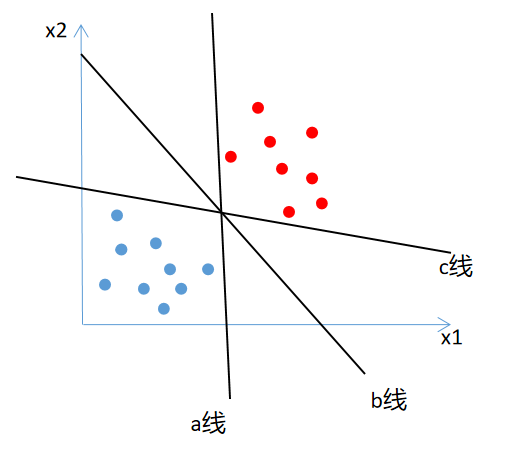
# 支持向量机SVM

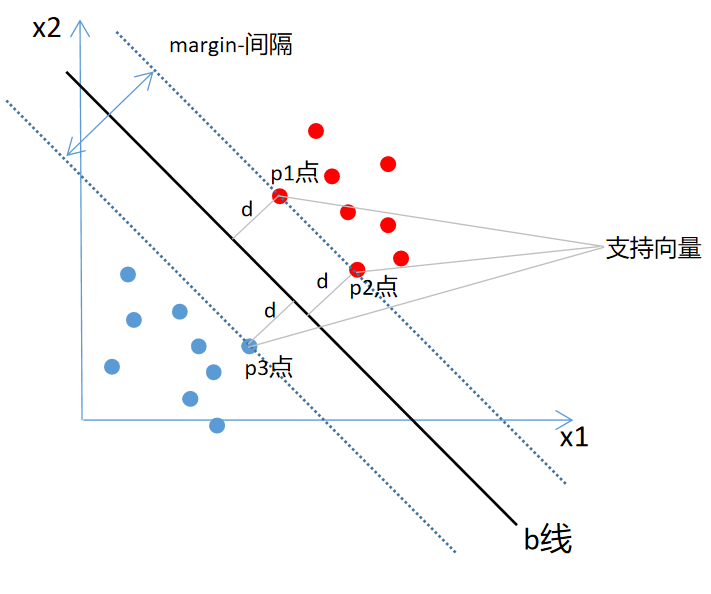
# （support vector machine）

## 支持向量机SVM

对两类样本点进行分类，如下图，有a线、b线、c线三条线都可以将两类样本点很好的分开类，我们可以观察到b线将两类样本点分类最好，原因是我们训练出来的分类模型主要应用到未知样本中，虽然a、b、c三条线将训练集都很好的分开类，但是当三个模型应用到新样本中时，b线抗干扰能力最强，也就是泛化能力最好，样本变化一些关系不大，一样能被正确的分类。那么如何确定b线的位置呢？我们可以使用支持向量机SVM来确定b线的位置。



支持向量机（support vector machines,SVM）是一种二分类算法，它的目的是寻找一个超平面来对样本进行分割，分割的原则是间隔最大化，如果对应的样本特征少，一个普通的SVM就是一条线将样本分隔开，但是要求线到两个类别最近样本点的距离要最大。

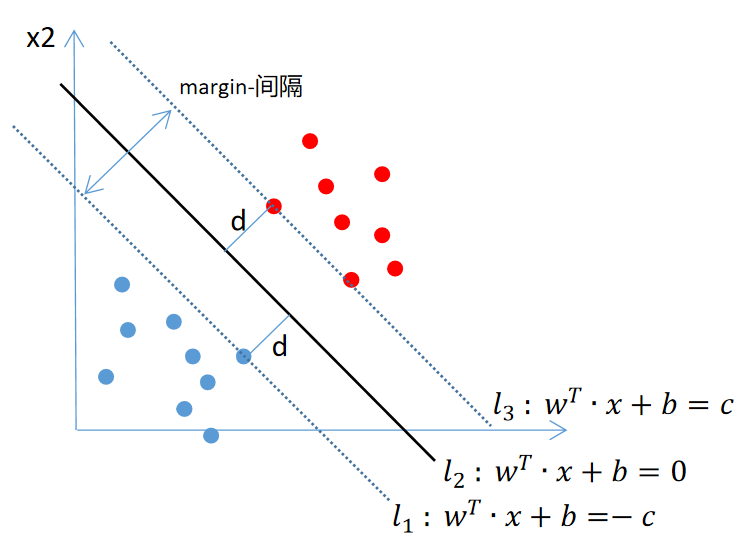


如上图所示，b线就是我们根据支持向量机要找到的分割线，这条线需要到两个类别最近的样本点最远，图上的距离就是d，由于这条线处于两个类别的中间，所以到两个类别的距离都是d。

支持向量机SVM算法就是在寻找一个最优的决策边界（上图中的两条虚线）来确定分类线b，所说的支持向量（support vector）就是距离决策边界最近的点（上图中p1、p2、p3点，只不过边界穿过了这些点）。如果没有这些支持向量点，b线的位置就要改变，所以**SVM就是根据这些支持向量点来最大化margin，来找到了最优的分类线（machine，分类器）**，这也是SVM分类算法名称的由来。

## 构建SVM目标函数

接着上面的分类问题来分析，假设支持向量机最终要找的线是，决策边界两条线是和，那么假设的方程为，这里w表示，x表示，我们想要确定直线，只需要确定w和b即可，此外，由于和线是的决策分类边界线，一定与是平行的，平行就意味着斜率不变，b变化即可，所以我们可以假设线的方程为线的方程为。



