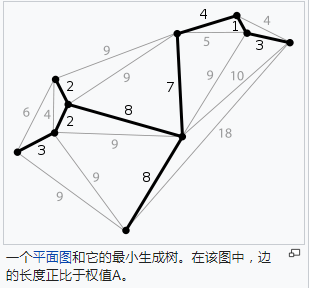
## 决策树

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）

决策树的创建可包括三部分：特征选择、决策树的生成以及决策树的剪枝；

**贪心法**可以解决一些[最优化](https://zh.wikipedia.org/wiki/%E6%9C%80%E4%BC%98%E5%8C%96)问题，如：求[图](https://zh.wikipedia.org/wiki/%E5%9B%BE)中的[最小生成树](https://zh.wikipedia.org/wiki/%E6%9C%80%E5%B0%8F%E7%94%9F%E6%88%90%E6%A0%91)、求[哈夫曼编码](https://zh.wikipedia.org/wiki/%E5%93%88%E5%A4%AB%E6%9B%BC%E7%BC%96%E7%A0%81)（查找最有效的二进制编码。由于无法证明哪个已有编码是最有效的，霍夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率[二叉树](https://zh.wikipedia.org/wiki/%E4%BA%8C%E5%8F%89%E6%A0%91)编码的想法，并很快证明了这个方法是最有效的。霍夫曼使用自底向上的方法构建二叉树，避免了次优算法[香农-范诺编码](https://zh.wikipedia.org/wiki/%E9%A6%99%E5%86%9C-%E8%8C%83%E8%AF%BA%E7%BC%96%E7%A0%81)的最大弊端──自顶向下构建树。）

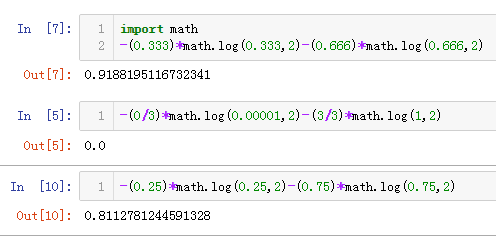
**最小生成树**是一副[连通](https://zh.wikipedia.org/wiki/%E8%BF%9E%E9%80%9A%E5%9B%BE)[加权无向图](https://zh.wikipedia.org/wiki/%E5%9B%BE)中一棵权值最小的[生成树](https://zh.wikipedia.org/wiki/%E7%94%9F%E6%88%90%E6%A0%91)。



**ID3算法**

<http://www.cnblogs.com/leoo2sk/archive/2010/09/19/decision-tree.html>

信息增益相当于在度量子节点属性→父节点属性的相关程度，相关性越大增益越大。



父节点属性分布越“混乱”熵越大，子节点属性越有规律熵越小，二者之差越大。

**3.4.2、关于剪枝**

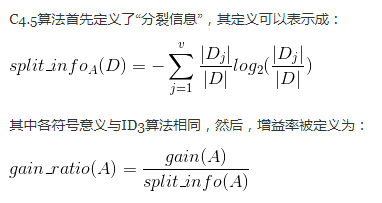
在实际构造决策树时，通常要进行[剪枝](http://en.wikipedia.org/wiki/Pruning_(decision_trees))，这时为了处理由于数据中的噪声和离群点导致的过分拟合问题。剪枝有两种：

      先剪枝——在构造过程中，当某个节点满足剪枝条件，则直接停止此分支的构造。

      后剪枝——先构造完成完整的决策树，再通过某些条件遍历树进行剪枝。

**C4.5算法**

C4.5决策树的提出完全是为了解决ID3决策树的一个缺点，当一个属性的可取值数目较多时，那么可能在这个属性对应的可取值下的样本只有一个或者是很少个，那么这个时候它的信息增益是非常高的，这个时候纯度很高，ID3决策树会认为这个属性很适合划分，但是较多取值的属性来进行划分带来的问题是它的泛化能力比较弱，不能够对新样本进行有效的预测。——容易过拟合

