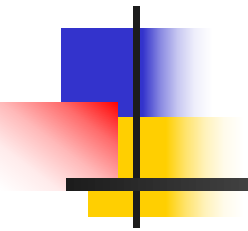
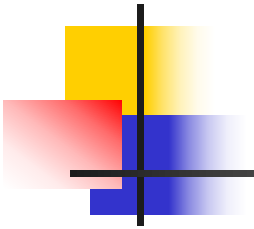


Support Vector Machine
支持向量机



内容

- SVM简介
- 线性分类器
- 核函数
- 松弛变量
- LIBSVM介绍
- 实验

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。

SVM简介

- 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）。

SVM简介

- VC维：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很适合用来解决像文本分类这样的问题，当然，有这样的能力也因为引入了核函数）。

SVM简介

- **结构风险最小原理：**就是追求“经验风险”与“置信风险”的**和**最小。

SVM简介

- **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），但毫无疑问，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）。

SVM简介

- **经验风险Remp(w)：**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) 。

SVM简介

以前的一些机器学习方法把经验风险最小化作为努力的目标，但后来发现很多分类函数能够在样本集上轻易达到100%的正确率，在真实分类时却不好（即所谓的推广能力差，或泛化能力差）。此时的情况是因为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。因为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。但实际上不太可能，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，不能保证在更大比例的真实文本上也没有误差。

SVM简介

- **泛化误差界：**为了解决刚才的问题，统计学提出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样本上分类的结果。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）。

SVM简介

- **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。

SVM简介

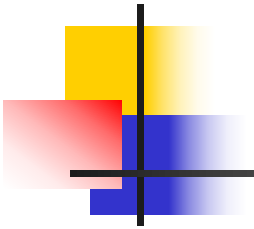
- 泛化误差界的公式为：

$$R(w) \leq R_{emp}(w) + \Phi(n/h)$$

公式中R(w)就是真实风险，$R_{emp}(w)$表示经验风险，$\Phi(n/h)$表示置信风险。此时目标就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简介

- **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。



SVM简介

- **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（也叫惩罚变量）和核函数技术来实现，这一部分是SVM的核心内容，后面会详细说明。

SVM简介

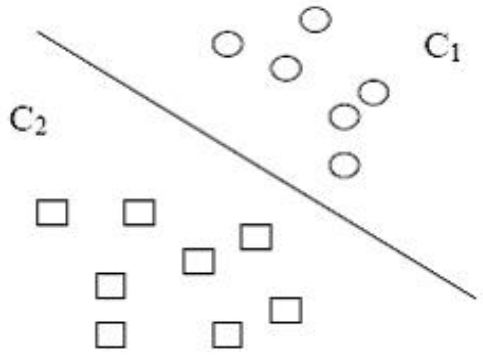
- **高维模式识别：**是指样本维数很高，SVM也可以应付。这主要是因为SVM 产生的分类器很简洁，用到的样本信息很少（仅仅用到那些称之为“支持向量”的样本），使得即使样本维数很高，也不会给存储和计算带来大麻烦。

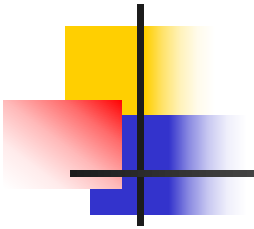
线性分类器

- **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。下面举例说明。

线性分类器

- 用一个二维空间里仅有两类样本的分类问题来举例子。如图所示：C1和C2是要区分的两个类别。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。





线性分类器

- 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）。

线性分类器

- 例如我们有一个线性函数

$$g(x)=wx+b$$

我们可以取阈值为0，这样当有一个样本x_i需要判别的时候，我们就看$g(x_i)$的值。若$g(x_i)>0$，就判别为类别C_1，若$g(x_i)<0$，则判别为类别C_2（等于的时候我们就拒绝判断）。此时也等价于给函数g(x)附加一个符号函数sgn()，即f(x)=sgn [g(x)]是我们真正的判别函数。

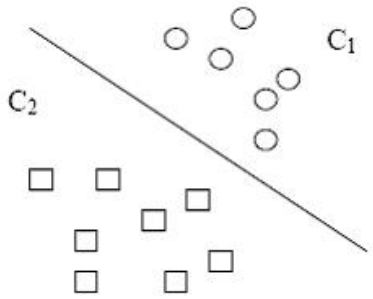
线性分类器

关于$g(x)=wx+b$这个表达式要注意三点：

1. 式中的x不是二维坐标系中的横轴，而是样本的向量表示，例如一个样本点的坐标是(3,8)，则$x^T=(3,8)$，而不是x=3（一般说向量都是说列向量）。
2. 这个形式并不局限于二维的情况，在n维空间中仍然可以使用这个表达式，只是式中的w成为了n维向量（在二维的这个例子中，w是二维向量，注意这里的w严格的说也应该是转置的形式，为了表示起来方便简洁，以下均不区别列向量和它的转置）。
3. g(x)不是中间那条直线的表达式，中间那条直线的表达式是g(x)=0，即wx+b=0，我们也把这个函数叫做分类面。

线性分类器

- **分类间隔：** 下图中间那条分界线并不是唯一的，把它稍微旋转一下，只要不把两类数据分错，仍然可以达到上面说的效果，稍微平移一下，也可以。此时就牵涉到一个问题，对同一个问题存在多个分类函数的时候，哪一个函数更好呢？显然必须要先找一个指标来量化“好”的程度，通常使用叫做“分类间隔”的指标。



线性分类器

比如在进行文本分类的时候，我们可以让计算机这样来看待我们提供给它的训练样本，每一个样本由一个向量（就是那些文本特征所组成的向量）和一个标记（标示出这个样本属于哪个类别）组成。如下：

$$Di=(x_i,y_i)$$

x_i就是文本向量（维数很高），y_i就是分类标记。

线性分类器

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）。有了这种表示法，我们就可以定义一个样本点到某个超平面的间隔：

$$\delta_i = y_i(wx_i + b)$$

首先如果某个样本属于该类别的话，那么$wx_i+b>0$，而y_i也大于0；若不属于该类别的话，那么$wx_i+b<0$，而y_i也小于0，这意味着$y_i(wx_i+b)$总是大于0的，而且它的值就等于$|wx_i+b|$，也就是$|g(x_i)|$。