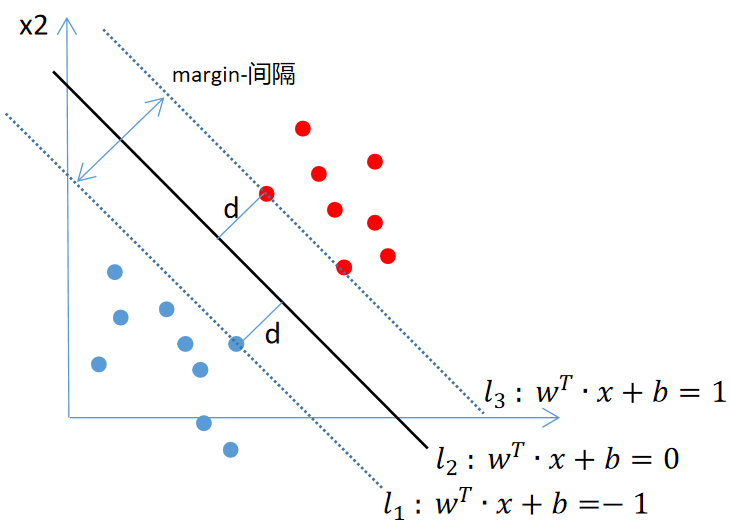
【知识补充】：

* **二维空间点到直线的距离公式为：**
* **两条平行线与之间的距离公式为：**

我们希望的是决策边界上的样本点到直线的距离d越大越好。我们可以直接计算上的点到直线的距离，假设将空间点扩展到n维空间后，点到直线（严格意义上来说直线变成了超平面）的距离为：  
  
以上，读做“w的模”。

由于、和三条直线平行，d也是两条平行线之间的距离，我们可以根据两条平行线之间的距离公式得到d值如下：

我们可以看到无论计算点到直线的距离还是两个平行线之间的距离，最终结果是一致的。对于直线，我们可以相对应成比例的缩小各项的系数，例如变成，这条直线还是原来本身的直线，现在我们可以把它变成，即改写后的直线还是原来的直线，同理，对于直线我们也可以变换成，所以上图可以改变成如下：



那么变换之后的d可以根据两条平行线之间的距离得到：

我们希望决策边界上的点到超平面距离d越大越好。我们将上方的点分类标记为“1”类别，将下方的点分类标记为“-1”类别，那么我们可以得到如下公式：  
  
两式两侧分别乘以对应的类别可以合并为一个式子：  
所以在计算d的最大值时，是有限制条件的，这个限制条件就是。

所以对于所有样本点我们需要在满足条件下，最大化，就是最小化，等效于最小化，这样转换的目的就是为了后期优化过程中求导时方便操作，不会影响最终的结果，所以在支持向量机SVM中要确定一个分类线，我们要做的就是在条件下最小化，记为:

以上n代表样本总数，缩写表示“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题，同时也是支持向量机的基本数学模型，这里我们称支持向量机的目标函数。

## 拉格朗日乘数法、KKT条件、对偶（扩展）

### 拉格朗日乘数法

**拉格朗日乘数法主要是将有等式约束条件优化问题转换为无约束优化问题**，拉格朗日乘数法如下：

假设是一个n维向量，和含有x的函数，我们需要找到满足条件下最小值，如下：

可以引入一个自变量，可以取任意值，则上式严格等价于下式：

叫做Lagrange函数（拉格朗日函数），叫做拉格朗日乘子（其实就是系数）。令对x的导数为0，对的导数为0，求解出 的值，那么x就是函数在附加条件下极值点。

以上就是拉格朗日乘数法，通俗理解**拉格朗日乘数法就是将含有等式条件约束优化问题转换成了无约束优化问题**构造出拉格朗日函数，让对未知数x和进行求导，得到一组方程式，可以计算出对应的x和结果，这个x对应的函数值就是在条件下的最小值点。

拉格朗日函数转换成等式过程的证明涉及到导数积分等数学知识，这里不再证明。

参考文章：

https://blog.csdn.net/Soft\_Po/article/details/118332454

### KKT条件

假设我们面对的是不等式条件约束优化问题，如下：

假设是一个n维向量，和含有x的函数，我们需要找到满足条件下最小值，如下：

针对上式，显然是一个不等式约束最优化问题，不能再使用拉格朗日乘数法，因为拉格朗日乘数法是针对等式约束最优化问题。

那我们可以考虑加入一个“松弛变量”让条件来达到等式的效果，即使条件变成，这里加上的原因是保证加的一定是非负数，即：，但是目前不知道这个值是多少，一定会找到一个合适的值使成立，所以将①式改写成如下：

以上②式变成了等式条件约束优化问题，我们就可以使用拉格朗日乘数法来求解满足等式条件下最小值对应的x值。

通过拉格朗日乘数法，将②式转换为如下：

**但是上式中值必须满足，由于变成了有条件的拉格朗日函数，这里需要求对应下的x值**。至于为什么及为什么计算的是下对应的x值，这里证明涉及到很多几何性质，这里也不再证明。参考文章：https://blog.csdn.net/Soft\_Po/article/details/118358564

所以将上式转换成如下拉格朗日函数：

以上我们可以看到将不等式条件约束优化问题加入“松弛变量”转换成等式条件约束优化问题，使用拉格朗日乘数法进行转换得到，我们先抛开min，下面我们可以对拉格朗日函数对各个未知数进行求导，让对应的导数为零，得到以下联立方程：