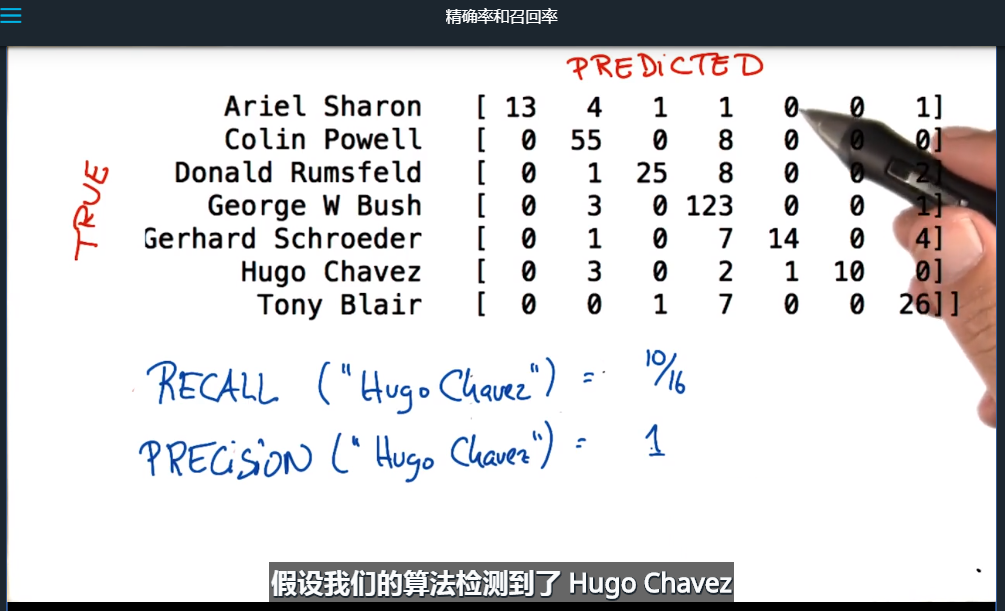


经过C4.5算法中“分裂信息”的处理，对可取值数目较少的属性有所偏好

请写一段二叉树中序遍历的代码：

<https://blog.csdn.net/bone_ace/article/details/46718683>

## 查全率与查准率



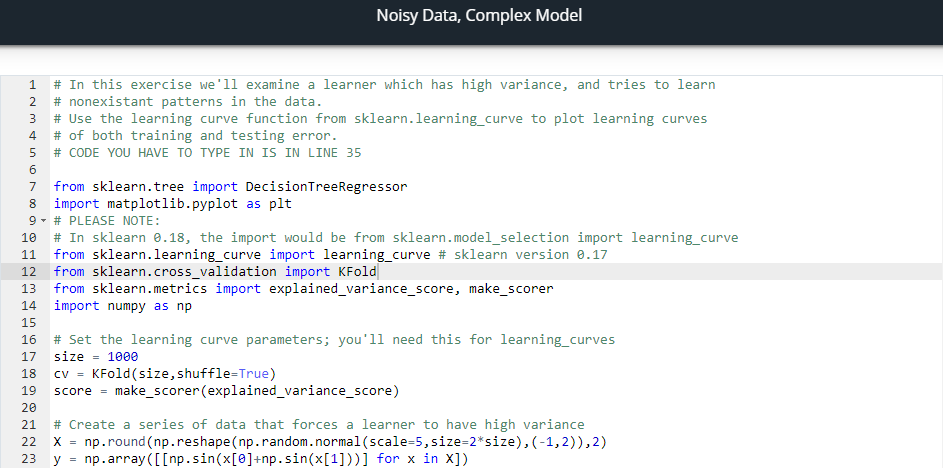
查全率（越大越全）：A类样本中有多少个被正确分到了A类（沿真实值方向）

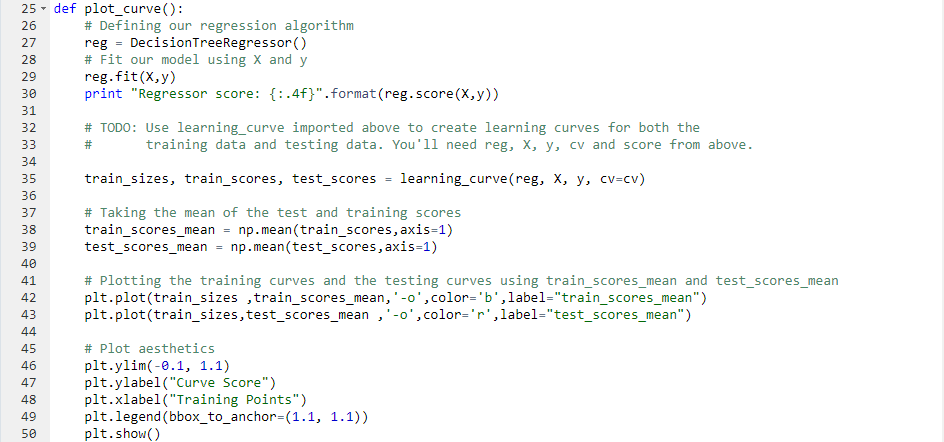
查准率（越大越准）： 被分为A类的样本中有多少正确的（沿预测值方向）

可将 F1 分数理解为精确率和召回率的加权平均值，其中 F1 分数的最佳值为 1、最差值为 0：

F1 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)

**决定系数**的含义是：模型所采用的预测模式下，目前所采用的特征能够在多大程度上解释目标特征的变化。







## 训练及测试

将数据集按一定比例分为训练用的数据集和测试用的数据集对学习算法有什么好处？

