的介绍2

代码：

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**import** matplotlib **as** mpl

**import** matplotlib.colors

**import** matplotlib.pyplot **as** plt

**def** *show\_accuracy*(a, b):

acc = a.ravel() == b.ravel()

*#print '正确率：%.2f%%' % (100\*float(acc.sum()) / a.size)*

**if** \_\_name\_\_ == "\_\_main\_\_":

data = np.loadtxt('14.bipartition.txt', dtype=np.float, delimiter='\t')

x, y = np.split(data, (2, ), axis=1)

y[y == 0] = -1

y = y.ravel()

*# 分类器*

clfs = [svm.SVC(C=0.3, kernel='linear'),

svm.SVC(C=10, kernel='linear'),

svm.SVC(C=5, kernel='rbf', gamma=1),

svm.SVC(C=5, kernel='rbf', gamma=4)]

titles = 'Linear,C=0.3', 'Linear, C=10', 'RBF, gamma=1', 'RBF, gamma=4'

x1\_min, x1\_max = x[:, 0].min(), x[:, 0].max() *# 第0列的范围*

x2\_min, x2\_max = x[:, 1].min(), x[:, 1].max() *# 第1列的范围*

x1, x2 = np.mgrid[x1\_min:x1\_max:500j, x2\_min:x2\_max:500j] *# 生成网格采样点*

grid\_test = np.stack((x1.flat, x2.flat), axis=1) *# 测试点*

cm\_light = matplotlib.colors.ListedColormap(['#77E0A0', '#FF8080'])

cm\_dark = matplotlib.colors.ListedColormap(['g', 'r'])

matplotlib.rcParams['font.sans-serif'] = [u'SimHei']

matplotlib.rcParams['axes.unicode\_minus'] = **False**

plt.figure(figsize=(10,8), facecolor='w')

**for** i, clf **in** enumerate(clfs):

clf.fit(x, y)

y\_hat = clf.predict(x)

show\_accuracy(y\_hat, y) *# 准确率*

*# 画图*

**print**('支撑向量的数目：', clf.n\_support\_)

**print**('支撑向量的系数：', clf.dual\_coef\_)

**print**('支撑向量：', clf.support\_)

*#print*

plt.subplot(2, 2, i+1)

grid\_hat = clf.predict(grid\_test) *# 预测分类值*

grid\_hat = grid\_hat.reshape(x1.shape) *# 使之与输入的形状相同*

plt.pcolormesh(x1, x2, grid\_hat, cmap=cm\_light, alpha=0.8)

plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', s=40, cmap=cm\_dark) *# 样本的显示*

plt.scatter(x[clf.support\_, 0], x[clf.support\_, 1], edgecolors='k', facecolors='none', s=100, marker='o') *# 支撑向量*

z = clf.decision\_function(grid\_test)

z = z.reshape(x1.shape)

plt.contour(x1, x2, z, colors=list('krk'), linestyles=['--', '-', '--'], linewidths=[1, 2, 1], levels=[-1, 0, 1])

plt.xlim(x1\_min, x1\_max)

plt.ylim(x2\_min, x2\_max)

plt.title(titles[i])

plt.grid()

plt.suptitle(u'SVM不同参数的分类', fontsize=18)

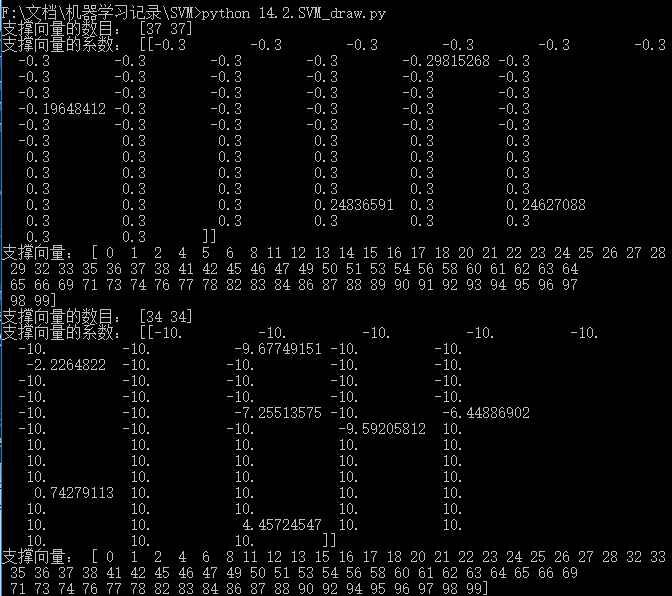
plt.tight\_layout(2)

plt.subplots\_adjust(top=0.92)

