2.41 决策树

2.41.1 决策树的基本原理

决策树是一种分而治之(Divide and Conquer)的决策过程。一个困难的预测问题，通过树的分支节点，被划分成两个或多个较为简单的子集，从结构上划分为不同的子问题。将依规则分割数据集的过程不断递归下去(Recursive Partitioning)。随着树的深度不断增加，分支节点的子集越来越小，所需要提的问题数也逐渐简化。当分支节点的深度或者问题的简单程度满足一定的停止规则(Stopping Rule)时，该分支节点会停止劈分，此为自上而下的停止阈值(Cutoff Threshold)法；有些决策树也使用自下而上的剪枝(Pruning)法。

2.41.2 决策树的三要素？

一棵决策树的生成过程主要分为以下 3 个部分:

特征选择：从训练数据中众多的特征中选择一个特征作为当前节点的分裂标准，如何选择特征有着很多不同量化评估标准标准，从而衍生出不同的决策树算法。

决策树生成：根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长。树结构来说，递归结构是最容易理解的方式。

剪枝：决策树容易过拟合，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有预剪枝和后剪枝两种。

2.41.3 决策树学习基本算法

机器学习算法 1 决策树学习基本算法

输入: 训练集$D=\{(\mathbf{x}_1,y_1),(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)\}$; 属性集$A=\{(a_1,a_2,\ldots,a_d)\}$

输出: 以$node$为根节点的一棵决策树

```
function TreeGenerate(D, A)
    生成节点node
    if D中的样本全属于同一类别C then
        将node标记为C类叶节点;return
    end if
    if A = ∅ OR D中样本再A上取值相同 then
        将node标记为叶节点，其类别标记为D中样本类最多的类;return
    end if
    从A中选择最优划分属性a_*
    for a_*的每一个值a_*^v do
        为node生成一个分支;令D_v表示D中在a_*上取值为a_*^v的样本子集;
        if D_v为空 then 将分支节点标记为叶节点，其类别标记为D中样本最多的类;return
        else 以TreeGenerate(D_v, A\{a_*})为分支节点
        end if
    end for
end function
```


2.41.4 决策树算法优缺点

决策树算法的优点：

1、理解和解释起来简单，决策树模型易想象。

2、相比于其他算法需要大量数据集而已，决策树算法要求的数据集不大。

3、决策树算法的时间复杂度较小，为用于训练决策树的数据点的对数。

4、相比于其他算法智能分析一种类型变量，决策树算法可处理数字和数据的类别。

5、能够处理多输出的问题。

6、对缺失值不敏感。

7、可以处理不相关特征数据。

8、效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

决策树算法的缺点：

1、对连续性的字段比较难预测。

2、容易出现过拟合。

3、当类别太多时，错误可能就会增加的比较快。

4、信息缺失时处理起来比较困难，忽略了数据集中属性之间的相关性。

5、在处理特征关联性比较强的数据时表现得不是太好。

6、对于各类别样本数量不一致的数据，在决策树当中,信息增益的结果偏向于那些具有更多数值的特征。

2.40.5 熵的概念以及理解

熵：度量随机变量的不确定性。

定义：假设随机变量 X 的可能取值有 x_1，$x_2,\cdots,x_n$，对于每一个可能的取值 x_i，其概率为 $P(X=x_i)=p_i,i=1,2,...,n$。随机变量的熵为：

$$H(X)=-\sum_{i=1}^{n}p_i\log_2 p_i$$

对于样本集合 D，假设样本有 k 个类别，每个类别的概率为 $\frac{|C_k|}{|D|}$，其中 $|C_k|$ 为类别为 k 的样本个数，$|D|$ 为样本总数。样本集合 D 的熵为：

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

2.40.6 信息增益的理解

定义： 以某特征划分数据集前后的熵的差值。

熵可以表示样本集合的不确定性，熵越大，样本的不确定性就越大。因此可以使用划分前后集合熵的差值来衡量使用当前特征对于样本集合 D 划分效果的好坏。

假设划分前样本集合 D 的熵为 $H(D)$。使用某个特征 A 划分数据集 D，计算划分后的数据子集的熵为 $H(D|A)$。

则信息增益为：

$$g(D,A)=H(D)-H(D|A)$$

注：在决策树构建的过程中我们总是希望集合往最快到达纯度更高的子集合方向发展，因此我们总是选择使得信息增益最大的特征来划分当前数据集 D。

思想：计算所有特征划分数据集 D，得到多个特征划分数据集 D 的信息增益，从这些信息增益中选择最大的，因而当前结点的划分特征便是使信息增益最大的划分所使用的特征。

另外这里提一下信息增益比相关知识：

信息增益比=惩罚参数×信息增益。

信息增益比本质：在信息增益的基础之上乘上一个惩罚参数。特征个数较多时，惩罚参数较小；特征个数较少时，惩罚参数较大。

惩罚参数：数据集 D 以特征 A 作为随机变量的熵的倒数。

2.40.7 剪枝处理的作用及策略？

剪枝处理是决策树学习算法用来解决过拟合的一种办法。

在决策树算法中，为了尽可能正确分类训练样本， 节点划分过程不断重复， 有时候会造成决策树分支过多，以至于将训练样本集自身特点当作泛化特点， 而导致过拟合。 因此可以采用剪枝处理来去掉一些分支来降低过拟合的风险。

剪枝的基本策略有预剪枝(prepruning)和后剪枝(postprunint)。

预剪枝：在决策树生成过程中，在每个节点划分前先估计其划分后的泛化性能， 如果不能提升，则停止划分，将当前节点标记为叶结点。

后剪枝：生成决策树以后，再自下而上对非叶结点进行考察， 若将此节点标记为叶结点可以带来泛化性能提升，则修改之。

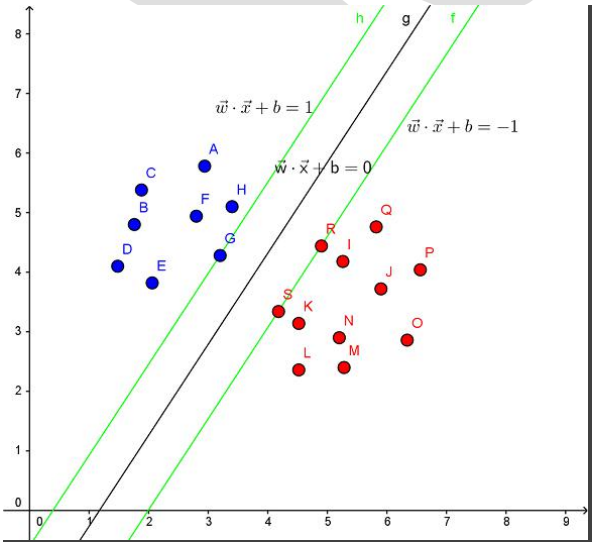
2.41 支持向量机

2.41.1 什么是支持向量机

SVM - Support Vector Machine。支持向量机，其含义是通过支持向量运算的分类器。其中“机”的意思是机器，可以理解为分类器。

什么是支持向量呢？在求解的过程中，会发现只根据部分数据就可以确定分类器，这些数据称为支持向量。

见下图，在一个二维环境中，其中点 R，S，G 点和其它靠近中间黑线的点可以看作为支持向量，它们可以决定分类器，也就是黑线的具体参数。



2.25.2 支持向量机解决的问题？

https://www.cnblogs.com/steven-yang/p/5658362.html

解决的问题：

线性分类

在训练数据中，每个数据都有 n 个的属性和一个二类类别标志，我们可以认为这些数据在一个 n 维空间里。我们的目标是找到一个 n-1 维的超平面（hyperplane），这个超平面可以将数据分成两部分，每部分数据都属于同一个类别。

