SVM入门

北邮计算机学院

传统的机器学习、统计的机器学习、结构风险、小样本、VC维

SVM的特点

SVM简介

- 支持向量机(Support Vector Machine)是Cortes和Vapnik于1995年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。
- 支持向量机方法是建立在统计学习理论的VC 维理论和结构风险最小原理基础上的，根据有限的样本信息在模型的复杂性（即对特定训练样本的学习精度，Accuracy）和学习能力（即无错误地识别任意样本的能力）之间寻求最佳折衷，以期获得最好的推广能力或称泛化能力）。

统计的机器学习和传统的机器学习

- 统计机器学习能够精确的给出学习效果，能够解答需要的样本数等等一系列问题。与统计机器学习的精密思维相比，传统的机器学习基本上属于摸着石头过河，用传统的机器学习方法构造分类系统完全成了一种技巧，一个人做的结果可能很好，另一个人差不多的方法做出来却很差，缺乏指导和原则。

什么是VC

- 所谓VC维是对函数类的一种度量，可以简单的理解为问题的复杂程度，VC维越高，一个问题就越复杂。正是因为SVM关注的是VC维，后面我们可以看到，SVM解决问题的时候，和样本的维数是无关的（甚至样本是上万维的都可以，这使得SVM很适合用来解决文本分类的问题，当然，有这样的能力也因为引入了核函数）。

经验风险

- 机器学习本质上就是一种对问题真实模型的逼近（我们选择一个我们认为比较好的近似模型，这个近似模型就叫做一个假设），真实模型一定是不知道的。既然真实模型不知道，那么我们选择的假设与问题真实解之间究竟有多大差距，我们就没法得知。
- 这个与问题真实解之间的误差，就叫做风险（更严格的说，误差的累积叫做风险）。我们选择了一个假设之后（更直观点说，我们得到了一个分类器以后），真实误差无从得知，但我们可以用某些可以掌握的量来逼近它。最直观的想法就是使用分类器在样本数据上的分类的结果与真实结果（因为样本是已经标注过的数据，是准确的数据）之间的差值来表示。这个差值叫做经验风险Remp(w)。

经验风险最小化

- 传统的机器学习：**基于经验风险最小化，但是泛化能力非常差。**
- 原因：选择了一个足够复杂的分类函数（它的VC维很高），能够精确的记住每一个样本，但对样本之外的数据一律分类错误。
- 经验风险最小化原则：适用的大前提是经验风险要确实能够逼近真实风险才行，但实际上能逼近么？答案是不能，因为样本数相对于现实世界要分类的文本数来说简直九牛一毛，经验风险最小化原则只在这占很小比例的样本上做到没有误差，当然不能保证在更大比例的真实文本上也没有误差。

泛化误差界

- 统计学习因此而引入了泛化误差界的概念，就是指真实风险应该由两部分内容刻画：
  - 一是经验风险，代表了分类器在给定样本上的误差；
  - 二是置信风险，代表了我们在多大程度上可以信任分类器在未知文本上分类的结果。
    - 很显然，第二部分是没有办法精确计算的，因此只能给出一个估计的区间，也使得整个误差只能计算上界，而无法计算准确的值（所以叫做泛化误差界，而不叫泛化误差）。

置信风险->结构风险

- 置信风险与两个量有关：
  - 一是样本数量，显然给定的样本数量越大，我们的学习结果越有可能正确，此时置信风险越小；
  - 二是分类函数的VC维，显然VC维越大，推广能力越差，置信风险会变大。
- 泛化误差界的公式为：
- $R(w) \leq Remp(w) + \Phi(n/h)$
  - 公式中R(w)就是真实风险，Remp(w)就是经验风险，Φ(n/h)就是置信风险。
  - **统计学习的目标从经验风险最小化变为了寻求经验风险与置信风险的和最小，即结构风险最小。**

SVM的特点：

- 基于最小化结构风险的算法；
- 小样本：
  - 并不是说样本的绝对数量少（实际上，对任何算法来说，更多的样本几乎总是能带来更好的效果），而是说与问题的复杂度比起来，SVM算法要求的样本数是相对比较少的。
- 非线性：
  - 是指SVM擅长应付样本数据线性不可分的情况，主要通过松弛变量（也有人叫惩罚变量）和核函数技术来实现，这一部分是SVM的精髓，

SVM与文本分类

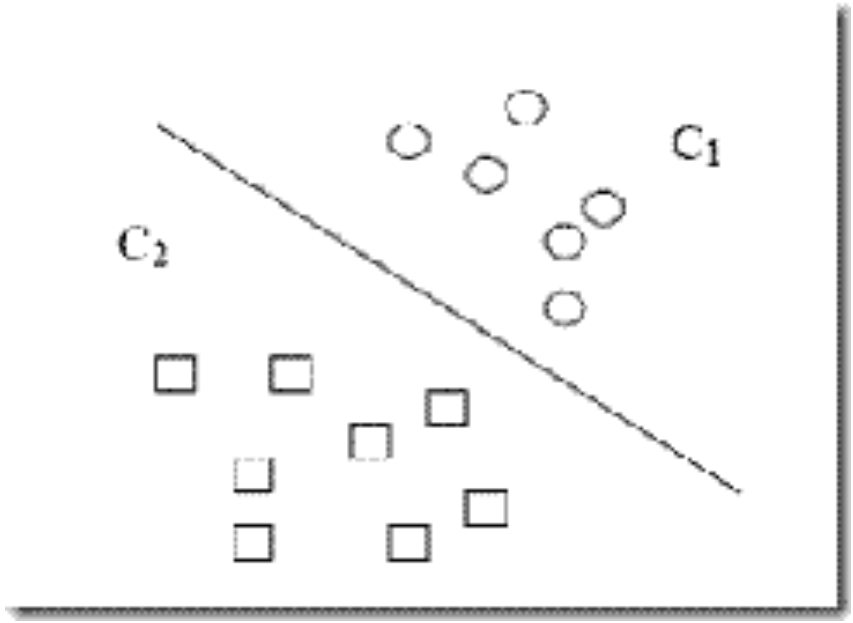
- 高维模式识别是指样本维数很高，例如文本的向量表示，如果没有经过降维处理，出现几万维的情况很正常，其他算法基本就没有能力应付了，SVM却可以，主要是因为SVM 产生的分类器很简洁，用到的样本信息很少（仅仅用到那些称之为“支持向量”的样本，此为后话），使得即使样本维数很高，也不会给存储和计算带来大麻烦（相对照而言，kNN算法在分类时就要用到所有样本，样本数巨大，每个样本维数再一高，这日子就没法过了.....）。

线性分类器(一定意义上,也可以叫做感知机) 是最简单也很有效的分类器形式.在一个线性分类器中,可以看到SVM形成的思路,并接触很多SVM的核心概念.