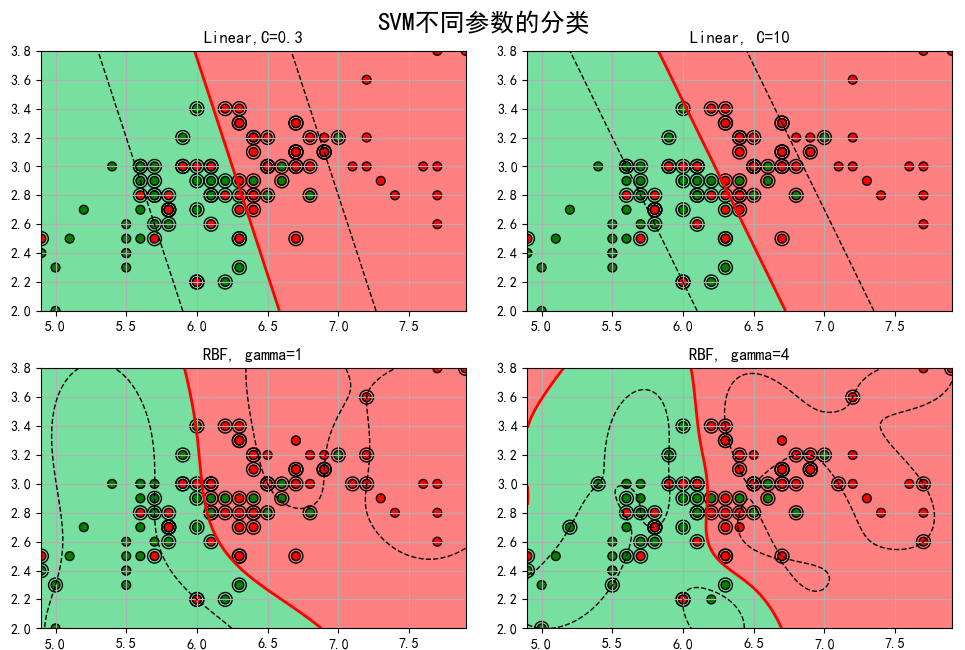
plt.show()

数据集：

结果：







## 3.关于分类器的常用评价指标

（这部分上面的SVM原理中没有讲到，以后可单独记录一下）

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.metrics **import** precision\_score, recall\_score, f1\_score, fbeta\_score

**from** sklearn.metrics **import** precision\_recall\_fscore\_support

**if** \_\_name\_\_ == "\_\_main\_\_":

y\_true = np.array([1, 1, 1, 1, 0, 0])

y\_hat = np.array([1, 0, 1, 1, 1, 1])

**print**('Accuracy：\t', accuracy\_score(y\_true, y\_hat))

*# The precision is the ratio 'tp / (tp + fp)' where 'tp' is the number of*

*# true positives and 'fp' the number of false positives. The precision is*

*# intuitively the ability of the classifier not to label as positive a sample*

*# that is negative.*

*# The best value is 1 and the worst value is 0.*

precision = precision\_score(y\_true, y\_hat)

**print**('Precision:\t', precision)

*# The recall is the ratio 'tp / (tp + fn)' where 'tp' is the number of*

*# true positives and 'fn' the number of false negatives. The recall is*

*# intuitively the ability of the classifier to find all the positive samples.*

*# The best value is 1 and the worst value is 0.*

recall = recall\_score(y\_true, y\_hat)

