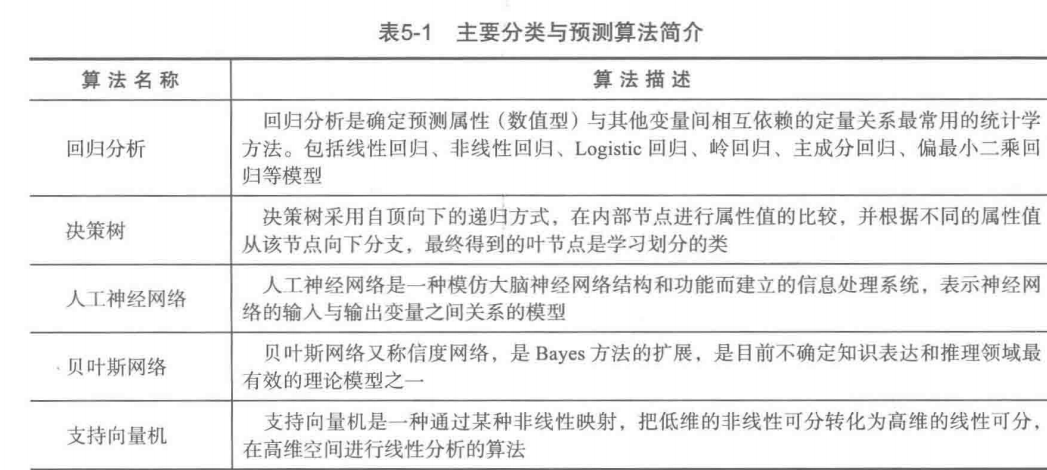
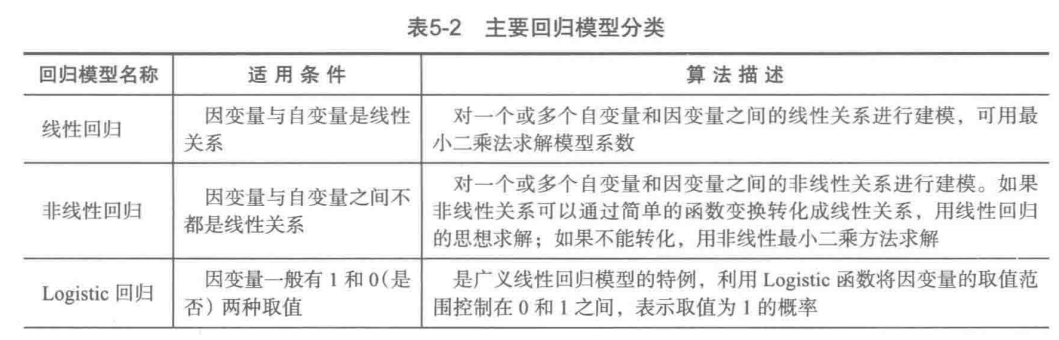
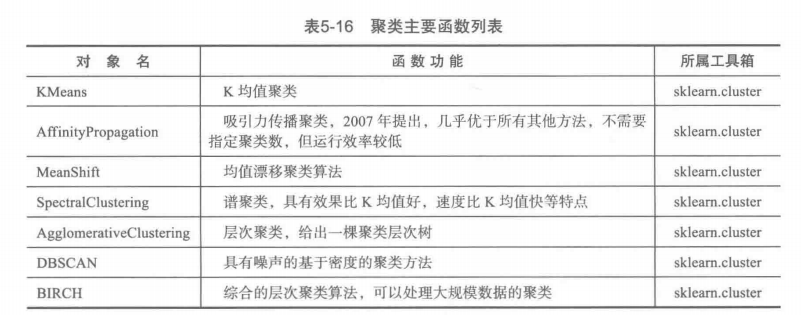
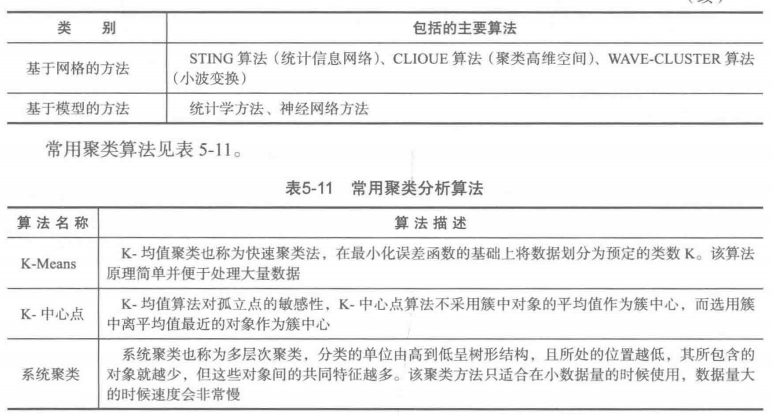
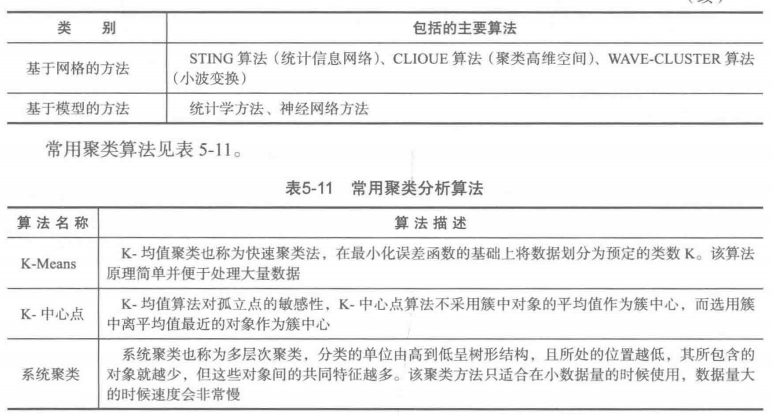
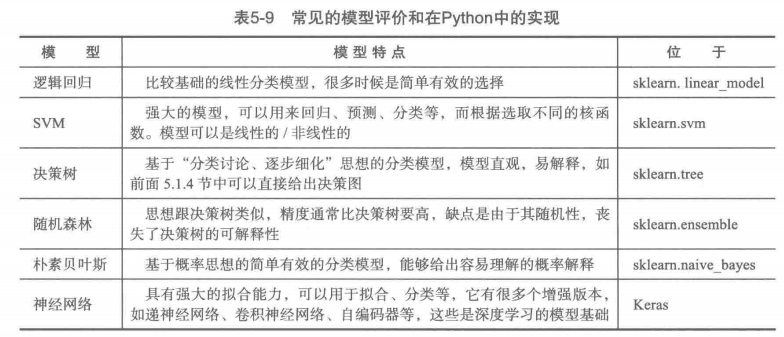
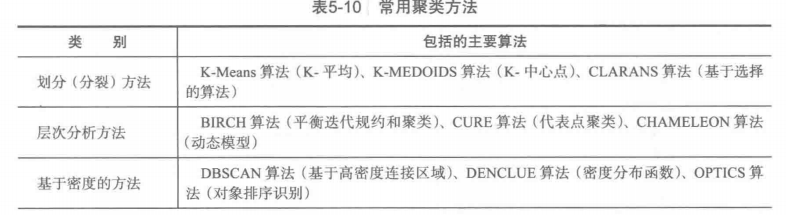
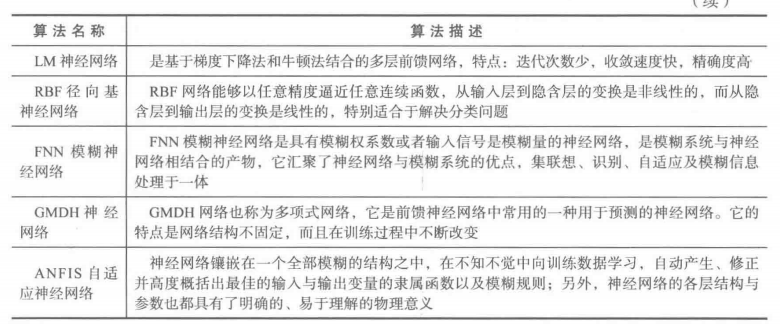
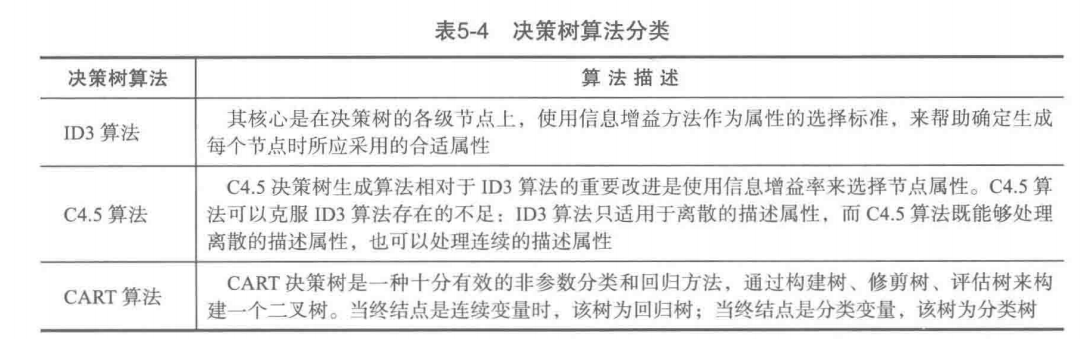
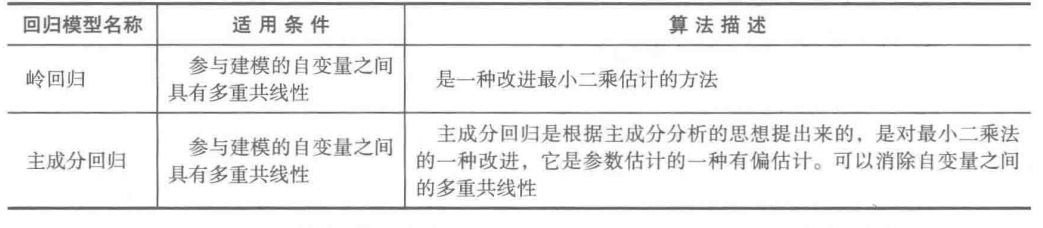
# 数据分析算法

* **黎建文，2018-03-03**

# 纲领







## 分类

## 回归

## 聚类分析

## 关联规则

## 异常检测

# 数据预处理——缺值处理

## EM算法

### 预定义

#### (1) Lazy Statistician规则

设是随机变量的函数，（是连续函数），那么

如果是离散型随机变量，它的分布律为，若绝对收敛，则有

如果是连续型随机变量，它的概率密度函数为，若绝对收敛，则有

#### (2) Jensen不等式

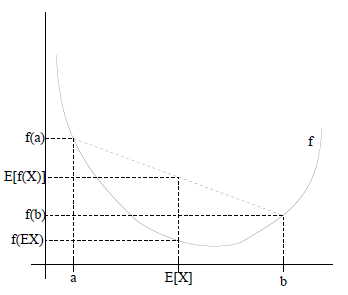
如果函数满足对定义域上任意两个数都有，那么为凸函数。开口向下的是凹,开口向上的是凸。