PART 1: 线性分类器

1. 线性可分

- C1和C2是要区分的两个类别，在二维平面中它们的样本如上图所示。中间的直线就是一个分类函数，它可以将两类样本完全分开。
- **一般的，如果一个线性函数能够将样本完全正确的分开，就称这些数据是线性可分的，否则称为非线性可分的。**
- 中间的分类函数不是唯一的。



2. 线性函数和超平面

- 线性函数：在一维空间里就是一个点，在二维空间里就是一条直线，三维空间里就是一个平面，可以如此想象下去，如果不关注空间的维数，这种线性函数还有一个统一的名称——**超平面（Hyper Plane）**！
- g(x)=wx+b：X为样本向量
  - 实值函数
  - 阈值
  - 离散型的输出结果

分类间隔：分类器的评价指标

- 文本分类的例子
  - 训练样本Di=(xi,yi)：
    - xi就是文本向量（维数很高），yi就是分类标记。
    - 在二元的线性分类中，这个表示分类的标记只有两个值，1和-1（用来表示属于还是不属于这个类）。
  - 样本点到某个超平面的间隔：
    - δi=yi(wxi+b) = |wxi+b|=|g(xi)|

$$\delta_i = \frac{1}{\|w\|}|g(x_i)|$$

几何间隔：解析几何中点xi到超平面g(xi)=0的欧式距离公式

几何间隔：样本点到超平面的距离

- 一个点的集合（就是一组样本）到某个超平面的距离为此集合中离超平面最近的点的距离。

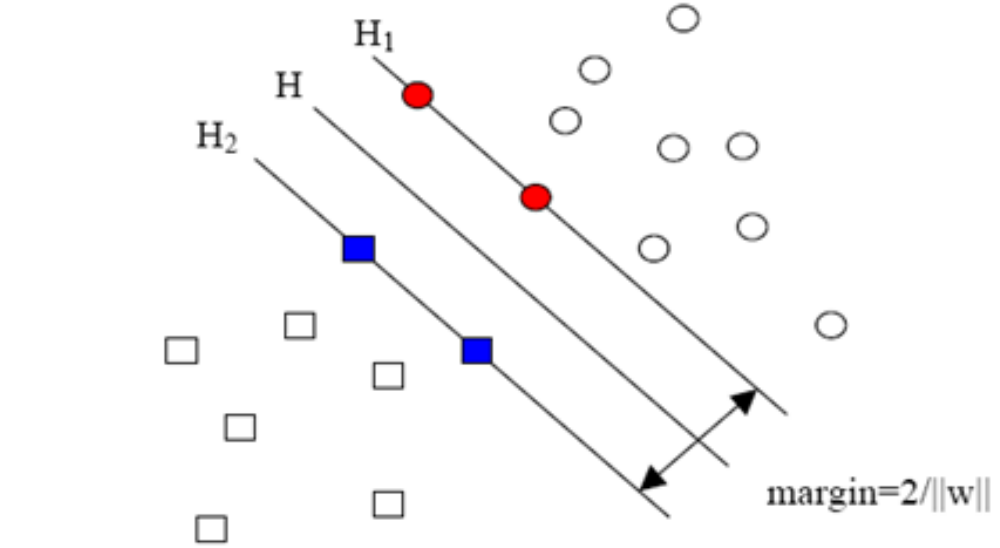


图2　线性可分情况下的最优分类线

误分次数与几何间隔

- 误分次数：代表分类器的误差。误分次数的上界由几何间隔决定！
- 几何间隔δ越大的解，它的误差上界越小。
- 因此最大化几何间隔成了我们训练阶段的目标。

$$\text{误分次数} \le \left(\frac{2R}{\delta}\right)^2$$

  - δ是样本集合到分类面的间隔；
  - R=max ||xi||　i=1,...,n，即R是所有样本中（xi是以向量表示的第i个样本）向量长度最长的值（也就是说代表样本的分布有多么广）

最大化集合间隔：寻找最小的||w||

$$\delta_{几何} = \frac{1}{\|w\|}|g(x)|$$

$$\min \|w\| \qquad \min \frac{1}{2}\|w\|^2$$

$$\min \quad \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i[(wx_i)+b]-1 \geq 0 \quad (i=1,2,\ldots,l) \quad (l\text{是样本数})$$

基于可行域的寻最优解

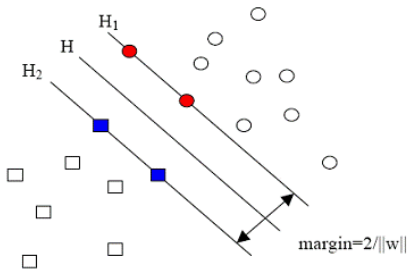


图2 线性可分情况下的最优分类线

线性分类器的求解——问题的描述 Part2

- 一般地，一个求最小值的问题就是一个优化问题（也叫寻优问题，或规划—Programming），它同样由两部分组成，目标函数和约束条件（函数C），可以用下面的式子表示：

$$\begin{aligned} &\min && f(x) \\ &\text{subject to} && c_i(x) \le 0 \quad i=1,2,\ldots,p \\ & && c_j(x) = 0 \quad j=p+1,p+2,\ldots,p+q \end{aligned}$$