**print**('Recall: \t', recall)

*# F1 score, also known as balanced F-score or F-measure*

*# The F1 score can be interpreted as a weighted average of the precision and*

*# recall, where an F1 score reaches its best value at 1 and worst score at 0.*

*# The relative contribution of precision and recall to the F1 score are*

*# equal. The formula for the F1 score is:*

*# F1 = 2 \* (precision \* recall) / (precision + recall)*

**print**('f1 score: \t', f1\_score(y\_true, y\_hat))

*#print 2 \* (precision \* recall) / (precision + recall)*

*# The F-beta score is the weighted harmonic mean of precision and recall,*

*# reaching its optimal value at 1 and its worst value at 0.*

*# The 'beta' parameter determines the weight of precision in the combined*

*# score. 'beta < 1' lends more weight to precision, while 'beta > 1'*

*# favors recall ('beta -> 0' considers only precision, 'beta -> inf' only recall).*

**print**('F-beta：')

**for** beta **in** np.logspace(-3, 3, num=7, base=10):

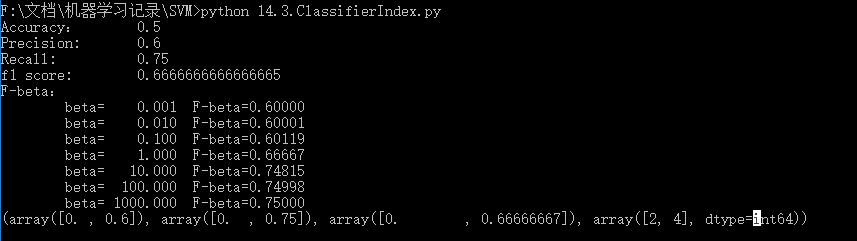
fbeta = fbeta\_score(y\_true, y\_hat, beta=beta)

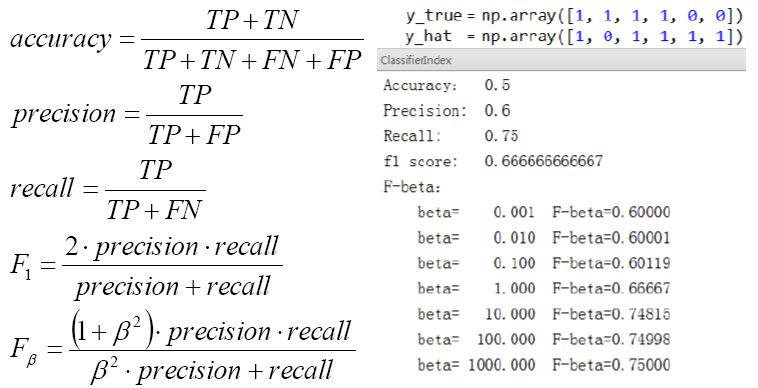
**print**('\tbeta=%9.3f\tF-beta=%.5f' % (beta, fbeta))

*#print (1+beta\*\*2)\*precision\*recall / (beta\*\*2 \* precision + recall)*

**print**(precision\_recall\_fscore\_support(y\_true, y\_hat, beta=1))

结果：





## 4.关于不平衡数据的处理

（其实到目前为止还没有一个统一的做法，一般都是case by case.）

代码：

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**import** matplotlib.colors

**import** matplotlib.pyplot **as** plt

**from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_score, f1\_score, fbeta\_score

**import** warnings

**def** *show\_accuracy*(a, b):

acc = a.ravel() == b.ravel()

**print**('正确率：%.2f%%' % (100\*float(acc.sum()) / a.size))

**def** *show\_recall*(y, y\_hat):

*# print y\_hat[y == 1]*

**print**('召回率：%.2f%%' % (100 \* float(np.sum(y\_hat[y == 1] == 1)) / np.extract(y == 1, y).size))

**if** \_\_name\_\_ == "\_\_main\_\_":

warnings.filterwarnings("ignore") *# UndefinedMetricWarning*

np.random.seed(0) *# 保持每次生成的数据相同*

c1 = 990

c2 = 10

N = c1 + c2

x\_c1 = 3\*np.random.randn(c1, 2)

x\_c2 = 0.5\*np.random.randn(c2, 2) + (4, 4)