其实这样的超平面有很多，我们要找到一个最佳的。因此，增加一个约束条件：这个超平面到每边最近数据点的距离是最大的。也成为最大间隔超平面（maximum-margin hyperplane）。这个分类器也成为最大间隔分类器（maximum-margin classifier）。

支持向量机是一个二类分类器。

非线性分类

SVM 的一个优势是支持非线性分类。它结合使用拉格朗日乘子法和 KKT 条件，以及核函数可以产生非线性分类器。

分类器 1 - 线性分类器

是一个线性函数，可以用于线性分类。一个优势是不需要样本数据。

classifier 1:

f(x)=xwT+b(1)

(1)f(x)=xwT+b

ww 和 bb 是训练数据后产生的值。

分类器 2 - 非线性分类器

支持线性分类和非线性分类。需要部分样本数据（支持向量），也就是 α i≠0 α i≠0 的数据。

∵∵

w=∑ni=1 α iyixiw=∑i=1n α iyixi

∴∴

classifier 2:

f(x)=∑ni=1 α iyiK(xi,x)+bherexi : training data iyi : label value of training data i α i : Lagrange multiplier of training data iK(x1,x2)=exp(−// x1−x2 // 22 σ 2) : kernel function(2)

(2)f(x)= Σ i=1n α iyiK(xi,x)+bherexi : training data iyi : label value of training data i α i : Lagrange multiplier of training data iK(x1,x2)=exp(− ‖ x1−x2 ‖ 22 σ 2) : kernel function

α α , σ σ 和 bb 是训练数据后产生的值。

可以通过调节 σ σ 来匹配维度的大小， σ σ 越大，维度越低。

2.25.2 核函数作用？

核函数目的：把原坐标系里线性不可分的数据用 Kernel 投影到另一个空间，尽量使得数据在新的空间里线性可分。

核函数方法的广泛应用,与其特点是分不开的：

1）核函数的引入避免了“维数灾难”,大大减小了计算量。而输入空间的维数 n 对核函数矩阵无影响，因此，核函数方法可以有效处理高维输入。

2）无需知道非线性变换函数Φ的形式和参数.

3）核函数的形式和参数的变化会隐式地改变从输入空间到特征空间的映射，进而对特征空间的性质产生影响，最终改变各种核函数方法的性能。

4）核函数方法可以和不同的算法相结合，形成多种不同的基于核函数技术的方法，且这两部分的设计可以单独进行，并可以为不同的应用选择不同的核函数和算法。

2.25.3 对偶问题

2.25.4 理解支持向量回归

http://blog.csdn.net/liyaohhh/article/details/51077082

2.25.5 理解 SVM（核函数）

http://blog.csdn.net/Love_wanling/article/details/69390047

2.25.6 常见的核函数有哪些？

http://blog.csdn.net/Love_wanling/article/details/69390047

本文将遇到的核函数进行收集整理，分享给大家。

http://blog.csdn.net/wsj998689aa/article/details/47027365

1. Linear Kernel

线性核是最简单的核函数，核函数的数学公式如下：

$$k(x,y)=x^{t}y$$

如果我们将线性核函数应用在 KPCA 中，我们会发现，推导之后和原始 PCA 算法一模一样，很多童鞋借此说“kernel is shit！！！”，这是不对的，这只是线性核函数偶尔会出现等价的形式罢了。

2. Polynomial Kernel

多项式核实一种非标准核函数，它非常适合于正交归一化后的数据，其具体形式如下：

$$k(x,y)=(ax^{t}y+c)^{d}$$

这个核函数是比较好用的，就是参数比较多，但是还算稳定。

3. Gaussian Kernel

这里说一种经典的鲁棒径向基核，即高斯核函数，鲁棒径向基核对于数据中的噪音有着较好的抗干扰能力，其参数决定了函数作用范围，超过了这个范围，数据的作用就“基本消失”。高斯核函数是这一族核函数的优秀代表，也是必须尝试的核函数，其数学形式如下：

$$k(x,y)=\exp(-\frac{\|x-y\|^{2}}{2\sigma^{2}})$$

虽然被广泛使用，但是这个核函数的性能对参数十分敏感，以至于有一大把的文献专门对这种核函数展开研究，同样，高斯核函数也有了很多的变种，如指数核，拉普拉斯核等。

4. Exponential Kernel

指数核函数就是高斯核函数的变种，它仅仅是将向量之间的 L2 距离调整为 L1 距离，这样改动会对参数的依赖性降低，但是适用范围相对狭窄。其数学形式如下：

$$k(x,y)=\exp(-\frac{\|x-y\|}{2\sigma^{2}})$$

5. Laplacian Kernel

拉普拉斯核完全等价于指数核，唯一的区别在于前者对参数的敏感性降低，也是一种径向基核函数。

$$k(x,y)=\exp(-\frac{\|x-y\|}{\sigma})$$

6. ANOVA Kernel

ANOVA 核也属于径向基核函数一族，其适用于多维回归问题，数学形式如下：

$$k(x,y)=\exp(-\sigma(x^k-y^k)^2)^d$$

7. Sigmoid Kernel

Sigmoid 核来源于神经网络，现在已经大量应用于深度学习，是当今机器学习的宠儿，它是 S 型的，所以被用作于“激活函数”。关于这个函数的性质可以说好几篇文献，大家可以随便找一篇深度学习的文章看看。

$$k(x,y)=\tanh(\alpha x^t y+c)$$

8. Rational Quadratic Kernel

二次有理核完完全全是作为高斯核的替代品出现，如果你觉得高斯核函数很耗时，那么不妨尝试一下这个核函数，顺便说一下，这个核函数作用域虽广，但是对参数十分敏感，慎用！！！！

$$k(x,y)=1-\frac{\|x-y\|^2}{\|x-y\|^2+c}$$

9. Multiquadric Kernel

多元二次核可以替代二次有理核，它是一种非正定核函数。

$$k(x,y)=(\|x-y\|^2+c^2)^{0.5}$$

10. Inverse Multiquadric Kernel

顾名思义，逆多元二次核来源于多元二次核，这个核函数我没有用过，但是据说这个基于这个核函数的算法，不会遇到核相关矩阵奇异的情况。

$$k(x,y)=(\|x-y\|^2+c^2)^{-0.5}$$

11. Circular Kernel

这个核函数没有用过，其数学形式如下所示：

$$k(x,y)=\frac{2}{\pi}\arccos(-\frac{\|x-y\|}{\sigma})-\frac{2}{\pi}\frac{\|x-y\|}{\sigma}(1-\frac{\|x-y\|^2}{\sigma})^{0.5}$$