从上式⑥式可知，我们分两种情况讨论：

由于，根据③式可知，约束不起作用，根据①式可知。

由于，根据③式可知。由于，约束条件起作用，根据⑤式可知，。

综上两个步骤我们可以得到，且在**约束条件起作用**时，；约束条件不起作用时，。

上面方程组中的⑥式可以改写成。由于，所以将⑤式也可以改写成，所以上面方程组也可以转换成如下：

**以上便是不等式约束优化问题的KKT（Karush-Kuhn-Tucker）条件**，我们回到最开始要处理的问题上，根据③式可知，我们需要找到合适的值使最小，但是合适的必须满足KKT条件。

我们对③式的优化问题可以进行优化如下：

由于且，一定大于零，所以进一步可以简化为如下：

满足最小化⑧式对应的值一定也要满足KKT条件，假设现在我们找到了合适的参数值使取得最小值P**【注意：这里根据①式来说，计算的最小值，这里假设合适参数值对应的的值P就是最小值，不存在比这更小的值。】**，由于⑧式中一定小于等于零，所以一定有：

也就是说一定有：

为了找到最优的值，我们一定想要接近P，即找到合适的最大化，可以写成，所以⑧式求解合适的值最终可以转换成如下：

### 对偶问题

对偶问题是我们定义的一种问题，对于一个不等式约束的原问题：

我们定义对偶问题为（对上面方程的求解等效求解下面方程）：

其实就是把min和max对调了一下，当然对应的变量也要变换。

对偶问题有什么好处呢？对于原问题，我们要先求里面的max，再求外面的min。而对于对偶问题，我们可以先求里面的min。有时候，先确定里面关于x的函数最小值，比原问题先求解关于的最大值，要更加容易解。

但是原问题跟对偶问题并不是等价的，这里有一个强对偶性、弱对偶性的概念，弱对偶性是对于所有的对偶问题都有的一个性质。

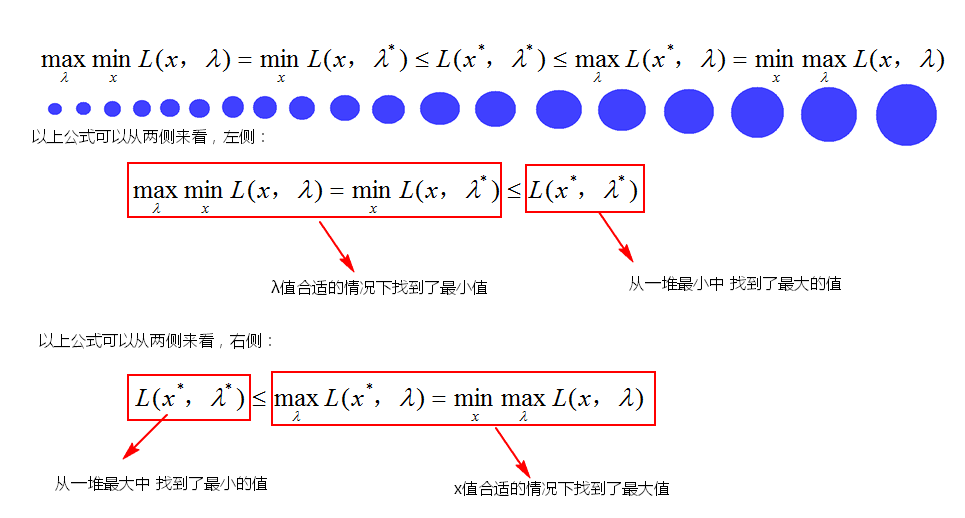
这里给出一个弱对偶性的推导过程:

其中 是函数取最大值最小值的时候对应的最优解，也就是说，原问题始终大于等于对偶问题:

如果两个问题是强对偶的，那么这两个问题其实是等价的问题:

所有的下凸函数都满足强对偶性，也就是说所有下凸函数都满足上式。

【备注】：对弱对偶性推导的理解如下图所示：



## 最小化SVM目标函数

通过1.2小节我们知道SVM目标函数如下：

根据拉格朗日乘数法、KKT条件、对偶问题我们可以按照如下步骤来计算SVM目标函数最优的一组w值。

### 构造拉格朗日函数

将SVM目标函数转换为如下:

根据1.3.2中的⑨式构建拉格朗日函数如下：

以上不等式转换成拉格朗日函数必须满足KKT条件，详见1.3.2 KKT条件，这里满足的KKT条件如下：

### 对偶转换

由于原始目标函数是个下凸函数，根据1.3.3中对偶问题可知一定是强对偶问题，所以可以将a式改写成如下：

针对b式，我们假设参数固定，使对参数w和b进行求导得到如下：

进一步可以得到：

按照解方程组的思想，我们现在将以上计算得到的结果代入到b式中得到：

即：

我们可以发现以上公式经过转换只含有的方程求极值问题，但是含有这些未知数（注意不是2个），那么我们要求的一组w如何得到呢？只要针对以上公式得到值后，我们可以根据c式进一步求解到对应的一组w值。针对上式求极值问题，我们常用 SMO(Sequential Minimal Optimization) 算法求解。

注意：d式中代表两个向量的点积，也叫内积，例如：那么两个向量点积结果为。

### SMO算法求解（了解）

SMO(Sequential Minimal Optimization)，序列最小优化算法，其核心思想非常简单：每次只优化一个参数，其他参数先固定住，仅求当前这个优化参数的极值。下面我们只说SVM中如何利用SMO算法进行求解最终结果的，这里不再进行公式推导。