x = np.vstack((x\_c1, x\_c2))

y = np.ones(N)

y[:c1] = -1

*# 显示大小*

s = np.ones(N) \* 30

s[:c1] = 10

*# 分类器*

clfs = [svm.SVC(C=1, kernel='linear'),

svm.SVC(C=1, kernel='linear', class\_weight={-1: 1, 1: 10}),

svm.SVC(C=0.8, kernel='rbf', gamma=0.5, class\_weight={-1: 1, 1: 2}),

svm.SVC(C=0.8, kernel='rbf', gamma=0.5, class\_weight={-1: 1, 1: 10})]

titles = 'Linear', 'Linear, Weight=50', 'RBF, Weight=2', 'RBF, Weight=10'

x1\_min, x1\_max = x[:, 0].min(), x[:, 0].max() *# 第0列的范围*

x2\_min, x2\_max = x[:, 1].min(), x[:, 1].max() *# 第1列的范围*

x1, x2 = np.mgrid[x1\_min:x1\_max:500j, x2\_min:x2\_max:500j] *# 生成网格采样点*

grid\_test = np.stack((x1.flat, x2.flat), axis=1) *# 测试点*

cm\_light = matplotlib.colors.ListedColormap(['#77E0A0', '#FF8080'])

cm\_dark = matplotlib.colors.ListedColormap(['g', 'r'])

matplotlib.rcParams['font.sans-serif'] = [u'SimHei']

matplotlib.rcParams['axes.unicode\_minus'] = **False**

plt.figure(figsize=(10, 8), facecolor='w')

**for** i, clf **in** enumerate(clfs):

clf.fit(x, y)

y\_hat = clf.predict(x)

*# show\_accuracy(y\_hat, y) # 正确率*

*# show\_recall(y, y\_hat) # 召回率*

**print**(i+1, '次：')

**print**('正确率：\t', accuracy\_score(y, y\_hat))

**print**(' 精度 ：\t', precision\_score(y, y\_hat, pos\_label=1))

**print**('召回率：\t', recall\_score(y, y\_hat, pos\_label=1))

**print**('F1Score：\t', f1\_score(y, y\_hat, pos\_label=1))

*# 画图*

plt.subplot(2, 2, i+1)

grid\_hat = clf.predict(grid\_test) *# 预测分类值*

grid\_hat = grid\_hat.reshape(x1.shape) *# 使之与输入的形状相同*

plt.pcolormesh(x1, x2, grid\_hat, cmap=cm\_light, alpha=0.8)

plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', s=s, cmap=cm\_dark) *# 样本的显示*

plt.xlim(x1\_min, x1\_max)

plt.ylim(x2\_min, x2\_max)

plt.title(titles[i])

plt.grid()

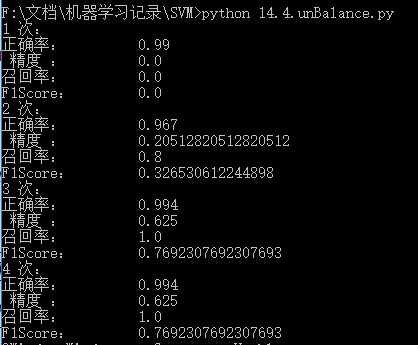
plt.suptitle(u'不平衡数据的处理', fontsize=18)