基于可行域的寻最优解

支持向量

- 支持向量：可行区域上的边界点
- 凸集：一个点的集合，其中任取两个点连一条直线，这条线上的点仍然在这个集合内部。
- 二次规划：　（能够找到最优解的规划）
  - 自变量就是w，而目标函数是w的二次函数，所有的约束条件都是w的线性函数因此这是一个二次规划的寻优。（Quadratic Programming，QP），
  - 由于它的可行域是一个凸集，因此它是一个凸二次规划。

$$\min \quad \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad y_i[(wx_i)+b]-1 \geq 0 \quad (i=1,2,\ldots,l) \quad (l\text{是样本数})$$

线性分类器的求解

- 问题的描述
  - 圆形的样本点定为正样本），方形的点定为负样例。求线性函数（在n维空间中的线性函数）：
  - **g(x)=wx+b**
  - 所有属于正类的点x_+代入以后有$g(x_+)\geq 1$，而所有属于负类的点x_-代入后有$g(x_-)\leq -1$
  - 代入g(x)后的值如果在1和-1之间，我们就拒绝判断。
  - 求这样的g(x)的过程就是求w（一个n维向量）和b（一个实数）两个参数的过程。

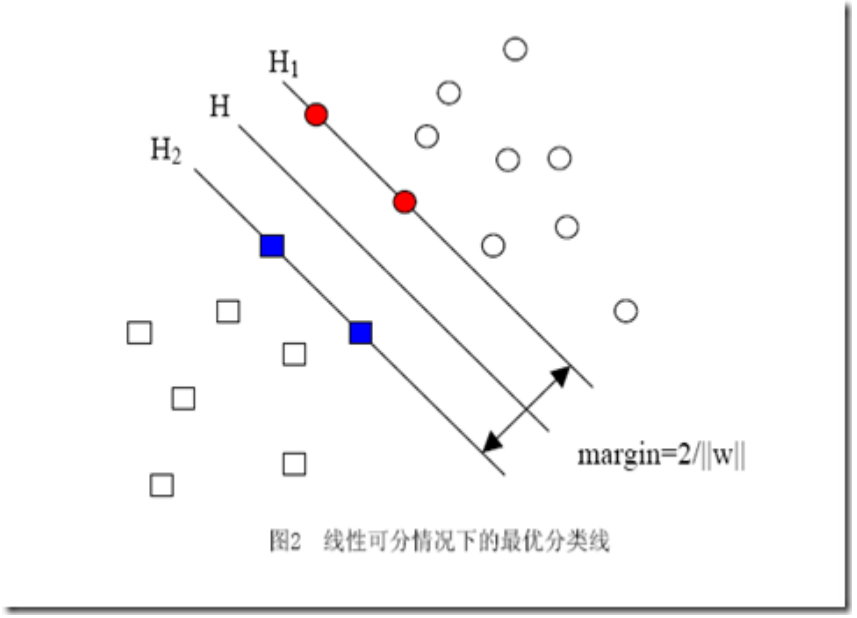


图2 线性可分情况下的最优分类线

求解W

- 样本确定了w，用数学的语言描述，就是w可以表示为样本的某种组合：
- $w=\alpha_1 x_1+\alpha_2 x_2+\ldots+\alpha_n x_n$
  - α被称为拉格朗日乘子
  - $\alpha_1 x_1$表示数和向量的乘积
  - g(x)的表达式严格的形式应该是

    $g(x)=<w,x>+b$

更严格的SVM中的求解定义

- w不仅跟样本点的位置有关，还跟样本的类别有关，因而严格的定义为：

$$w=\alpha_1 y_1 x_1+\alpha_2 y_2 x_2+\ldots+\alpha_n y_n x_n \quad \text{（式1）}$$

- 用求和符号简写：

$$w=\sum_{i=1}^{n}(\alpha_i y_i x_i)$$

$$g(x)=<w,x>+b$$

$$=<\sum_{i=1}^{n}(\alpha_i y_i x_i),x>+b$$

$$g(x) = < w, x > + b$$

$$= < \sum_{i=1}^{n} (\alpha_i y_i x_i), x > + b$$

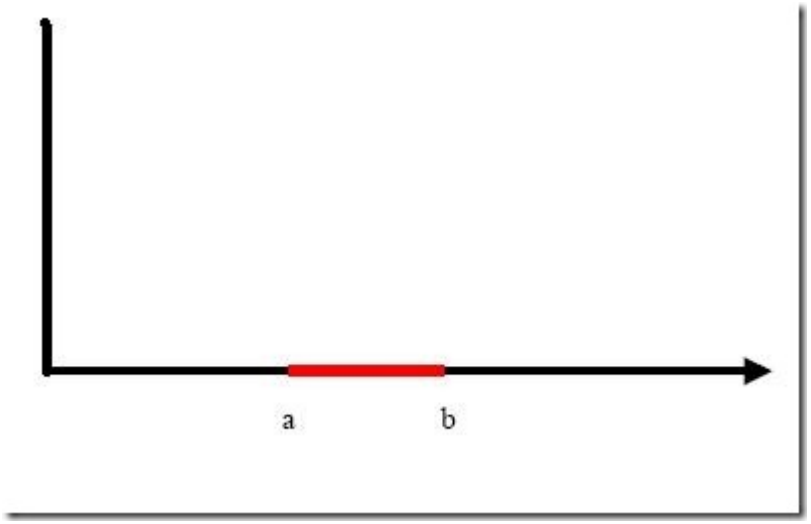
- x是变量，即待分类的文档，而x_i是已知的样本。
- 注意到式子中只有x_i和x是向量，因此，得到g(x)的式子为：