我们根据d式可以看到有 很多参数，由于优化目标中含有约束条件：，此约束条件中，代表每个样本所属类别-1或者1。

根据SMO算法假设将参数固定，那么之间的关系也确定了，可以将约束条件：改写成  
  
我们可以将上式中c看成是乘以对应的类别得到的常数，根据e式我们发现之间有等式关系，即：，也就是说我们可以使用的表达式代替，这样我们就可以将d式看成只含有这一个条件下最优化问题，并且我们可以将代入到d式，d式中只含有一个未知数求极值问题，可以对函数进行求导，令导数为零，得到的值，然后根据的值计算出的值。通过以上固定参数方式多次迭代计算得到一组值。

### 计算分割线w和b的值

根据1.4.2中的c式，如下：

我们可以知道获取了一组的值，我们可以得到一组w对应的值。根据1.4.1中的KKT条件：

根据f式中式可知，当时，一定有，对于这样的样本就属于支持向量点，就在分类边界上，我们将对应的w代入到下式中  
  
一定可以计算出b的值。我们将g式两边乘以，得到  
  
由于，所以得到最终b的结果如下：

假设我们有S个支持向量，以上b的计算可以使用任意一个支持向量代入计算即可，如果数据严格是线性可分的，这些b结果是一致的。对于数据不是严格线性可分的情况，参照后面的软间隔问题，一般我们可以采用一种更健壮的方式，将所有支持向量对应的b都求出，然后将其平均值作为最后的结果，最终b的结果如下：