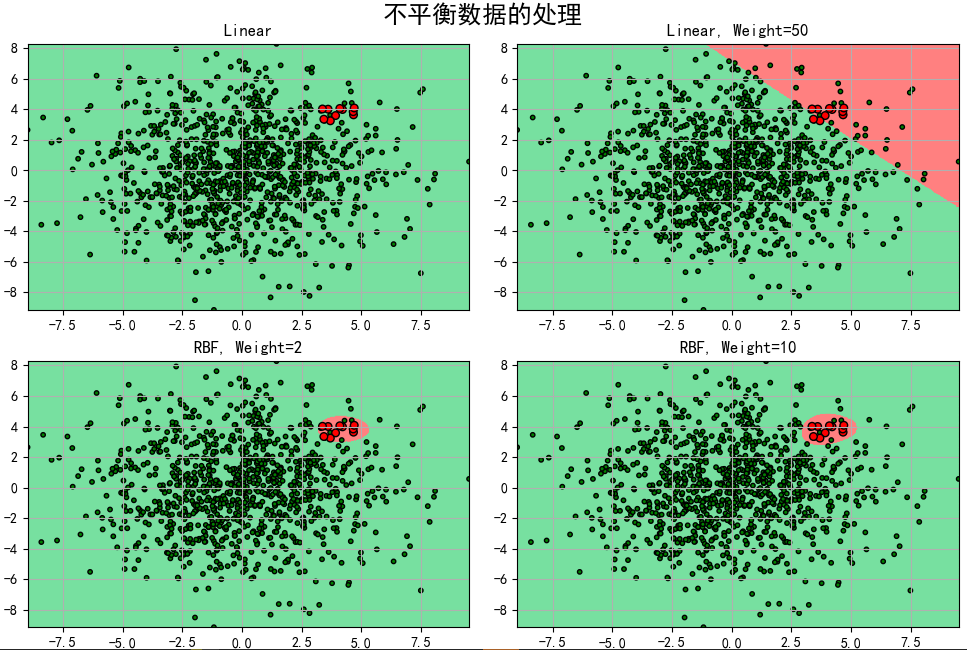
plt.tight\_layout(1.5)

plt.subplots\_adjust(top=0.92)

plt.show()

结果：





## 5.手写数字识别

代码：

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**import** matplotlib.colors

**import** matplotlib.pyplot **as** plt

**from** PIL **import** Image

**import** os

**def** *show\_accuracy*(a, b, tip):

acc = a.ravel() == b.ravel()

**print**(tip + '正确率：%.2f%%' % (100\*np.mean(acc)))

**def** *save\_image*(im, i):

im \*= 15.9375

im = 255 - im

a = im.astype(np.uint8)

output\_path = '.\\HandWritten'

**if** **not** os.path.exists(output\_path):

os.mkdir(output\_path)

Image.fromarray(a).save(output\_path + ('\\%d.png' % i))

**if** \_\_name\_\_ == "\_\_main\_\_":

**print**('Load Training File Start...')

data = np.loadtxt('14.optdigits.tra', dtype=np.float, delimiter=',')

x, y = np.split(data, (-1, ), axis=1)

images = x.reshape(-1, 8, 8)

y = y.ravel().astype(np.**int**)

**print**('Load Test Data Start...')

data = np.loadtxt('14.optdigits.tes', dtype=np.float, delimiter=',')

x\_test, y\_test = np.split(data, (-1, ), axis=1)

images\_test = x\_test.reshape(-1, 8, 8)

y\_test = y\_test.ravel().astype(np.**int**)

**print**('Load Data OK...')

*# x, x\_test, y, y\_test = train\_test\_split(x, y, random\_state=1)*

*# images = x.reshape(-1, 8, 8)*

*# images\_test = x\_test.reshape(-1, 8, 8)*

matplotlib.rcParams['font.sans-serif'] = [u'SimHei']

matplotlib.rcParams['axes.unicode\_minus'] = **False**

plt.figure(figsize=(15, 9), facecolor='w')

**for** index, image **in** enumerate(images[:16]):

plt.subplot(4, 8, index + 1)

plt.imshow(image, cmap=plt.cm.gray\_r, interpolation='nearest')

plt.title(u'训练图片: %i' % y[index])

**for** index, image **in** enumerate(images\_test[:16]):

plt.subplot(4, 8, index + 17)

plt.imshow(image, cmap=plt.cm.gray\_r, interpolation='nearest')

save\_image(image.copy(), index)

plt.title(u'测试图片: %i' % y\_test[index])

plt.tight\_layout()

plt.show()

clf = svm.SVC(C=1, kernel='rbf', gamma=0.001) *# ~ kNN*

**print**('Start Learning...')

clf.fit(x, y)

**print**('Learning is OK...')

y\_hat = clf.predict(x)

show\_accuracy(y, y\_hat, '训练集')

y\_hat = clf.predict(x\_test)