线性分类器

现在把w和b进行归一化处理，即用w/||w||和b/||w||分别代替原来的w和b，那么间隔就可以写成：

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

这就是解析几何中点x_i到直线g(x)=0的距离公式，也就是到超平面g(x)=0的距离。

线性分类器

||w||叫做向量w的范数，范数是对向量长度的一种度量。我们常说的向量长度其实指的是它的2-范数，范数最一般的表示形式为p-范数，可以写成如下表达式

向量w=(w1, w2, w3,…… wn)

它的p-范数为：

$$\|w\|_p = \sqrt[p]{w_1^p + w_2^p + \_ + w_n^p}$$

当我们不指明p的时候，就意味着我们不关心p的值，用几范数都可以。 当用归一化的w和b代替原值之后的间隔有一个专门的名称，叫几何间隔，表示的是点到超平面的欧氏距离。

线性分类器

下面这张图直观的展示出了几何间隔的现实含义：

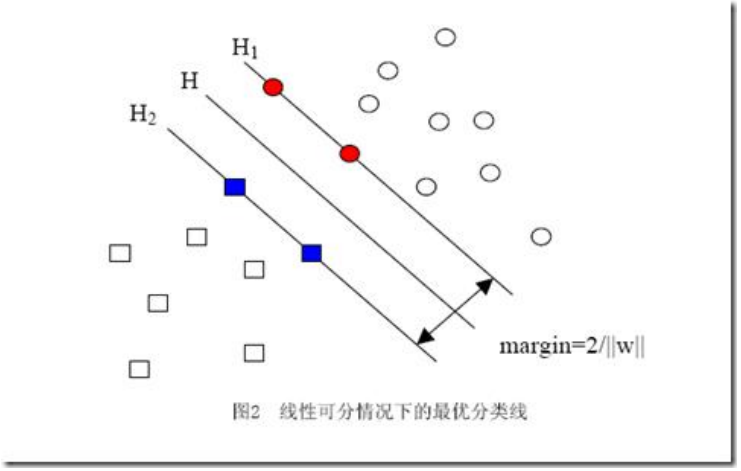


图2　线性可分情况下的最优分类线

H是分类面，而H1和H2是平行于H，且过离H最近的两类样本的直线，H1与H，H2与H之间的距离就是几何间隔。

线性分类器

之所以如此关心几何间隔这个东西，是因为几何间隔与样本的误分次数间存在关系：

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

其中的δ是样本集合到分类面的几何间隔，$R=\max ||x_i||$ $i=1,...,n$，即R是所有样本中向量长度最长的值（也就是说代表样本的分布有多么广）。误分次数一定程度上代表分类器的误差。而从上式可以看出，在样本已知的情况下，误分次数的上界由几何间隔决定！几何间隔越大的解，它的误差上界越小。因此最大化几何间隔成了训练阶段的目标。

线性分类器

间隔：δ=y(wx+b)=|g(x)|

几何间隔：

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

可以看出δ=||w||δ几何。几何间隔与||w||是成反比的，因此最大化几何间隔与最小化||w||完全是一回事。而我们常用的方法并不是固定||w||的大小而寻求最大几何间隔，而是把所有样本点中间隔最小的那一点的间隔固定（例如固定为1），寻找最小的||w||。

线性分类器

如果直接来解这个求最小值问题，当||w||=0的时候就得到了目标函数的最小值。但是无论给什么样的数据，都是这个解！反映在图中，就是H1与H2两条直线间的距离无限大，这个时候，所有的样本点都跑到了H1和H2中间，进入了无法分类的灰色地带。

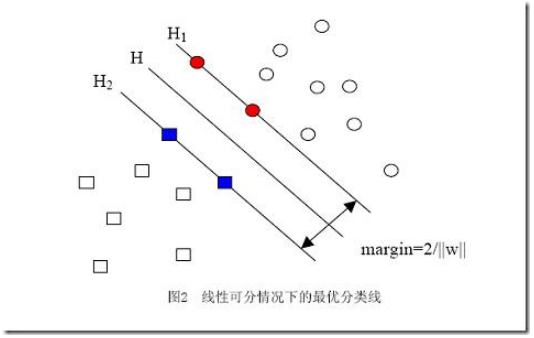


图2 线性可分情况下的最优分类线

造成这种结果的原因是在描述问题的时候只考虑了目标，而没有加入约束条件 。

线性分类器

之前把所有样本点中间隔最小的那一点的间隔定为1，这就相当于让下面的式子总是成立：

$y_i[(w \cdot x_i)+b] \geq 1 \ (i=1,2,\ldots,l)$ （l是总的样本数）

即：

$y_i[(w \cdot x_i)+b]-1 \geq 0 \ (i=1,2,\ldots,l)$ （l是总的样本数）

因此我们的两类分类问题也被我们转化成了它的数学形式，一个带约束的最小值的问题：

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

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

线性分类器

从最一般的定义上说，一个求最小值的问题就是一个优化问题（也叫规划），它同样由两部分组成，目标函数和约束条件，可以用下面的式子表示：

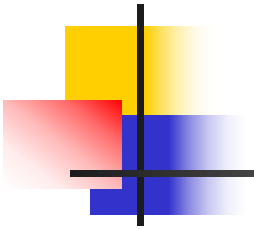
$$\begin{aligned} &\min && f(x) \\ &\text{subject to} && c_i(x) \le 0 \quad i=1,2,\ldots,p \\ & && c_j(x) = 0 \quad j=p+1,p+2,\ldots,p+q \end{aligned}$$

约束条件用函数c来表示，就是constrain的意思。一共有p+q个约束条件，其中p个是不等式约束，q个等式约束。

线性分类器

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

这个式子中的x是自变量，但不限定它的维数必须为1（视乎你解决的问题空间维数）。要求f(x)在哪一点上取得最小值，但不是在整个空间里找，而是在约束条件所划定的可行域里找。注意可行域中的每一个点都要求满足所有p+q个条件，同时可行域边界上的点有一个额外好的特性，它们可以使不等式约束取得等号！而边界内的点不行。