S代表支持向量点的集合，也就是在边界上点的集合。

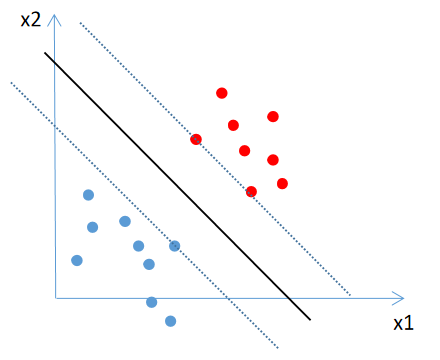
综上所述，我们可以的到最终的w和b的值如下：

确定w和b之后，我们就能构造出最大分割超平面：，新来样本对应的特征代入后得到的结果如果是大于0那么属于一类，小于0属于另一类。

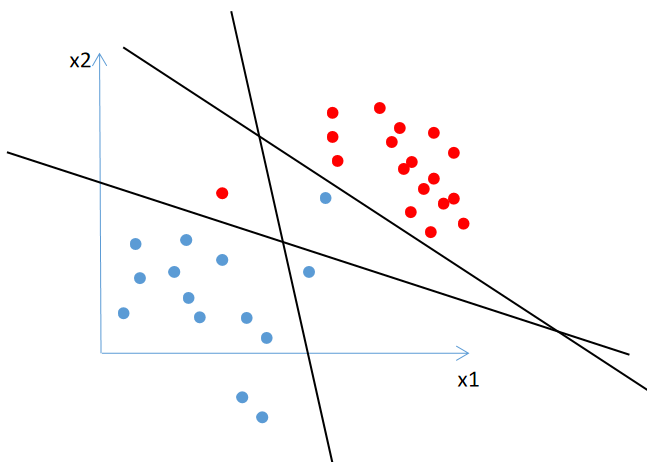
## 软间隔及优化

### 软间隔问题

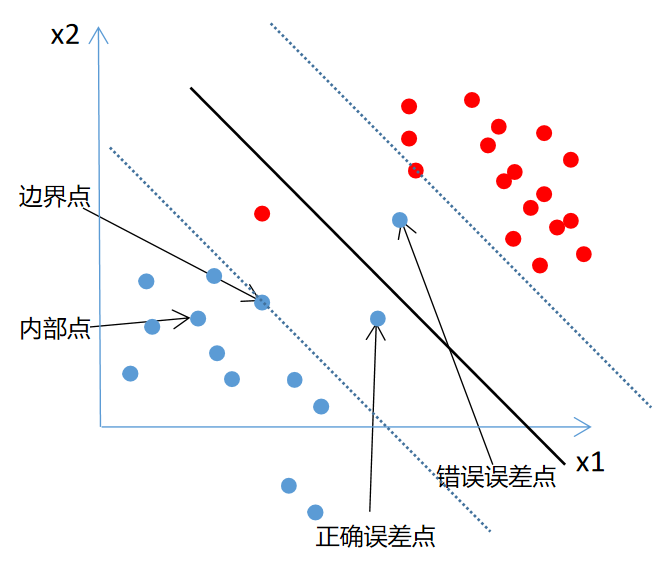
以上讨论问题都是基于样本点完全的线性可分，我们称为硬间隔如下图：



如果存在部分样本点不能完全线性可分，大部分样本点线性可分的情况，如下图所示：

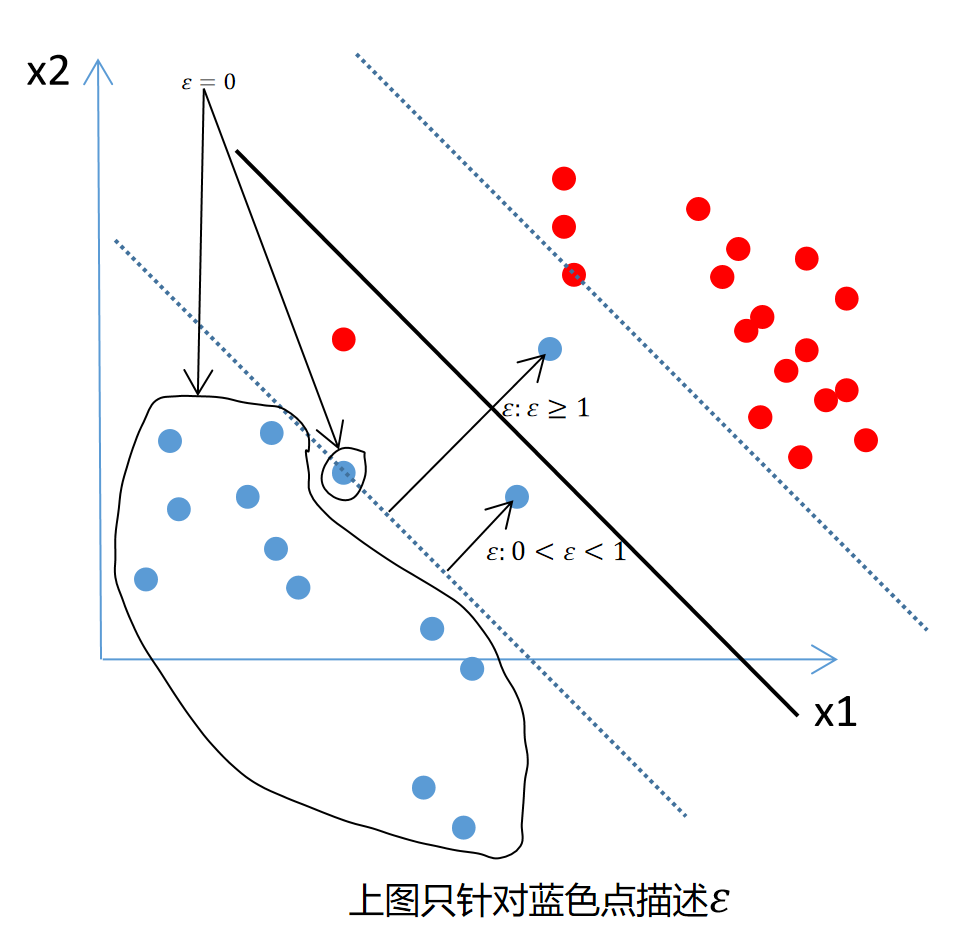


那么我们就需要用到软间隔，相比于硬间隔的苛刻条件，我们允许个别样本点出现在间隔带里面，比如：



即我们允许部分样本点不满足约束条件：

为了度量这个间隔软到何种程度，我们为每个样本引入一个“**松弛变量**”，（中文：克西）加入松弛变量后，我们的约束条件变成，这样不等式就不会那么严格，数据点在不同的约束条件下的情况如下：



* 内部点：,即
* 边界点：,即
* 正确误差点：,即
* 错误误差点：,即

### 优化SVM目标函数

加入软间隔后我们的目标函数变成了：

其中C是一个大于0的常数，这里在原有的目标函数中加入可以理解为错误样本的惩罚程度，我们希望在对应条件下变小，对应的也是希望变小，当C为无穷大，必然无穷小，这样的话SVM就变成一个完全线性可分的SVM。如果C为有对应的值时，对应的会有一个大于0的值，这样的SVM就是允许部分样本不遵守约束条件。

接下来我们对新的目标函数h式求解最优化问题，步骤与1.4 SVM“硬间隔”完全线性可分步骤完全一样。

1. **构造拉格朗日函数**

将h式目标函数改写成如下：

构造拉格朗日函数如下：

上式中是拉格朗日乘子，是我们要计算的主问题参数。

1. **对偶关系转换**

目标函数是个下凸函数，对应的拉格朗日函数复合强对偶性，所以可以根据强对偶向，将对偶问题转换为：

即：使用j式对求导，并令导数为零，得到如下关系：

整理之后得到：  
  
将以上关系代入到j式：

我们发现上式中求与没有关系，但是为什么可以消掉是因为，所以在约束条件中一定有此条件，所以我们得到最终的目标函数可以写成如下：

推导到以上之后，我们发现结果和硬间隔结果一样，参照1.4.3 d式，只是结果中多了约束条件。

1. **求解结果**

同样我们也可以利用SMO算法对式**L**求解得到一组合适的值和值，由于综上所述，我们可以的到最终的w和b的值如下：

这里得到的w和b虽然和硬分割结果一样，但是这是加入松弛变量之后得到的w和b的值，根据**L**式的条件和，结合h式，C越大，必然导致松弛越小，如果C无穷大，那么就意味着模型过拟合，在训练SVM时，C是我们需要调节的参数。

那么确定w和b之后，我们就能构造出最大分割超平面：，新来样本对应的特征代入后得到的结果如果是大于0那么属于一类，小于0属于另一类。

### SVM代码

SVM代码如下：

|  |
| --- |
| **from** sklearn **import** svm **import** numpy **as** np **def** read\_data(path):  **with** open(path) **as** f :  lines=f.readlines()  lines=[eval(line.strip()) **for** line **in** lines]  X,y=zip(\*lines)  X=np.array(X)  y=np.array(y)  **return** X,y  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  X\_train,y\_train=read\_data(**"./data/train\_data"**)  X\_test,y\_test=read\_data(**"./data/test\_data"**)   *#C对样本错误的容忍,这里就是SVM中加入松弛变量对应的C，* model= svm.SVC(C=1.0)  model.fit(X\_train,y\_train.ravel())  *#打印支持向量* print(model.support\_vectors\_)  *#打印支持向量下标* print(model.support\_)   *#使用SVM应用到测试集，返回正确率* score = model.score(X\_test,y\_test)  print(score) |