**print**(y\_hat)

**print**(y\_test)

show\_accuracy(y\_test, y\_hat, '测试集')

err\_images = images\_test[y\_test != y\_hat]

err\_y\_hat = y\_hat[y\_test != y\_hat]

err\_y = y\_test[y\_test != y\_hat]

**print**(err\_y\_hat)

**print**(err\_y)

plt.figure(figsize=(10, 8), facecolor='w')

**for** index, image **in** enumerate(err\_images):

**if** index >= 12:

**break**

plt.subplot(3, 4, index + 1)

plt.imshow(image, cmap=plt.cm.gray\_r, interpolation='nearest')

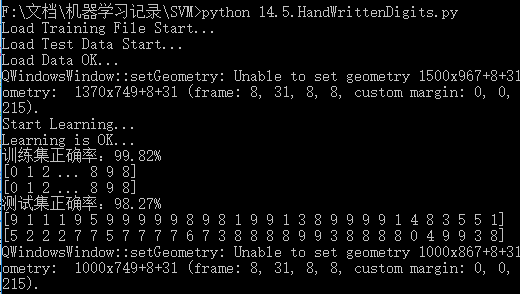
plt.title(u'错分为：%i，真实值：%i' % (err\_y\_hat[index], err\_y[index]))

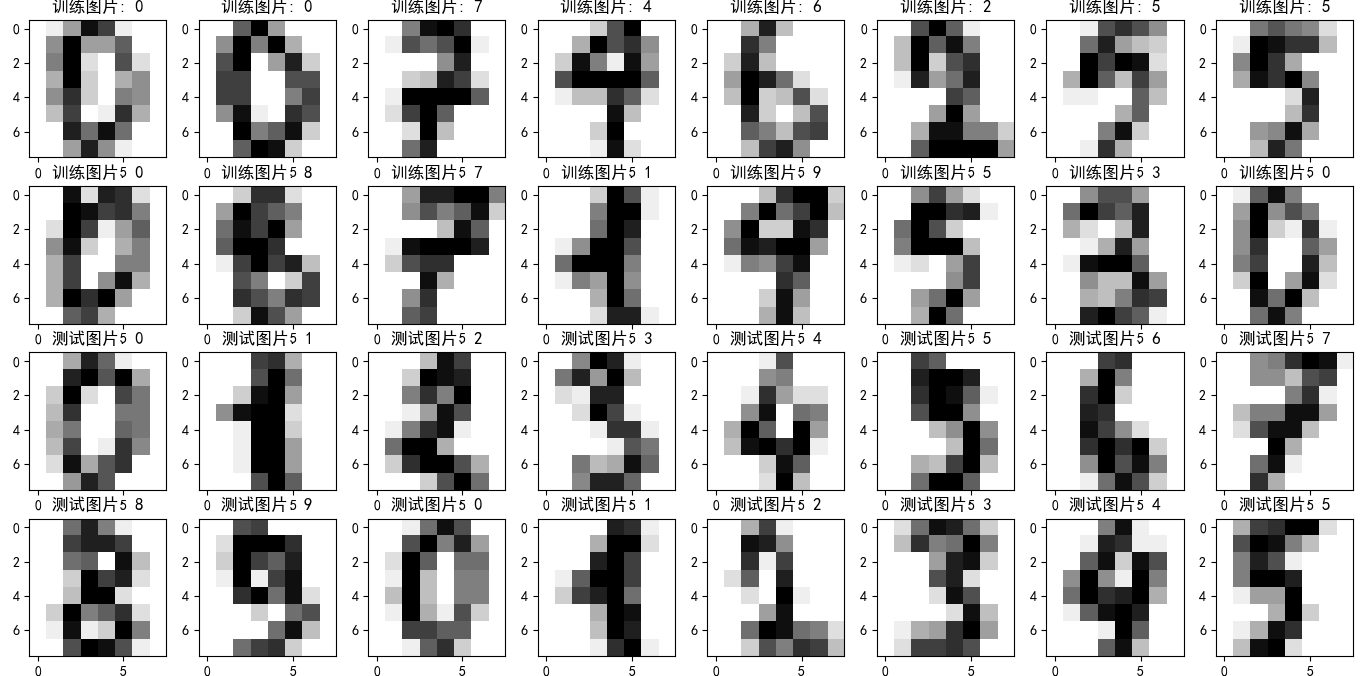
plt.tight\_layout()