如果用模型已经见过的数据，例如部分训练集数据进行测试，又有什么坏处？

**回答:**

学习算法的任务就是从已知进行学习，对未知进行预测，将用学习中没有见过的数据对模型进行测试更符合实际需求，测试的结果能真正反映出模型预测的准确性。

如果用模型已经见过的数据，那么即使模型从已知数据中总结出了不准确的模式，但这种不准确的模式仍然符合它学习过的数据集，使用部分训练集数据进行测试就无法发现这个问题。但当使用模型没有见过的新的数据集进行测试时则能够发现模型并不能准确预测新的数据。

例如图像分类的时候，训练集中戴眼镜的人大部分是男人，当模型错误地学到一个模式把戴眼镜的人判断为男人，那么你用训练集测试不能发现这个问题，表现仍然很好，只有用新数据模型没有学习过的数据才可能发现问题。

## 模型复杂度

## 交叉验证

## KNN

K Nearest Neighbor算法又叫KNN算法（未用到BP？）

**懒惰学习器：**lazy learning machine 把从已有实例泛化所要查询的新实例的工作推迟到每个新实例出现时。属于后学习的学习器，而不是像决策树、神经网络是先从训练集的实例学到一个模式，遇到新实例的时候应用已经学到的模式进行预测。因此有效地索引训练样例， 以减少查询时所需计算是一个重要的问题。