对于任意的凸函数以及其定义域上个数那么都有

对于任意的凹函数以及其定义域上个数,那么都有

如果上面凹凸是严格的，那么不等式的等号只有才成立

用简洁的话来说，对于凸函数来说，函数值的期望值大于等于自变量期望值的函数值。



#### (3) 极大似然

假设随机变量服从参数为的某种分布，密度函数为，的值未知。现在观测到一组样本数据、，……，，要用样本数据估算这一组的值。有多少个参数要估计，就应该要有多少组样本数组。根据不同的样本数据组，估算出来的参数是不一样的。

取似然函数

取似然函数的极值点，联立解方程组

，得到这个

极大似然法的思想是：每组样本是独立的，观察组样本，这组样本联合分布率为各个样本分布率的乘积(独立发生)。而既然取到样本了，证明事件发生了，因此取联合分布律最大值时的取值，就是对的估算结果了。

#### (4) 算法原理

假设随机变量的概率分布为，未知。需要估算。

取似然函数



假设我们的观察数据中还有一个隐变量，在次取样中的值分别为，

含隐变量的似然函数为（边缘分布）



**由于含有隐变量，无法使用极大似然法来求解。要求解，先要知道。但要求解又先要知道。这是鸡生蛋，蛋生鸡的问题。为了打破这个循环，EM算法的思路是：先假设一个的先验分布以及观察到的和的联合分布，根据贝叶斯法则，估算出的后验分布后，参与到的计算中，以取得极大似然点的一个最小值。得到的估算，然后根据和观察到的和的联合分布，再进行下一轮的迭代，直至最终收敛。**

我们假设满足某个参数已知的分布，。



上述这个推导是根据边缘分布、Lazy Statistician规则和Jensen不等式变换出来的。重点理解这一变换。

首先，由对数的性质：，得到①->②的转换：



接着，做一个小处理，得到②->③的转换：



然后，我们假定是已知的，构造出满足分布为的的一个函数，根据Lazy Statistician规则，有，令,log函数为凹函数，根据Jensen不等式，有，展开为原式，得到：

，即有



所以，两边取和式，得到③->④：



也就是说是似然函数的一个下界，当

，为一个常数时，取得下界。

将移到右边，有，左右两边积分， ，由于，，所以有，将的替换为，就得到，为已知的，根据贝叶斯公式， 的本质就是，**就是后验概率**。这时，****就可以计算了。

#### (5) 收敛性

http://www.cnblogs.com/jerrylead/archive/2011/04/06/2006936.html

### 算法描述

(1) E步：对于每一个，计算

(2) M步：根据极大似然法则计算的估算



### 算法适用场景

缺点：对初始值敏感：EM算法需要初始化参数，而参数的选择直接影响收敛效率以及能否得到全局最优解。

## 缺失数据添加

### 神经网络拟合

### 添补法

### 卡尔曼滤波

### EM算法

# 数据预处理——离散连续处理

## 离散数据连续化

## 连续数据离散化

## 枚举数据编码

### One-Hot

### 哑编码

# 数据预处理——异常值处理

# 数据预处理——多重共线性

解释变量理论上的高度相关与观测值高度相关没有必然关系，有可能两个解释变量理论上高度相关，但观测值未必高度相关，反之亦然。所以多重共线性本质上是数据问题。

## 产生原因

造成多重共线性的原因有一下几种：

1、解释变量都享有共同的时间趋势；

2、一个解释变量是另一个的滞后，二者往往遵循一个趋势；

3、由于数据收集的基础不够宽，某些解释变量可能会一起变动；

4、某些解释变量间存在某种近似的线性关系；

## 判断方法

1、计算模型中各对自变量之间的相关系数，如果一个或多个相关系数是显著的，就表示存在多重共线性。

2、当模型的线性关系检验(F检验)显著时，几乎所有回归系数β的t检验却不显著。

3、回归系数的正负号与预期的相反。

4、容忍度(tolerance)与方差扩大因子(VIF)。某个自变量的容忍度等于1减去该自变量为因变量而其他自变量为预测变量时所得到的线性回归模型的判定系数。容忍度越小，多重共线性越严重。通常认为容忍度小于0.1时，存在严重的多重共线性。方差扩大因子等于容忍度的倒数。显然，VIF越大，多重共线性越严重。一般认为VIF大于10时，存在严重的多重共线性。

## 处理原则

1、多重共线性是普遍存在的，轻微的多重共线性问题可不采取措施；

2、严重的多重共线性问题，一般可根据经验或通过分析回归结果发现。如影响系数符号，重要的解释变量t值很低。要根据不同情况采取必要措施。

3、如果模型仅用于预测，则只要拟合程度好，可不处理多重共线性问题，存在多重共线性的模型用于预测时，往往不影响预测结果；

## 处理手段

1、将一个或多个相关的自变量从模型中删除。实际操作中常用逐步法作为自变量筛选筛选按方法。

2、如果哟啊在模型中保留所有的自变量，那就应该：避免根据t统计量对单个参数β进行检验；对因变量y值得推断限定在自变量样本值的范围内。

3、主成份分析：PCA

4、偏最小二乘法。偏最小二乘法≈多元线性回归分析+典型相关分析+主成分分析

5、岭回归法。岭回归法是通过最小二乘法的改进允许回归系数的有偏估计量存在而补救多重共线性的方法。

6、增加样本容量。多重共线性问题的实质是样本信息的不充分而导致模型参数的不能精确估计，因此追加样本信息是解决该问题的一条有效途径。

# 数据预处理——特征筛选

## 最佳特征数目的确定

## WOE和IV

https://zhuanlan.zhihu.com/p/30026040

### 特征筛选的考虑

挑选入模变量过程是个比较复杂的过程，需要考虑的因素很多，比如：变量的预测能力，变量之间的相关性，变量的简单性（容易生成和使用），变量的强壮性（不容易被绕过），变量在业务上的可解释性（被挑战时可以解释的通）等等。但是，其中最主要和最直接的衡量标准是变量的预测能力。

可以用来衡量自变量的预测能力

### WOE的定义

是对原始自变量的一种编码形式。

     要对一个变量进行编码，需要首先把这个变量进行分组处理/离散化处理（等宽切割，等高切割，或者利用决策树来切割）。分组后，对于第组，的计算公式如下：

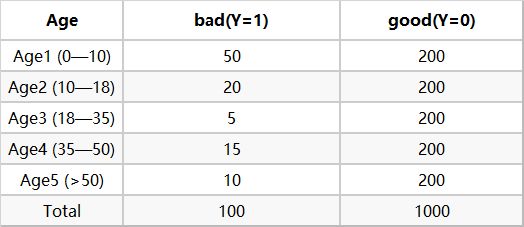


表示某变量第组对应的值；表示某变量第组属性对应的正样本数，表示某变量第组属性对应的负样本数，表示样本中总正样本数，表示样本中总正样本数。

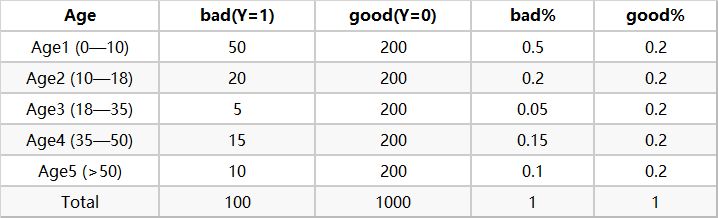
### WOE的计算例子

以信用评分卡的建模场景为例：X是客户样本字段，Y表示客户逾期与否，其中Y=1代表逾期，Y=0代表未逾期。 我们希望能用客户已知的信息来预测客户借款后发生逾期的概率，以此来决定是否放贷。

下面我们拿Age(年龄)这个变量来计算相关的 ，首先对每个level分层统计【表1】：



然后计算各分层的好坏占比【表2】



最后通过好坏占比计算【表3】



以上三个表就是计算的过程，简单易懂。

### WOE的作用

表示的实际上是“当前分组中坏客户占所有坏客户的比例”和“当前分组中好客户占所

有坏客户的比例”的差异。转化公式以后，也可以理解为：当前这个组中坏客户和好客户的比值，和所有样本中这个比值的差异。这个差异为这两个比值的比值，再取对数来表示的。

越大，这种差异越大，这个分组里的样本坏样本可能性就越大，越小，差异越小，这个分组里的坏样本可能性就越小。

当前分组中，坏样本比例越大，值越大

当前分组的正负，由当前分组坏样本和好样本的比例，与样本整体坏样本和好样本的比例的大小关系决定，当前分组的比例小于样本整体比例时，为负，当前分组的比例大于整体比例时，为正，当前分组的比例和整体比例相等时，为0。

的取值范围是全体实数。

其实描述了变量当前这个分组，对判断个体是否属于坏样本所起到影响方向和大小。当为正时，变量当前取值对判断个体是否会响应起到的正向的影响，当为负时，起到了负向影响。而值的大小，则是这个影响的大小的体现。

### IV的计算

一个变量的值：



### IV值为+∞处理

其实有一个缺点，就是不能自动处理变量的分组中出现响应比例为0或100%的情况。遇到坏样本比例为0或者100%的情况，建议如下：

如果可能，直接把这个分组做成一个规则，作为模型的前置条件或补充条件；

重新对变量进行离散化或分组，使每个分组的响应比例都不为0且不为100%，尤其是当一个分组个体数很小时（比如小于100个），强烈建议这样做，因为本身把一个分组个体数弄得很小就不是太合理。

如果上面两种方法都无法使用，建议人工把该分组的响应数和非响应的数量进行一定的调整。如果响应数原本为0，可以人工调整响应数为1，如果非响应数原本为0，可以人工调整非响应数为1。

### IV信息量大小与指标判别力有一个经验的规则

若信息量取值小于0.02，认为该指标对因变量没有预测能力，应该被剔除。

若信息量取值在0.02与0.1之间，认为该指标对因变量有较弱的预测能力。

若信息量取值在0.1与0.3之间，认为该指标对因变量的预测能力一般。

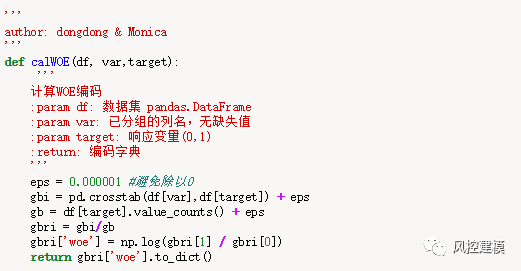
若信息量取值大于0.3，认为该指标对因变量有较强的预测能力。

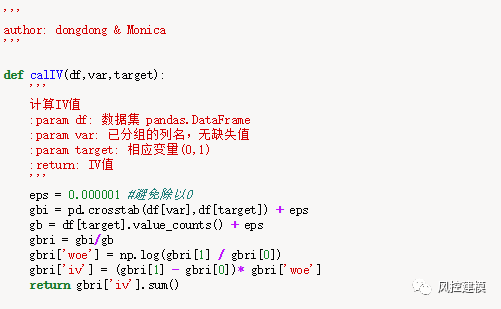
实际应用时，可以保留值大于0.1的指标。

### Python计算

<https://www.sohu.com/a/227312722_479788>









# 数据预处理——特征提取

## 降维必要性

降维的必要性：

(1) 随着数据维度不断降低，数据存储所需的空间也会随之减少。

(2) 低维数据有助于减少计算/训练用时。

(3) 一些算法在高维度数据上容易表现不佳，降维可提高算法可用性。

(4) 降维可以用删除冗余特征解决**多重共线性**问题。

(5) 降维有助于数据可视化。

<https://www.jqr.com/article/000439>

## 降维方法的比较

PCA：计算代价高昂，特征向量得存在线性相关。

核PCA: 特征向量是非线性相关也可以。

SVD：比PCA更能解释数据，因为是直接作用于原数据集，不会像PCA一样，将相关变量转换为一系列不相干的变量。另外，PCA是单模因子分析方法，行列代表的是相同的实体， 而SVD是双模因子（即适用两类实体矩阵），可以运用在文本挖掘中，行对应词，列对应文档。