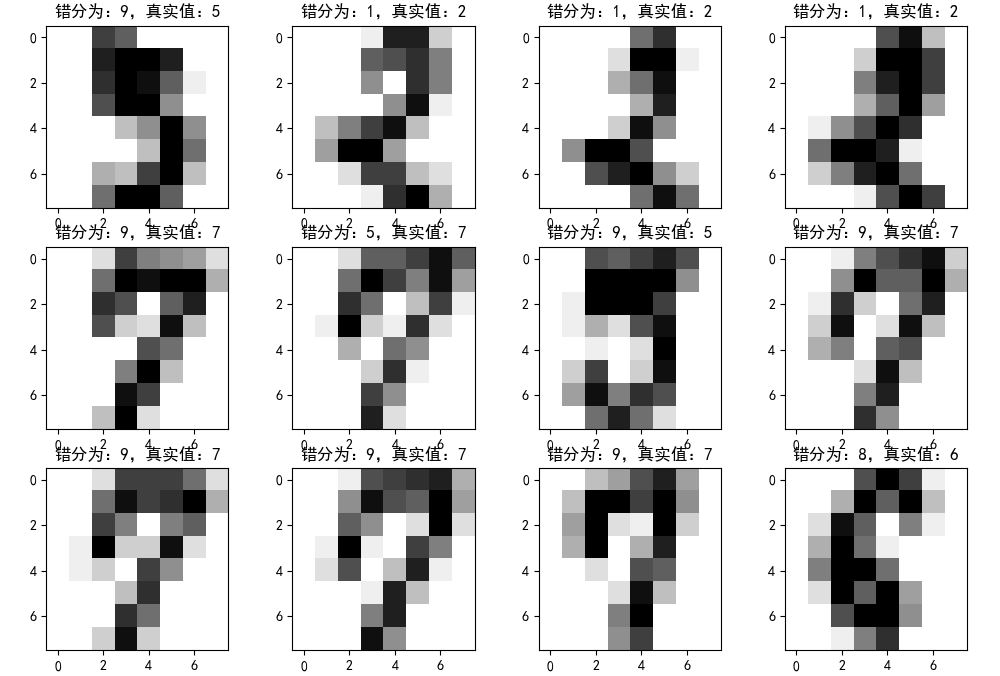
plt.show()

数据集：

结果：







## 6.SVM的推广

用支持向量(support vector)做回归问题------（SVR）

代码：

*#!/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**import** matplotlib.pyplot **as** plt

**if** \_\_name\_\_ == "\_\_main\_\_":

N = 50

np.random.seed(0)

x = np.sort(np.random.uniform(0, 6, N), axis=0)

y = 2\*np.sin(x) + 0.1\*np.random.randn(N)

x = x.reshape(-1, 1)

**print**('x =\n', x)

**print**('y =\n', y)

**print**('SVR - RBF')

svr\_rbf = svm.SVR(kernel='rbf', gamma=0.2, C=100)

svr\_rbf.fit(x, y)

**print**('SVR - Linear')

svr\_linear = svm.SVR(kernel='linear', C=100)

svr\_linear.fit(x, y)

**print**('SVR - Polynomial')

svr\_poly = svm.SVR(kernel='poly', degree=3, C=100)

svr\_poly.fit(x, y)