**维度灾难：**随着特征数量增多，想要准确泛化新实例所需要的训练集样本数会指数增长，因此使用KNN时尽可能提供更多实例而不是更多的特征。

**做回归：**比如取K个近邻的目标特征的**平均值**作为新实例的预测值，从一种近邻关系映射到另一个特征上，是平滑进行的。

可以把KNN算法作为一种提取近邻的算法，提取出局部相似的一组数据后可以进行其他算法的操作，比如决策树、神经网络、AdaBoost等等

**在相似的局部事物附近构建函数或概念**

<https://blog.csdn.net/icamera0/article/details/77961341>

1. **def** knnClassify(inX, dataSet, labels, k):
2. dataSetSize = dataSet.shape[0] #获取训练样本集dataSet的个数，即矩阵dataSet的行数N
3. #tile(inX, (dataSetSize, 1))是建立一个N行1列的数组，但其元素是一个1行M列的向量，最后返回一个N行M列的数组
4. #N行M列的数组中每一行都是输入的测试样本，它与训练样本集相减，则得到NxM的数组值之差
5. diffMat = tile(inX, (dataSetSize, 1)) - dataSet
6. sqDiffMat = diffMat\*\*2
7. sqDistances = sqDiffMat.sum(axis = 1) #axis=1是对存储NxM个元素对应的平方差的数组，将每一行的值累加起来，返回一个Nx1的数组
8. distances = sqDistances\*\*0.5 #求得测试样本与各个训练样本的欧式距离
10. #对distances中N个元素进行从小到大排序，之后返回一个N个元素的一维数组，存储distances排序后各个元素在原来数组中的index
11. #eg. distances=[2,1,3,0], argsort的返回值为[3，1，0，2]
12. sortedDistIndicies = distances.argsort()
14. classCount={} #定义一个空的字典变量
15. **for** i **in** range(k): #i的取值为[0，k - 1]
16. voteLabel = labels[sortedDistIndicies[i]] #返回测试样本与训练样本欧式距离第i小的训练样本所对应的类的标签
17. #classCount.get(voteLabel, 0)获取classCount中voteLabel为index的元素的值，找不到则返回0
18. classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
20. #classCount.iteritems()返回字典classCount中所有项，按照迭代器的方式
21. #operator.itemgetter()用于获取对象的哪些维的数据，参数为一些序号；此处参数为1，即按照字典变量classCount各项的第二个元素进行排序
22. #reverse = True表示按照逆序排序，即从大到小的次序；False表示按照从小到大的次序排序
23. sortedClassCount = sorted(classCount.iteritems(), key = operator.itemgetter(1), reverse = True)
25. #返回k个近邻元素中所对应类最多那个类的标签，即测试样本所属那个类
26. **return** sortedClassCount[0][0]

## SVM（support vector machines）

<https://blog.csdn.net/leonis_v/article/details/50688766>

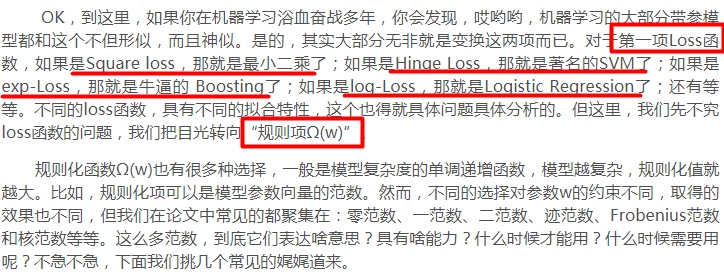
这种算法的思想是将输入空间**映射到高维空间**中，然后再根据内积的公式进行计算；

而使用核函数相当于直接在原来的**低维空间中进行计算**，不会显式地写出映射后的结果。避开了直接在高维空间中进行计算，而**结果却是等价的**

把这里的计算两个向量在隐式映射过后的空间中的**内积**的函数叫做核函数，而SVM 里需要计算的地方数据向量**总是以内积的形式出现**的。

因此可以说SVM算法与核函数kernel二者完美搭配。

## 范数L0L1L2



<https://blog.csdn.net/zouxy09/article/details/24971995>

**为什么L1范数会使权值稀疏？**有人可能会这样给你回答“它是L0范数的最优凸近似”。实际上，还存在一个更美的回答：任何的规则化算子，如果他在Wi=0的地方不可微，并且可以分解为一个“求和”的形式，那么这个规则化算子就可以实现稀疏。

L0范数很难优化求解（NP难问题），二是L1范数是L0范数的最优凸近似，

**一句话总结：**L1范数和L0范数可以实现稀疏，L1因具有比L0更好的优化求解特性而被广泛应用。

为什么要稀疏？让我们的参数稀疏有什么好处呢？这里扯两点：

1）特征选择(Feature Selection)：

大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现**特征的自动选择**。

2）可解释性(Interpretability)：

例如患某种病的概率是y，然后我们收集到的数据x是1000维的，也就是我们需要寻找这1000种因素到底是怎么影响患上这种病的概率的。通过学习，如果最后学习到的w\*就只有很少的非零元素，例如只有5个非零的wi，那么我们就有理由相信，这些对应的特征在患病分析上面提供的信息是巨大的，决策性的。也就是说，患不患这种病只和这5个因素有关，那医生就好分析多了。

**L2规则化**

对**condition number**来个一句话总结：condition number是一个矩阵（或者它所描述的线性系统）的稳定性或者敏感度的度量，如果一个矩阵的condition number在1附近，那么它就是well-conditioned的，如果远大于1，那么它就是ill-conditioned的，如果一个系统是ill-conditioned的(系统的解对系数矩阵A或者b太敏感)

另外，如果使用**迭代优化**的算法，condition number 太大仍然会导致问题：它会拖慢迭代的收敛速度，而规则项从优化的角度来看，实际上是将目标函数变成λ-strongly convex（λ强凸）的.

