数据处理：如何处理缺失数据(missing value)? 各种处理方法有什么利弊？

数据处理：如何将描述变量(categorical variables)转为连续变量(continuous variables)数据处理：如何进行选择特征选择？如何进行数据压缩（特征提取）？

特征选择：包裹式，过滤式，嵌入式

数据压缩：主成分分析，自编码等

模型解释：试解释什么是欠拟合与过拟合？如何应对这两种情况？

模型解释：什么是偏差与方差分解(Bias Variance Decomposition)？与欠拟合与过拟合有什么联系？

评估模型一般有什么手段？

分类问题评估方法

回归问题评估方法

数据不均衡的评估方法

深度学习是否比其他学习模型都好？为什么？（没有免费的午餐定理）

只有少量的有标签数据的情况下，如何构建一个反保险欺诈系统？

如果面试者回答先用监督学习来做那么：

这种情况下数据是不均衡的，你是采用过采样还是欠采样？如何调整代价函数和阈值？

如果面试者提到了集成学习，那么也会追问一下为什么集成学习适合数据不平衡？

如果面试者回答用无监督学习，那我们可能会问：

比如使用One-class SVM？那么我们可能会追问一下SVM相关的问题，比如什么是最大间隔分类器啊什么是Kernel，如何选择Kernel等。

为什么K-Means不适合异常值检测？K-Means和GMM（聚类）是什么关系？是否可以用FMM（因子分解积模型）来直接拟合异常值。

如何可以得到无监督学习中的分类规则？

比如其他答案提到的L1和L2正则化，我们可能希望面试者在白板上画图介绍为什么L1可以得到稀疏解，以及和嵌入式特征选择有什么联系。

我们曾把Kaggle上泰坦尼克的那个数据集处理过以后，让面试者谈谈幸存率到底和什么有关，如何分析。同样的，我们不追求完美答案，只是希望看到求职者可以解释一些简单的模型结果，从中攫取商业价值。

**项目流程分析**

1. 读取数据（多用pandas构建DataFrame对于数据的处理十分方便）把构建的DataFrame用df指向
2. 调用df.info()查看数据的情况，主要看数据行列数，每个特征（列）其值的缺失情况。若为要执行分类的数据，调用df[‘label’].value\_counts()查看数据中代表类别的列数据的分布情况主要是看数据的分布是否均衡。
3. 进行数据预处理。填补缺失值，对于数值型的缺失值（数据量不是非常大）用随机森林进行预测进行填补，对于字符型的缺失值映射到高维的空间（例如加个no）然后进行独热编码。若牵扯到距离计算还应进行归一化处理
4. 若数据量很大，特征很多应进行特征提取，特征选择。
5. 选择合适的模型训练数据，注意要是数据不均衡应该在模型中进行相应的操作。
6. 用合适的方法评估模型，并对模型性能进行度量。不断调节优化超参数使模型泛化能力达到最强。

1.什么是过采样？

过抽样方法通过增加少数类样本来提高少数类的分类性能 ，最简单的办法是简单复制少数类样本，缺点是可能导致过拟合，没有给少数类增加任何新的信息。改进的过抽样方法通过在少数类中加入随机高斯噪声或产生新的合成样本等方法。（采样的个数大于该类样本的个数）

2.什么是欠采样？

欠抽样方法通过减少多数类样本来提高少数类的分类性能，最简单的方法是通过随机地去掉一些多数类样本来减小多数类的规模，缺点是会丢失多数类的一些重要信息，不能够充分利用已有的信息。（采样的次数少于该类样本的个素）

1. 如何解决分类问题中的样本不平衡问题

具体说明数据不均衡会带来的问题：

1. 在一个二分类问题中，训练集中class 1的样本数比class 2的样本数是60:1。使用逻辑回归进行分类，最后训练出的模型可能会忽略了class 2，即模型可能会将所有的训练样本都分类为class 1。

2）在分类任务的数据集中，有三个类别，分别为A，B，C。在训练集中，A类的样本占70%，B类的样本占25%，C类的样本占5%。最后我的分类器对类A的样本过拟合了，而对其它两个类别的样本欠拟合。

1. 过抽样

抽样是处理不平衡数据的最常用方法，基本思想就是通过改变训练数据的分布来消除或减小数据的不平衡。过抽样方法通过增加少数类样本来提高少数类的分类性能 ，最简单的办法是简单复制少数类样本，缺点是可能导致过拟合，没有给少数类增加任何新的信息。改进的过抽样方法通过在少数类中加入随机高斯噪声或产生新的合成样本等方法。

如何解决过采样中只是简单的复制少数类样本所带来的过拟合缺点？

采用过采样的典型算法SMOTE（它是通过对训练集里的小样本类别进行插值来产生额外的小样本类别数据）

1. 欠抽样

欠抽样方法通过减少多数类样本来提高少数类的分类性能，最简单的方法是通过随机地去掉一些多数类样本来减小多数类的规模，缺点是会丢失多数类的一些重要信息，不能够充分利用已有的信息。

如何解决欠采样带来的无法充分利用数据问题？

可以把样本数较多的那一类分成与少类样本相同的多份，让不同的学习器使用，最后利用bagging思想。

1. 采用代价敏感方法（常用,还无自己的具体实现需调用包中的）

重构训练集的方法。不改变已有算法，而是根据样本的不同错分代价给训练集中的每一个样本赋一个权值，接着按权重对原始样本集进行重构。引入代价敏感因子，设计出代价敏感的分类算法。通常对小样本赋予较高的代价，大样本赋予较小的代价，期望以此来平衡样本之间的数目差异。