### 网格搜索

网格搜索就是手动给定模型中想要改动的所有参数，程序自动使用穷举法来将所有的参数或者参数组合都运行一遍，将对应得到的模型做K折交叉验证，将得分最高的参数或参数组合当做最佳结果，并得到模型。也就是说网格搜索用来获取最合适的一组参数，不需要我们手动一个个尝试。使用网格搜索由于针对每个参数下的模型需要做K折交叉验证最终导致加大了模型训练时间，所以一般我们可以在小的训练集中使用网格搜索找到对应合适的参数值，再将合适的参数使用到大量数据集中训练模型。例如:逻辑回归中使用正则项时，我们可以先用一小部分训练集确定合适的惩罚系数，然后将惩罚系数应用到大量训练集中训练模型。

代码如下：

|  |
| --- |
| **from** sklearn **import** svm **import** numpy **as** np **from** sklearn.model\_selection **import** GridSearchCV **def** read\_data(path):  **with** open(path) **as** f :  lines=f.readlines()  lines=[eval(line.strip()) **for** line **in** lines]  X,y=zip(\*lines)  X=np.array(X)  y=np.array(y)  **return** X,y  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  X\_train,y\_train=read\_data(**"./data/train\_data"**)  X\_test,y\_test=read\_data(**"./data/test\_data"**)   *#np.logspace(0,5,6) 创建等比数列，logspace(开始值，结束值，数据个数)  #np.logspace(0,5,6)得到的就是10^0,10^1,10^2,10^3,10^4,10^5* search\_space = {**'C'**: np.logspace(-3, 3, 7)}  print(search\_space[**'C'**])  model= svm.SVC()  *#网格搜索 ,cv的值就是指定K折交叉验证中的k* gridsearch = GridSearchCV(model, param\_grid=search\_space,cv=5)  gridsearch.fit(X\_train,y\_train.ravel())  *#交叉验证中测试的最好的结果* cv\_performance = gridsearch.best\_score\_  print(**"交叉验证最好结果："**,cv\_performance)  test\_performance = gridsearch.score(X\_test, y\_test)  *#交叉验证中测试的最好的参数* print(**"C"**,gridsearch.best\_params\_[**'C'**]) |

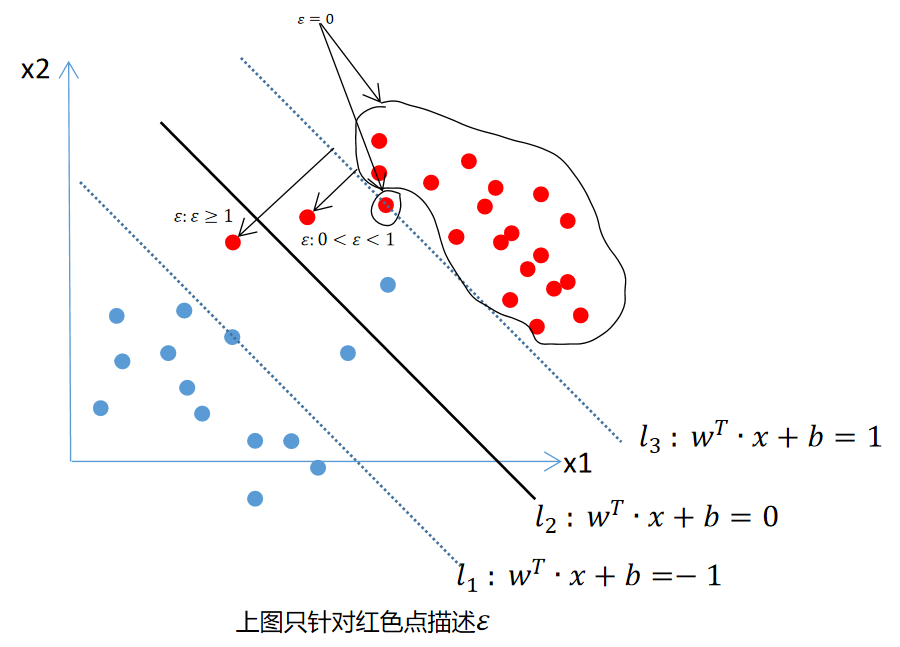
## SVM Hinge Loss 损失函数（了解）

软间隔情况下，SVM的目标函数如下:

其实这里我们说SVM的损失主要说的就是上式中的松弛变量损失，当所有样本总的为0时那么就是一个完全线性可分的SVM；如果SVM不是线性可分，那么我们希望总体样本的松弛变量越小越好。

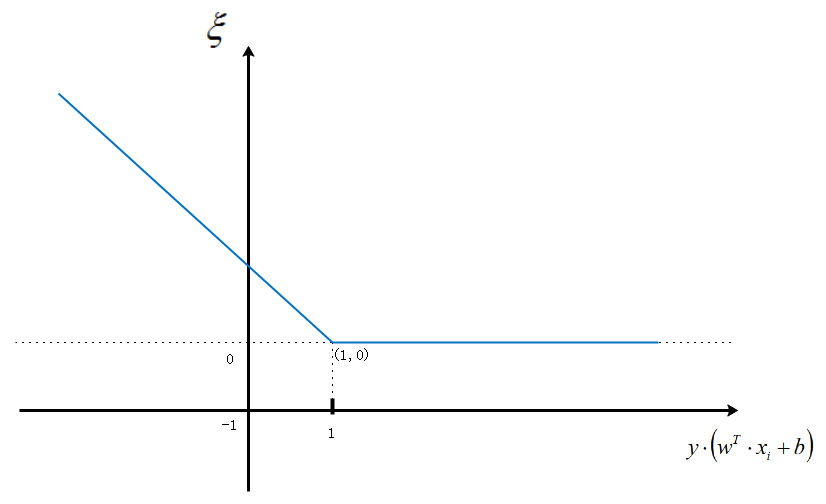
根据h式的条件我们可以改写成如下：

根据下图结合以上两个条件我们可以继续改写成如下：



上式等价于:

以上公式就是SVM的损失函数，称为“Hinge Loss”，也被叫做“合页损失函数”。其图像如下：



特点是当时损失为0。

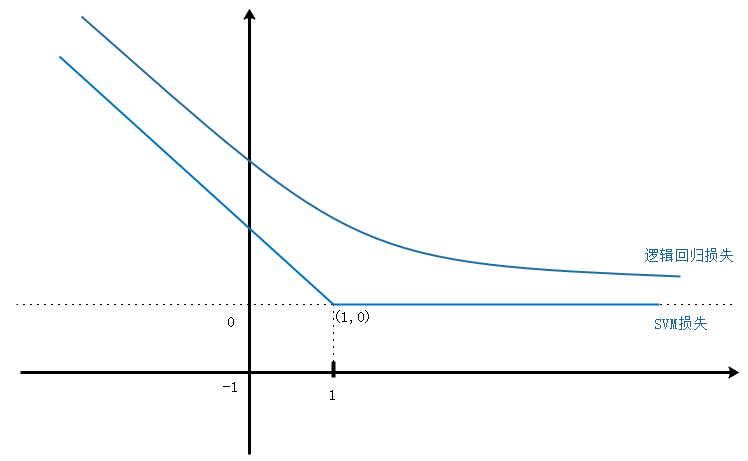
我们可以将m式代入到h式得到新的目标函数，暂时先不关注约束条件，新的目标函数如下：

针对新的目标函数我们同样也是最小化整体目标函数。我们发现以上目标函数与逻辑回归L2正则化的损失函数在形式上非常类似，将上式可以改写成如下：

逻辑回归L2正则化损失函数如下：

也可以改写成如下：

如果将逻辑损失函数图像与“Hinge Loss”图像放在一起图像如下：

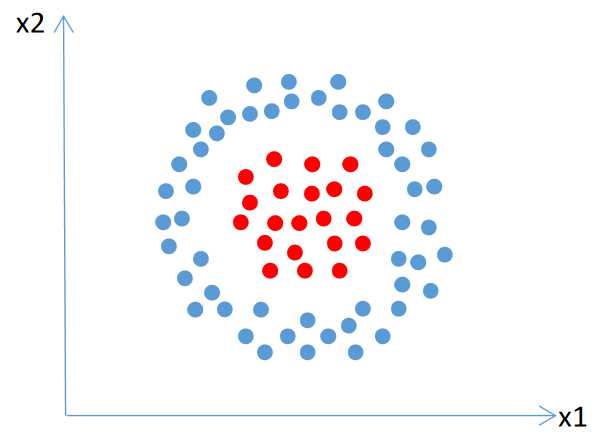


通过上图我们可以看出，在训练逻辑回归时，损失永远不会为0，因此即使分类正确，逻辑回归仍然会不停的训练，以求得更小的损失和更大的概率，但是极有可能出现过拟合，这就是为什么我们需要在梯度下降中提前终止的原因。但是在SVM中，如果分类正确可以直接达到，不再进行对分类无益的训练，即学习到一定程度就不再学习。

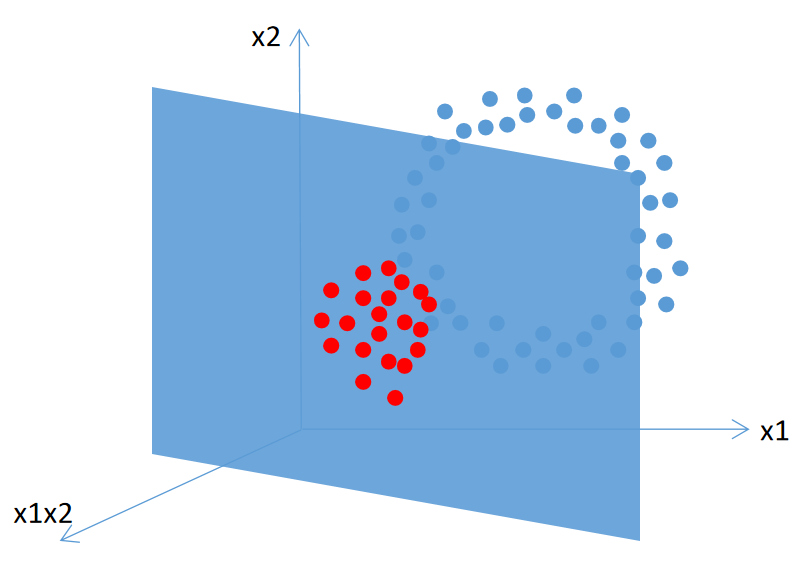
## 核函数

### 非线性SVM

在前面SVM讨论的硬间隔和软间隔都是指的样本完全可分或者大部分样本点线性可分的情况。那么如果样本完全线性不可分如下图所示：



我们可以使用升维方式来解决这种线性不可分的问题，例如目前只有两个维度，我们可以基于已有的维度进行升维得到更多维度，例如：，这样我们可以将以上问题可以转换成高维可分问题，如下图：



SVM中线性不可分问题就是非线性SVM问题，对这种问题我们可以将样本映射到高维空间，再通过间隔最大化的方式学习得到支持向量机。

### 核函数（Kernel Function）

假设原有特征有如下三个，那么基于三个已有特征进行两两组合（二阶交叉）我们可以得到如下更多特征：  
  
我们在训练模型时将以上组合特征可以参与到模型的训练中，其代价是运算量过大，比如原始特征有m个，那么二阶交叉的维度为个，这种量级在实际应用中很难实现，那么有没有一种方式可以在升维的同时可以有效的降低运算量，在非线性SVM中，核函数就可以实现在升维的同时可以有效降低运算量，那么核函数是什么？