L2范数不但可以防止过拟合，还可以让我们的优化求解变得**稳定和快速**。

一句话总结就是：L1会趋向于产生少量的特征，而其他的特征都是0，而L2会选择更多的特征，这些特征都会接近于0，从而避免condition number过大的问题

## Boosting & Adaboost

## K-means

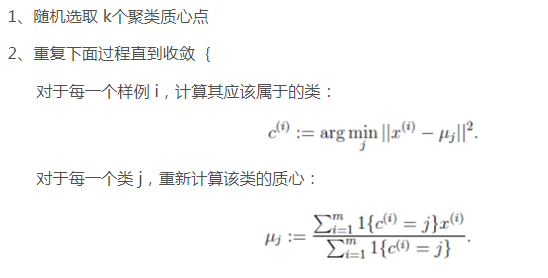
与KNN：两种算法之间有一点相似，都是**计算最近邻**，一般都用**欧氏距离**。

对k个**初始质心**的选择比较敏感，容易陷入**局部最小值**。

**复杂度**是O（n,k,t）n 是所有对象的数目, k 是簇的数目, t 是迭代的次数

<https://blog.csdn.net/zouxy09/article/details/17589329> 含代码

<https://www.cnblogs.com/mindpuzzle/archive/2013/04/24/3036447.html> GMM



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

创建k个点作为初始的质心点（随机选择）

当任意一个点的簇分配结果发生改变时

对数据集中的每一个数据点

对每一个质心

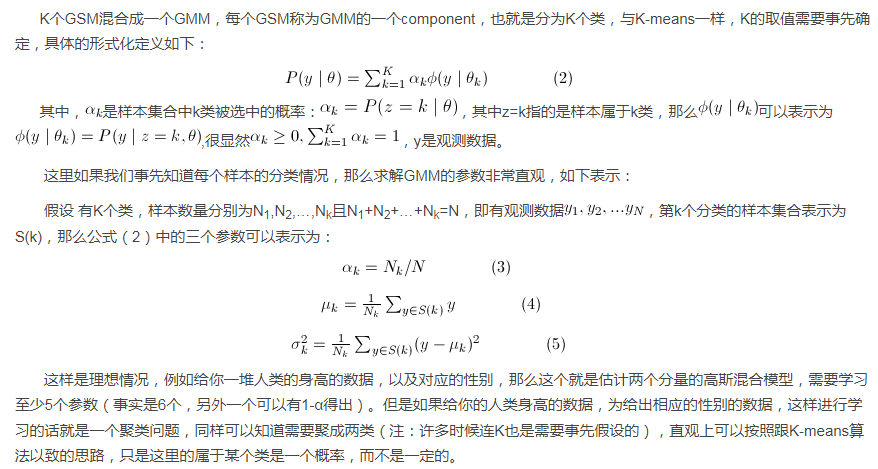
计算质心与数据点的距离

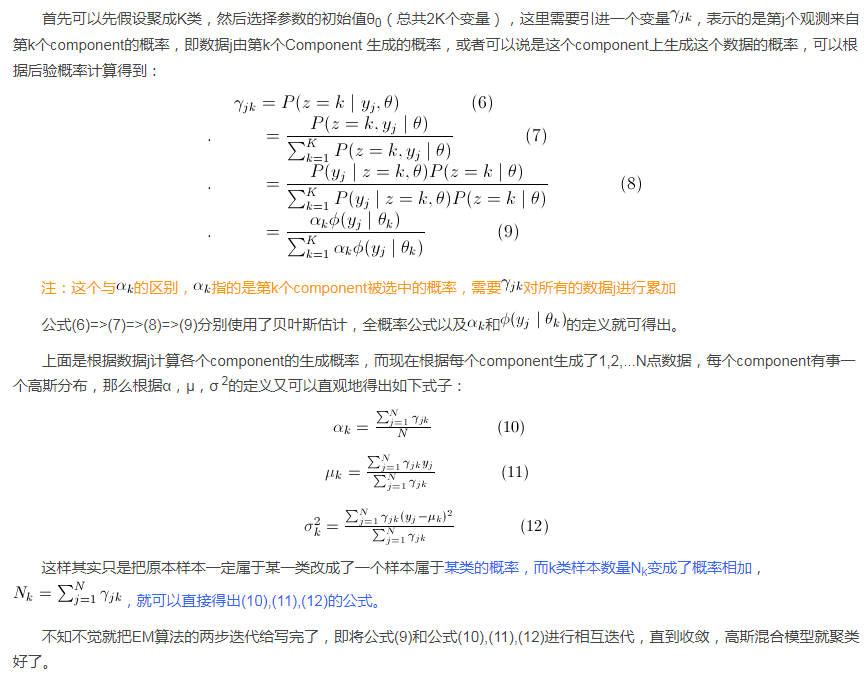
将数据点分配到距离最近的簇

对每一个簇，计算簇中所有点的均值，并将均值作为质心

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**软聚类**：结合了高斯分布的概念





## 算法复杂度



## PCA主成分分析

一个数据集可能有很多特征，但往往只有一小部分特征真正驱动着整个数据集，主成分分析可以将很多特征合并得到几个起到主要影响的“隐含特征”

一个数据集的主成分是由**数据中具有最大方差的方向**决定的

## 人脸识别

能有很多特征，但往往只有一小部分特征真正驱动着整个数据集，主成分分析可以将很多特征合并得到几个起

## 特征选择：wrapping & filter

特征选择算法可以被视为**搜索技术和评价指标**的结合。前者提供候选的新特征子集，后者为不同的特征子集打分。 最简单的算法是测试每个特征子集，找到究竟哪个子集的错误率最低。这种算法需要穷举搜索空间，难以算完所有的特征集，只能涵盖很少一部分特征子集。 选择何种评价指标很大程度上影响了算法。而且，通过选择不同的评价指标，可以把特征选择算法分为三类：包装类、过滤类和嵌入类方法[[3]](https://zh.wikipedia.org/wiki/%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9#cite_note-guyon-intro-3)