高斯随机映射：速度快，利用欧氏距离降维，但数据多会有内存问题，可以考虑稀疏随机映射代替。

NMF：常见于推荐系统，输入矩阵A = 降维矩阵（行）A\_dash \* 成本矩阵（列） F。

## 特征选择和特征抽取

特征选择即经过某种法则直接扔掉某些特征，特征抽取即利用映射的方法，将高维度的样本映射至低维度。

## 低方差滤波

## 高相关滤波

## 随机森林

## 前向特征选择

## 反向特征消除

## 因子分析

## PCA

### 预定义

#### (1) 降维

设数据的维度为，希望降到的维度为。

根据线性代数的结论：维向量的表示，可以看做是维单位正交基{、、}的坐标，寻找一个合适的线性变换，使得。这个线性变换，也可以看做是个维向量，每一行是一个维行向量，每一行的表示可以看做是维单位正交基{、、}的坐标。而的表示，是相对于向量组来说的。因此，有：

，和的本质是一样的。只不过，是以单位正交基为参照系来表示，而是以为参照系来表示。而本身的描述却是以单位正交基为参照系来表示。

这就是矩阵和向量相乘的意义：将列向量变换到左边矩阵中每一行行向量为基所表示的空间中去。选择不同的基可以对同样一组数据给出不同的表示，而且如果基的数量少于向量本身的维数，则可以达到**降维**的效果。

#### (2) 特定的基下表达一组数据所需要的维度

假设一组数据有3个样本点，并且是3个属性值，分别为，，，如果一组基是，那么表达这组数据至少需要三维才能保证信息不丢失。如果这组基是或与共线的任意基，那么表达这组数据至少需要一维才能保证信息不丢失。如果这组基是共平面但非共线的两个基，那么表达这组数据至少需要两维才能保证信息不丢失。

**因此，表达一组数据至少需要多少维度才能不丢失信息，需要根据这组数据的分布来构造特定的基，使得信息不丢失的情况下，维度最少。理论上，对于特定的分布，在信息不丢失的情况下，可以计算出最少维度的特定基。当工程需要的维度比这个不丢失信息的最少维度还要少的时候，就要发生信息丢失了。对于特定的这个维度，也是可以计算出特定的基并选择这组基下的特定维度，使得信息丢失尽可能少。**

#### (3) 方差

方差反映的是某个维度上数据的分散程度



通常，在PCA解决过程中，先对数据进行0均值处理，会使计算变得简单

#### (4) 协方差

协方差计算的是不同维度之间的协方差，而不是不同样本之间的协方差。协方差反映一组数据中两个不同属性间变化方向一致性。在0均值处理后：



为了让协方差为0，我们选择第二个基时只能在与第一个基正交的方向上选择。因此最终选择的两个方向一定是正交的。

#### (5) 协方差矩阵

假设一组数据有个样本，每个样本用这个属性描述，用矩阵表示这组数据

这组数据的协方差矩阵为：

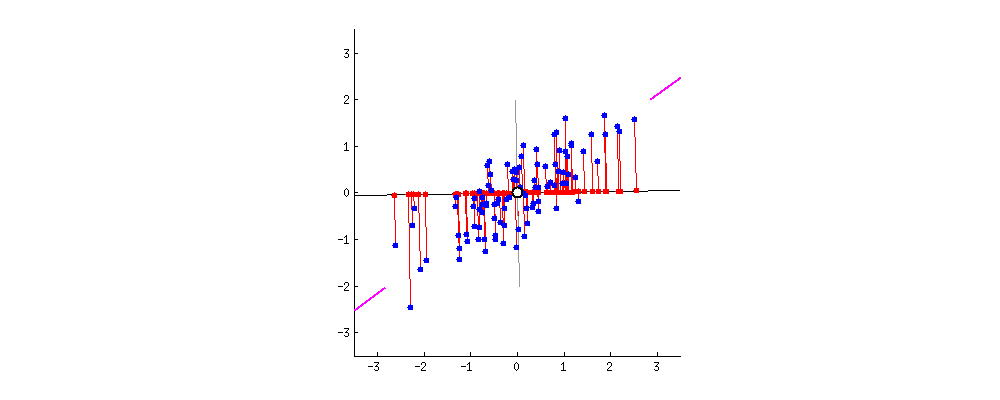


对进行0均值处理后，

为协方差矩阵

#### (6) PCA原理

设原始数据经0均值处理后用矩阵表达为，为行列构成的数据矩阵(表示有个样本，每个样本有个属性)，新的基为，为行列的矩阵，的每一行为一个基。为原数据在新基下的表达矩阵，，为行列的矩阵，每一列为变换后的样本数据。当时，达到降维的效果。



怎样的降维才是好的？我们希望表达的样本数据，经变换后，信息不失真，并且不同样本点能够最大可能地区分。从协方差的视角，希望表达的样本数据，不同属性间两两协方差为0，而每个属性的方差尽可能大。也就是以构造的协方差矩阵，对角线上尽量不为零，非对角线上均为0，也就是为一个对角阵。另为以构造的协方差矩阵。

根据等价变换：



我们的目标是寻找使以构造的协方差矩阵得以对角化的，并且对角元素按从大到小依次排列。对角元素的排序不同，所产生的也不同。为什么要从小到大排序呢？因为上面说了，对角元素是的属性值的方差，我们希望方差越大越好，自然希望对角元素按左上角到右下角的顺序从大到小排序，这样，我们就知道哪些属性分量携带的信息比较多，当迫于无奈必须进行取舍的时候，我们可以舍弃携带信息比较少的维度，也就是比较小的对角元所对应的维度。假如对角元素按左上角到右下角的顺序从大到小排序后，我们取前个维度，实际上，这个维度的数据是根据的前行生成的。

剩下的问题就是根据构造了，而是一个实对称矩阵。在线性代数里有比较好的性质。

**说白了，PCA就是选择一个合适的投影方法，将数据从高维投影到低维，使得数据的主要特征信息能够尽可能得到保留。**

#### (7) 实对称矩阵的一些好的性质

设为阶实对称矩阵，必然存在正交阵，使得

实对称矩阵不同特征值对应的向量必然正交

设特征向量的重数为，则必然存在个线性无关的的特征向量对应于，因此可以将这个特征向量单位正交化。

#### (8) 求P

根据样本数据构造阶的协方差矩阵，一定可以找到个单位正交的特征向量，用其构造出，使得，而所求的。

### 算法描述

设有条维数据。

(1) 将原始数据按列组成行列矩阵。

(2) 将的每一行（代表一个属性字段）进行零均值化，即减去这一行的均值。并进行标准化处理(归一)

(3) 求出协方差矩阵。

(4) 求出协方差矩阵的特征值及对应的特征向量。

(5) 将特征向量按对应特征值大小从上到下按行排列成矩阵，取前行组成矩阵。

(6) 即为从维降到维后的数据。

### 算法适用场景

PCA其实是最简单的降维方法之一了，很明显的劣势是它仅去除数据之间的线性相关性(多重共线性)。对线性的改善往往通过kernel技术拓展到非线性的应用上。另外，PCA的这种降维不一定有助于分类，用于分类的降维方法之一就是LDA。从另一方面说，PCA是一种线性投影，保留了数据与数据之间的欧式距离，即原来欧式距离大的两点在降维后的空间中距离也应大（这样才好保证方差大）。而事实上数据有可能呈现某种流型结构，用PCA降维后数据将不能保持原有的流型结构。

**PCA是单因子分析方法，被分析的是同一类实体，这些实体有共同的属性集。跟SVD不同，SVD是双因子分析方法。**

### 算法实现

|  |
| --- |
| *#coding=utf-8* **from** numpy **import** \*  **class** PCA:  *'''通过方差的百分比来计算将数据降到多少维是比较合适的，函数传入的参数是特征值和百分比percentage，返回需要降到的维度数num'''* @staticmethod  **def** eig\_val\_pct(eig\_vals, percentage):  *# 使用numpy中的sort()对特征值按照从小到大排序* sortArray=sort(eig\_vals)   *# 特征值从大到小排序* sortArray=sortArray[-1::-1]   *# 数据全部的方差arraySum* arraySum = sum(sortArray)   tempSum=0  num=0  **for** i **in** sortArray:  tempSum += i  num += 1  **if** tempSum >= arraySum\*percentage:  **return** num   **return** num   @staticmethod  **def** pca\_by\_dim(high\_dim\_data\_matrix, dim = 2):   *# 对每一列求平均值，因为协方差的计算中需要减去均值* mean\_matrix = mean(high\_dim\_data\_matrix, axis = 0)   *# 0均值处理* zeromean\_matrix = high\_dim\_data\_matrix - mean\_matrix   *# 求最大最小值* matrix\_min, matrix\_max = zeromean\_matrix.min(), zeromean\_matrix.max()  *# (矩阵元素-最小值)/(最大值-最小值),得到标准化的数据(归一)* normaliz\_matrix = (zeromean\_matrix - matrix\_min) / (matrix\_max - matrix\_min)   *# 协方差矩阵* cov\_matrix = cov(normaliz\_matrix, rowvar = 1)   *# 利用numpy中寻找特征值和特征向量的模块linalg中的eig()方法  # mat函数是将对象转换为matrix类型* eig\_vals, eig\_vects = linalg.eig(mat(cov\_matrix))   k = dim   *# 对特征值eig\_vals从小到大排序* eig\_val\_ind = argsort(eig\_vals)   *# 从排好序的特征值，从后往前取k个，这样就实现了特征值的从大到小排列* eig\_val\_ind = eig\_val\_ind[:-(k+1):-1]   *# 返回排序后特征值对应的特征向量red\_eig\_vects（主成分）* red\_eig\_vects = eig\_vects[:,eig\_val\_ind]   *# 将原始数据投影到主成分上得到新的低维数据low\_dim\_data\_matrix* low\_dim\_data\_matrix = red\_eig\_vects.T \* normaliz\_matrix   **return** low\_dim\_data\_matrix, eig\_vals, eig\_vects,eig\_val\_ind,k,red\_eig\_vects.T    **'''pca函数有两个参数，其中high\_dim\_data\_matrix是已经转换成矩阵matrix形式的数据集，列表示特征；其中的percentage表示取前多少个特征需要达到的方差占比，默认为0.9'''** @staticmethod  **def** pca\_by\_percentage(high\_dim\_data\_matrix, percentage = 0.9):   *# 对每一列求平均值，因为协方差的计算中需要减去均值* mean\_matrix = mean(high\_dim\_data\_matrix, axis = 0)   *# 0均值处理* zeromean\_matrix = high\_dim\_data\_matrix - mean\_matrix   *# 求最大最小值* matrix\_min, matrix\_max = zeromean\_matrix.min(), zeromean\_matrix.max()  *# (矩阵元素-最小值)/(最大值-最小值),得到标准化的数据(归一)* normaliz\_matrix = (zeromean\_matrix - matrix\_min) / (matrix\_max - matrix\_min)   *# 协方差矩阵* cov\_matrix = cov(normaliz\_matrix, rowvar = 1)   *# 利用numpy中寻找特征值和特征向量的模块linalg中的eig()方法  # mat函数是将对象转换为matrix类型* eig\_vals, eig\_vects = linalg.eig(mat(cov\_matrix))   *# 要达到方差的百分比percentage，需要前k个向量* k = PCA.eig\_val\_pct(eig\_vals, percentage)    *# 对特征值eig\_vals从小到大排序* eig\_val\_ind = argsort(eig\_vals)   *# 从排好序的特征值，从后往前取k个，这样就实现了特征值的从大到小排列* eig\_val\_ind = eig\_val\_ind[:-(k+1):-1]   *# 返回排序后特征值对应的特征向量red\_eig\_vects（主成分）* red\_eig\_vects = eig\_vects[:,eig\_val\_ind]   *# 将原始数据投影到主成分上得到新的低维数据low\_dim\_data\_matrix* low\_dim\_data\_matrix = red\_eig\_vects.T \* normaliz\_matrix   **return** low\_dim\_data\_matrix, eig\_vals, eig\_vects,eig\_val\_ind,k,red\_eig\_vects.T |