线性分类器

这对一般的优化问题可能提供不了什么帮助，但对 SVM来说，边界上的点有其特殊意义，实际上是它们唯一决定了分类超平面，这些点（就是以前的图中恰好落在H1和H2上的点，在文本分类问题中，每一个点代表一个文档，因而这个点本身也是一个向量）就被称为支持向量。

线性分类器

回头再看线性分类器问题的描述：

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

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

在这个问题中，自变量就是w，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（不要把x_i当成变量，它代表样本，是已知的），这种规划问题也叫做二次规划（Quadratic Programming，QP）。而且，由于它的可行域是一个凸集，因此它是一个凸二次规划。凸二次规划的优点在于它有全局最优解。

线性分类器

但是实际上我们并不知道该怎么解一个带约束的优化问题。我们可以轻松的解一个不带任何约束的优化问题（实际上就是函数求极值，求导再找0点），我们甚至还会解一个只带等式约束的优化问题，就是求条件极值，通过添加拉格朗日乘子，构造拉格朗日函数，来把这个问题转化为无约束的优化问题。

如果只带等式约束的问题可以转化为无约束的问题来求解，那么可不可以把带不等式约束的问题向只带等式约束的问题转化而得以求解呢？答案是可以。

线性分类器

我们想求得这样一个线性函数（在n维空间中的线性函数）：

$$g(x)=wx+b$$

求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（但实际上只需要求w，求得以后找某些样本点代入就可以求得b）。因此在求g(x)的时候，w才是变量。

线性分类器

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：

$$w=a_1x_1+a_2x_2+\ldots+a_nx_n$$

式子中的a_i是一个一个的数（在严格的证明过程中，这些a被称为拉格朗日乘子），而x_i是样本点，因而是向量，n就是总样本点的个数。为了方便描述，以下开始严格区别数字与向量的乘积和向量间的乘积，用a_1x_1表示数字和向量的乘积，而用$<x_1,x_2>$表示向量x_1,x_2的内积。因此g(x)的表达式严格的形式应该是：

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

线性分类器

但是上面的式子还不够好，如果我不动所有点的位置，而只是把其中一个正样本点定为负样本点（也就是把一个点的形状从圆形变为方形），那么三条直线都必须移动。这说明w不仅跟样本点的位置有关，还跟样本的类别有关因此用下面这个式子表示才算完整：

$$w=\alpha_1 y_1 x_1+\alpha_2 y_2 x_2+\ldots+\alpha_n y_n x_n$$

其中的yi就是第i个样本的标签，它等于1或者-1。其实以上式子的不等于0（不等于0才对w起决那一堆拉格朗日乘子中，只有很少的一部分定作用），这部分不等于0的拉格朗日乘子后面所乘的样本点，其实都落在H1和H2上，也正是这部分样本唯一的确定了分类函数。更严格的说，这些样本的一部分就可以确定，因为例如确定一条直线，只需要两个点就可以。这部分样本点，就叫做支持（撑）向量！

线性分类器

式子也可以用求和符号简写一下：

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

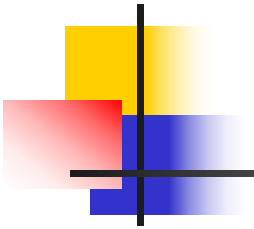
因此原来的g(x)表达式可以写为：

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

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

注意式子中x才是变量，如果要分类哪篇文档，就把该文档的向量表示代入到x的位置，而所有的x_i统统都是已知的样本。还注意到式子中只有x_i和x是向量，因此一部分可以从内积符号中拿出来，得到g(x)的式子为：

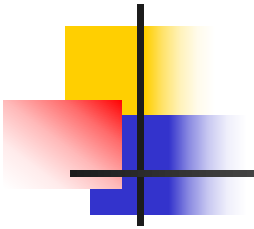
$$g(x)=\sum_{i=1}^{n}a_i y_i <x_i,x>+b \quad (式2)$$



线性分类器

至此w不见了，从求w变成了求α。看似没有简化问题，其实简化了原来的问题，因为以这样的形式描述问题以后，我们的优化了不等式约束。之后的求解就变得很容易了。

下面遇到一个问题：如果提供的样本线性不可分，怎么办？所以必须要提到SVM中比较重要的内容——核函数。

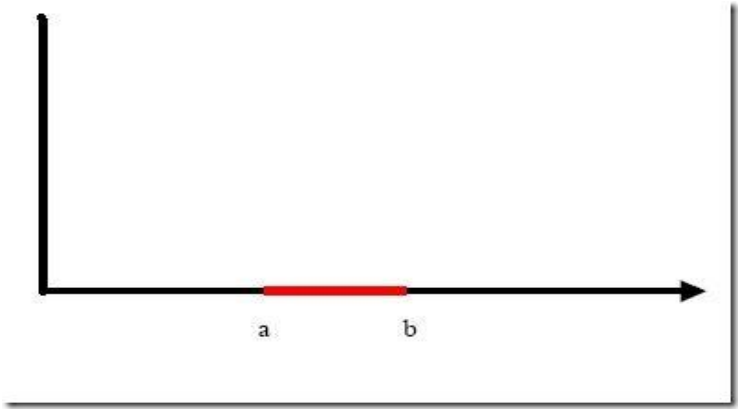


核函数

之前一直在讨论的线性分类器。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。这必然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，那么就必须寻找让线性不可分的数据变得线性可分的方法。

核函数

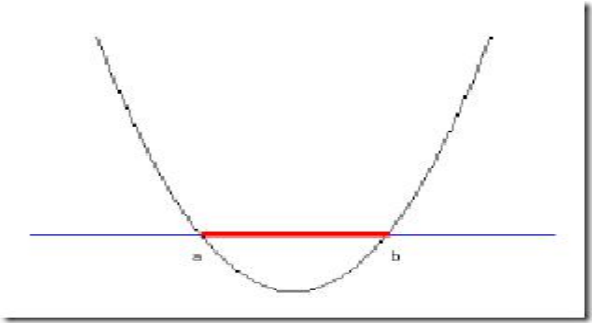
用一个二维平面中的分类问题作例子，如图：



把横轴上端点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

但我们可以找到一条曲线，例如下面这一条：



显然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。这条曲线就是我们熟知的二次曲线，它的函数表达式是：