1. 尝试产生人工数据样本

一种简单的人工样本数据产生的方法便是，对该类下的所有样本每个属性特征的取值空间中随机选取一个组成新的样本，即属性值随机采样。你可以使用基于经验对属性值进行随机采样而构造新的人工样本，或者使用类似朴素贝叶斯方法假设各属性之间互相独立进行采样，这样便可得到更多的数据，但是无法保证属性之前的线性关系（如果本身是存在的）。   
  有一个系统的构造人工数据样本的方法SMOTE(Synthetic Minority Over-sampling Technique)。SMOTE是一种过采样算法，它构造新的小类样本而不是产生小类中已有的样本的副本，即该算法构造的数据是新样本，原数据集中不存在的。它基于距离度量选择小类别下两个或者更多的相似样本，然后选择其中一个样本，并随机选择一定数量的邻居样本对选择的那个样本的一个属性增加噪声，每次处理一个属性。这样就构造了更多的新生数据。

1. 大样本/小样本个数=L对大样本进行L聚类

将大类中样本划分到L个聚类中，然后训练L个分类器，每个分类器使用大类中的一个簇与所有的小类样本进行训练得到。最后对这L个分类器采取少数服从多数对未知类别数据进行分类，如果是连续值（预测），那么采用平均值。

1. 小样本个为N对大样本进行N聚类

设小类中有N个样本。将大类聚类成N个簇，然后使用每个簇的中心组成大类中的N个样本，加上小类中所有的样本进行训练。

1. 选择合适的评价指标

常规的分类评价指标可能会失效，比如将所有的样本都分类成大类，那么准确率、精确率等都会很高。这种情况下，AUC时最好的评价指标。

4.什么是噪声？

样本中的错误信息称之为噪声。

5.什么是过拟合与欠拟合，如何解决这两类问题？

所谓过拟合就是：把训练样本自身的一些特点当作了所有潜在样本都会具有的一般性质，这样就会导致训练出的模型其泛化能力降低，这就是过拟合。

如何解决？

1. Early stopping

Early stopping便是一种通过提前截断迭代次数来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。

Early stopping方法的具体做法是，在每一个Epoch结束时（一个Epoch集为对所有的训练数据的一轮遍历）计算validation data的accuracy，当accuracy不再提高时，就停止训练。这种做法很符合直观感受，因为accurary都不再提高了，在继续训练也是无益的，只会提高训练的时间。那么该做法的一个重点便是怎样才认为validation accurary不再提高了呢？并不是说validation accuracy一降下来便认为不再提高了，因为可能经过这个Epoch后，accuracy降低了，但是随后的Epoch又让accuracy又上去了，所以不能根据一两次的连续降低就判断不再提高。一般的做法是，在训练的过程中，记录到目前为止最好的validation accuracy，当连续10次Epoch（或者更多次）没达到最佳accuracy时，则可以认为accuracy不再提高了。此时便可以停止迭代了（Early Stopping）。这种策略也称为“No-improvement-in-n”，n即Epoch的次数，可以根据实际情况取，如10、20、30……

简单来说就是训练时设定一个训练轮的次数，每一轮训练结束后计算一次准确率，纪录最高准确率，若又经过多轮训练准确率不再提高，则说明原来最高准确率那轮可以提前停止修改总训练轮数。

1. 数据集扩增

通俗得讲，数据机扩增即需要得到更多的符合要求的数据，即和已有的数据是独立同分布的，或者近似独立同分布的。一般有以下方法：

* 从数据源头采集更多数据
* 复制原有数据并加上随机噪声
* 重采样
* 根据当前数据集估计数据分布参数，使用该分布产生更多数据等

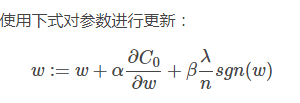
1. 正则化

即在对模型的目标函数（objective function）或代价函数（cost function）加上正则项。

描述正则化及其能预防过拟合原因很好的总结

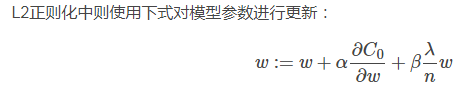
<http://blog.csdn.net/heyongluoyao8/article/details/49429629>

**L1正则化为什么可以防止过拟合？：**



从上式可以看出，当w为正时，更新后w会变小；当w为负时，更新后w会变大；因此L1正则项是为了使得那些原先处于零（即|w|≈0）附近的参数w往零移动，使得部分参数为零，从而降低模型的复杂度（模型的复杂度由参数决定。同时权重系数w≈0说明此特征和最终结果关系不大,可能是某一样本本身的特征而不是训练集中的数据普遍具有的特性，故不应加入整体模型的训练中，所以把其设为0可以提高模型的泛化能力），从而防止过拟合，提高模型的泛化能力。

**L2正则化为什么可以防止过拟合？：**



L2正则项起到使得参数w变小加剧的效果,更小的参数值w意味着模型的复杂度更低，对训练数据的拟合刚刚好（奥卡姆剃刀），不会过分拟合训练数据，从而使得不会过拟合，以提高模型的泛化能力。

1. Dropout

深度学习中用，随机的剔除隐藏层中的节点

所谓欠拟合：就是模型没有很好地捕捉到数据特征，不能够很好地拟合数据。

如何解决？

1. 添加其他特征项

有时候我们模型出现欠拟合的时候是因为特征项不够导致的，可以添加其他特征项来很好地解决。例如，“组合”、“泛化”、“相关性”三类特征是特征添加的重要手段

1. 减少正则化参数

正则化的目的是用来防止过拟合的，但是现在模型出现了欠拟合，则需要减少正则化参数。

1. 什么是偏差-方差分解，与欠拟合与过拟合有什么联系？

偏差-方差分解是解释学习算法泛化性能的一种重要工具，它试图对学习算法的期望泛化错误率进行分解。泛化误差=偏差+方差+噪声

具体解释见7.