## KPCA

https://www.cnblogs.com/engineerLF/p/5393087.html

## LDA

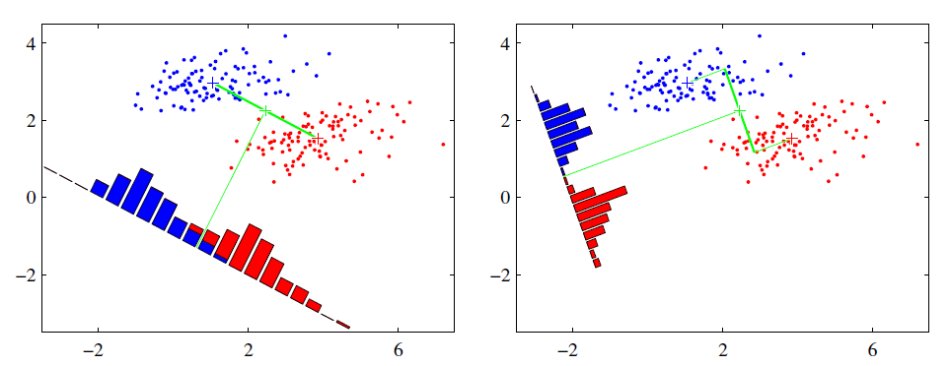
参照https://www.cnblogs.com/engineerLF/p/5393119.html

### 预定义

#### (1) 带类标签的降维

PCA算法，从高维到低维，是不带有类标签的。但有一些场景，从高维降到低维，希望保留类标签，并且低维数据也线性可分。

如下面，将二维降到一维后，左边的降维方式，降维后的数据是线性不可分的。而

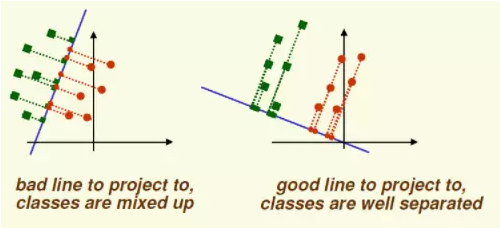


### 算法描述

### 算法适用场景

LDA假设：1.样本数据服从正态分布，2.各类得协方差相等。虽然这些在实际中不一定满足，但是LDA被证明是非常有效的降维方法，其线性模型对于噪音的鲁棒性效果比较好，不容易过拟合。

LDA和PCA的区别：



(1) 左边是 PCA，属于无监督方法，当数据没有标签时可以用它。右边是 LDA，属于监督方法。考虑了数据的分类信息，这样数据在低维空间上就可以分类了，减少了很多的运算量。

(2) PCA 主要是从特征的协方差角度考虑，追求的是在降维之后能够最大化保持数据的内在信息。它不考虑分类信息，因此，降低维度后，信息损失降到最低，但分类上可能会变得更加困难。LDA 追求的是降维后的数据点尽可能容易被区分。降维后的样本数据在新的维度空间有最大的类间距离和最小的类内方差，数据在低维空间有最佳的可分离性。

(3) PCA 后的维度数目是和数据维度相关的，原始数据是 n 维，那么 PCA 后维度为 1、2～n 维。LDA 后的维度数目是和类别的个数相关的，原始数据是 n 维，一共有 C 个类别，那么 LDA 后维度为 1、2～C-1 维。

(4) PCA 投影的坐标系都是正交的。LDA 关注分类能力，不保证投影到的坐标系是正交的。

### 算法实现

## SVD

### 预定义

#### (1) 矩阵(方阵)的特征分解(EVD)

一个方阵，会存在一组特征值及其对应的特征向量(这组特征向量两两正交)，使得,其中，是以各特征值为主元的对角阵，而是以中按序的特征值对应的特征向量为列向量组成的矩阵。这种的分解为特征值分解。

可以理解为，一个方阵是一个线性变换，如果其对应个特征值和个特征向量，那么这个特征向量就是的个变化方向，而对应的特征值就表示这个变化的强度，也就是这个变化有多重要。

#### (2) 矩阵的奇异值分解(SVD)

特征值只能针对方阵，但现实中很多矩阵都不是方阵。奇异值分解是一个能适用于任意矩阵特征提取的一种分解的方法。

的SVD分解为：寻找和,使得：，其中和都是正交矩阵，且，，，，除了主对角线上的元素以外全为0

 和 的列分别叫做的 左奇异向量和右奇异向量， 的对角线上的值叫做  的奇异值。

的列由的单位化过的特征向量构成

的列由的单位化过的的特征向量构成

的对角元素来源于或的特征值的平方根，并且是按从大到小的顺序排列。

求解SVD分解的过程为：

求的特征值和特征向量，用单位化的特征向量构成

求的特征值和特征向量，用单位化的特征向量构成

将或者的特征值求平方根，然后构成。

通过SVD可以得到PCA相同的结果，但是SVD通常比直接使用PCA更稳定。

#### (3) 矩阵的LU分解

#### (4) 矩阵的QR分解

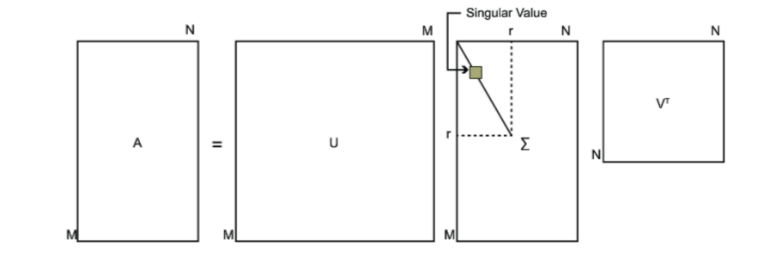
#### (5) 矩阵的极分解

#### (6) SVD原理

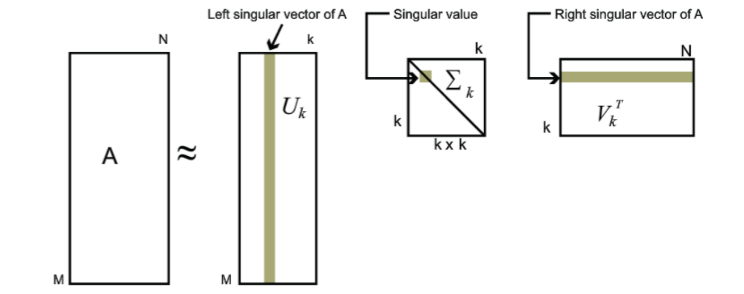
参考：<https://blog.csdn.net/zhongkejingwang/article/details/43053513>

https://www.cnblogs.com/pinard/p/6251584.html

若将矩阵看做一个变换，那么任何这样的变换可以看做是两个旋转和一个缩放变换的复合。，通过对进行调整实现降维，通过通过和在高维和低维间转换。



对于奇异值,它跟我们特征分解中的特征值类似，在奇异值矩阵中也是按照从大到小排列，而且奇异值的减少特别的快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上的比例。也就是说，我们也可以用最大的个的奇异值和对应的左右奇异向量来近似描述矩阵。也就是说：



### 算法描述

### 算法适用场景

### 算法实现

## ICA

## IOSMAP

## t-SNE

## UMAP

## Sammon映射

## LLE

## CCA

## MVU

## 拉普拉斯特征图

## MDS

## 稀疏自编码Autoencoder

# 数据预处理——离群点分析

# 建模——聚类

https://blog.csdn.net/xiayto/article/details/79562503

## 距离公式

### 闵可夫斯基距离



当p为1时是曼哈顿距离：

当p为2时是欧式距离：

当p为无穷大时，是切比雪夫距离：

### 余弦相似度



### KL距离



### Lance距离

### 杰卡德相似系数



### Pearson相关系数



## 聚类衡量指标

### 均一性（类似于精确率）：p

一个簇中只包含一个类别的样本，则满足均一性。其实也可以认为就是正确率(每个 聚簇中正确分类的样本数占该聚簇总样本数的比例和)

### 完整性（类似于召回率）：r

同类别样本被归类到相同簇中，则满足完整性;每个聚簇中正确分类的样本数占该

### V-measure



### 轮廓系数

簇内不相似度：计算样本i到同簇其它样本的平均距离为ai，应尽可能小。

簇间不相似度：计算样本i到其它簇Cj的所有样本的平均距离bij，应尽可能大。

轮廓系数：si值越接近1表示样本i聚类越合理，越接近-1，表示样本i应该分类到 另外的簇中，近似为0，表示样本i应该在边界上;所有样本的si的均值被成为聚类结果的轮廓系数。



### 操作上

sklearn中的Silhouette Coefficient和Calinski-Harabaz Index

## 聚类的公理化描述

例如聚类是什么？聚类算法和输入一个空间输出一个空间分布的任意函数的区别是什么？聚类有没有一些基本性质是独立于具体算法或任务的。

回答这些问题的一种方式是公理化方法，考虑一个聚类函数 ，将任意有限域及不相似函数作为输入，返回的一个划分，考虑这类函数的三种特性：

1、尺度不变形()

对任意的域集，不相似函数，以及任意的，下式成立：

尺度不变形是一种非常自然的要求，因为如果聚类函数输出的结果依赖于测量点之间的距离测度单元，将会显得十分奇怪和不合理。

2、丰富性()

对任意的优先集和划分（划分到非空子集），存在多种不相似函数使得

丰富性要求主要想说明聚类函数的输出是由函数全权决定，也是一种十分直观的特征。

3、一致性()

如果和都是上的不相似函数，对任一，根据：

如果属于同一类，则

如果不属于同一类，则

那么

一致性要求是和聚类基本定义相关的要求，我们希望相似的点聚到一类，不相似的点分属不同类，因此共享同类的点更相似，已经分离的点不相似，聚类函数应当对之前的聚类决策有很强的“支撑”作用。

不存在一个函数同时满足上述三种属性：尺度不变性、丰富性和一致性。

## 对聚类的典型泛化要求

不同的算法有着不同的应用背景，有的适合大数据集，可以发现任意形状的聚类；有的算法思想简单直观，适用于小数据集。总的来说，算法都试图从不同途径实现对数据集进行高效、可靠的聚类。数据挖掘对聚类的典型要求包括：