**包装类方法**使用预测模型给特征子集打分。每个新子集都被用来训练一个模型，然后用验证数据集来测试。通过计算验证数据集上的错误次数（即模型的错误率）给特征子集评分。由于包装类方法为每个特征子集训练一个新模型，所以计算量很大。不过，这类方法往往能为特定类型的模型找到性能最好的特征集。

**过滤类方法**采用代理指标，而不根据特征子集的错误率计分。所选的指标算得快，但仍然能估算出特征集好不好用。常用指标包括[互信息](https://zh.wikipedia.org/wiki/%E4%BA%92%E4%BF%A1%E6%81%AF)[[3]](https://zh.wikipedia.org/wiki/%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9#cite_note-guyon-intro-3)、逐点互信息[[4]](https://zh.wikipedia.org/wiki/%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9#cite_note-textcat-4)、皮尔逊积矩相关系数、每种分类/特征的组合的帧间/帧内类距离或[显著性测试](https://zh.wikipedia.org/wiki/%E5%81%87%E8%A8%AD%E6%AA%A2%E5%AE%9A)评分。[[4]](https://zh.wikipedia.org/wiki/%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9#cite_note-textcat-4)[[5]](https://zh.wikipedia.org/wiki/%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9#cite_note-5) 过滤类方法计算量一般比包装类小，但这类方法找到的特征子集不能为特定类型的预测模型调校。由于缺少调校，过滤类方法所选取的特征集会比包装类选取的特征集更为通用，往往会导致比包装类的预测性能更为低下。不过，由于特征集不包含对预测模型的假设，更有利于暴露特征之间的关系。许多过滤类方法提供特征排名，而非显式提供特征子集。要从特征列表的哪个点切掉特征，得靠[交叉验证](https://zh.wikipedia.org/wiki/%E4%BA%A4%E5%8F%89%E9%A9%97%E8%AD%89)来决定。过滤类方法也常常用于包装方法的预处理步骤，以便在问题太复杂时依然可以用包装方法。

## CNN卷积神经网络

<https://blog.csdn.net/zouxy09/article/details/8781543>

## 最大期望（EM）算法

最大期望（EM）算法是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐性变量。 最大期望算法经常用在机器学习和计算机视觉的数据聚类（Data Clustering）领域