一般偏差与方差是有冲突的，这称为偏差-方差窘境。当训练不足时，学习器的拟合能力不强，训练数据的扰动不足以使学习器产生显著的变化，这时候偏差较大，偏差主导了泛化错误率，同时会导致欠拟合。随着训练程度的加深，学习器的拟合能力逐渐增强，训练数据发生的扰动逐渐能被学习器学到，方差变大，逐渐主导了泛化错误率，一定程度上会导致过拟合。

即为通过偏差-方差分解后，偏差主导泛化错误率时会导致欠拟合，方差主导泛化错误率时会导致过拟合。

7.什么是泛化误差、偏差（残差）、方差和噪声？

这里首先解释一下bias和variance的概念。模型的Error = Bias + Variance，Error反映的是整个模型的准确度，Bias反映的是模型在样本上的输出与真实值之间的误差，即模型本身的精准度，Variance反映的是模型每一次输出结果与模型输出期望之间的误差，即模型的稳定性。

机器学习的目标是使得泛化误差最小。

泛化误差=偏差+方差+噪声

对测试样本，令Yd为x在数据集中的标记，y为x的真实标记

噪声=（Yd-y）的平方

偏差度量了学习算法的期望预测与真实结果的偏离程度，即刻画了学习算法本身的拟合能力；方差度量了同样大小的训练集的变动所导致的学习性能的变化，即刻画了数据扰动所造成的影响。噪声则表达了当前任务上任何学习算法所能达到的期望泛化误差的下界，即刻画了学习问题本身的难度。

8.目标函数、损失函数、代价函数的区别与联系

为了评估模型拟合的好坏，通常用损失函数（觉得严格来说相当于下面的目标函数）来度量拟合的程度。损失函数极小化，意味着拟合程度最好，对应的模型参数即为最优参数。

每一个算法都有一个目标函数（objective function），算法就是让这个目标函数达到最优。对于分类的算法，都会有对错。错了就会有代价，或者说是损失。分类算法的目标就是让它错的最少，也就是代价最小。

损失函数又叫误差函数（预测值与真实值之间的误差），用来衡量算法的运行情况。损失函数只适用于衡量算法在单个训练样本中的表现。它主要是配合反向传播使用的，为使得在反向转播中可以找到最小值，所以损失函数必须是可导的。

但是我们需要衡量算法在全部训练样本上的表现，这就需要我们定义一个代价函数（又称成本函数），代价函数是m个损失函数求和再除以m。

损失函数再加上正则项或者别的什么优化项就叫目标函数。

9.缺失值的处理方法以及各种方法的优劣

1. 用数值进行填充

用平均值、中值、分位数、众数、随机值等替代。简便快速但是效果一般，因为等于人为增加了噪声。

data\_train.fillna(0)所有缺失值用0填充

data\_train.fillna（data\_train.mean()）所有缺失值用各自列的均值填充

data\_train.fillna（data\_train.mean()[‘a’:’b’]）只对a,b列

data\_train[‘a’].fillna（data\_train.mean())只对a列

data\_train.fillna（method=’pad’）#用前一个数据填充

data\_train.fillna（method=’bfill’）#用后一个数据填充

1. 用算法拟合进行填充（常用的是随机森林算法）

相对一较为准确。但是有一个根本缺陷，如果其他变量和缺失变量无关，则预测的结果无意义。如果预测结果相当准确，则又说明这个变量是没必要加入建模的。

age = test\_data[[**'Age'**, **'Sex'**, **'Parch'**, **'SibSp'**, **'Pclass'**]]  
age\_notnull = age.loc[(test\_data.Age.notnull())]  
age\_isnull = age.loc[(test\_data.Age.isnull())]  
X = age\_notnull.values[:, 1:]  
Y = age\_notnull.values[:, 0]  
rfr = RandomForestRegressor(n\_estimators=1000, n\_jobs=-1)  
rfr.fit(X, Y)  
predictAges = rfr.predict(age\_isnull.values[:, 1:])  
test\_data.loc[(test\_data.Age.isnull()), **'Age'**] = predictAges

1. 对于缺失值很大的列直接删除，或者是映射到高维

映射到高维举例方法如下：若性别一列缺失较大，则可映射为男0，女1，不知2

**def** fill\_sex(df):  
 df.ix[train\_data.Sex == **'male'**, **'Sex'**] = 0  
 df.ix[train\_data.Sex == **'female'**, **'Sex'**] = 1  
 df.ix[train\_data.Sex.isnull(), **'Sex'**] = 2  
 **return** df  
train\_data=fill\_sex(train\_data)

这样做的好处是完整保留了原始数据的全部信息、不用考虑缺失值、不用考虑线性不可分之类的问题。缺点是计算量大大提升。而且只有在样本量非常大的时候效果才好，否则会因为过于稀疏，效果很差。

10.机器学习中模型的评估方法、性能度量和结果的检验方法

模型评估与选择中用于评估测试的数据称为“验证集”，在研究对比不同算法的泛化性能时，我们用测试集上的判别效果来估计模型在实际使用时的泛化能力，而把训练数据另外划分为训练集与验证集，基于验证集上的性能来进行模型选择与调参。

在对算法进行训练时，我们必须要有相应的数据。我们并不能在所有数据上进行训练，否则就没有数据来对算法的性能进行验证了。这就涉及到训练集与测试集划分的问题，即评估方法。