$$g(x) = \sum_{i=1}^{n} \alpha_i y_i < x_i, x > + b \quad （式2）$$

核函数

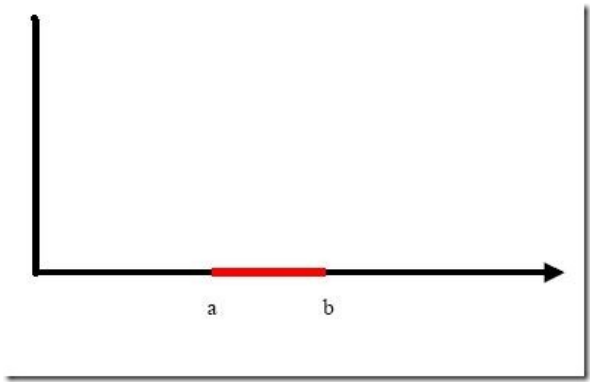
- 线性分类器，只能对线性可分的样本做处理。
- 如果提供的样本线性不可分，结果很简单，线性分类器的求解程序会无限循环，永远也解不出来。
- 这使得它的适用范围大大缩小，而它的很多优点我们实在不原意放弃，怎么办呢？
- **是否有某种方法，让线性不可分的数据变得线性可分呢？**

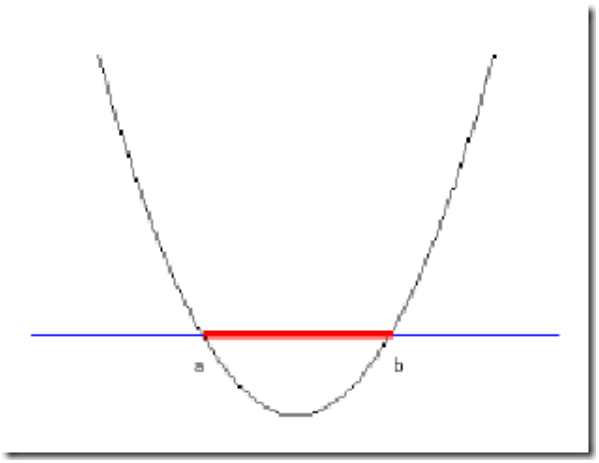
一个线性不可分的例子



- 横轴上端点a和b之间红色部分里的所有点定为正类，两边的黑色部分里的点定为负类。试问能找到一个线性函数把两类正确分开么？
- 不能，因为二维空间里的线性函数就是指直线，显然找不到符合条件的直线。

核函数：升维后的线性可分

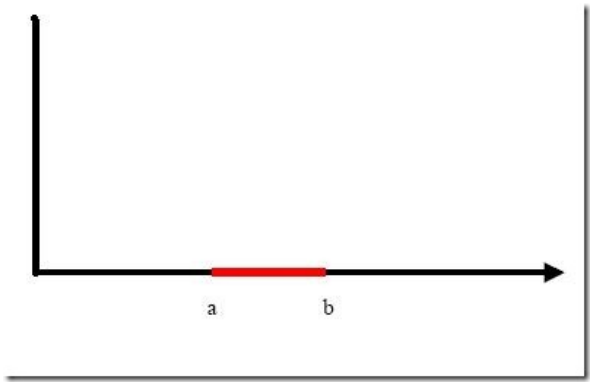


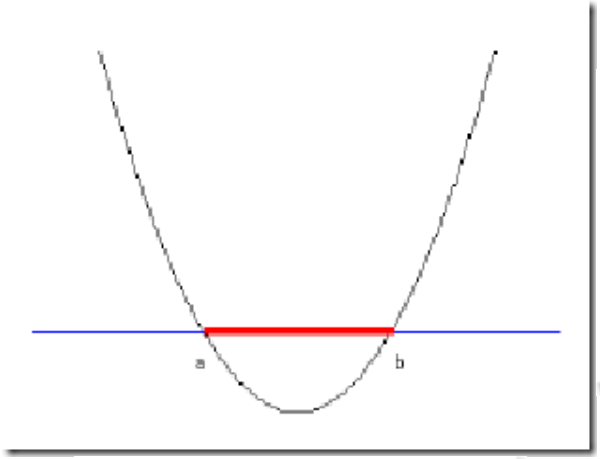


- 通过点在这条曲线的上方还是下方就可以判断点所属的类别，即一条二次曲线：

$$g(x) = c_0 + c_1 x + c_2 x^2$$

核函数：升维后的线性可分





$$g(x) = c_0 + c_1 x + c_2 x^2$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

f(y)=<a,y>

g(x)=f(y)=ay

- 解决线性不可分问题的基本思路——
  - 向高维空间转化，使其变得线性可分。
  - 找到x映射到y的方法

g(x)=K(w,x)+b （低纬空间下将x输入到K核函数中，得到与f(x')一样的值
f(x′)=<w′,x′>+b

核函数：升维后的线性可分

$$g(x) = c_0 + c_1 x + c_2 x^2$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}, \quad a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

f(y)=<a,y>

g(x)=f(y)=ay

- 一个文本分类的例子来看看这种向高维空间映射从而分类的方法如何运作:
  - 文本分类问题的原始空间是1000维的（即每个要被分类的文档被表示为一个1000维的向量），在这个维度上问题是线性不可分的。现在我们有一个2000维空间里的线性函数

f(x′)=<w′,x′>+b

- 它能够将原问题变得可分。式中的 w′和x′都是2000维的向量，w′是定值，而x′是变量，
- 现在输入是一个1000维的向量x，分类的过程是先把x变换为2000维的向量x′，然后求这个变换后的向量x′与向量w′的内积，再把这个内积的值和b相加，就得到了结果，看结果大于阈值还是小于阈值就得到了分类结果。

- 我们其实只关心那个高维空间里内积的值，那个值算出来了，分类结果就算出来了.