### 可伸缩性

当聚类对象由几百上升到几百万，我们希望最后的聚类结果的准确度能保持一致

### 处理不同类型属性的能力

有些聚类算法，其处理对象的属性的数据类型只能是数值类型，但是在实际应用场景中，我们往往会遇到其他类型的数据（例如二元数据），分类数据等等，虽然我们也可以在预处理数据时将这些其他类型的数据转换成数值型数据，但是在聚类效率上或者聚类准确度上往往会有折损

### 发现任意形状的类簇

因为许多聚类算法是基于距离（例如欧式距离或曼哈顿距离）来量化实例对象之间的相似度的，基于这种方式，我们往往只能发现相似尺寸和密度的球状类簇或者凸形类簇。但是在很多场景下，类簇的形状可能是任意的。

### 对聚类算法初始化参数的知识需求的最小化

很多算法在分析过程中需要开发者提供一定的参数（例如期望的类簇K个数、类簇初始质心），这导致了聚类结果对这些参数是十分敏感的，这不仅加重了开发者的负担，也非常影响聚类结果的准确性

### 处理噪声数据的能力

所谓的噪声数据，可以理解为影响聚类结果的干扰数据，这些噪声数据的存在会造成聚类结果的“畸变”，最终导致低质量的聚类

### 增量聚类和对输入次序的不敏感

一些聚类算法不能将新加入的数据插入到已有的聚类结果，输入次序的敏感是指，对于给定的数据对象集合，以不同的次序提供输入对象时，最终产生的聚类结果的差异会比较大

**7. 高维性**

有些算法只适合处理2维或者3维的数据，而对高维数据的处理能力很弱，因为在高维空间中数据的分布可能十分稀疏，而且高度倾斜。

**8. 基于约束的聚类**

在实际应用中可能需要在各种条件下进行聚类，因为同一个聚类算法，在不同的应用场景中所带来的聚类结果也是各异的，因此找到满足“特定约束”的具有良好聚类特性的数据分组是十分有挑战的。这里最困难的问题就在于如何是识别我们要解决的问题中隐含的“特定约束”具体是什么，以及该使用什么算法来最好的“适配”这种约束

**9. 可解释性和可用性**

希望得到的聚类结果都能用特定的语义、知识进行解释，和实际的应用场景相联系

## 数据分布

### 聚类趋势与霍普金斯统计量Hopkins

如果数据的分布是均匀的，就没有必要聚类了。

均匀地从数据集的空间中抽出个点，也就是空间中每个点都以相同的概率包含在这个样本中。对于每个点，我们找出在中的最近邻，并令为与它在中最近邻之间的欧式距离，即。

均匀地从的一个子集中抽到个点，对于每个点，我们找出在中的最近邻，并令为与它在中最近邻之间的欧式距离，即。

计算霍普金斯统计量

霍普金斯统计量告诉我们数据集有多大可能遵循数据空间的均匀分布

如果是均匀分布，和会很接近，因而得到的的值大约为0.5。

如果是高度倾斜的，则会显著小于，因而的值会接近0

|  |
| --- |
| **import** numpy **as** np  *#读取数据* X = [] f = open(**'cluster.txt'**)  **for** v **in** f:  X.append(  [float(v.split(**','**)[-2]),  float(v.split(**','**)[-1])]) *#转换成numpy array* X = np.array(X)  *#归一化* a = X[:,:1]/18.68\*100 b = X[:,1:]/17.54\*100  X = np.concatenate((a,b),axis = 1)  *#随机挑选出5个* pn = X[np.random.choice(X.shape[0],5,replace = False),:] xn = [] **for** i **in** pn:  distance\_min = 100  **for** j **in** X:  **if** np.array\_equal(i, j):  **continue** distance = np.linalg.norm(j-i)  **if** distance\_min>distance:  distance\_min = distance  xn.append(distance\_min)  *#随机挑选5个* qn = X[np.random.choice(X.shape[0],5,replace = False),:] yn = [] **for** i **in** qn:  distance\_min = 100  **for** j **in** X:  **if** np.array\_equal(i, j):  **continue** distance = np.linalg.norm(j-i)  **if** distance\_min>distance:  distance\_min = distance  yn.append(distance\_min)  H = float(np.sum(yn))/(float(np.sum(xn))+float(np.sum(yn))) **print**(H) |

### 簇的形状

高维空间判断簇的形状是很难的。

一般是降维到2维或者3维空间来看

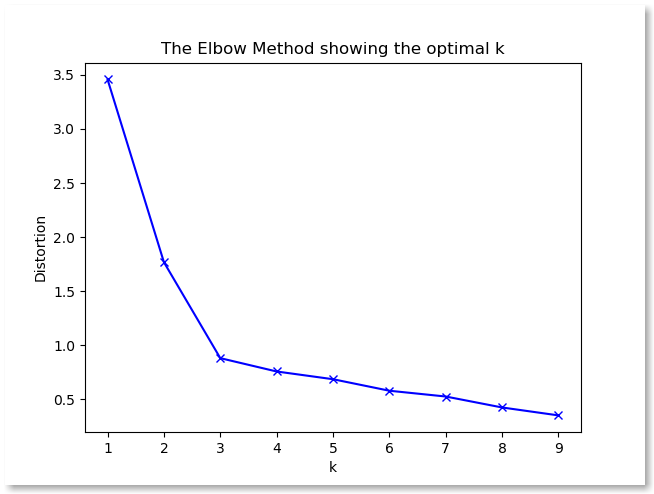
### 簇的数目

https://www.leiphone.com/news/201710/bULnj1olCI0dd3Pb.html

#### (1) 肘部法则(Elbow Method)

肘部法则会把不同值的成本函数值画出来。随着值的增大，平均畸变程度会减小；每个类包含的样本数会减少，于是样本离其重心会更近。但是，随着值继续增大，平均畸变程度的改善效果会不断减低。值增大过程中，畸变程度的改善效果下降幅度最大的位置对应的值就是肘部。

|  |
| --- |
| **from** sklearn.cluster **import** KMeans **from** sklearn **import** metrics **from** scipy.spatial.distance **import** cdist **import** numpy **as** np **import** matplotlib.pyplot **as** plt  *# 样本数据* x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8]) x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])  *# 查看样本数据分布* plt.plot() plt.xlim([0, 10]) plt.ylim([0, 10]) plt.title(**'Dataset'**) plt.scatter(x1, x2) plt.show()  *# 创建绘图和数据* plt.plot() X = np.array(list(zip(x1, x2))).reshape(len(x1), 2) colors = [**'b'**, **'g'**, **'r'**] markers = [**'o'**, **'v'**, **'s'**]  *# 迭代地去尝试K的取值和误差* distortions = [] K = range(1,10) **for** k **in** K:  kmeanModel = KMeans(n\_clusters=k).fit(X)  kmeanModel.fit(X)  distortions.append(sum(np.min(cdist(X, kmeanModel.cluster\_centers\_, **'euclidean'**), axis=1)) / X.shape[0])  *# Plot the elbow* plt.plot(K, distortions, **'bx-'**) plt.xlabel(**'k'**) plt.ylabel(**'Distortion'**) plt.title(**'The Elbow Method showing the optimal k'**) plt.show() |



#### (2) 经验法则



#### (3) Silhouette Method

#### (4) 信息论方法与信息准则

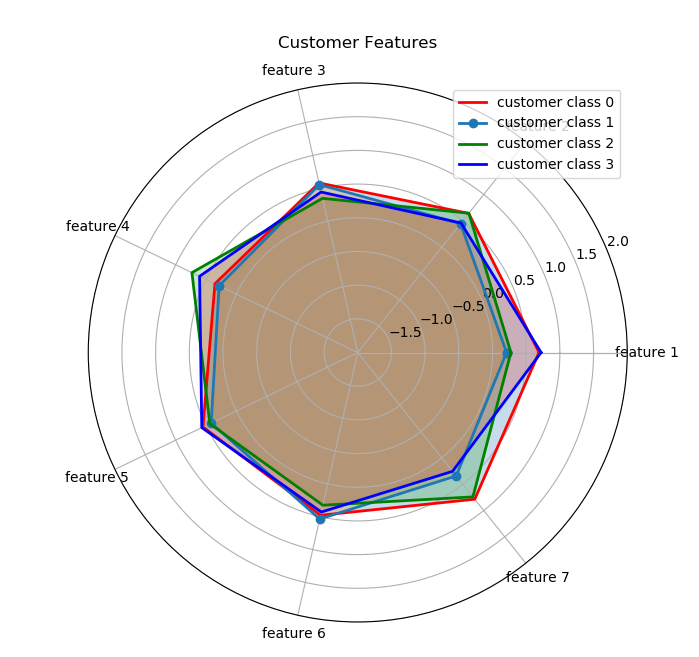
#### (5) 率失真理论

率失真理论 （Rate distortion theory）应用于选择k，通过信息理论标准最小化误差的同时最大化效率。 该策略通过运行一个标准的聚类算法为输入数据在k值从1到n生成一个失真曲线（distortion curve），接着基于数据维数选择的a negative power对失真曲线处理，最后寻找跳跃最大的点作为k。

#### (6) 交叉验证法

## 聚类结果的可视化

|  |
| --- |
| **import** matplotlib.pyplot **as** plt **import** numpy **as** np **from** sklearn.cluster **import** KMeans data = np.random.rand(100, 7) *#生成一个随机数据，样本大小为100, 特征数为5  #假如我要构造一个聚类数为3的聚类器* estimator = KMeans(n\_clusters=4)*#构造聚类器* estimator.fit(data)*#聚类* label\_pred = estimator.labels\_ *#获取聚类标签* centroids = estimator.cluster\_centers\_ *#获取聚类中心* inertia = estimator.inertia\_ *# 获取聚类准则的总和* N = len(estimator.cluster\_centers\_[0]) angles=np.linspace(0, 2\*np.pi, N, endpoint=**False**) *# 设置雷达图的角度，用于平分切开一个圆面* angles=np.concatenate((angles, [angles[0]])) *# 为了使雷达图一圈封闭起来* fig = plt.figure(figsize=(7,7)) *# 设置画布大小* ax = fig.add\_subplot(111, polar=**True**) *# 这里一定要设置为极坐标格式* sam = [**'r-'**, **'o-'**, **'g-'**, **'b-'**, **'p-'**] *# 样式* lab = [] *# 图例标签名* **for** i **in** range(len(estimator.cluster\_centers\_)):  values = estimator.cluster\_centers\_[i]  feature = [**'feature 1'**,**'feature 2'**,**'feature 3'**,**'feature 4'**,**'feature 5'**,**'feature 6'**,**'feature 7'**] *# 设置各指标名称  # 为了使雷达图一圈封闭起来，需要下面的步骤* values=np.concatenate((values,[values[0]]))  ax.plot(angles, values, sam[i], linewidth=2) *# 绘制折线图* ax.fill(angles, values, alpha=0.25) *# 填充颜色* ax.set\_thetagrids(angles \* 180/np.pi, feature) *# 添加每个特征的标签* ax.set\_ylim(-2, 2) *# 设置雷达图的范围* plt.title(**'Customer Features'**) *# 添加标题* ax.grid(**True**) *# 添加网格线* lab.append(**'customer class '**+str(i)) plt.legend(lab) plt.show() *# 显示图形* |