算法在训练好了之后，需要将其在数据集上进行测试，如何来衡量测试的结果，这就是性能的度量。

有了实验评估方法和性能度量，看起来就能对学习算法的性能进行评估比较了：先使用某种实验评估方法测得学习算法的某个性能度量结果，然后对这些结果进行比较。但怎样来做这个比较呢？是直接取得性能度量的值然后比较吗？实际上，机器学习中性能比较这件事要复杂的多。这里涉及几个重要的因素：首先，我们希望比较的是泛化性能，然而通过实验评估方法获得的是测试集上的性能，两者对比的结果可能未必相同；第二，测试集上的性能与测试集本身的选择有很大的关系，且使用不同大小的测试集会得到不同的结果，即便使用同样大小的测试集，若包含的测试用例不一样，测试的结果也会有不同；第三，很多机器学习算法本身有一定的随机性，即便使用相同的参数设置在同一个测试集上多此运行，其结果也会有不同。所以要有检验的方法来获取结果的可信度。

1. 评估方法

通常我们可以通过实验测试来对机器学习的泛化误差进行评估并进而做出选择，为此，需要用到一个验证集来测试学习器对新样本的判别能力，然后以验证集上的“验证误差”作为泛化误差的近似。

留出法：

直接将数据集划分为两个互斥的集合，一个作为训练集一个作为验证集。

首先需要考虑的就是两个集合中样本数的比例。如果训练集中数据太多，则估计的误差结果可能不准确。如果训练数据太少，则有可能会造成欠拟合。一般来说，训练集的比例大约是2/3 ~ 4/5。

这种划分要尽可能保持数据分布的一致性，避免因数据划分引入额外的偏差而对最终的结果产生影响。即在训练集与测试集中每个类别的样本的比例应该尽可能与原数据集一致。

在给定两个集合中样本的比例之后，分割的方法又是多种多样的。一般采用若干次随机划分、重复进行实验评估后取平均值作为评估结果。

K折交叉验证：

K折交叉验证的极端是：若有m个样本把样本划分为m个子集然后循环以其中一个样本作为验证集其它作为训练集。这样训练出的模型是比较准确的，但是若数据量过大时其计算开销太大。

自助法：

给定包含m个样本的数据集D，我们对它进行采样产生数据集D`,每次随机从D中挑选一个样本，将其拷贝放入D`,然后再将该样本放回初始数据集D中，使得该样本在下次采样时仍有可能被采到，这个过程重复执行m次后，我们可得到包含m个样本的数据集D`作为训练集，因为D~中必然会出现重复的数据，那么那些在D中而在D`中未出现过的数据会被作为测试集。

自助法在数据集较小，难以有效划分训练集与测试集时很有用，此外通过自助法能从初始数据中产生多个不同的训练集，这对集成学习等方法有很大好处。

1. 性能度量

性能度量是衡量模型泛化能力的评价标准，模型的好坏是相对的，模型的好坏不仅取决于算法和数据，还决定于任务的需求。

**回归任务中常用的性能度量是：**均方误差（越小越好），解释方差分（越接近1越好）

**分类任务中常用的性能度量：**

最常用的是错误率（分类错误的样本数占样本总数的比例）和精度（分类正确的样本数占样本总数的比例）

查准率（precision）亦称准确率，查全率（recall）亦称召回率

查准率=真正例/（真正例+假正例）“检索出的信息中有多少比例是用户感兴趣的”，在推荐系统中为了尽少的打扰用户，更希望推荐的内容是用户感兴趣的，此时查准率更重要。

查全率=真正例/（真正例+假反例）“用户感兴趣的信息中有多少被检索出来了”

查准率与查全率是一对矛盾的度量。一般度量学习器的好坏用F1得分

F1=（2\*真正例）/（样例总数+真正例-真反例）

ROC曲线

11.狭义的最小二乘方法、广义的最小二乘准则、梯度下降的区别

详细见：https://www.zhihu.com/question/20822481

通常我们所说的狭义的最小二乘，指的是在线性回归下采用最小二乘准则（或者说叫做最小平方），进行线性拟合参数求解的、矩阵形式的公式方法。所以，这里的「最小二乘法」应叫做「最小二乘算法」或者「最小二乘方法」，百度百科「最小二乘法」词条中对应的英文为「The least square method」。

最好的情况莫过于在有一些模型中，能用一种非数值方法求得全局精确解。这样的模型比如说orinigal linear regression model，它已经简单到只要你直接将假设好的模型代入到最小二乘法的公式里，利用矩阵微积分就能得到解的精确表示。

这里，基于线性回归，有两个细节比较重要：

　　第一，线性回归的模型假设，这是最小二乘方法的优越性前提，否则不能推出最小二乘是最佳（即方差最小）的无偏估计，具体请参考高斯-马尔科夫定理。特别地，当随机噪声服从正态分布时，最小二乘与最大似然等价。

　　第二，由于是线性回归/拟合，因此可以很容易的求出全局最优的闭式解close form solution，也即我们通常看到的那几个矩阵形式，给了input data可以一步到位算拟合参数，而不是像梯度下降法或者牛顿法那样一点点地迭代优化调参最后到达极值点。

　　而广义的最小二乘，指的是上文提到过的最小二乘准则，本质上是一种evaluation rule或者说objective funcion，这里的「最小二乘法」应叫做「最小二乘法则」或者「最小二乘准则」，英文可呼为LSE（least square error）。

　　举个例子，我要优化一个深度神经网络DNN（Deep neural network）的网络参数（换言之，优化此网络对于已知数据拟合结果的正确性），可不可以用最小二乘准则去衡量某一拟合结果相对于标准答案的偏差程度呢？可以。而同时，由于DNN模型本身的复杂性，我们没有办法像线性拟合时那样，在理论和公式的层面求出一个close form solution，因此需要引入所谓的BP算法（实质上就是梯度下降法）进行参数的迭代求解。