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

核函数

问题只是它不是一个线性函数，但是，做一下变换，新建一个向量y和a：

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

这样g(x)就可以转化为f(y)=<a,y>，你可以把y和a分别回带一下，看看等不等于原来的g(x)。用内积的形式写你可能看不太清楚，实际上f(y)的形式就是：

$$g(x)=f(y)=ay$$

在任意维度的空间中，这种形式的函数都是一个线性函数，因为自变量y的次数不大于1。

原来在二维空间中一个线性不可分的问题，映射到高维空间后，变成了线性可分的！因此也形成了我们最初想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。

核函数

用一个具体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，如果我们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，只不过w'是定值，而x'是变量。现在我们的输入呢，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。

核函数

所以只需要关心那个高维空间里内积的值。而从理论上说，x'是经由x变换来的，因此广义上可以把它叫做x的函数（因为有一个x，就确定了一个x'），而w'是常量，它是一个低维空间里的常量w经过变换得到的，所以给了一个w 和x的值，就有一个确定的f(x')值与其对应。所以，需要这样一种函数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'>。

也就是当给了一个低维空间的输入x以后：

$$g(x)=K(w,x)+b$$

$$f(x')=\langle w',x'\rangle+b$$

这两个函数的计算结果就完全一样，我们也就用不着费力找那个映射关系，直接拿低维的输入往g(x)里面代就可以了。

核函数

这样的函数确实存在，它被称作核函数（核，kernel），而且不止一个，事实上，只要是满足了Mercer条件的函数，都可以作为核函数。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。这就是说，尽管给的问题是线性不可分的，但是就硬当它是线性问题来求解，只不过求解过程中，凡是要求内积的时候就用选定的核函数来算。这样求出来的α再和选定的核函数一组合，就得到分类器。 。

核函数

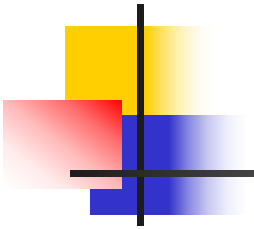
几个比较常用的核函数如下：

1) 线性核函数（linear kernel） $k(\mathbf{x},\mathbf{x}_i)=(\mathbf{x}\cdot\mathbf{x}_i)$

2) 多项式核函数(polynomial kernel) $k(\mathbf{x},\mathbf{x}_i)=(s(\mathbf{x}\cdot\mathbf{x}_i)+c)^d$ ，其中s, c d为参数。显然，线性核函数可以看作多项式核函数的一种特殊情况。

3) 径向基核函数(radical basis function，rbf) $k(\mathbf{x},\mathbf{x}_i)=\exp(-\gamma|\mathbf{x}-\mathbf{x}_i|^2)$ ，其中 γ 为参数。

4) Sigmoid 核函数(Sigmoid tanh) $k(\mathbf{x},\mathbf{x}_i)=\tanh(s(\mathbf{x}\cdot\mathbf{x}_i)+c)$，其中s, c为参数。

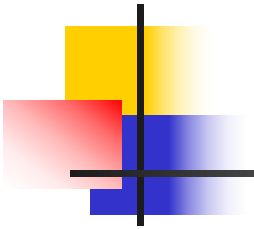


核函数

接下来还有两个问题：

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？



核函数

对核函数的选择，现在还缺乏指导原则！各种实验的观察结果（不光是文本分类）的确表明，某些问题用某些核函数效果很好，用另一些就很差，但是一般来讲，径向基核函数（RBF）是不会出太大偏差的一种。

核函数

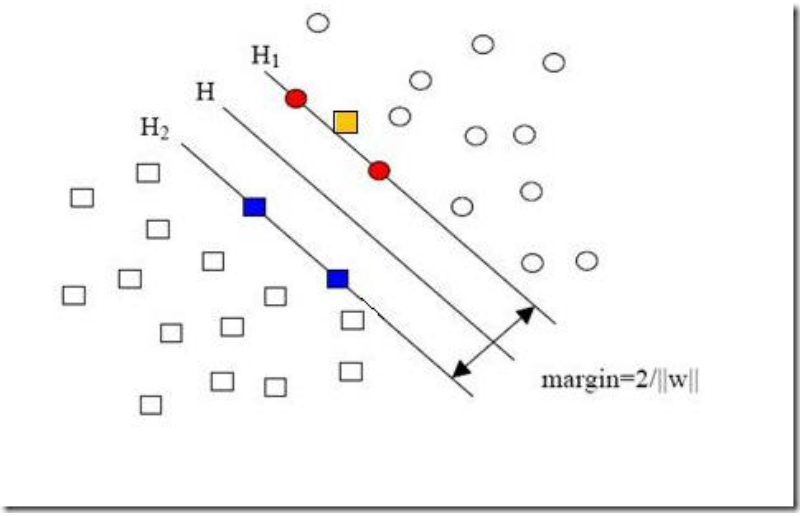
在常用的核函数中，应用最广泛的是具有较好学习能力的RBF 核，无论低维、高维、小样本、大样本等情况，RBF核均适应，具有较宽的收敛域，是较为理想的分类依据函数。Keerthi S S 等人证明了线性核和多项式核是RBF 核的特殊情况。 Lin C J等说明了在某些参数情况下， Sigmoid 核同RBF 核具有相似的性能。

松弛变量

- 解决第二个问题：

举个例子，比如我们已经把一个本来线性不可分的文本分类问题，通过映射到高维空间而变成了线性可分的。现在有这样一个训练集，只比原先这个训练集多了一篇文章，映射到高维空间以后也就多了一个样本点，但是这个样本的位置如下图：