## 聚类的方法

### 划分方法

发现球形互斥的簇

基于距离

可以用均值或中心点等代表簇中心

对中小规模数据集有效

### 层次方法

聚类是一个层次分解(即多层)

不能纠正错误的合并或分析

可以集成其他技术，如微聚类或考虑对象"连接"

### 基于密度的方法

可以发现任意形状的簇

簇是对象空间中被低密度区域分割的稠密区域

簇密度：每个点的"领域"内必须具有最少个数的点

可能过滤离群点

### 基于网格的方法

使用一种多分辨率网格数据结构

快速处理

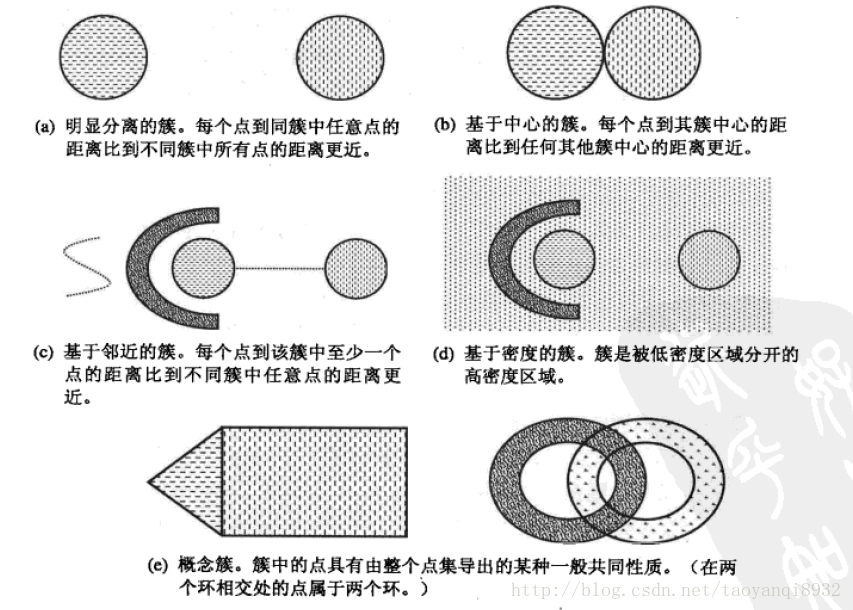
### 基于概率模型

## 基于划分的聚类——k-means

### 预定义

#### 簇类型

聚类旨在发现有用的对象簇，在现实中我们用到很多的簇的类型，使用不同的簇类型划分数据的结果是不同的，如下的几种簇类型。



### 算法描述

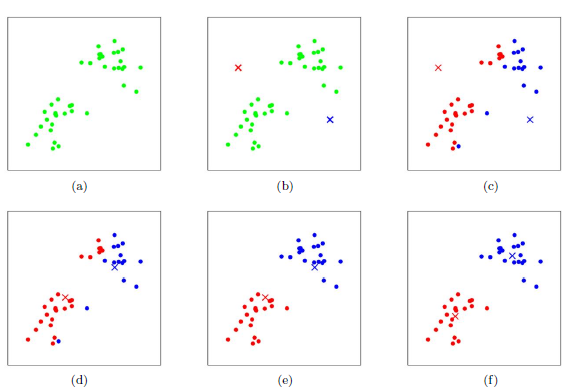
其聚类过程类似于梯度下降算法，建立代价函数并通过迭代使得代价函数值越来越小。算法分为4步：

(1) 适当选择c个类的初始中心；

(2) 在第k次迭代中，对任意一个样本，求其到c个中心的距离，将该样本归到距离最短的中心所在的类；

(3) 利用均值等方法更新该类的中心值；

(4) 对于所有的c个聚类中心，如果利用(2)(3)的迭代法更新后，值保持不变，则迭代结束，否则继续迭代。



### 算法适应场景

K-means是机器学习中一个比较常用的算法，属于无监督学习算法，其常被用于数据的聚类，只需为它指定簇的数量即可自动将数据聚合到多类中，相同簇中的数据相似度较高，不同簇中数据相似度较低。

优点：原理简单、速度快、对大数据集有比较好的伸缩性。

缺点：

(1) K值需要预先给定，属于预先知识，很多情况下K值的估计是非常困难的，对于像计算全部微信用户的交往圈这样的场景就完全的没办法用K-Means进行。对于可以确定K值不会太大但不明确精确的K值的场景，可以进行迭代运算，然后找出Cost Function最小时所对应的K值，这个值往往能较好的描述有多少个簇类。

(2) K-Means算法对初始选取的聚类中心点是敏感的，不同的随机种子点得到的聚类结果完全不同

(3) K均值算法并不是很所有的数据簇。它不能处理非球形簇、不同尺寸和不同密度的簇，银冠指定足够大的簇的个数是他通常可以发现纯子簇。

(4) 对离群点的数据进行聚类时，K均值也有问题，这种情况下，离群点检测和删除有很大的帮助。

### 算法实现

## 基于划分的聚类——k-means++

## 基于划分的聚类——k-modes

## 基于划分的聚类——k-prototypes

## 基于划分的聚类——k-medoids

## 基于划分的聚类——PAM

## 基于划分的聚类——CLARA

## 基于划分的聚类——CLARANS

## 基于划分的聚类——PCM

## 期望最大值算法EM

http://www.cnblogs.com/jerrylead/archive/2011/04/06/2006936.html

## 基于层次的聚类——BIRCH

## 基于层次的聚类——Chameleon

## 基于层次的聚类——CURE

## 基于层次的聚类——ROCK

## 模糊聚类

## 谱聚类

参照

https://www.cnblogs.com/Leo\_wl/p/3156049.html

https://www.cnblogs.com/pinard/p/6221564.html

### 预定义

### 算法描述

输入：样本集，相似矩阵的生成方式，降维后的维度，聚类后的维度

输出：簇划分

过程：

(1) 根据输入的相似矩阵的生成方式构建样本的相似矩阵

(2) 根据相似矩阵构建邻接矩阵，构建度矩阵

(3) 计算出拉普拉斯矩阵

(4) 构建标准化后的拉普拉斯矩阵

(5) 计算最小的个特征值所各自对应的特征向量

(6) 将各自对应的特征向量组成的矩阵按行标准化，最终组成维的特征矩阵

(7) 将中的每一行作为一个维的样本，共个样本，用输入的剧了里方法进行聚类，聚类维数为

(8) 得到簇划分

### 算法适应场景

### 算法实现

## 基于密度的聚类——DBSCAN

### 预定义

(1) 核心对象：若某个点的密度达到算法设定的阈值则称为核心点。

(2) 距离阈值：设定的半径r。

(3) 直接密度可达：若某点p在点q的r邻域内，且q是核心密度点，则pq直接密度可达。

(4) 密度可达：直接密度可达的相互传播。

(5) 密度相连：若从某核心点q出发，点p和k都能与q核心密度可达，则p和k是密度相连的。

(6) 边界点：属于某个类的非核心点，不能发展下线。

(7) 噪声点：不属于任何一个类簇的点，从任何一个点出发都不可能密度可达。

### 算法描述

(1) 标记所有对象是unvisited

(2) 随机选择一个unvisited的对象p,把他变成visited

(3) 如果p某个半径长度r的邻域内至少有m个对象。

(4) 创建新簇C，然后把p加入到C中；

(5) 令N为p的半径r的邻域内的所有对象集合。

(6) 对N中的每个点s，如果s是unvisted的，把它变为visited，如果s的r邻域内至少有m个对象，把这些对象添加到N，如果s不是任何簇的成员，将其添加到C，然后输出C

(7) 如果第三步没有那么多对象，将其变成噪声点。

(8) 然后不断循环。

### 算法适应场景

**优点：**不需要指定簇个数，可以发现任意形状的簇，擅长找到离群点。

**缺点：**高维度的数据计算慢（可以做降维，和数据消减），参数难以选择。

DBScan中含有两个参数：eps和min\_samples, 如果存在min\_samples个其他的数据样本与当前样本的距离为eps，我们就认为当前样本为核心样本，其他的min\_samples个样本被作为这个核心样本的邻居。一个簇是由一系列的核心样本组建的，递归地通过一个核心样本发现它的所有的邻居，它的所有邻居也是核心样本，以此类推。一个簇和可以有非核心样本，它们是核心样本的邻居但是它们本身并不是核心样本。直观地看，这些非核心样本都在簇的边缘部分。 **一个大的min\_samples和小的eps表明只有密度很大的样本才能组成一个簇。**

### 算法实现

## 基于密度的聚类——OPTICS

## 基于密度的聚类——DENCLUE

## 基于密度的聚类——DBCLASD

## 基于密度的聚类——CURD

## 基于网格的聚类——STING

## 基于网格的聚类——CLIQUE

## 基于网格的聚类——WaveCluster

## 非负矩阵分解算法

## 隐狄利克雷分布

## 超级大群的处理

80%的数据分布在1%的空间内，而剩下的20%的数据分布在99%的空间内。聚类时，分布在1%空间内的大部分数据会被聚为一类，剩下的聚为一类。当不断增加K值时，模型一般是对99%空间内的数据不断进行细分，因为这些数据之间的空间距离比较大。

而对分布在1%空间内的数据则很难进一步细分，或者即使细分了，也只是剥离出了外侧少量数据。

解决办法：那么为了解决这个问题，一种可行的方法是是对特征取LOG，减轻长尾问题。经过这两种方法处理后，都能较好的对玩家进行分类。下图是上图中的数据点取LOG后得到的分布图。

缺点：取LOG的方法的缺点在于，会使数据变得不直观，不好理解。

# 建模——支持向量机SVM

# 建模——K近邻——KNN

# 建模——决策树

只适用于标称型数据，对连续性数据处理得不好；

学习出一颗决策树首要考虑一个问题，即 根据数据集构建当前树应该选择哪种属性作为树根，即划分标准？

考虑最好的情况，一开始选择某个特征，就把数据集划分成功，即在该特征上取某个值的全是一类。

考虑最坏的情况，不断选择特征，划分后的数据集总是杂乱无章，就二分类任务来说，总是有正类有负类，一直到特征全部用完了，划分的数据集合还是有正有负，这时只能用投票法，正类多就选正类作为叶子，否则选负类。

单独使用决策树算法时，有容易过拟合缺点。抑制单颗决策树的复杂度的方法有很多，比如限制树的最大深度、限制叶子节点的最少样本数量、限制节点分裂时的最少样本数量、吸收 bagging 的思想对训练样本采样（subsample），在学习单颗决策树时只使用一部分训练样本、借鉴随机森林的思路在学习单颗决策树时只采样一部分特征、在目标函数中添加正则项惩罚复杂的树结构等。

## 迭代二叉树3代：ID3算法

可以参考下<https://www.cnblogs.com/pinard/p/6056319.html>

### 预定义

熵这个概念最早起源于物理学，在物理学中是用来度量一个热力学系统的无序程度，而在信息学里面，熵是对不确定性的度量。在1948年，香农引入了信息熵，将其定义为离散随机事件出现的概率，一个系统越是有序，信息熵就越低，反之一个系统越是混乱，它的信息熵就越高。所以信息熵可以被认为是系统有序化程度的一个度量。宇宙是熵增的，也就是说，从有序逐渐逐渐无序。