　　But（^\_^），上面虽然给出了最小二乘准则+梯度下降法串联使用的例子，但实际的拟合效果必定会比较一般，原因在于DNN这一体系相当于非线性回归，因此最小二乘不好，反而是logistic回归+最大似然=交叉熵准则Cross Entropy在DNN参数优化算法中的更有效和广泛一些。当然，这就是另一个话题了。  
综上：  
　　狭义的最小二乘方法，是线性假设下的一种有闭式解的参数求解方法，最终结果为全局最优；  
　　梯度下降法，是假设条件更为广泛（无约束）的，一种通过迭代更新来逐步进行的参数优化方法，最终结果为局部最优；  
　　广义的最小二乘准则，是一种对于偏差程度的评估准则，与上两者不同。

12.常见的损失函数有哪些？

具体见：

https://blog.csdn.net/iqqiqqiqqiqq/article/details/77413541

1）0-1损失函数

记录分类错误的次数。

2）绝对值损失函数

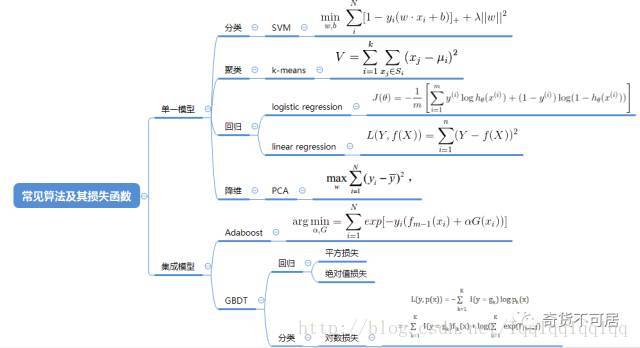
通常用于回归中

3）平方损失函数

即真实值与预测值之差的平方和。通常用于线性模型中，如线性回归模型。之所以采用平方的形式，而非绝对值或三次方的形式，是因为最大似然估计（求损失函数的极小值）与最小化平方损失是等价的。

4）对数损失

5）指数损失函数



13.常用的优化方法有哪些？

对损失函数的优化：

当我们对分类的Loss进行改进的时候，我们要通过梯度下降，每次优化一个step大小的梯度，这个时候我们就要求Loss对每个权重矩阵的偏导，然后应用链式法则。

最小二乘法（主要是说线性回归中的优化算法）梯度下降法、牛顿法、拟牛顿法、共轭梯度法

在求解机器学习算法的模型参数，即无约束优化问题时，梯度下降（Gradient Descent）是最常采用的方法之一，梯度下降不一定能够找到全局的最优解，有可能是一个局部最优解。当然，如果损失函数是凸函数，梯度下降法得到的解就一定是全局最优解。



1. 梯度

在微积分里面，对多元函数的参数求∂偏导数，把求得的各个参数的偏导数以向量的形式写出来，就是梯度。

那么这个梯度向量求出来有什么意义呢？他的意义从几何意义上讲，就是函数变化增加最快的地方。或者说，沿着梯度向量的方向，更加容易找到函数的最大值。反过来说，沿着梯度向量相反的方向，也就是 -(∂f/∂x0, ∂f/∂y0)T的方向，梯度减少最快，也就是更加容易找到函数的最小值。

1. 梯度下降与梯度上升

在机器学习算法中，在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，通过启发式的方式一步步迭代求解函数的最小值,得到最小化的损失函数，和模型参数值。反过来，如果我们需要求解损失函数的最大值，这时就需要用梯度上升法来迭代了。

　　梯度下降法和梯度上升法是可以互相转化的。比如我们需要求解损失函数f(θ)的最小值，这时我们需要用梯度下降法来迭代求解。但是实际上，我们可以反过来求解损失函数 -f(θ)的最大值，这时梯度上升法就派上用场了。

1. 梯度下降的算法调优

在使用梯度下降时，需要进行调优。

第一、算法的步长选择。在前面的算法描述中，我提到取步长为1，但是实际上取值取决于数据样本，可以多取一些值，从大到小，分别运行算法，看看迭代效果，如果损失函数的值在变小，说明取值有效，否则要增大步长。前面说了。步长太大，会导致迭代过快，甚至有可能错过最优解。步长太小，迭代速度太慢，很长时间算法都不能结束。所以算法的步长需要多次运行后才能得到一个较为优的值。

第二、算法参数的初始值选择。初始值不同，获得的最小值也有可能不同，因此梯度下降求得的只是局部最小值；当然如果损失函数是凸函数则一定是最优解。由于有局部最优解的风险，需要多次用不同初始值运行算法，观测损失函数的最小值，选择损失函数最小化的初值。

第三、归一化。由于样本不同特征的取值范围不一样，可能导致迭代很慢，为了减少特征取值的影响，可以对特征数据归一化，也就是对于每个特征x，求出它的期望x¯和标准差std(x)，然后转化为x−x¯¯¯std(x)x−x¯std(x)

这样特征的新期望为0，新方差为1，迭代次数可以大大加快。

1. 梯度下降的种类
2. 批量梯度下降法。每次对参数的更新都需要用到整个训练数据集，所以导致在训练集很大时训练速度很慢。
3. 随机梯度下降法。他是每次 只选用一个样本进行迭代，训练速度固然得到很大提升，但是准确度下降了，得到的很可能不是最优解。
4. 小批量（部分）梯度下降法，它是以上两者的结合。