松弛变量



就是图中黄色那个点，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

按照常识判断，如果有一万个点都符合某种规律，只有一个点不符合，那这个样本点可能是噪声。所以即便简单的忽略这个样本点，仍然使用原来的分类器，其效果丝毫不受影响。

但程序并没有这种容错性。由于在原本的优化问题的表达式中，确实要考虑所有的样本点，并在此基础上寻找正负类之间的最大几何间隔，而几何间隔代表的是距离，是非负的，像上面这种有噪声的情况会使得整个问题无解。这种解法其实也叫做**“硬间隔”**分类法，因为他硬性的要求所有样本点都满足和分类平面间的距离必须大于某个值。

解决方法也很明显，就是仿照人的思路，允许一些点到分类平面的距离不满足原先的要求。

松弛变量

为此引入一个非负的松弛项，有两种常用的方式：

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

另一种是：

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

其中l都是样本的数目。两种方法没有大的区别。如果选择了第一种，得到的方法的就叫做**一阶软间隔分类器**，第二种就叫做**二阶软间隔分类器**。

松弛变量

把损失加入到目标函数里的时候，就需要一个惩罚因子（cost，也就是libSVM的诸多参数中的C），原来的优化问题就变成了下面这样：

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

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

$$\zeta_i \geq 0$$

这个式子有这么几点要注意：

1. 并非所有的样本点都有一个松弛变量与其对应。只有“离群点”才有。
2. 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。

松弛变量

3. 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，C越大，对目标函数的损失也越大，此时就暗示着非常不愿意放弃这些离群点，最极端的情况是把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。

4. 是惩罚因子C不是一个变量，整个优化问题在解的时候，C是一个必须事先指定的值，指定这个值以后，解一下，就得到一个分类器，然后用测试数据看看结果好不好，不好再换一个C的值。如此就是一个参数寻优的过程。

另外，当遇到数据集偏斜问题时，也是通过调整惩罚因子来解决。

松弛变量

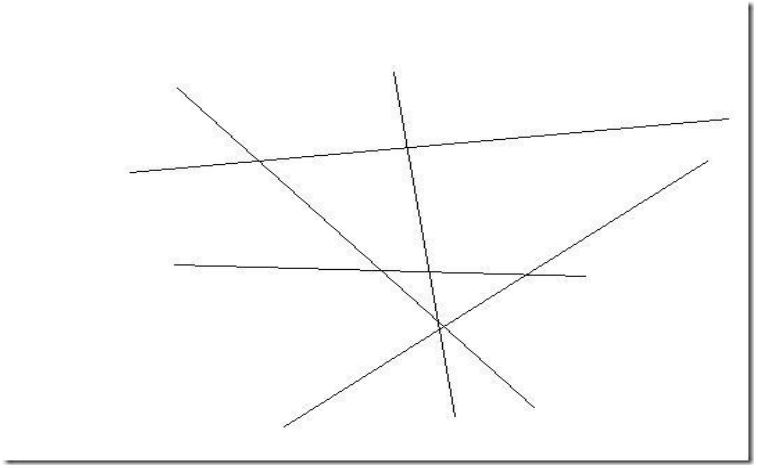
核函数与松弛变量作用的区别：

虽然二者的引入都是为了解决线性不可分的问题。但两者还有微妙的不同。

以文本分类为例。在原始的低维空间中，样本相当的不可分，无论怎么找分类平面，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，就是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的离群点，就简单有效得多了。

SVM用于多类分类

一次性得到多个分类面的方法： 就是真的一次性考虑所有样本，并求解一个多目标函数的优化问题，一次性得到多个分类面，如下图：



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。

SVM用于多类分类

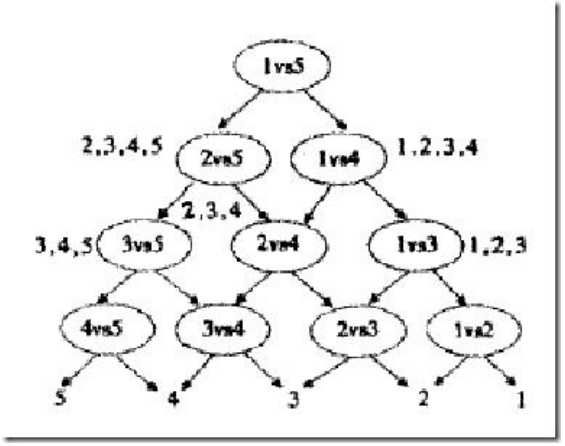
- **“一类对其余”的方法：**比如有5个类别，第一次就把类别1的样本定为正样本，其余2，3，4，5的样本合起来定为负样本，得到一个两类分类器，它能够指出一篇文章是还是不是第1类的；第二次我们把类别2 的样本定为正样本，把1，3，4，5的样本合起来定为负样本，得到一个分类器，如此下去。但这种方法容易导致分类重叠现象和不可分类现象人为造成“数据集偏斜”问题。

SVM用于多类分类

- **“一对一单挑”的方法：**每次选一个类的样本作正类样本，而负类样本则变成只选一个类。因此第一个只回答“是第1类还是第2类”，第二个只回答“是第1类还是第3类”，如此下去。在真正用来分类的时候，把一篇文章扔给所有分类器，让每一个都投上自己的一票，最后统计票数，如果类别“1”得票最多，就判这篇文章属于第1类。这种方法使得两类分类器数目为k(k-1)/2）。类别数如果是1000，要调用的分类器数目会上升至约500000个，过于复杂。

SVM用于多类分类

所以必须在分类的时候下功夫。举个例子，还是像一对一方法那样来训练，只是在对一篇文章进行分类之前，先按照下面图的样子来组织分类器：



分类时，先问分类器“1对5”（意思是它能够回答“是第1类还是第5类”），如果回答5，就往左走，再问“2对5”这个分类器，如果还是“5”，继续往左走，这样一直下去，就可以得到分类结果。

SVM用于多类分类

这是一个有向无环图，因此这种方法也叫做DAG SVM），调用了k-1（K是类别数）个分类器。速度快，且没有分类重叠和不可分类现象。缺点是假如最一开始的分类器回答错误，那么后面的分类器是无法纠正它的错误。对下面每一层的分类器都存在这种错误向下累积的现象。但是DAG方法好于它们的地方就在于，累积的上限，不管是大是小，总是有定论的，有理论可以证明。并且可以通过调整**“根节点的选取”**及输出**“置信度”**来改善效果。