- 是否能有这样一种函数K(w,x),他接受低维空间的输入值，却能算出高维空间的内积值 <w′,x′>?
- **g(x)=K(w,x)+b**
  - **低纬空间下将x输入到K核函数中，得到与f(x')一样的值　　f(x′)=<w′,x′>+b**

核函数:核函数的基本作用就是接受两个低维空间里的向量，能够计算出经过某个变换后在高维空间里的向量内积值

原始的经拉格朗日分解的线性分类器

$$f(x') = \sum_{i=1}^{n} \alpha_i y_i < x_i' , x' > + b$$

- 利用核函数，求向量内积

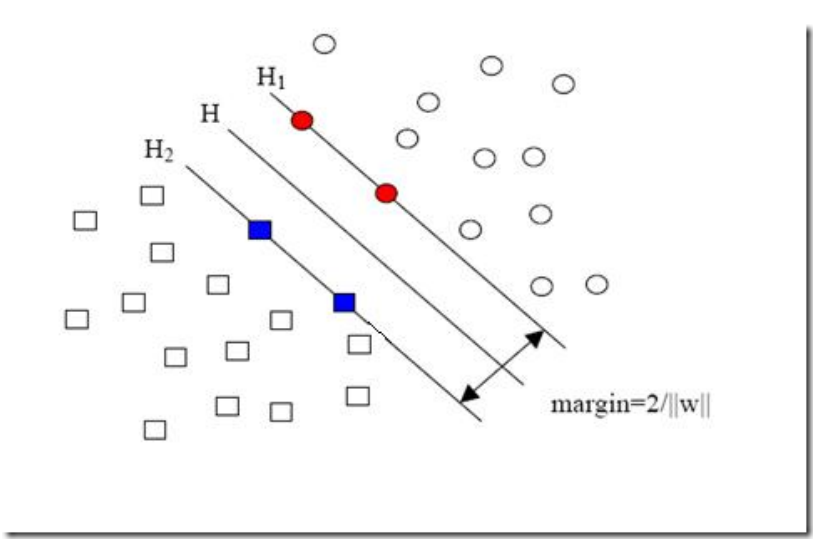
$$g(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b$$