对于损失函数与模型整体训练的理解

损失函数（loss function）是用来估量你模型的预测值f(x)与真实值Y的不一致程度，它是一个非负实值函数,通常使用L(Y, f(x))来表示，损失函数越小，模型的鲁棒性就越好。损失函数是经验风险函数的核心部分，也是结构风险函数重要组成部分。模型的结构风险函数包括了经验风险项和正则项，通常可以表示成如下式子：

IMG_256

　　通常损失函数(严格来说是目标函数)由上面公式的两部分组成，前部分就是计算算法预测的值和训练样本真实标签之间的距离，不同的距离计算方式代表了不同的计算损失函数的方法。第二部分J(f)代表了正则化选项，当训练的出函数过于复杂时，可能会导致训练的参数过拟合了，这时需要引入正则化因子来控制模型的复杂度。防止过拟合的产生。

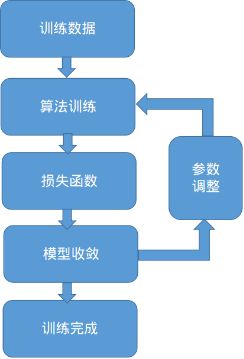
　介绍完损失函数我们介绍一下参数学习的方式，

　　　　　　　　ωj = ωj - λ ∂L(ωj) / ∂ωj

通过计算损失函数关于w参数的梯度来逐步调整w参数，使损失函数越来越小，完成模型的训练，参数达到收敛。

**算法模型的训练：**

算法模型的训练就是通过利用损失函数来衡量模型是否已经收敛，如果未收敛，通过计算损失函数的梯度，沿着梯度下降的方向不断调整参数的值，重新计算算法的输出。周而复始，不断迭代直到模型收敛，损失函数达到一个极小值为止。



14.常用的特征选择方法有哪些？

**特征选择是从样本集中选取重要的特征子集**

比较有名的特征选择有过滤法（Filter），包裹法(Wrapper)，嵌入法(Embedded)

先介绍几个和特征选择相关的名词：

**特征发散：**如果特征不发散，也就是说特征的方差趋近于0，则代表这个特征上不同样本之间没有差异性，对区分样本的作用基本不存在。

**特征与目标的相关性：**所谓相关性，就是说特征和目标值之间存在正相关（随着目标值的变大特征值也逐渐变大）或者负相关的特性。代表了特征值和目标值之间具有很强的数据上的因果关系。

1. 过滤法

过滤法就是按照发散性或者相关性对各个特征进行评分，设定阈值或者选择阈值的个数，完成特征选择。

1. **方差法：**这种方法通过计算每个特征的的均值和方差，设定一个基础阈值，当该维度的特征方差小于基础阈值时，则丢弃该特征。这种方法简单高效的过滤了一些低方差的特征，但是存在一个问题就是阈值的设定是一个先验条件，当设置过低时，保留了过多低效的特征，设置过高则丢弃了过多有用的特征。

对应的sklearn用法：方差为0的特征会被自动移除。剩下的特征按设定的方差的阈值进行选择。

#导入sklearn库中的VarianceThreshold

from sklearn.feature\_selection import VarianceThreshold

#设置方差的阈值为0.8

sel = VarianceThreshold(threshold=0.8)

#选择方差大于0.8的特征

X\_sel=sel.fit\_transform(X\_scaler)

1. **单变量特征选择：**单变量特征选择能够对每一个特征进行测试，衡量该特征和响应变量之间的关系，根据得分扔掉不好的特征。单变量特征选择方法,独立的衡量每个特征与响应变量之间的关系

**卡方检验：**对于回归和分类问题可以采用卡方检验等方式对特征进行测试。导入特征选择库中的SelectKBest和chi2用来计算特征与结果间的相关性，并选择相关性最高的特征。

#导入sklearn库中的SelectKBest和chi2

from sklearn.feature\_selection import SelectKBest ,chi2

#返回的是训练数据中与类别相关性最高的前5个特征

X\_chi2 = SelectKBest(chi2, k=5).fit\_transform(X, y)

**互信息法选择特征样例**：

from sklearn.feature\_selection import SelectKBest

from minepy import MINE #由于MINE的设计不是函数式的，定义mic方法将其为函数式的，返回一个二元组，二元组的第2项设置成固定的P值0.5

def mic(x, y):

m = MINE()

m.compute\_score(x, y)

return (m.mic(), 0.5)

#选择K个最好的特征，返回特征选择后的数据

SelectKBest(lambda X, Y: array(map(lambda x:mic(x, Y), X.T)).T, k=2).fit\_transform(iris.data, iris.target)

**基于学习模型的特征排序 (Model based ranking)：**

这种方法的思路是直接使用你要用的机器学习算法，针对每个单独的特征和响应变量建立预测模型。要注意过拟合问题，因此树的深度最好不要太大，再就是运用交叉验证。具体例子如下：

from sklearn.cross\_validation import cross\_val\_score, ShuffleSplit

from sklearn.datasets import load\_boston

from sklearn.ensemble import RandomForestRegressor

#Load boston housing dataset as an example

boston = load\_boston()

X = boston["data"]

Y = boston["target"]

names = boston["feature\_names"]

Rf=RandomForestRegressor(n\_estimators=20, max\_depth=4)

scores = []

for i in range(X.shape[1]):

score = cross\_val\_score(rf, X[:, i:i+1], Y, scoring="r2",cv=ShuffleSplit(len(X), 3, .3))

scores.append((round(np.mean(score),3),names[i]))

print sorted(scores, reverse=True)