12. Spherical Kernel

这个核函数是上一个的简化版，形式如下所示

$$k(x,y)=1-\frac{3}{2}\frac{\|x-y\|}{\sigma}+\frac{1}{2}(\frac{\|x-y\|^2}{\sigma})^3$$

13. Wave Kernel

这个核函数没有用过，其适用于语音处理场景。

$$k(x,y)=\frac{\theta}{||x-y||}\sin(\frac{||x-y||}{\theta})$$

14. Triangular Kernel

三角核函数感觉就是多元二次核的特例，数学公式如下：

$$k(x,y)=-||x-y||^d$$

15. Log Kernel

对数核一般在图像分割上经常被使用，数学形式如下：

$$k(x,y)=-\log(1+||x-y||^d)$$

16. Spline Kernel

$$k(x,y)=1+x^ty+x^ty\min(x,y)-\frac{x+y}{2}\min(x,y)^2+\frac{1}{3}\min(x,y)^3$$

17. Bessel Kernel

$$k(x,y)=\frac{J_{v+1}(\sigma||x-y||)}{||x-y||^{-n(v+1)}}$$

18. Cauchy Kernel

柯西核来源于神奇的柯西分布，与柯西分布相似，函数曲线上有一个长长的尾巴，说明这个核函数的定义域很广泛，言外之意，其可应用于原始维度很高的数据上。

$$k(x,y)=\frac{1}{||x-y||^2/\sigma+1}$$

19. Chi-Square Kernel

卡方核，这是我最近在使用的核函数，让我欲哭无泪，在多个数据集上都没有用，竟然比原始算法还要差劲，不知道为什么文献作者首推这个核函数，其来源于卡方分布，数学形式如下：

$$k(x,y)=1-\sum_{k=1}^{n}\frac{(x_k-y_k)^2}{0.5(x_k+y_k)}$$

它存在着如下变种：

$$k(x,y)=\frac{x^ty}{||x+y||_{L1}}$$

其实就是上式减去一项得到的产物，这个核函数基于的特征不能够带有赋值，否则性能会急剧下降，如果特征有负数，那么就用下面这个形式：

$$sign(x^t y)k(|x|,|y|)$$

20. Histogram Intersection Kernel

直方图交叉核在图像分类里面经常用到，比如说人脸识别，适用于图像的直方图特征，例如 extended LBP 特征其数学形式如下，形式非常的简单

$$k(x,y)=\sum_{k=1}^{n}\min(x_k,y_k)$$

21. Generalized Histogram Intersection

顾名思义，广义直方图交叉核就是上述核函数的拓展，形式如下：

$$k(x,y)=\sum_{k=1}^{n}\min(|x_k|^{\alpha},|y_k|^{\beta})$$

22. Generalized T-Student Kernel

TS 核属于 mercer 核，其数学形式如下，这个核也是经常被使用的

$$k(x,y)=\frac{1}{1+||x-y||^{d}}$$

23. Bayesian Kernel

贝叶斯核函数还没有用到过。

2.25.6 软间隔与正则化

2.25.7 SVM 主要特点及缺点？

http://www.elecfans.com/emb/fpga/20171118582139_2.html

3.3.2.1 SVM 有如下主要几个特点：

(1)非线性映射是 SVM 方法的理论基础,SVM 利用内积核函数代替向高维空间的非线性映射；

(2)对特征空间划分的最优超平面是 SVM 的目标,最大化分类边际的思想是 SVM 方法的核心；

(3)支持向量是SVM的训练结果,在SVM分类决策中起决定作用的是支持向量。(4)SVM 是一种有坚实理论基础的新颖的小样本学习方法。它基本上不涉及概率测度及大数定律等,因此不同于现有的统计方法。从本质上看,它避开了从归纳到演绎的传统过程,实现了高效的从训练样本到预报样本的“转导推理”,大大简化了通常的分类和回归等问题。

(5)SVM 的最终决策函数只由少数的支持向量所确定,计算的复杂性取决于支持向量的数目,而不是样本空间的维数,这在某种意义上避免了“维数灾难”。

(6)少数支持向量决定了最终结果,这不但可以帮助我们抓住关键样本、“剔除”大量冗余样本,而且注定了该方法不但算法简单,而且具有较好的“鲁棒”性。这种“鲁棒”性主要体现在:

①增、删非支持向量样本对模型没有影响;

②支持向量样本集具有一定的鲁棒性;

③有些成功的应用中,SVM 方法对核的选取不敏感

3.3.2.2 SVM 的两个不足:

(1) SVM 算法对大规模训练样本难以实施

由 于 SVM 是借助二次规划来求解支持向量,而求解二次规划将涉及 m 阶矩阵的计算(m 为样本的个数),当 m 数目很大时该矩阵的存储和计算将耗费大量的机器内存 和运算时间。针对以上问题的主要改进有有 J.Platt 的 SMO 算法、T.Joachims 的 SVM、C.J.C.Burges 等的 PCGC、张学工的 CSVM 以及 O.L.Mangasarian 等的 SOR 算法。

(2) 用 SVM 解决多分类问题存在困难

经典的支持向量机算法只给出了二类分类的算法,而在数据挖掘的实际应用中,一般要解决多类的分类问题。可以通过多个二类支持向量机的组合来解决。主要有一对多组合模式、一对一组合模式和 SVM 决策树; 再就是通过构造多个分类器的组合来解决。主要原理是克服 SVM 固有的缺点,结合其他算法的优势,解决多类问题的分类精度。如:与粗集理论结合,形成一种优势互补的多类问题的组合分类器。

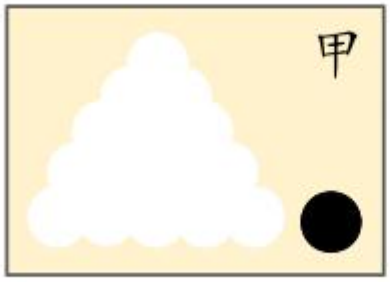
2.26 贝叶斯

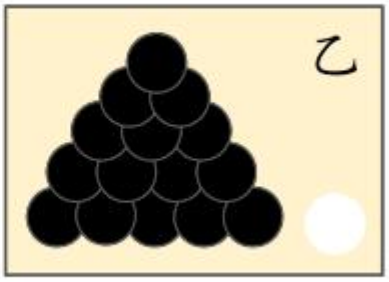
2.26.1 图解极大似然估计

极大似然估计 http://blog.csdn.net/zengxiantao1994/article/details/72787849

极大似然估计的原理,用一张图片来说明,如下图所示:

最大似然原理





- 例：有两个外形完全相同的箱子，甲箱中有99只白球，1只黑球；乙箱中有99只黑球，1只白球。一次试验取出一球，结果取出的是黑球。
- 问：黑球从哪个箱子中取出？
- 人们的第一印象就是：“此黑球最像是从乙箱中取出的”，这个推断符合人们的经验事实。“最像”就是“最大似然”之意，这种想法常称为“最大似然原理”（maximum-likelihood）。