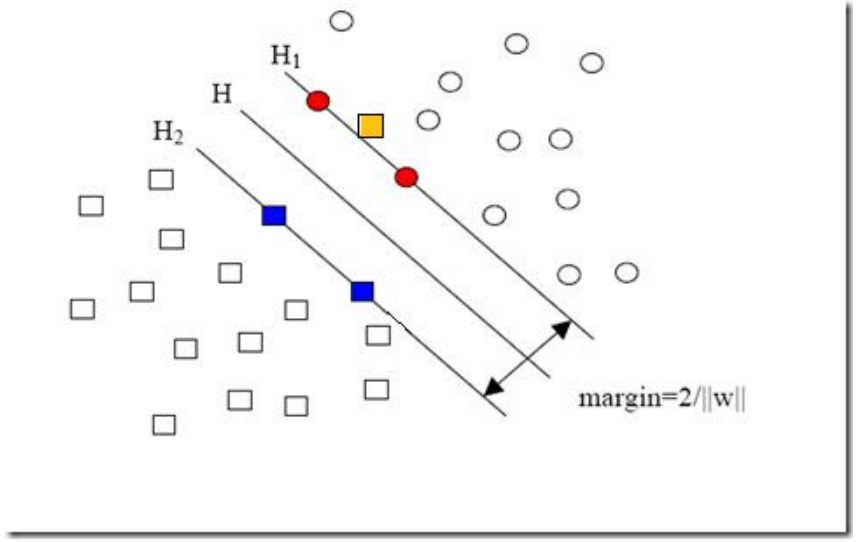
两个问题：

1. 既然有很多的核函数，针对具体问题该怎么选择？

2. 如果使用核函数向高维空间映射后，问题仍然是线性不可分的，那怎么办？

近似线性可分问题

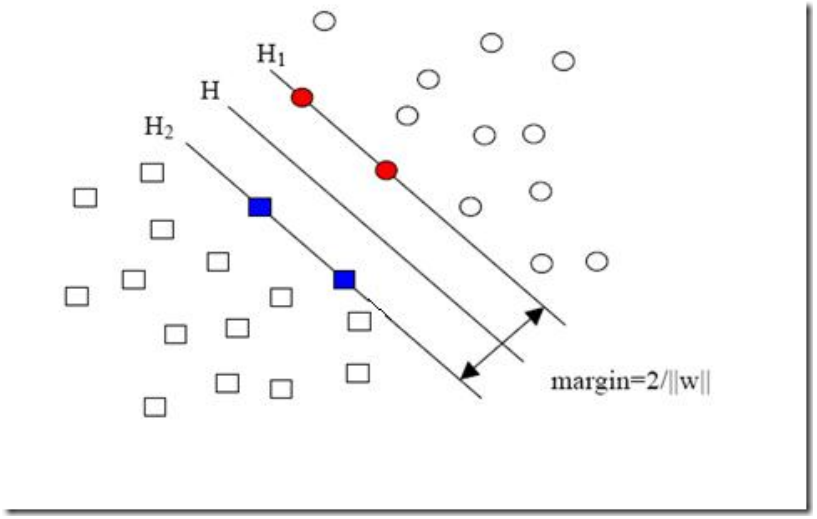


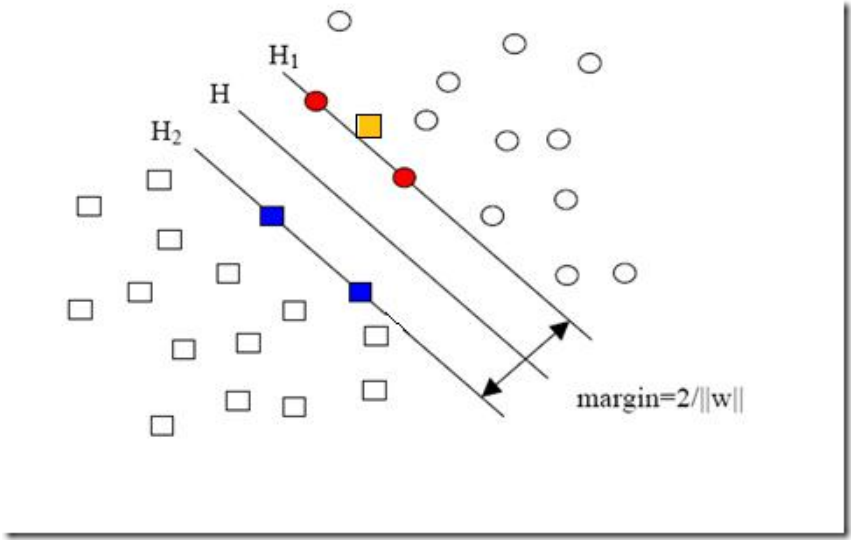


想象我们有另一个训练集，只比原先这个训练集多了一篇文章，映射到高维空间以后（当然，也使用了相同的核函数），也就多了一个样本点，但是这个样本的位置是这样的：

图中黄色那个点，它是方形的，因而它是负类的一个样本，这单独的一个样本，使得原本线性可分的问题变成了线性不可分的。这样类似的问题（仅有少数点线性不可分）叫做“近似线性可分”的问题。

松弛变量-增加计算机的容噪方法





$y_i[(wx_i)+b] \geq 1$ (i=1,2, . . . ,l) (l是样本数)

$y_i[(wx_i)+b] \geq 1-\zeta_i$ (i=1,2, . . . ,l) (l是样本数)

$\zeta_i \geq 0$

$\sum_{i=1}^{l} \zeta_i^2$

$\sum_{i=1}^{l} \zeta_i$

$$\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \zeta_i$$

subject to $y_i[(wx_i)+b] \geq 1-\zeta_i$ (i=1,2, . . . ,l) (l是样本数) (式1)

$\zeta_i \geq 0$

关于松弛变量（1）

- 并非所有的样本点都有一个松弛变量与其对应。
  - 实际上只有“离群点”才有，所有没离群的点松弛变量都等于0
- 松弛变量的值实际上标示出了对应的点到底离群有多远，值越大，点就越远。
- 惩罚因子C决定了你有多重视离群点带来的损失，显然当所有离群点的松弛变量的和一定时，你定的C越大，对目标函数的损失也越大，此时就暗示着你非常不愿意放弃这些离群点，最极端的情况是你把C定为无限大，这样只要稍有一个点离群，目标函数的值马上变成无限大，马上让问题变成无解，这就退化成了硬间隔问题。

关于松弛变量（2）

- 惩罚因子C不是一个变量，整个优化问题在解的时候，C是一个你必须事先指定的值；
- 指定这个值以后，得到一个分类器，然后用测试数据看看结果怎么样，如果不够好，换一个C的值，再解一次优化问题，得到另一个分类器，再看看效果，如此就是一个参数寻优的过程，
- 这与优化问题本身决不是一回事，优化问题在解的过程中，C一直是定值，要记住。

关于松弛变量（3）

- 惩罚因子C不是一个变量，整个优化问题在解的时候，C是一个你必须事先指定的值；
- 指定这个值以后，得到一个分类器，然后用测试数据看看结果怎么样，如果不够好，换一个C的值，再解一次优化问题，得到另一个分类器，再看看效果，如此就是一个参数寻优的过程，
- 这与优化问题本身决不是一回事，优化问题在解的过程中，C一直是定值，要记住。

松弛变量的经验之谈

- 优化问题求解的过程：
  - 就是先试着确定一下w，也就是确定了前面图中的三条直线，这时看看间隔有多大，又有多少点离群，把目标函数的值算一算，再换一组三条直线
  - 分类的直线位置如果移动了，有些原来离群的点会变得不再离群，而有的本来不离群的点会变成离群点），再把目标函数的值算一算，如此往复（迭代），直到最终找到目标函数最小时的w。

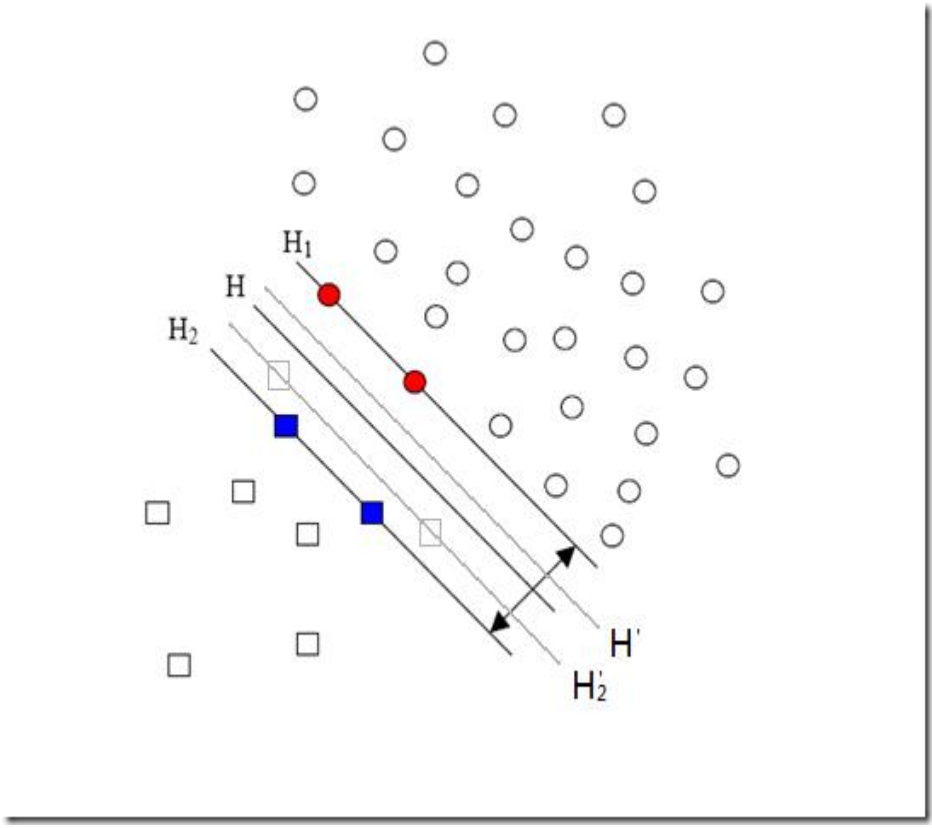
核函数和松弛变量

- 两者都是解决线性不可分的方法。
- 先用核函数：在原始的低维空间中，样本相当的不可分，无论你怎么找分类平面，总会有大量的离群点，此时用核函数向高维空间映射一下，虽然结果仍然是不可分的，但比原始空间里的要更加接近线性可分的状态
- 达到了近似线性可分的状态，此时再用松弛变量处理那些少数“冥顽不化”的离群点