1. **包裹法(Wrapper)**

所谓包裹法就是选定特定算法，然后再根据算法效果来选择特征集合。

就是通过不断的启发式方法来搜索特征，主要分为如下两类。

方法一：选择一些特征，逐步增加特征保证算法模型精度是否达标。

方法二：删除一些特征，然后慢慢在保持算法精度的条件下，缩减特征。

**即为选用那些本就提供特征重要性测量的模型，直接调用相应方法进行特征选择。**

1. **利用线性回归模型。**

from sklearn.linear\_model import LinearRegression

size = 100np.random.seed(seed=5)

X\_seed = np.random.normal(0, 1, size)X1 = X\_seed + np.random.normal(0, .1, size)X2 = X\_seed + np.random.normal(0, .1, size)X3 = X\_seed + np.random.normal(0, .1, size)

Y = X1 + X2 + X3 + np.random.normal(0,1, size)X = np.array([X1, X2, X3]).T

lr = LinearRegression()

lr.fit(X,Y)

print "Linear model:", pretty\_print\_linear(lr.coef\_)

输出为：Linear model: -1.291 \* X0 + 1.591 \* X1 + 2.747 \* X2

这个不常用，因为真实数据的线性关系不是很好，故应选择能处理非线性的随机森林模型，它精确度更高，也提供预测特征重要性的方法。主要是为了学习pretty\_print\_linear(lr.coef\_)这句代码的作用。

1. **利用随机森林模型**

随机森林提供了两种特征选择的方法：mean decrease impurity和mean decrease accuracy。

1. 平均不纯度减少mean decrease impurity：

*#encoding=utf-8***from** sklearn.datasets **import** load\_boston  
**from** sklearn.ensemble **import** RandomForestRegressor  
**import** numpy **as** np  
*#Load boston housing dataset as an example*boston = load\_boston()  
X = boston[**"data"**]  
Y = boston[**"target"**]  
names = boston[**"feature\_names"**]  
rf = RandomForestRegressor()  
rf.fit(X, Y)  
print(names)  
print(rf.feature\_importances\_)  
print(**"Features sorted by their score:"**)  
print(sorted(zip(map(**lambda** x: round(x, 4), rf.feature\_importances\_), names), reverse=**True**))

#若没指定key，则以前面的值作为key

Features sorted by their score:

[(0.4647, 'RM'), (0.3579, 'LSTAT'), (0.0446, 'DIS'), (0.0369, 'CRIM'), (0.0303, 'NOX'), (0.017, 'AGE'), (0.0135, 'PTRATIO'), (0.0128, 'B'), (0.0127, 'TAX'), (0.0041, 'INDUS'), (0.0038, 'RAD'), (0.0012, 'ZN'), (0.0005, 'CHAS')]

1. 平均精度减少具体见：<http://www.17bigdata.com/%E7%BB%93%E5%90%88scikit-learn%E4%BB%8B%E7%BB%8D%E5%87%A0%E7%A7%8D%E5%B8%B8%E7%94%A8%E7%9A%84%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9%E6%96%B9%E6%B3%95.html>
2. **以GBDT为基模型结合SelectFromModel进行特征选择**

**from** sklearn.ensemble **import** GradientBoostingClassifier

**from** sklearn.feature\_selection **import** SelectFromModel

ss=SelectFromModel(GradientBoostingClassifier()).fit\_transform(X,y)

返回的数据即为经过特征选择后的训练数据，以此为X结合y进行数据集划分。SelectFromModel（）就第一个参数有用即为指定的基模型。

1. **用SelectKBest()进行选择**

SelectKBest也是要基于类别标签（先验值）的

在sklearn中的用法如下

**from** sklearn.feature\_selection **import** SelectKBest

selection = SelectKBest()

#注意SelectKBest()接收两个参数，一个是得到恰当特征的得分函数（默认为[f\_classif](http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html" \l "sklearn.feature_selection.f_classif" \o "sklearn.feature_selection.f_classif)）

#另外一个是选择的特征个数k（默认为10）

#返回的是选择出的最佳k个特征

features=Selection.fit\_transform(X,y)

1. **嵌入法(Embedded)**

就是利用正则化的思想，将部分特征属性的权重调整到0，则这个特性相当于就是被舍弃了。（其实就是在损失函数上再加入正则项，不断的利用梯度下降极小化损失函数，调整一些特征的权重，有些权重变为0了则相当于被舍弃了，没被舍弃的相当于被选择出来的向量。）

嵌入法的基本原理是通过在均方误差的基础上加入的正则化的矫正因子（L1,L2都属于这一类）通过不断通过求最小值的方法优化这个混合的损失函数，可以得出有些w权重因子为0，完成损失函数的正则化，避免了过拟合

具体见：<http://www.17bigdata.com/结合scikit-learn介绍几种常用的特征选择方法.html的3.1>

**L1正则方法具有稀疏解的特性，因此天然具备特征选择的特性，但是要注意，L1没有选到的特征不代表不重要，原因是两个具有高相关性的特征可能只保留了一个，**如果要确定哪个特征重要应再通过L2正则方法交叉检验；

1. **顶层特征选择法**

具体见：[http://www.17bigdata.com/结合scikit-learn介绍几种常用的特征选择方法.html的5.1](http://www.17bigdata.com/结合scikit-learn介绍几种常用的特征选择方法.html的3.1)

5）**通过特征组合之后再选择特征**如对用户id和用户特征最组合来获得较大的特征集再来选择特征，这种做法在推荐系统和广告系统中比较常见，这也是所谓亿级甚至十亿级特征的主要来源，原因是用户数据比较稀疏，组合特征能够同时兼顾全局模型和个性化模型

15.常用的特征提取方法有哪些？

常用的方法有主成分分析（PCA），独立成分分析（ICA），线性判别分析（LDA）一般数据是有类别的，最好先考虑用LDA降维。也可**先用小幅度的PCA降维消除噪声**再用LDA降维，若训练数据没有类别优先考虑PCA。