有两个向量，我们分别对两个向量进行相同升维分别记为：

之后计算两者内积如下：

另外我们计算m和n的内积如下：

对结果进行平方得到：

【备注】：

我们发现o式和p式结果一样，也就是  
  
以上在m和n向量中只有3个元素时推导成立，那么在m和n中如果有多个元素时也一样成立。

我们发现，如果最终目标只是计算向量升维后的内积，根据q式可知，我们可以先计算原始向量的内积，再进行平方运算，而不需要真正的进行高运算量的升维操作。这种通过两个低维向量进行数学运算得到它们投影至高维空间时向量的内积方法叫做核方法，相应的算法叫做核函数，上面案例核函数记为：

以上核函数叫做多项式核函数，常用的核函数有如下几种，不同的核函数的区别是将向量映射到高维空间时采用的升维方法不同，不过高维向量都是不需要计算出来的。

* 线性核函数：
* 多项式核函数：
* 高斯核函数：

### 核函数在SVM中的应用

在非线性SVM中，我们可以对原始特征进行升维，原有的超平面变成了，根据l式SVM目标函数：

可知最终经过对偶处理后的目标函数如下：

以上公式与l式不同的是只是将转换成了，这样我们可以将上式进而转换成下式：

这样我们可以通过核函数将数据映射到高维空间，但是核函数是在低维上进行计算，而将实质上的分类效果表现在了高维空间上，来解决在原始空间中线性不可分的问题。

在SVM中我们可以使用常用的核函数：线性核函数、多项式核函数、高斯核函数来进行升维解决线性不可分问题，应用最广泛的就是高斯核函数，无论是小样本还是大样本、高维或者低维，高斯核函数均适用，它相比于线性核函数来说，线性核函数只是高斯核函数的一个特例，线性核函数适用于数据本身线性可分，通过线性核函数来达到更理想情况。高斯核函数相比于多项式核函数，多项式核函数阶数比较高时，内部计算时会导致一些元素值无穷大或者无穷小，而在高斯核函数会减少数值的计算困难。

综上，我们在非线性SVM中使用核函数时，先使用线性核函数，如果不行尝试换不同的特征，如果还不行那么可以直接使用高斯核函数。

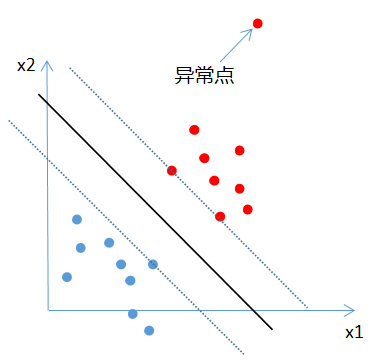
非线性SVM使用核函数代码如下：

|  |
| --- |
| **from** sklearn **import** svm **from** sklearn **import** datasets **from** sklearn.model\_selection **import** train\_test\_split **as** ts **''' ‘rbf’：高斯核函数 ‘linear’:线性核函数 ‘poly’：多项式核函数 '''  if** \_\_name\_\_ == **'\_\_main\_\_'**:  *#import our data* iris = datasets.load\_iris()  X = iris.data  y = iris.target   *#split the data to 7:3* X\_train,X\_test,y\_train,y\_test = ts(X,y,test\_size=0.3)  print(y\_test)  *# select different type of kernel function and compare the score   # kernel = 'rbf'* clf\_rbf = svm.SVC(kernel=**'rbf'**)  clf\_rbf.fit(X\_train,y\_train)  score\_rbf = clf\_rbf.score(X\_test,y\_test)  print(**"The score of rbf is : %f"**%score\_rbf)   *# kernel = 'linear'* clf\_linear = svm.SVC(kernel=**'linear'**)  clf\_linear.fit(X\_train,y\_train)  score\_linear = clf\_linear.score(X\_test,y\_test)  print(**"The score of linear is : %f"**%score\_linear)   *# kernel = 'poly'* clf\_poly = svm.SVC(kernel=**'poly'**)  clf\_poly.fit(X\_train,y\_train)  score\_poly = clf\_poly.score(X\_test,y\_test)  print(**"The score of poly is : %f"**%score\_poly) |

## 支持向量机SVM特点

### 抗干扰能力强

根据一些样本点，我们可以通过拉格朗日乘数法、KKT条件、对偶问题、SMO算法计算出支持向量机SVM的分类线,其中  
  
S代表的是边界上的样本点。得到的分割下如下图所示：



我们可以看出SVM在训练过程中找到的是两类点的分割线，计算样本中有少量异常点，在训练模型时依然能很正确的找到中间分割线，因为训练SVM时考虑了全量数据，确定这条线的b时只与支持向量点（边界上的点）有关。同样在测试集中就算有一些样本点是异常点也不会影响其正常分类，SVM具有抗干扰能力强的特点。

此外，由于训练SVM需要所有样本点参与模型训练，不然不能缺点这条线，所以当数据量大时，SVM训练占用内存非常大。这时SVM模型的缺点。

### 二分类及多分类

SVM同样支持多分类，我们可以将一个多分类问题拆分成多个二分类问题，例如有A,B,C三类，我们使用SVM训练时可以训练3个模型，第一个模型针对是A类和不是A类进行训练。第二个模型针对是B类和不是B类进行训练。第三个模型针对是C类和不是C类进行训练。这样可以解决多分类问题。