LIBSVM介绍

- LIBSVM是台湾大学林智仁教授2001年开发的一套支持向量机的库，运算速度快，可以很方便的对数据做分类或回归。由于LIBSVM程序小，运用灵活，输入参数少，并且是开源的，易于扩展，因此成为目前国内应用最多的SVM的库。这套库目前已经发展到2.9版。

LIBSVM介绍

工具包里主要有5个文件夹：

1. **Java**主要是应用于java平台；
2. **Python**是用来参数优选的工具，稍后介绍；
3. **svm-toy**是一个可视化的工具，用来展示训练数据和分类界面，里面是源码，其编译后的程序在windows文件夹下；
4. **tools**—主要包含四个python文件，用来数据集抽样(subset)，参数优选（grid），集成测试(easy), 数据检查(check data)；
5. **windows**包含libSVM四个exe程序包，我们所用的库就是他们。其中svm-scale.exe是用来对原始样本进行缩放的；svm-train.exe主要实现对训练数据集的训练，并可以获得SVM模型；svmpredict 是根据训练获得的模型，对数据集合进行预测。还有一个svm-toy.exe之前已经交待过，是一个可视化工具。

LIBSVM介绍

libSVM的**数据格式**如下：

Label 1:value 2:value …

Label是类别的标识，比如上节train.model中提到的1 -1，你可以自己随意定，比如-10，0，15。如果是回归，这是目标值，就要实事求是了。

Value就是要训练的数据，从分类的角度来说就是特征值，数据之间用空格隔开，比如：

-15 1:0.708 2:1056 3:-0.3333

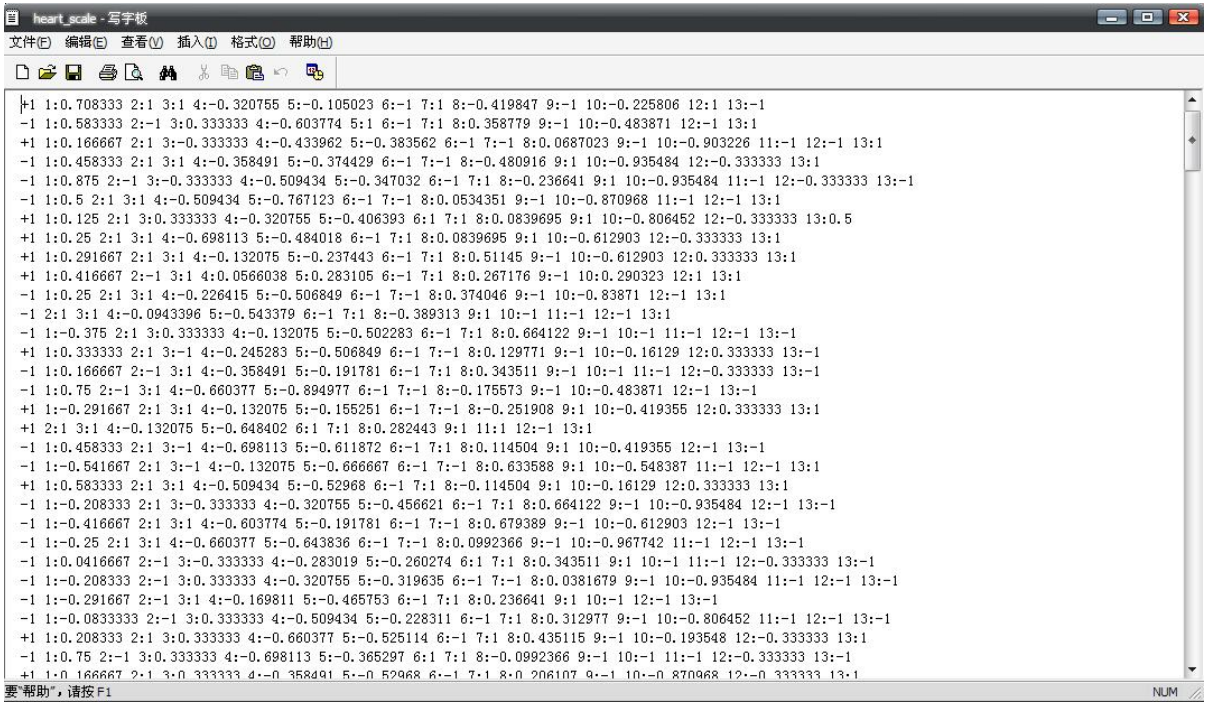
需要注意的是，如果特征值为0，特征冒号前面的(姑且称做序号)可以不连续。如：

-15 1:0.708 3:-0.3333

表明第2个特征值为0，从编程的角度来说，这样做可以减少内存的使用，并提高做矩阵内积时的运算速度。

实验

实验样本选择工具包下存在的样本文件：heart_scale。内容截图如下：

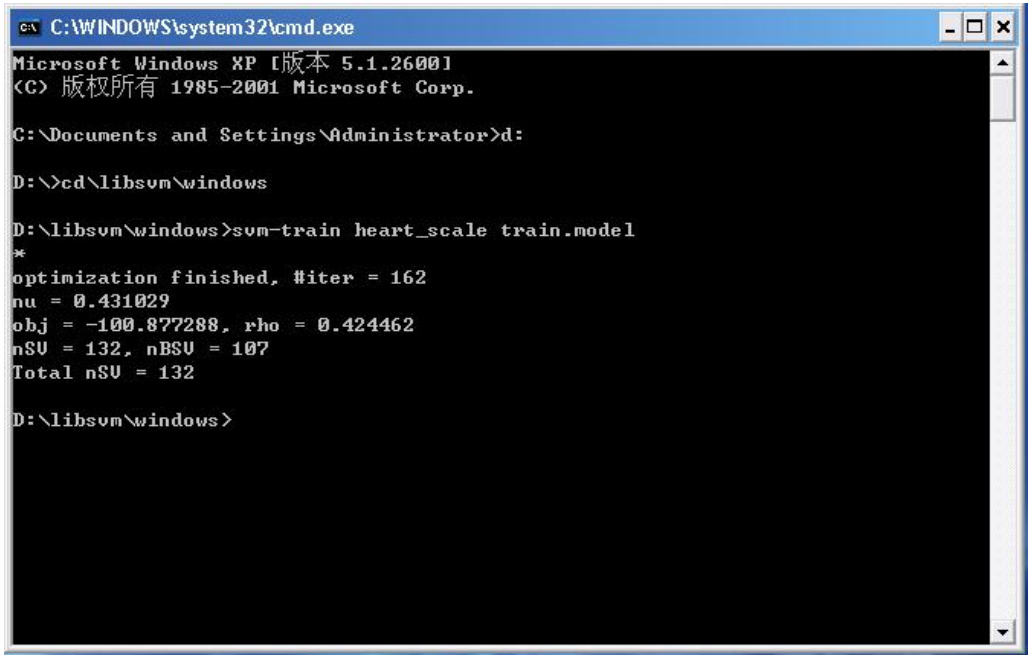


实验