随机变量取值为，概率为，那么的熵定义为。一个随机变量的变化情况可能越多，那么它携带的信息量就越大，它的熵也越大。

对于一个分类系统来说，类别的取值为，每类的数目为，总的个数为，那么类别出现的概率为。此时分类系统的熵定义为：。

对于一个分类训练器来说，有个样本，有个属性，其中，属性的取值为，每个属性的可取值个数的都不一样。个样本的类别的取值为，如下例子：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Windy | Play? |
| sunny | hot | hight | false | no |
| sunny | hot | high | true | no |
| overcast | hot | high | false | yes |
| rain | mid | high | false | yes |
| rain | cool | normal | false | yes |
| rain | cool | normal | true | no |
| overcast | cool | normal | true | yes |
| sunny | mid | high | false | no |
| sunny | cool | normal | false | yes |
| rain | mid | normal | false | yes |
| sunny | mid | normal | true | yes |
| overcast | mid | high | true | yes |
| overcast | hot | normal | false | yes |
| rain | mid | high | true | no |

，

，











如果样本作为一个随机变量，在初始没有属性信息辅助的时候，，，

假如现在增加的信息，，，

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Windy | Play |
| sunny | hot | hight | false | no |
| sunny | hot | high | true | no |
| overcast | hot | high | false | yes |
| rain | mid | high | false | yes |
| rain | cool | normal | false | yes |
| rain | cool | normal | true | no |
| overcast | cool | normal | true | yes |
| sunny | mid | high | false | no |
| sunny | cool | normal | false | yes |
| rain | mid | normal | false | yes |
| sunny | mid | normal | true | yes |
| overcast | mid | high | true | yes |
| overcast | hot | normal | false | yes |
| rain | mid | high | true | no |

，

，

，

当时，



同理，时，

时，

最终有，



增加属性的信息，带来的的熵减少为

为选择属性的信息熵增益。

同理，如果先增加属性的信息，再增加属性，那么到第二步，增加两个属性带来的熵的减少为：



每一步选择增加一个属性，到最后，所有属性的信息都充分利用，给出所有属性值唯一确定一个分类。

### 算法描述

实际上，对于有个属性的样本，最多经过步的判断，就可以唯一确定样本的分类。先选择哪个属性，后选择哪个属性，是有讲究的。在选择属性的顺序上，ID3算法采用了自顶向下的贪心设计的原则：

在最开始，没有利用任何属性，

然后第一步，对于可供选择的属性集，分别计算，计算最大的那个，选择对应的属性作为第一步的选择的属性。这个谓之最大信息增益。

然后是第二步，从可供选择的属性集里剔除第一步选择的属性后，在对剩余的属性集重复同样的过程，选择出第二个属性。实际上，某个属性的熵的值，跟进行到第几步是无关的。

直到所有属性均已选择。

这个过程自然而然地就构造出一颗树，谓之ID3树，也就是决策树。这棵树的每一个内部节点是一个属性，每一个叶子是一个分类结果。每一个内部节点代表的属性有多少种取值，就意味着内部节点有多少棵子树。

### 算法适用场景

(1) ID3没有考虑连续特征，比如长度，密度都是连续值，无法在ID3运用。这大大限制了ID3的用途。

(2) ID3采用信息增益大的特征优先建立决策树的节点。很快就被人发现，在相同条件下，取值比较多的特征比取值少的特征信息增益大。比如一个变量有2个值，各为1/2，另一个变量为3个值，各为1/3，其实他们都是完全不确定的变量，但是取3个值的比取2个值的信息增益大。

(3) ID3算法对于缺失值的情况没有做考虑

(4) 没有考虑过拟合的问题

C4.5算法解决了这些问题。

### 算法实现

#### ID3算法的C++实现



#### ID3算法的R实现

#### ID3算法的Python实现

## 分类回归树：C4.5算法

### 预定义

### 算法描述

#### (1) 缺失值处理

缺失值处理有三种情况：

a) 在具有缺失值的属性上如何计算信息增益率？

第一种：忽略该类样本

第二种：选择常用值或均值填充

第三种：依据缺失比例，折算信息增益/信息增益率

第四种：对缺失值赋予独特的值，参与训练

b) 具有缺失值的样本在进行数据分裂时，分配给哪个子数据集？

第一种：忽略该类样本

第二种：选择常用值或均值填充

第三种：根据其他非缺失属性的比例，分配到子数据集中

第四种：为缺失值建立单独分支

第五种：确定最可能的取值，按比例仅分配给一个子数据集

c) 对新样本进行分类时，缺失值导致样本到达叶子节点，怎么处理？

第一种：有缺失值单独分支，走单独分支

第二种：走最常见的值的的分支

第三种：确定最可能取值，走相应分支

第四种：走所有分支，根据不同输出结果的概率进行组合

第五种：不进行分类，直接赋给最有可能的值

#### (2) 多属性特征过于优先的处理

#### (3) 连续值处理

对连续型属性进行排序，得到多个候选阀值，选取产生最大信息增益的阀值作为分裂阀值。

#### (4) 过拟合处理

有监督的算法需要考虑泛化能力，在有限样本的条件下，决策树超过一定规模后，训练错误率减小，但测试错误率会增加。控制决策树规模的方法称为剪枝，一种是先剪枝，一种是后剪枝。所谓先剪枝，实际上是控制决策树的生产；后剪枝是指，对完全生成的决策树进行修剪。

先剪枝：

a) 数据划分法：划分数据成训练样本和测试样本，使用用训练样本进行训练，使用测试样本进行树生长检验

b) 阀值法：当某节点的信息增益小于某阀值时，停止树生长

c) 信息增益的统计显著性分析。从已有节点获得的所有信息增益统计其分布，如果继续生长得到的信息增益与该分布相比不显著，则停止树的生长。

后剪枝：

a) 减少分类错误修建发。使用独立的剪枝集估计剪枝前后的分类错误率，基于此进行剪枝。

b) 最小代价与复杂性折中的剪枝。对剪枝后的树综合评价错误率和复杂性，决定是否剪枝。

c) 最小描述长度准则。最简单的树就是最好的树。对决策树进行编码，通过剪枝得到编码最小的树。

d) 规则后剪枝。将训练完的决策树转换成规，通过删除不会降低估计精度的前件修剪每一条规则。

### 算法适用场景

### 算法实现

## C5.0

## 单层决策树

## M5模型

## 随机森林

## 条件决策树

## Boosting提升算法

Boosting算法是将“弱学习算法“提升为“强学习算法”的过程，主要思想是“三个臭皮匠顶个诸葛亮”。一般来说，找到弱学习算法要相对容易一些，然后通过反复学习得到一系列弱分类器，组合这些弱分类器得到一个强分类器。Boosting算法要涉及到两个部分，加法模型和前向分步算法。加法模型就是说强分类器由一系列弱分类器线性相加而成。组合形式如下：



其中，就是一个个的弱分类器，是弱分类器学习到的最优参数，就是弱学习在强分类器中所占比重，是所有和的组合。这些弱分类器线性相加组成强分类器。前向分步就是说在训练过程中，下一轮迭代产生的分类器是在上一轮的基础上训练得来的。也就是可以写成这样的形式：



## Adaboost

### 预定义

#### (1) AdaBoost原理

基于Boosting的理解，对于AdaBoost，要明白两点：

第一点：每一次迭代的弱分类器有何 ，如何学习？

第二点：弱分类器权值如何确定？

对于第一点，AdaBoost改变了训练数据的权重，也就是样本的概率分布，其思想是将关注点放在被错误分类的样本上，减少上一轮被正确分类的样本的权值，提高那些被错误分类的样本的权值。然后再根据所采用的一些基本机器学习算法进行学习，比如逻辑回归。

对于第二点，AdaBoost采用加权多数表决的方法，加大分类误差率小的弱分类器的权重，减少分类误差率大的弱分类器的权重。即，正确率高分的好的弱分类器在强分类器中应该要有较大的发言权。

#### (1) 各弱分类器权重更新的推导

https://www.cnblogs.com/ScorpioLu/p/8295990.html

https://www.cnblogs.com/qwj-sysu/p/5989282.html

### 算法描述

输入：训练数据集，其中，，，迭代次数

1、初始化训练样本的权值分布：，

2、对于：

(1) 适用具有权值分布的训练数据集进行学习，得到弱分类器

(2) 计算在训练数据集上的分类误差率：

(3) 计算在强分类器中所占的权重：

(4) 更新训练数据集的权重分布(这里，是归一化因子，为了使样本的概率分布和为1)

,

3、最终分类器为：



### 算法适用场景

### 算法实现

## GBDT

## XGboost

# 建模——回归分析

# 建模——关联分析

## Apriori算法

## FP-Growth算法

## eclat算法

# 建模——贝叶斯

## 贝叶斯定理

### 贝叶斯定理的推导

[条件概率](http://link.zhihu.com/?target=https%3A//zh.wikipedia.org/wiki/%25E6%259D%25A1%25E4%25BB%25B6%25E6%25A6%2582%25E7%258E%2587)，描述的是事件在另一个事件已经发生条件下的概率，记作 ， 和可能是相互独立的两个事件，也可能不是。



表示和事件同时发生的概率，如果和是相互独立的两个事件，那么：



上面的反过来证明了如果和是相互独立的事件，那么事件 发生的概率与无关。

稍微改变一下，



考虑到先验条件的多种可能性，引入全概率公式：

，这里是的补集。

根据条件概率和全概率公式，很容易推导出贝叶斯公式：



特别地，

如果是的一个划分，



当然，从逻辑上，和也可以无类别上的关系。仅仅是观察到的两个因素。

### 先验概率和后验概率

如果从先验和后验去理解：



对于贝叶斯公式，本质上是用先验概率去修正后验概率，其中，

是先验概率，是不考虑任何方面的因素。

是标准化常量

是似然度，是观察到的内容，是相关的证据。

后验概率 = (似然度×先验概率) / 标准化常量。也就是说，后验概率与先验概率和似然度的乘积成正比。

### 机器学习的视角理解贝叶斯



### 帮助理解贝叶斯定理的一个例子

在生活中，几乎所有人（包括统计学者）都会无意识地将两个事件的后验概率混淆，即：

最经典的一个例子就是疾病检测，假设某种疾病在所有人群中的感染率是0.1%，医院现有的技术对于该疾病检测准确率为 99%（已知患病情况下， 99% 的可能性可以检查出阳性；正常人 99% 的可能性检查为正常），如果从人群中随机抽一个人去检测，医院给出的检测结果为阳性，那么这个人实际得病的概率是多少？

很多人会脱口而出 "99%"，但真实概率远低于此，因为他们把两个后验概率搞混了，如果用表示这个人患有该疾病，用表示医院检测的结果是阳性，那么 表示的是「已知一个人得病的情况下医院检测出阳性的概率」，而我们现在问的是「对于随机抽取的这个人，已知检测结果为阳性的情况下这个人患病的概率」，即 。