总结起来，最大似然估计的目的就是：利用已知的样本结果，反推最有可能（最大概率）导致这样结果的参数值。

原理：极大似然估计是建立在极大似然原理的基础上的一个统计方法，是概率论在统计学中的应用。极大似然估计提供了一种给定观察数据来评估模型参数的方法，即：“模型已定，参数未知”。通过若干次试验，观察其结果，利用试验结果得到某个参数值能够使样本出现的概率为最大，则称为极大似然估计。

由于样本集中的样本都是独立同分布，可以只考虑一类样本集 D，来估计参数向量θ。记已知的样本集为：

$$D=\{x_1,x_2,\cdots,x_N\}$$

似然函数（linkehood function）：联合概率密度函数$p(D|\theta)$称为相对于$\{x_1,x_2,\cdots,x_N\}$的θ的似然函数。

$$l(\theta)=p(D|\theta)=p(x_1,x_2,\cdots,x_N|\theta)=\prod_{i=1}^{N}p(x_i|\theta)$$

如果$\hat{\theta}$是参数空间中能使似然函数$l(\theta)$最大的θ值，则$\hat{\theta}$应该是“最可能”的参数值，那么$\hat{\theta}$就是θ的极大似然估计量。它是样本集的函数，记作：

$$\hat{\theta}=d(x_1,x_2,\cdots,x_N)=d(D)$$

$\hat{\theta}(x_1,x_2,\cdots,x_N)$称作极大似然函数估计值

2.26.2 朴素贝叶斯分类器和一般的贝叶斯分类器有什么区别？

2.26.4 朴素与半朴素贝叶斯分类器

2.26.5 贝叶斯网三种典型结构

2.26.6 什么是贝叶斯错误率

2.26.7 什么是贝叶斯最优错误率

2.27 EM 算法解决问题及实现流程

1. EM 算法要解决的问题

我们经常会从样本观察数据中，找出样本的模型参数。 最常用的方法就是极大化模型分布的对数似然函数。

但是在一些情况下，我们得到的观察数据有未观察到的隐含数据，此时我们未知的有隐含

数据和模型参数，因而无法直接用极大化对数似然函数得到模型分布的参数。怎么办呢？这就是 EM 算法可以派上用场的地方了。

EM 算法解决这个的思路是使用启发式的迭代方法，既然我们无法直接求出模型分布参数，那么我们可以先猜想隐含数据（EM 算法的 E 步），接着基于观察数据和猜测的隐含数据一起来极大化对数似然，求解我们的模型参数（EM 算法的 M 步)。由于我们之前的隐藏数据是猜测的，所以此时得到的模型参数一般还不是我们想要的结果。不过没关系，我们基于当前得到的模型参数，继续猜测隐含数据（EM 算法的 E 步），然后继续极大化对数似然，求解我们的模型参数（EM 算法的 M 步)。以此类推，不断的迭代下去，直到模型分布参数基本无变化，算法收敛，找到合适的模型参数。

从上面的描述可以看出，EM 算法是迭代求解最大值的算法，同时算法在每一次迭代时分为两步，E 步和 M 步。一轮轮迭代更新隐含数据和模型分布参数，直到收敛，即得到我们需要的模型参数。

一个最直观了解 EM 算法思路的是 K-Means 算法，见之前写的 K-Means 聚类算法原理。在 K-Means 聚类时，每个聚类簇的质心是隐含数据。我们会假设 KK 个初始化质心，即 EM 算法的 E 步；然后计算得到每个样本最近的质心，并把样本聚类到最近的这个质心，即 EM 算法的 M 步。重复这个 E 步和 M 步，直到质心不再变化为止，这样就完成了 K-Means 聚类。

当然，K-Means 算法是比较简单的，实际中的问题往往没有这么简单。上面对 EM 算法的描述还很粗糙，我们需要用数学的语言精准描述。

EM 算法流程

现在我们总结下 EM 算法的流程。

输入：观察数据 x=(x(1),x(2),...x(m))x=(x(1),x(2),...x(m))，联合分布 p(x,z|θ)p(x,z|θ)，条件分布 p(z|x,θ)p(z|x,θ)，最大迭代次数 JJ。

1) 随机初始化模型参数θθ的初值θ0θ0。

2） for j from 1 to J 开始 EM 算法迭代：

a) E 步：计算联合分布的条件概率期望：

Qi(z(i))=P(z(i)|x(i)，θj))Qi(z(i))=P(z(i)|x(i)，θj))

L(θ,θj)=∑i=1m∑z(i)Qi(z(i))logP(x(i)，z(i)|θ)L(θ,θj)=∑i=1m∑z(i)Qi(z(i))logP(x(i)，z(i)|θ)

b) M 步：极大化 L(θ,θj)L(θ,θj),得到θj+1θj+1:

θj+1=argmaxθL(θ,θj)θj+1=argmaxθL(θ,θj)

c) 如果θj+1θj+1 已收敛，则算法结束。否则继续回到步骤 a)进行 E 步迭代。

输出：模型参数θθ。

2.28 为什么会产生维数灾难？

http://blog.csdn.net/chenjianbo88/article/details/52382943

假设地球上猫和狗的数量是无限的。由于有限的时间和计算能力，我们仅仅选取了 10 张照片作为训练样本。我们的目的是基于这 10 张照片来训练一个线性分类器，使得这个线性分类器可以对剩余的猫或狗的照片进行正确分类。我们从只用一个特征来辨别猫和狗开始：