训练：首先用svm-train进行训练，输入以下命令：

svm-train heart_scale train.model

得到一个结果文件：train.model



实验

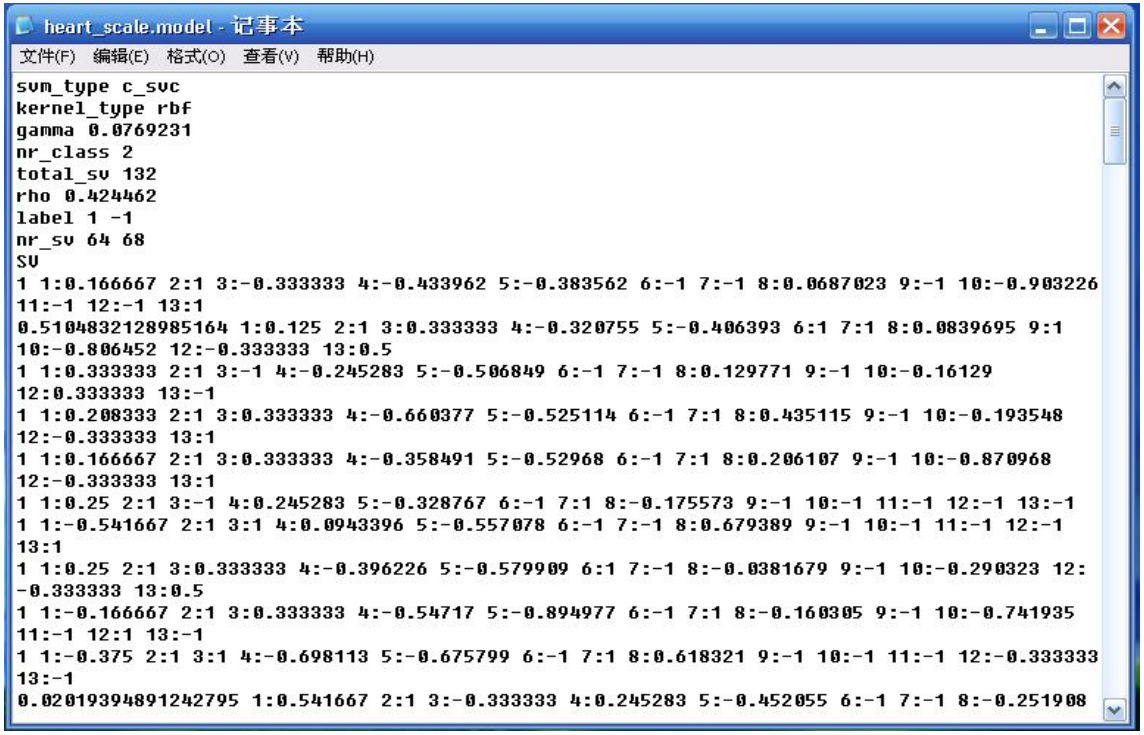
可以看到结果：

optimization finished, #iter = 162
nu = 0.431029
obj = -100.877288, rho = 0.424462
nSV = 132, nBSV = 107
Total nSV = 132

其中，#iter为迭代次数，nu 是选择的核函数类型的参数，obj为SVM文件转换为的二次规划求解得到的最小值，rho为判决函数的偏置项b，nSV 为标准支持向量个数(0<a[i]<c)，nBSV为边界上的支持向量个数(a[i]=c)，Total nSV为支持向量总个数（对于两类来说，因为只有一个分类模型Total nSV = nSV，但是对于多类，这个是各个分类模型的nSV之和）。

实验

- train.model文件用记事本打开如下图：



实验

svm_type c_svc　　//所选择的svm类型，默认为c_svc
kernel_type rbf　　//训练采用的核函数类型，此处为RBF核
gamma 0.0769231　　//RBF核的参数γ
nr_class 2　　//类别数，此处为两分类问题
total_sv 132　　//支持向量总个数
rho 0.424462　　//判决函数的偏置项b
label 1 -1　　//原始文件中的类别标识
nr_sv 64 68　　//每个类的支持向量机的个数
SV　　//以下为各个类的权系数及相应的支持向量

1 1:0.166667 2:1 3:-0.333333 … 10:-0.903226 11:-1 12:-1 13:1
0.5104832128985164 1:0.125 2:1 3:0.333333 … 10:-0.806452 12:-0.333333 13:0.5
……
-1 1:-0.375 2:1 3:-0.333333…. 10:-1 11:-1 12:-1 13:1
-1 1:0.166667 2:1 3:1 …. 10:-0.870968 12:-1 13:0.5