支持向量机就是使用了核函数的软间隔线性分类法

数据偏斜问题

方形的点是负类。H，H_1，H_2是根据给的样本算出来的分类面，由于负类的样本很少很少，所以有一些本来是负类的样本点没有提供，比如图中两个灰色的方形点，如果这两个点有提供的话，那算出来的分类面应该是H’，H_2’和H_1，他们显然和之前的结果有出入，实际上负类给的样本点越多，就越容易出现在灰色点附近的点，我们算出的结果也就越接近于真实的分类面。但现在由于偏斜的现象存在，使得数量多的正类可以把分类面向负类的方向“推”，因而影响了结果的准确性。



数据集偏斜（unbalanced），它指的是参与分类的两个类别（也可以指多个类别）样本数量差异很大

惩罚因子C

$$\min \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \zeta_i$$

$$\text{subject to} \quad y_i[(wx_i)+b] \geq 1-\zeta_i \quad (i=1,2,\ldots,l) \quad (l\text{是样本数}) \quad (\text{式}1)$$

$$\zeta_i \geq 0$$

- C所起的作用：表征你有多么重视离群点，C越大越重视，越不想丢掉它们
- 没有任何规定说必须对所有的松弛变量都使用同一个惩罚因子，我们完全可以给每一个离群点都使用不同的C，这时就意味着你对每个样本的重视程度都不一样

数据偏斜的处理

- 数据集偏斜问题的解决方法之一：使用惩罚因子。
  - 给样本数量少的负类更大的惩罚因子，表示我们重视这部分样本
  - 因此目标函数中因松弛变量而损失的部分就变成了：

$$C_+\sum_{i=1}^{p}\zeta_i + C_-\sum_{j=p+1}^{p+q}\zeta_j$$

$$\zeta_i \geq 0$$

如何确定C_+和C_-

- 它们的大小是试出来的（参数调优），但是他们的比例可以有些方法来确定
  - 按照正负样例的数量比例确定
  - 按照正负样例的分布来确定
    - 计算正负类样本在空间中占据了多大的体积，例如给负类找一个超球——就是高维空间里的球——它可以包含所有负类的样本，再给正类找一个，比比两个球的半径，就可以大致确定分布的情况。
    - 显然半径大的分布就比较广，就给小一点的惩罚因子

SVM的后续

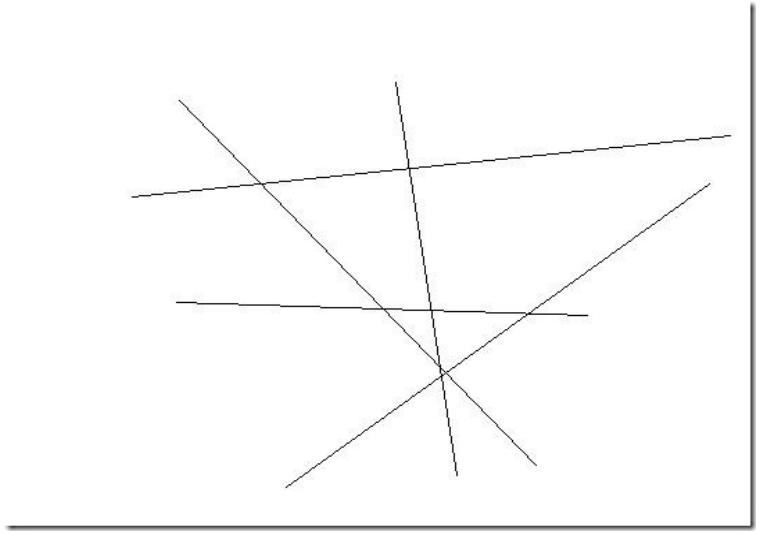
- 如何让二值的SVM完成多类分类任务？
- 如何使用SVM

SVM用于多类分类

- SVM是一种典型的两类分类器，即它只回答属于正类还是负类的问题。而现实中要解决的问题，往往是多类的问题
- 如何由两类分类器得到多类分类器，就是一个值得研究的问题。