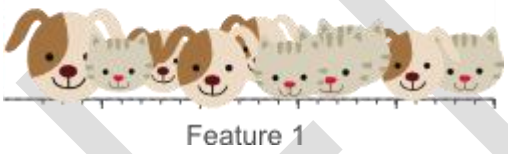


图 2

从图 2 可以看到，如果仅仅只有一个特征的话，猫和狗几乎是均匀分布在这条线段上，很难将 10 张照片线性分类。那么，增加一个特征后的情况会怎么样：

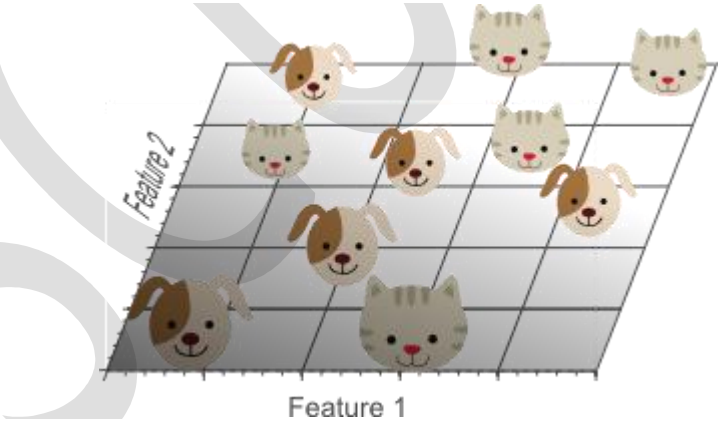


图 3

增加一个特征后，我们发现仍然无法找到一条直线将猫和狗分开。所以，考虑需要再增加一个特征：

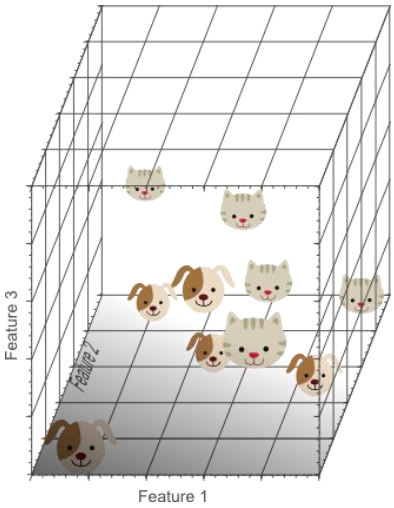


图 3

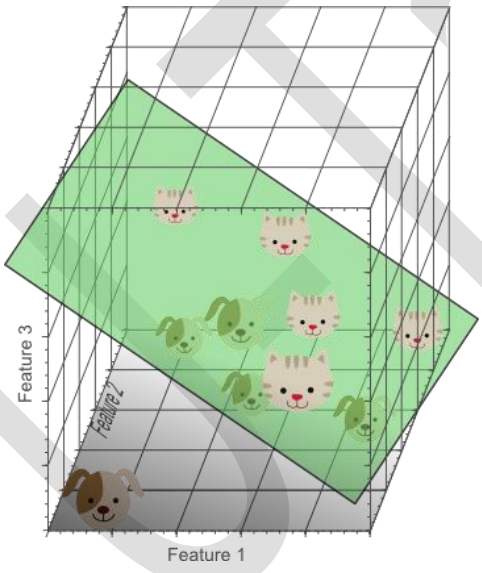


图 4

此时，我们终于找到了一个平面将猫和狗分开。需要注意的是，只有一个特征时，假设特征空间是长度为 5 的线段，则样本密度是 10/5=2。有两个特征时，特征空间大小是 5*5=25，样本密度是 10/25=0.4。有三个特征时，特征空间大小是 5*5*5=125，样本密度是 10/125=0.08。如果继续增加特征数量，样本密度会更加稀疏，也就更容易找到一个超平面将训练样本分开。因为随着特征数量趋向于无限大，样本密度非常稀疏，训练样本被分错的可能性趋向于零。当我们将高维空间的分类结果映射到低维空间时，一个严重的问题出现了：

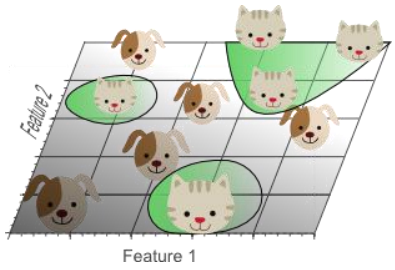


图 5

从图 5 可以看到将三维特征空间映射到二维特征空间后的结果。尽管在高维特征空间时训练样本线性可分，但是映射到低维空间后，结果正好相反。事实上，增加特征数量使得高维空间线性可分，相当于在低维空间内训练一个复杂的非线性分类器。不过，这个非线性分类

器太过“聪明”，仅仅学到了一些特例。如果将其用来辨别那些未曾出现在训练样本中的测试样本时，通常结果不太理想。这其实就是我们在机器学习中学过的过拟合问题。

尽管图 6 所示的只采用 2 个特征的线性分类器分错了一些训练样本，准确率似乎没有图 4 的高，但是，采用 2 个特征的线性分类器的泛化能力比采用 3 个特征的线性分类器要强。因为，采用 2 个特征的线性分类器学习到的不只是特例，而是一个整体趋势，对于那些未曾出现过的样本也可以比较好地辨别开来。换句话说，通过减少特征数量，可以避免出现过拟合问题，从而避免“维数灾难”。

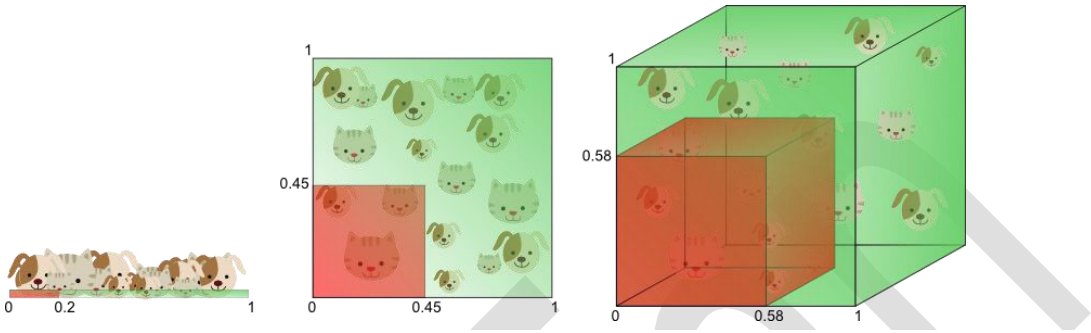


图 7

图 7 从另一个角度诠释了“维数灾难”。假设只有一个特征时，特征的值域是 0 到 1，每一只猫和狗的特征值都是唯一的。如果我们希望训练样本覆盖特征值值域的 20%，那么就需要猫和狗总数的 20%。我们增加一个特征后，为了继续覆盖特征值值域的 20%就需要猫和狗总数的 45%(0.45^2=0.2)。继续增加一个特征后，需要猫和狗总数的 58%(0.58^3=0.2)。随着特征数量的增加，为了覆盖特征值值域的 20%，就需要更多的训练样本。如果没有足够的训练样本，就可能会出现过拟合问题。

通过上述例子，我们可以看到特征数量越多，训练样本就会越稀疏，分类器的参数估计就会越不准确，更加容易出现过拟合问题。“维数灾难”的另一个影响是训练样本的稀疏性并不是均匀分布的。处于中心位置的训练样本比四周的训练样本更加稀疏。

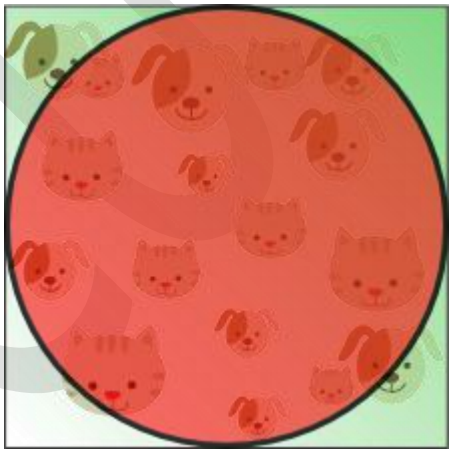


图 8

假设有一个二维特征空间，如图 8 所示的矩形，在矩形内部有一个内切的圆形。由于越接近圆心的样本越稀疏，因此，相比于圆形内的样本，那些位于矩形四角的样本更加难以分类。那么，随着特征数量的增加，圆形的面积会不会变化呢？这里我们假设超立方体(hypercube)的边长 d=1，那么计算半径为 0.5 的超球面(hypersphere)的体积(volume)的公式为：