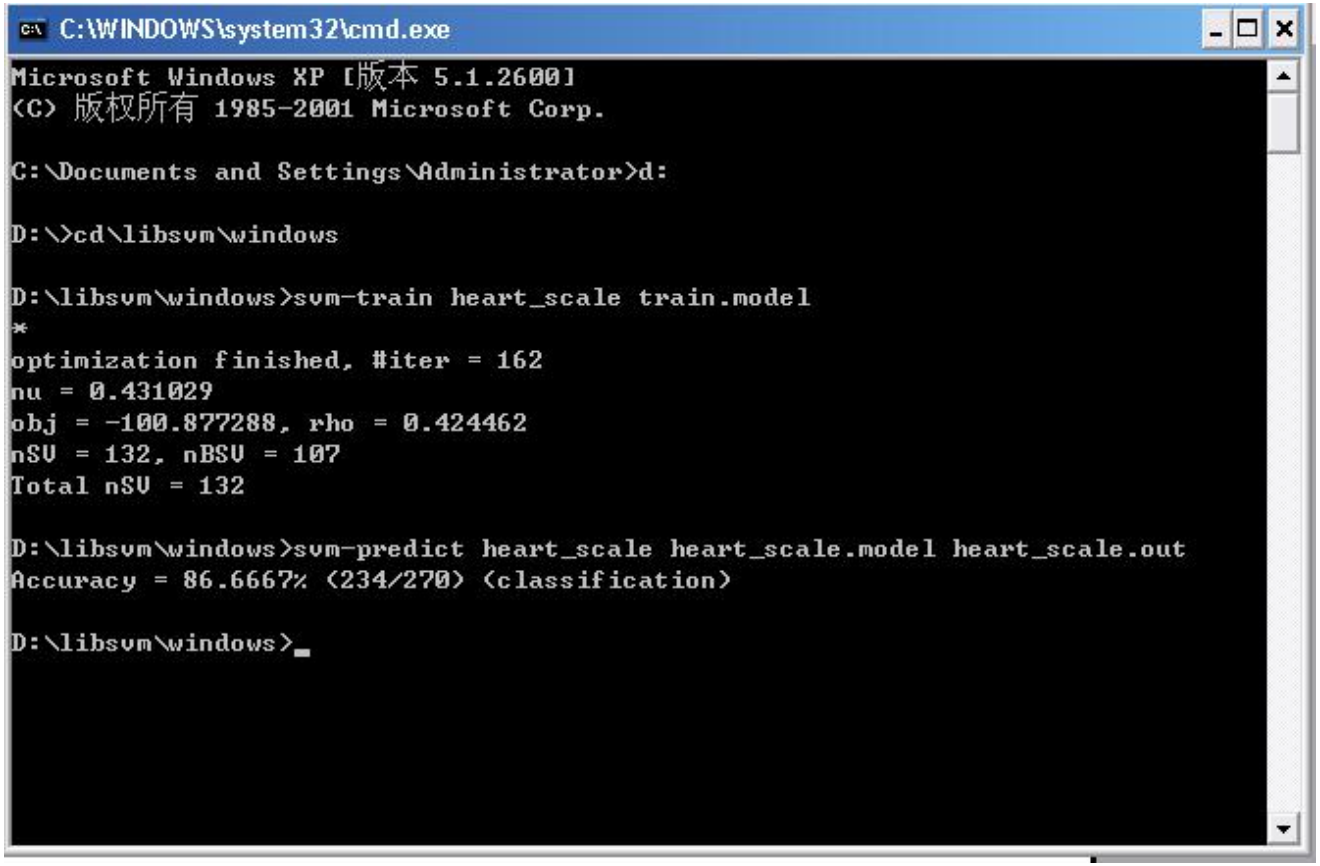
实验

使用svm-predict，根据训练获得的模型，对数据集合进行预测。输入命令：

svm-predict heart_scale heart_scale.model heart_scale.out

得到准确率是：86.667%。结果如下图所示：

实验


C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.
C:\Documents and Settings\Administrator>d:
D:\>cd\libsvm\windows
D:\libsvm\windows>svm-train heart_scale train.model
*
optimization finished, #iter = 162
nu = 0.431029
obj = -100.877288, rho = 0.424462
nSV = 132, nBSV = 107
Total nSV = 132
D:\libsvm\windows>svm-predict heart_scale heart_scale.model heart_scale.out
Accuracy = 86.6667% (234/270) (classification)
D:\libsvm\windows>

应用简介

应用

Boser，Guyon 等人利用美国邮政标准手写数字库进行的对比实验，使用多项式核函数对16×16 的手写体阿拉伯数字进行了识别，训练集共7300 个样本，测试集有2000 个样本。 实验结果表明采用SVM 方法比采用其他神经网络(具有五层神经网络复杂结构) 算法效果要好。针对手写汉字、针对印刷汉字分别利用SVM 技术进行了字体识别，识别率非常高。Guodong Guo 等人采用两个人脸数据库，一个是含有400 幅图象，另一个含有1079 幅图象，并将SVM 方法与其他算法进行了比较，证明SVM 方法的错识率最低。 此外SVM还在其他数据分类、线性回归等方面得到应用。

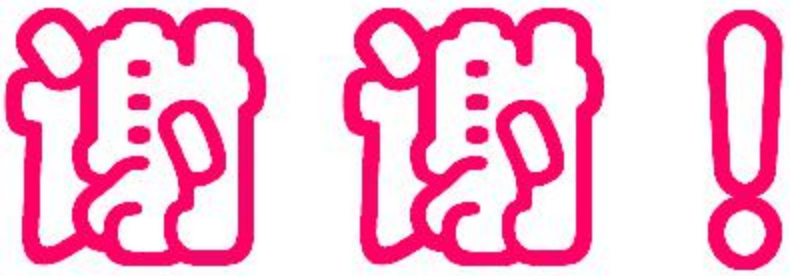
结论

SVM 在分类、函数模拟等领域中的作用越来越明显。但从发展到完善时间很短，还有很多缺欠。

（1）核函数的选定非常关键，它的选择好坏直接影响到算法的实现与效果。目前对这方面的研究还不够，缺乏相应的理论根据；

（2）SVM 分类器的设计依赖于少量的支持向量，使得分类效果受噪声的影响非常大。如何构造容噪性能强的 SVM 分类器尤为关键；

（3）SVM 的计算速度较慢，尤其是训练样本很大时，传统的求解优化方法难以满足实时性要求；尽管如此，支持向量机较强的分类性能和适应性能使得这种方法能够在实际工程中发挥重要作用。


謝謝！