**print**('Fit OK.')

*# 思考：系数1.1改成1.5*

x\_test = np.linspace(x.min(), 1.1\*x.max(), 100).reshape(-1, 1)

y\_rbf = svr\_rbf.predict(x\_test)

y\_linear = svr\_linear.predict(x\_test)

y\_poly = svr\_poly.predict(x\_test)

plt.figure(figsize=(9, 8), facecolor='w')

plt.plot(x\_test, y\_rbf, 'r-', linewidth=2, label='RBF Kernel')

plt.plot(x\_test, y\_linear, 'g-', linewidth=2, label='Linear Kernel')

plt.plot(x\_test, y\_poly, 'b-', linewidth=2, label='Polynomial Kernel')

plt.plot(x, y, 'mo', markersize=6)

plt.scatter(x[svr\_rbf.support\_], y[svr\_rbf.support\_], s=130, c='r', marker='\*', label='RBF Support Vectors')

plt.legend(loc='lower left')

plt.title('SVR', fontsize=16)

plt.xlabel('X')

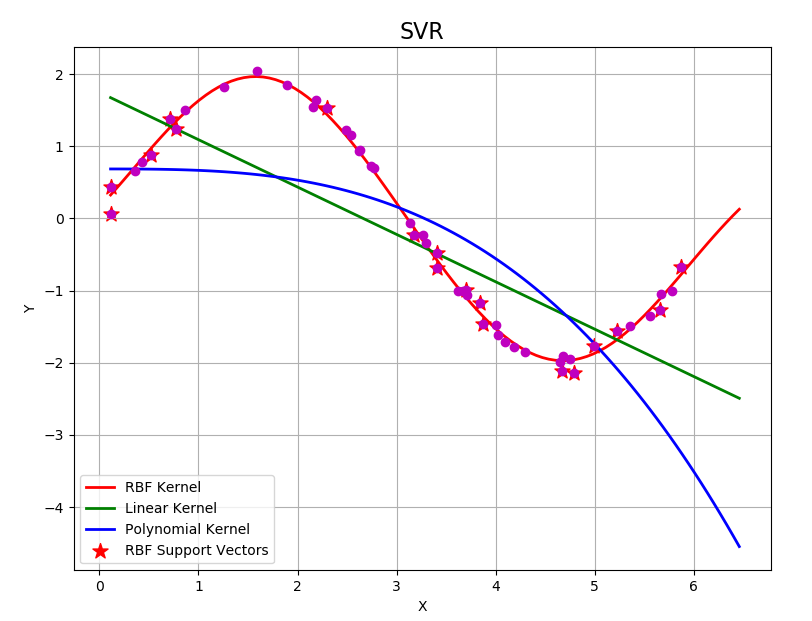
plt.ylabel('Y')

plt.grid(**True**)

plt.show()

数据集：

结果：



## 7.交叉验证（cross validation）

CV----自动搜寻最优参数

代码：

*# !/usr/bin/python*

*# -\*- coding:utf-8 -\*-*

**import** numpy **as** np

**from** sklearn **import** svm

**from** sklearn.model\_selection **import** GridSearchCV *# 0.17 grid\_search*

**import** matplotlib.pyplot **as** plt

**if** \_\_name\_\_ == "\_\_main\_\_":

N = 50

np.random.seed(0)

x = np.sort(np.random.uniform(0, 6, N), axis=0)

y = 2\*np.sin(x) + 0.1\*np.random.randn(N)

x = x.reshape(-1, 1)

**print**('x =\n', x)

**print**('y =\n', y)

model = svm.SVR(kernel='rbf')

c\_can = np.logspace(-2, 2, 10)

gamma\_can = np.logspace(-2, 2, 10)

svr = GridSearchCV(model, param\_grid={'C': c\_can, 'gamma': gamma\_can}, cv=5)

svr.fit(x, y)

**print**('验证参数：\n', svr.best\_params\_)

x\_test = np.linspace(x.min(), x.max(), 100).reshape(-1, 1)

y\_hat = svr.predict(x\_test)

sp = svr.best\_estimator\_.support\_

plt.figure(facecolor='w')

plt.scatter(x[sp], y[sp], s=120, c='r', marker='\*', label='Support Vectors', zorder=3)

plt.plot(x\_test, y\_hat, 'r-', linewidth=2, label='RBF Kernel')

plt.plot(x, y, 'go', markersize=5)

plt.legend(loc='upper right')

plt.title('SVR', fontsize=16)

plt.xlabel('X')

plt.ylabel('Y')

plt.grid(**True**)

plt.show()

数据集：

结果：