$$V(d) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} 0.5^d.$$

公式 1

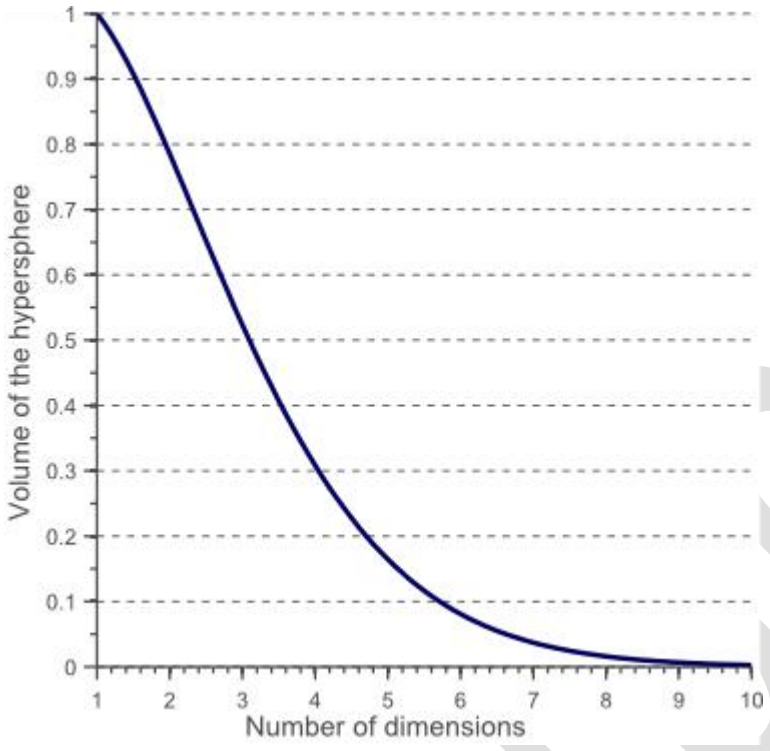


图 9

从图 9 可以看出随着特征数量的增加，超球面的体积逐渐减小直至趋向于零，然而超立方体的体积却不变。这个结果有点出乎意料，但部分说明了分类问题中的"维数灾难"：在高维特征空间中，大多数的训练样本位于超立方体的角落。

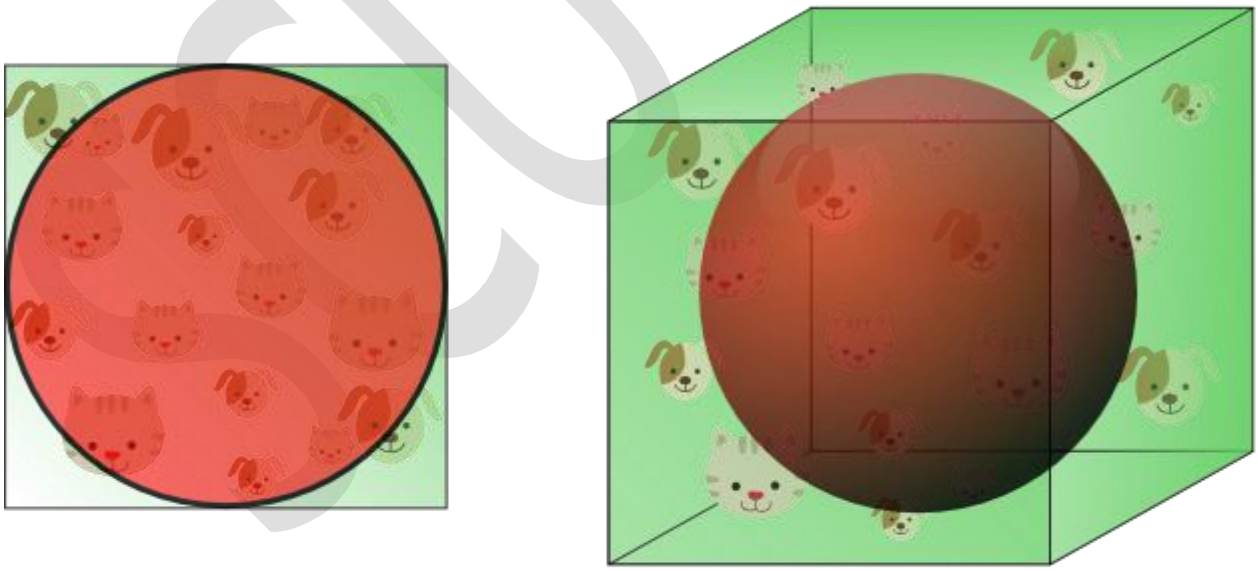


图 10

图 10 显示了不同维度下，样本的分布情况。在 8 维特征空间中，共有 2^8=256 个角落，而 98%的样本分布在这些角落。随着维度的不断增加，公式 2 将趋向于 0，其中 dist_max 和 dist_min 分别表示样本到中心的最大与最小距离。

$$\lim_{d\to\infty} \frac{\text{dist}_{\max} - \text{dist}_{\min}}{\text{dist}_{\min}} \to 0$$

公式 2

因此，在高维特征空间中对于样本距离的度量失去意义。由于分类器基本都依赖于如 Euclidean 距离，Manhattan 距离等，所以在特征数量过大时，分类器的性能就会出现下降。

所以，我们如何避免"维数灾难"？图 1 显示了分类器的性能随着特征个数的变化不断增加，过了某一个值后，性能不升反降。这里的某一个值到底是多少呢？目前，还没有方法来确定分类问题中的这个阈值是多少，这依赖于训练样本的数量，决策边界的复杂性以及分类器的类型。理论上，如果训练样本的数量无限大，那么就不会存在"维数灾难"，我们可以采用任意多的特征来训练分类器。事实上，训练样本的数量是有限的，所以不应该采用过多的特征。此外，那些需要精确的非线性决策边界的分类器，比如 neural network，knn，decision trees 等的泛化能力往往并不是很好，更容易发生过拟合问题。因此，在设计这些分类器时应当慎重考虑特征的数量。相反，那些泛化能力较好的分类器，比如 naive Bayesian，linear classifier 等，可以适当增加特征的数量。

如果给定了 N 个特征，我们该如何从中选出 M 个最优的特征？最简单粗暴的方法是尝试所有特征的组合，从中挑出 M 个最优的特征。事实上，这是非常花时间的，或者说不可行的。其实，已经有许多特征选择算法(feature selection algorithms)来帮助我们确定特征的数量以及选择特征。此外，还有许多特征抽取方法(feature extraction methods)，比如 PCA 等。交叉验证(cross-validation)也常常被用于检测与避免过拟合问题。

参考资料：

[1] Vincent Spruyt. The Curse of Dimensionality in classification. Computer vision for dummies. 2014. [Link]