我们可以用贝叶斯定理来计算这个人实际得病的概率：



其中：

，被检测者患病的概率

，被检测未者患病的概率

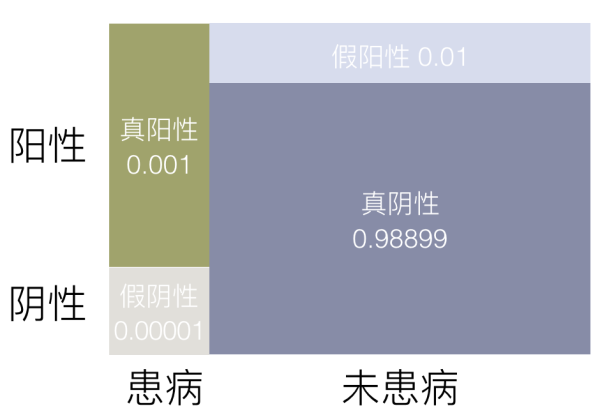
，已知患病的情况下检测为阳性的概率

，已知未患病的情况下检测为阳性的概率

将上面的概率代入到贝叶斯公式中，可得：



这个公式在这里的实际意义是什么？让我们用图来解释（图中概率经过四舍五入，考虑到图片的尺寸，面积并没有和概率严格对应起来）：



从贝叶斯的角度来看，随意选取的一个被测者，由于信息并不充分，未检测之前有假阳性、真阳性、假阴性和真阴性四种可能，这些可能性由检测技术和该疾病的感染率决定，当检测结果为阳性的时候，只剩下真阳性和假阳性两种可能，而真阳性的概率仅为假阳性的十分之一，贝叶斯公式在这里的实际意义是：



即使被医院检测为阳性，实际患病的概率其实还不到10%，有很大可能是假阳性，往往需要复检来确定是否真的患病，让我们再来计算初检和复检结果都为阳性时，患病的可能性。假设两次检查的准确率相同，都是99%，这里令为第一次检测结果为阳性，为第二次检测结果为阳性，为被检测者患病，那么两次检测结果都是阳性患病的概率可以表示为：



其中：

，被检测者患病的概率

，被检测未者患病的概率

，已知患病情况下连续两次检测结果为阳性的概率

，已知未患病情况下连续两次检测结果为阳性的概率

代入后可得：



可见复检结果大大提高了检测的可信度，联系上面的图，复检的意义在于大幅减少假阳性的可能（0.01 -> 0.0001）从而提高阳性检测的准确性。

## 朴素贝叶斯

<https://www.cnblogs.com/pinard/p/6069267.html>

### 预定义

#### (1) 朴素贝叶斯分类器的原理

已有的样本数据：

，有个样本，每个样本有个特征，特征输出有个类别，定义未

从样本中可以学到朴素贝叶斯的先验分布，接着学习到条件概率分布：



根据贝叶斯定理：



其中，**朴素贝叶斯的假设是：各个特征之间是独立无关的**。基于这一假设，



是根据样本数据的各个特征取值和对应的类别出现的频数可以构造出来。作为极大似然。

是作为一个先验。实际上，这个分类的概率应该是一个先验知识，而不是根据样本数据的类别频数构造出来。这里朴素贝叶斯分类器用样本的频数作为先验了。

有了和，分类器就可以构造了。

分类器构造完毕，就可以对一个新样本进行分类了。

对于样本，对于，计算

，







哪个大，就认为新样本属于哪个分类。至于实际上是不用计算的，因为每个公式里都有它，作为一个相同值的因子不影响计算结果。

#### (2) 朴素贝叶斯的参数估计

直接根据训练样本的频数计算即可。

但对于测试样本或者验证样本的计算就比较复杂了。

如果第个特征是离散的值，如果出现在训练样本中，，其中，为样本类别的样本的个数，为样本类别第个特征取值为的样本个数。如果没有出现在训练样本中，需要引入拉普拉斯平滑，此时有，其中为一个大于0的常数，常常取为1。为第个特征的取值个数。如果是非常稀疏的离散值，即各个特征出现的概率很低，这时可以假设符合伯努利分布，即特征出现记为1，不出现记为0。即只要出现即可，不用关注的次数。是在样本类别中，出现的频率。

，取值为0或者1。

如果第个特征是连续的值，通常认为的先验分布为正态分布，即在样本类别为的样本中，的值符合正态分布，对于取值为的概率为

，是取值为的样本中所有的方差， 是取值为的样本中所有的均值。

### 算法描述

### 算法适用场景

朴素贝叶斯的主要优点有：

　　(1) 朴素贝叶斯模型发源于古典数学理论，有稳定的分类效率。

　　(2) 对小规模的数据表现很好，能个处理多分类任务，适合增量式训练，尤其是数据量超出内存时，我们可以一批批的去增量训练。

　　(3) 对缺失数据不太敏感，算法也比较简单，常用于文本分类。

朴素贝叶斯的主要缺点有：

　　(1) 理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯模型给定输出类别的情况下,假设属性之间相互独立，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。而在属性相关性较小时，朴素贝叶斯性能最为良好。对于这一点，有半朴素贝叶斯之类的算法通过考虑部分关联性适度改进。

　　(2) 需要知道先验概率，且先验概率很多时候取决于假设，假设的模型可以有很多种，因此在某些时候会由于假设的先验模型的原因导致预测效果不佳。

　　(3) 由于我们是通过先验和数据来决定后验的概率从而决定分类，所以分类决策存在一定的错误率。

　　(4) 对输入数据的表达形式很敏感。

### 算法实现

## 高斯朴素贝叶斯

## 多项式朴素贝叶斯

## 平均单依赖分类器

## 贝叶斯信念网络

### 预定义

#### (1) 条件独立性

给定第三个事件，如果，则称和是条件独立事件，符号表示为

若和关于事件条件独立，则有以下一些理解：

1) 事件的发生，使本来可能不独立的事件和事件变得[独立](https://baike.baidu.com/item/%E7%8B%AC%E7%AB%8B/3415220)起来；

2) 事件的出现或发生，解开了和的依赖关系。

另外，有：若，则

对条件独立性的理解：

给定三个事件、、：

：明天下雨

：今天的地面是湿的；

：今天是否下雨；

事件的成立，对和均有影响，然而，在事件成立的前提下，今天的地面情况对明天是否下雨没有影响。即，在已知的前提下，和是相互独立的，即和是条件独立的。

#### (2) 图结构

在图模型中，条件独立有以下三种典型情况： 

1) 头到尾连接：顺连结构

三个事件可以顺序连接，如

。

联合概率为：

若未知，无法推出，和不独立。

若作为条件已知：



则与是条件独立的，即

另一方面，知道就知道的一切；知道的状态并不能为增加附加知识，记作



一般说，阻塞(block) 了从到的路径，换句话说，分开了和，意指如果删掉，则就不存在和之间的路径。

2) 尾到尾连接：分连结构

可能是两个节点和的父节点，



联合密度可以写作：



若未知，无法推出，和不独立。

若作为条件已知：

，

当的值已知时，它阻塞了和之间的路径，换言之，分开了和。

3) 头到头连接：汇连结构

在头到头连接中，有两个父节点和连接到单个节点，



则联合密度记作：



若未知，



两边消去可得到：和是独立的：

 。这个独立是边际独立

当知道时，和变成依赖的。这种情况与阻塞或分开的概念不同。当观察不到时，和之间的路径被阻塞；当 (或者它的任意后代)被观测到时，和便不再是阻塞的、独立的。

#### (2) 贝叶斯网络

对比起朴素贝叶斯假设**各个特征之间是独立无关的，**贝叶斯网络的假设要强一些，它假设每个属性与它的非后裔属性是独立的。





### 算法描述

### 算法适用场景

#### (1) 贝叶斯网的意义

贝叶斯网是一种系统地描述随机变量之间关系的语言，构造贝叶斯网的主要目的是进行概率推理，即计算一些时间发生的的概率。要在一些随机变量之间进行概率推理，理论上只需要一个联合概率分布即可。但是，联合概率分布的复杂度相对于变量个数成指数增长，所以当变量众多时不可行。贝叶斯网络把复杂的联合概率分布分解为一系列相对简单的模块，从而大大降低了知识获取的难度和概率推理的复杂度。

#### (1) 构造贝叶斯信念网络的三种方式

1) 由领域专家确定贝叶斯网的变量（有时也称为影响因子）节点，然后通过专家的知识来确定贝叶斯网络的结构，并指定它的分布参数。这种方式构造的贝叶斯网完全在专家的指导下进行，由于人类获得知识的有限性，导致构建的网络与实践中积累下的数据具有很大的偏差。

2) 由领域专家确定贝叶斯网络的节点，通过大量的训练数据，**来学习贝叶斯网的结构和参数**。这种方式完全是一种数据驱动的方法，具有很强的适应性，而且随着人工智能、数据挖掘和机器学习的不断发展，使得这种方法成为可能。如何从数据中学习贝叶斯网的结构和参数，已经成为贝叶斯网络研究的热点。

3) 由领域专家确定贝叶斯网络的节点，通过专家的知识来指定网络的结构，而通过机器学习的方法从数据中学习网络的参数。这种方式实际上是前两种方式的折衷，当领域中变量之间的关系较明显的情况下，这种方法能大大提高学习的效率。

#### (1) 贝叶斯信念网络的优点

通过提供图形化的方法来表示和运算概率知识，贝叶斯网络克服了基于规则的系统所具有的许多概念上和计算上的困难。贝叶斯网络与统计技术相结合，使得其在数据分析方面拥有了许多优点，与规划挖掘、决策树、人工神经网络、密度估计、分类、回归和聚类等方法相比，贝叶斯网络的优点主要体现在：

1) 贝叶斯网络使用图形的方法描述数据间的相互关系，语义清晰，易于理解。图形化的知识表示方法使得保持概率知识库的一致性和完整性变得容易，可以方便地针对条件的改变进行网络模块的重新配置。

2) 贝叶斯网络易于处理不完备数据集。对于传统标准的监督学习算法而言必须知道所有可能的数据输入，如果缺少其中的某一输入就会对建立的模型产生偏差，贝叶斯网络的方法反映的是整个数据库中数据间的概率关系模型，缺少某一数据变量仍然可以建立精确的模型 。

3) 贝叶斯网络允许学习变量间的因果关系。在以往的数据分析中，一个问题的因果关系在干扰较多时，系统就无法做出精确的预测。而这种因果关系己经包含在贝叶斯网络模型中。贝叶斯方法具有因果和概率性语义，可以用来学习数据中的因果关系，并根据因果关系进行学习。

4) 贝叶斯网络与贝叶斯统计相结合能够充分利用领域知识和样本数据的信息。贝叶斯网络用弧表示变量间的依赖关系，用概率分布表来表示依赖关系的强弱，将先验信息与样本知识有机结合起来，促进了先验知识和数据的集成，这在样本数据稀疏或数据较难获得的时候特别有效。

### 算法实现

## 隐马尔科夫模型

## 条件随机场

# 基于实例的方法

## 学习向量量化(LVQ)

## 自组织映射(SOM)

## 自组织映射(SOM)

# 回归分析

## 线性回归

## 岭回归

## 对数几率回归

## 多元自适应回归样条法

## 局部散点平滑估计

# 正则化

从**使用正则化解决了一个什么问题**的角度来看：**正则化是为了防止过拟合， 进而增强泛化能力**。用白话文转义，**泛化误差**（generalization error）= 测试误差（test error），其实就是使用训练数据训练的模型在测试集上的表现（或说性能 performance）不好。

可以了解：

https://charlesliuyx.github.io/2017/10/03/%E3%80%90%E7%9B%B4%E8%A7%82%E8%AF%A6%E8%A7%A3%E3%80%91%E4%BB%80%E4%B9%88%E6%98%AF%E6%AD%A3%E5%88%99%E5%8C%96/

## L1正则化

## L2正则化

# 神经网络

# 深度学习

## CNN

## RNN

## LSTM

## 强化学习

## 神经对抗网络