多个超平面把空间划分为多个区域，每个区域对应一个类别，给一篇文章，看它落在哪个区域就知道了它的分类。

？只可惜这种算法还基本停留在纸面上，因为一次性求解的方法计算量实在太大，大到无法实用的地步。

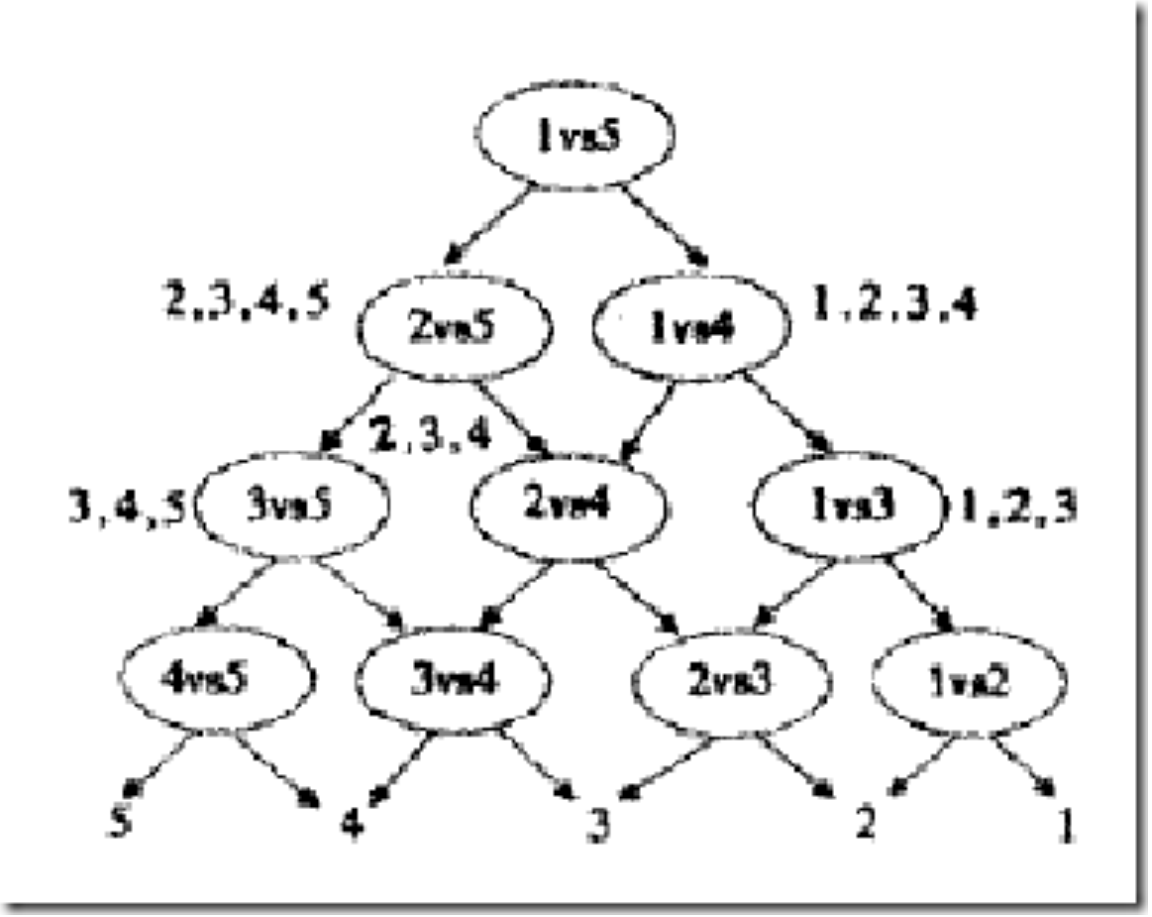


方法

- 1 VS Rest
  - 分类重叠
  - 不可分类
- 1 VS 1
  - 分类器的数量：如果有k个类别，则总的两类分类器数目为k(k-1)/2
  - 有分类重叠现象，但不出现不可分类
  - 训练时间短，但测试时间需要调用所有分类器

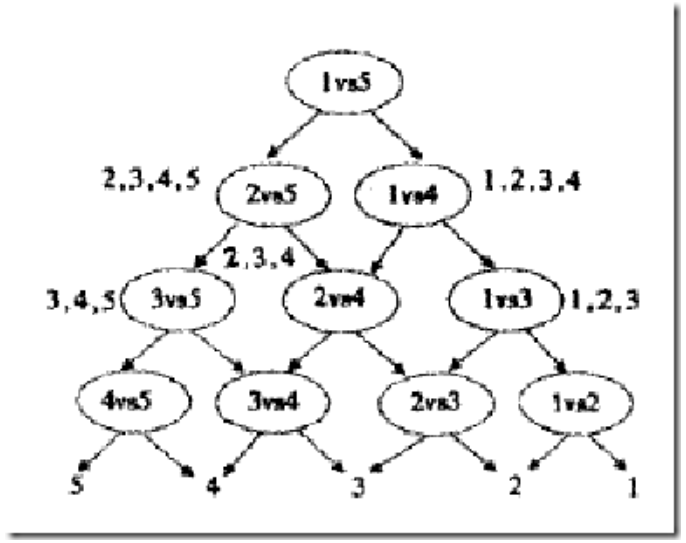
DAG SVM

- 1 VS 1方式训练，按照下面方式组织分类器进行测试



DAG SVM的优缺点？

- 调用分类器少：如果类别数是k，则只调用k-1个
- 没有分类重叠和不可分类
- 泛化误差限明确
- 存在错误向下累积现象



SVM的计算复杂度

- 使用SVM进行分类的时候，实际上是训练和分类两个完全不同的过程，因而讨论复杂度就不能一概而论，我们这里所说的主要是训练阶段的复杂度，即解那个二次规划问题的复杂度。对这个问题的解，基本上要划分为两大块，解析解和数值解。

解析解

- 解析解就是理论上的解，它的形式是表达式，因此它是精确的，一个问题只要有解那它的解析解是一定存在的。
- 当然存在是一回事，能够解出来，或者可以在可以承受的时间范围内解出来，就是另一回事了。
- 对SVM来说，求得解析解的时间复杂度最坏可以达到$O(N_{sv}^3)$，其中N_{sv}是支持向量的个数，而虽然没有固定的比例，但支持向量的个数多少也和训练集的大小有关。

数值解

- 数值解就是可以使用的解，是一个一个的数，往往都是近似解。
- 求数值解的过程非常像穷举法，从一个数开始，试一试它当解效果怎样，不满足一定条件（叫做停机条件，就是满足这个以后就认为解足够精确了，不需要继续算下去了）就试下一个，当然下一个数不是乱选的，也有一定章法可循。
- 有的算法，每次只尝试一个数，有的就尝试多个，而且找下一个数字（或下一组数）的方法也各不相同，停机条件也各不相同，最终得到的解精度也各不相同，可见对求数值解的复杂度的讨论不能脱开具体的算法。

一个问题

- 一对一与一对其余那个速度更快？