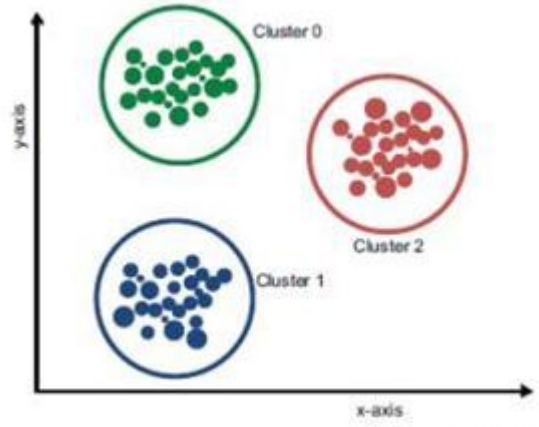
2.29 怎样避免维数灾难

解决维度灾难问题：
主成分分析法 PCA，线性判别法 LDA
奇异值分解简化数据、拉普拉斯特征映射
Lassio 缩减系数法、小波分析法、

2.30 聚类和降维有什么区别与联系？

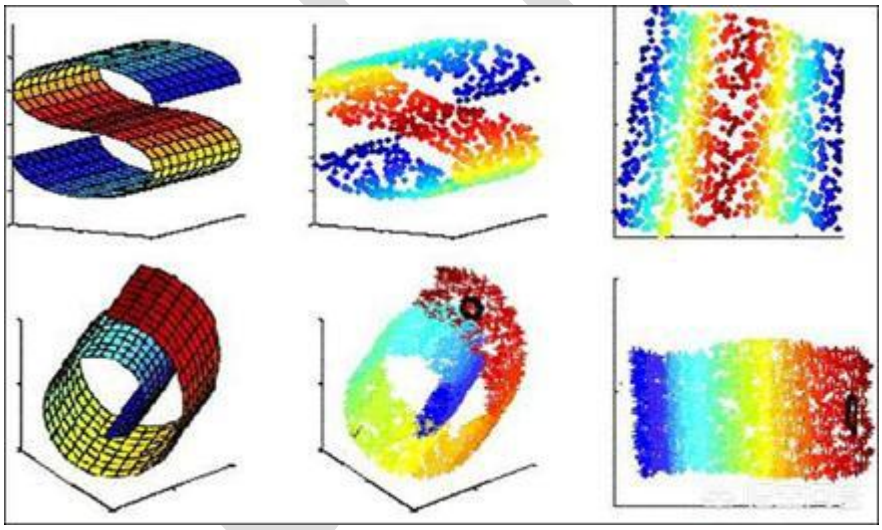
聚类用于找寻数据内在的分布结构,既可以作为一个单独的过程，比如异常检测等等。也可作为分类等其他学习任务的前驱过程。聚类是标准的无监督学习。

1) 在一些推荐系统中需确定新用户的类型，但定义"用户类型"却可能不太容易,此时往往可先对原油的用户数据进行聚类,根据聚类结果将每个簇定义为一个类,然后再基于这些类训练分类模型,用于判别新用户的类型。



2）而降维则是为了缓解维数灾难的一个重要方法，就是通过某种数学变换将原始高维属性空间转变为一个低维“子空间”。其基于的假设就是，虽然人们平时观测到的数据样本虽然是高维的，但是实际上真正与学习任务相关的是个低维度的分布。从而通过最主要的几个特征维度就可以实现对数据的描述，对于后续的分类很有帮助。比如对于 Kaggle 上的泰坦尼克号生还问题。通过给定一个人的许多特征如年龄、姓名、性别、票价等，来判断其是否能在海难中生还。这就需要首先进行特征筛选，从而能够找出主要的特征，让学习到的模型有更好的泛化性。

聚类和降维都可以作为分类等问题的预处理步骤。



但是他们虽然都能实现对数据的约减。但是二者适用的对象不同，聚类针对的是数据点，而降维则是对于数据的特征。另外它们着很多种实现方法。聚类中常用的有 K-means、层次聚类、基于密度的聚类等；降维中常用的则 PCA、Isomap、LLE 等。

2.31 GBDT 和随机森林的区别

GBDT 和随机森林的相同点：

1、都是由多棵树组成

2、最终的结果都是由多棵树一起决定

GBDT 和随机森林的不同点：

1、组成随机森林的树可以是分类树，也可以是回归树；而 GBDT 只由回归树组成

2、组成随机森林的树可以并行生成；而 GBDT 只能是串行生成

3、对于最终的输出结果而言，随机森林采用多数投票等；而 GBDT 则是将所有结果累加起来，或者加权累加起来

4、随机森林对异常值不敏感，GBDT 对异常值非常敏感

5、随机森林对训练集一视同仁，GBDT 是基于权值的弱分类器的集成

6、随机森林是通过减少模型方差提高性能，GBDT 是通过减少模型偏差提高性能

2.32 四种聚类方法之比较

http://www.cnblogs.com/William_Fire/archive/2013/02/09/2909499.html

聚类分析是一种重要的人类行为，早在孩提时代，一个人就通过不断改进下意识中的聚类模式来学会如何区分猫狗、动物植物。目前在许多领域都得到了广泛的研究和成功的应用，如用于模式识别、数据分析、图像处理、市场研究、客户分割、Web 文档分类等[1]。

聚类就是按照某个特定标准(如距离准则)把一个数据集分割成不同的类或簇，使得同一个簇内的数据对象的相似性尽可能大，同时不在同一个簇中的数据对象的差异性也尽可能地大。即聚类后同一类的数据尽可能聚集到一起，不同数据尽量分离。

聚类技术[2]正在蓬勃发展，对此有贡献的研究领域包括数据挖掘、统计学、机器学习、空间数据库技术、生物学以及市场营销等。各种聚类方法也被不断提出和改进，而不同的方法适合于不同类型的数据，因此对各种聚类方法、聚类效果的比较成为值得研究的课题。

1 聚类算法的分类

目前，有大量的聚类算法[3]。而对于具体应用，聚类算法的选择取决于数据的类型、聚类的目的。如果聚类分析被用作描述或探查的工具，可以对同样的数据尝试多种算法，以发现数据可能揭示的结果。

主要的聚类算法可以划分为如下几类：划分方法、层次方法、基于密度的方法、基于网格的方法以及基于模型的方法[4-6]。

每一类中都存在着得到广泛应用的算法，例如：划分方法中的 k-means[7]聚类算法、层次方法中的凝聚型层次聚类算法[8]、基于模型方法中的神经网络[9]聚类算法等。

目前,聚类问题的研究不仅仅局限于上述的硬聚类，即每一个数据只能被归为一类，模糊聚类[10]也是聚类分析中研究较为广泛的一个分支。模糊聚类通过隶 属函数来确定每个数据隶属于各个簇的程度，而不是将一个数据对象硬性地归类到某一簇中。目前已有很多关于模糊聚类的算法被提出，如著名的 FCM 算法等。

本文主要对 k-means 聚类算法、凝聚型层次聚类算法、神经网络聚类算法之 SOM,以及模糊聚类的 FCM 算法通过通用测试数据集进行聚类效果的比较和分析。

2 四种常用聚类算法研究

2.1 k-means 聚类算法

k-means 是划分方法中较经典的聚类算法之一。由于该算法的效率高，所以在对大规模数据进行聚类时被广泛应用。目前，许多算法均围绕着该算法进行扩展和改进。

k-means 算法以 k 为参数，把 n 个对象分成 k 个簇，使簇内具有较高的相似度，而簇间的相似度较低。k-means 算法的处理过程如下：首先，随机地 选择 k 个对象，每个对象初始地代表了一个簇的平均值或中心;对剩余的每个对象，根据其与各簇中心的距离，将它赋给最近的簇;然后重新计算每个簇的平均值。 这个过程不断重复，直到准则函数收敛。通常，采用平方误差准则，其定义如下：

$$E=\sum_{i=1}^{k}\sum_{p\in C_i}|p-m_i|^2$$

这里 E 是数据库中所有对象的平方误差的总和，p 是空间中的点，mi 是簇 Ci 的平均值[9]。该目标函数使生成的簇尽可能紧凑独立，使用的距离度量是欧几里得距离,当然也可以用其他距离度量。k-means 聚类算法的算法流程如下：

输入：包含 n 个对象的数据库和簇的数目 k;

输出：k 个簇，使平方误差准则最小。

步骤：

(1) 任意选择 k 个对象作为初始的簇中心；

(2) repeat；

(3) 根据簇中对象的平均值，将每个对象(重新)赋予最类似的簇；

(4) 更新簇的平均值，即计算每个簇中对象的平均值；

(5) until 不再发生变化。

2.2 层次聚类算法

根据层次分解的顺序是自底向上的还是自上向下的，层次聚类算法分为凝聚的层次聚类算法和分裂的层次聚类算法。

凝聚型层次聚类的策略是先将每个对象作为一个簇，然后合并这些原子簇为越来越大的簇，直到所有对象都在一个簇中，或者某个终结条件被满足。绝大多数层次聚类属于凝聚型层次聚类，它们只是在簇间相似度的定义上有所不同。四种广泛采用的簇间距离度量方法如下：

最小距离：

$$d_{\min}(c_i,c_j)=\min_{p\in c_i,p'\in c_j}|p-p'|$$

最大距离：

$$d_{\max}(c_i,c_j)=\max_{p\in c_i,p'\in c_j}|p-p'|$$

平均值的距离：

$$d_{\text{mean}}(c_i,c_j)=|m_i-m_j|$$

平均距离：

$$d_{\text{avg}}(c_i,c_j)=\frac{1}{n_i n_j}\sum_{p\in c_i}\sum_{p'\in c_j}|p-p'|$$

这里，$|p-p'|$是两个对象 p 和 p'之间的距离，m_i 是簇 c_i 的平均值，n_i 是簇 c_i 中对象的数目。

这里给出采用最小距离的凝聚层次聚类算法流程：

(1) 将每个对象看作一类，计算两两之间的最小距离；

(2) 将距离最小的两个类合并成一个新类；
(3) 重新计算新类与所有类之间的距离；
(4) 重复(2)、(3)，直到所有类最后合并成一类。

2.3 SOM 聚类算法

SOM 神经网络[11]是由芬兰神经网络专家 Kohonen 教授提出的，该算法假设在输入对象中存在一些拓扑结构或顺序，可以实现从输入空间(n 维)到输出平面(2 维)的降维映射，其映射具有拓扑特征保持性质,与实际的大脑处理有很强的理论联系。

SOM 网络包含输入层和输出层。输入层对应一个高维的输入向量，输出层由一系列组织在 2 维网格上的有序节点构成，输入节点与输出节点通过权重向量连接。 学习过程中，找到与之距离最短的输出层单元，即获胜单元，对其更新。同时，将邻近区域的权值更新，使输出节点保持输入向量的拓扑特征。

算法流程：

(1) 网络初始化，对输出层每个节点权重赋初值；
(2) 将输入样本中随机选取输入向量，找到与输入向量距离最小的权重向量；
(3) 定义获胜单元，在获胜单元的邻近区域调整权重使其向输入向量靠拢；
(4) 提供新样本、进行训练；
(5) 收缩邻域半径、减小学习率、重复，直到小于允许值，输出聚类结果。

2.4 FCM 聚类算法

1965 年美国加州大学柏克莱分校的扎德教授第一次提出了‘集合’的概念。经过十多年的发展，模糊集合理论渐渐被应用到各个实际应用方面。为克服非此即彼的分类缺点，出现了以模糊集合论为数学基础的聚类分析。用模糊数学的方法进行聚类分析，就是模糊聚类分析[12]。

FCM 算法是一种以隶属度来确定每个数据点属于某个聚类程度的算法。该聚类算法是传统硬聚类算法的一种改进。

设数据集 $X=\{x_1,x_2,\cdots x_n\}$，它的模糊 c 划分可用模糊矩阵 $\boldsymbol{U}=[u_{ij}]$ 表示，矩阵 $\boldsymbol{U}$ 的元素 u_{ij} 表示第 $j(j=1,2,\cdots,n)$ 个数据点属于第 $i(i=1,2,\cdots,c)$ 类的隶属度，u_{ij} 满足如下条件：

$$\forall j,\sum_{i=1}^{c}u_{ij}=1;\forall i,ju_{ij}\in[0,1];\forall i,\sum_{j=1}^{n}u_{ij}>0$$

目前被广泛使用的聚类准则为取类内加权误差平方和的极小值，即：

$$(\min)J_m(U,V)=\sum_{j=1}^{n}\sum_{i=1}^{c}u_{ij}^m d_{ij}^2(x_j,v_i)$$

其中 V 为聚类中心，m 为加权指数，

$$d_{ij}(x_j,v_i)=\|v_i-x_j\|$$

算法流程：

(1) 标准化数据矩阵；
(2) 建立模糊相似矩阵，初始化隶属矩阵；
(3) 算法开始迭代，直到目标函数收敛到极小值；
(4) 根据迭代结果，由最后的隶属矩阵确定数据所属的类，显示最后的聚类结果。

3 四种聚类算法试验

3.1 试验数据

实验中，选取专门用于测试分类、聚类算法的国际通用的 UCI 数据库中的 IRIS[13]数据集，IRIS 数据集包含 150 个样本数据，分别取自三种不同 的莺尾属植物 setosa、versicolor 和 virginica 的花朵样本,每个数据含有 4 个属性，即萼片长度、萼片宽度、花瓣长度，单位为 cm。在数据集上执行不同的聚类算法，可以得到不同精度的聚类结果。

3.2 试验结果说明

文中基于前面所述各算法原理及算法流程，用 matlab 进行编程运算，得到表 1 所示聚类结果。

表 1 三种聚类方法的实验对比结果

聚类方法	聚错样本数	运行时间/s	平均准确度/(%)
k－means	17	0.146 001	89
层次聚类	51	0.128 744	66
FCM	12	0.470 417	92
SOM	22	5.267 283	86

如表 1 所示，对于四种聚类算法，按三方面进行比较：(1)聚错样本数：总的聚错的样本数，即各类中聚错的样本数的和；(2)运行时间：即聚类整个 过程所耗费的时间，单位为 s；(3)平均准确度：设原数据集有 k 个类,用 ci 表示第 i 类，ni 为 ci 中样本的个数，mi 为聚类正确的个数,则 mi/ni 为 第 i 类中的精度，则平均精度为：

$$\text{avg}=\frac{1}{k}\sum_{i=1}^{k} m_i/n_i$$