**特征提取是由原始输入形成较少的新特征**，为了使得训练出的模型更加健壮，若不是数据量很大特征种类很多，一般不要用特征提取。

1. PCA

**作为一个非监督学习的降维方法**，它只需要特征值分解，就可以对数据进行压缩，去噪。因此在实际场景应用很广泛。为了克服PCA的一些缺点，出现了很多PCA的变种，比如**第六节的为解决非线性降维的KPCA，**还有解决内存限制的增量PCA方法Incremental PCA，以及解决稀疏数据降维的PCA方法Sparse PCA等。

**PCA是最常用的线性降维方法**，它的目标是通过某种线性投影，将高维的数据映射到低维的空间中表示，并期望在所投影的维度上数据的方差最大（样本的分布最散乱）以使用较少的数据维度同时保留住较多的原数据点的特征。**注意：线性降维必然会损坏原数据的特性，所以只有在数据特别多或者特征特别多时再使用。**

在sklearn中的应用：

**from** sklearn.decomposition **import** PCA

pca=PCA(n\_components=**'mle'**)  
fit\_data=pca.fit\_transform(X)  
X\_train,X\_val,y\_train,y\_val=train\_test\_split(fit\_data,y,test\_size=0.25,random\_state=5)  
rf=RandomForestClassifier()  
rf.fit(X\_train,y\_train)  
yHat=rf.predict(X\_val)  
acc=metrics.accuracy\_score(yHat,y\_val)

降维后准确度降低，因为特征过少。

使用时需注意的参数：

n\_components=**'mle'，**则由系统自主的选择合适的降维后的维度。

n\_components=**3，表示降维到3维（由原始数据构建出3个新的特征）**

除了这一输入参数外，有两个PCA类的成员值得关注。第一个是explained\_variance\_，它代表降维后的各主成分的方差值。方差值越大，则说明越是重要的主成分。第二个是explained\_variance\_ratio\_，**它代表降维后的各主成分的方差值占总方差值的比例，这个比例越大，则越是重要的主成分。应该根据这一值选择合适的降维后的维度。**

第三个是n\_components\_表示降维后的维度即特征的个数。

PCA只适用于对稠密矩阵进行降维，如果是稀疏矩阵（例如在推荐系统中用户和物品构成的矩阵，由于评论很少，组成的是稀疏矩阵）应该用SparePCA，它是PCA的变种，适用于对稀疏矩阵进行降维，sklearn对其也有支持。

**PCA的优缺点分析：**

优点：

第一、仅仅需要以方差衡量信息量，不受数据集以外的因素影响。第二、各主成分之间正交，可消除原始数据成分间的相互影响的因素。第三、计算方法简单，主要运算是特征值分解，易于实现。

缺点：

第一、主成分各个特征维度的含义具有一定的模糊性，不如原始样本特征的解释性强。第二、PCA会消除一些类信息，但是方差小的非主成分也可能含有对样本差异的重要信息，因降维丢弃可能对后续数据处理有影响。

1. LDA

LDA是一种监督学习的降维技术，也就是说它的数据集的每个样本是有类别输出的。LDA的思想可以用一句话概括，就是“投影后类内方差最小，类间方差最大”。什么意思呢？ 我们要将数据在低维度上进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。

**LDA的优缺点分析：**

LDA算法的主要优点有：

　　　　1）在降维过程中可以使用类别的先验知识经验，而像PCA这样的无监督学习则无法使用类别先验知识。

　　　　2）LDA在样本分类信息依赖均值而不是方差的时候，比PCA之类的算法较优。

　　　　LDA算法的主要缺点有：

　　　　1）LDA不适合对非高斯分布样本进行降维，PCA也有这个问题。

　　　　2）LDA降维最多降到类别数k-1的维数，如果我们降维的维度大于k-1，则不能使用LDA。当然目前有一些LDA的进化版算法可以绕过这个问题。

　　　　3）LDA在样本分类信息依赖方差而不是均值的时候，降维效果不好。

　　　　4）LDA可能过度拟合数据。

**LDA与PCA的比较：**

首先我们看看相同点：

　　　　1）两者均可以对数据进行降维。

　　　　2）两者在降维时均使用了矩阵特征分解的思想。

　　　　3）两者都假设数据符合高斯分布。

　　　　我们接着看看不同点：

　　　　1）LDA是有监督的降维方法，而PCA是无监督的降维方法

　　　　2）LDA降维最多降到类别数k-1的维数，而PCA没有这个限制。

　　　　3）LDA除了可以用于降维，还可以用于分类。

　　　　4）LDA选择分类性能最好的投影方向，而PCA选择样本点投影具有最大方差的方向。

LDA的sklearn应用：

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n\_components=2)

lda.fit(X,y)

X\_new = lda.transform(X)

**LDA与PCA的选择：**

一般来说，如果我们的数据是有类别标签的，那么优先选择LDA去尝试降维；当然也可以使用PCA做很小幅度的降维去消去噪声，然后再使用LDA降维。如果没有类别标签，那么肯定PCA是最先考虑的一个选择了。

特征提取和特征选择的区别

特征选择和降维有着些许的相似点，这两者达到的效果是一样的，就是试图去减少特征数据集中的属性(或者称为特征)的数目；但是两者所采用的方式方法却不同：降维的方法主要是通过属性间的关系，如组合不同的属性得到新的属性，这样就改变了原来的特征空间；而特征选择的方法是从原始特征数据集中选择出子集，是一种包含的关系，没有更改原始的特征空间。

特征组合解释

而wrapper是将子集的选择看作是一个搜索寻优问题，生成不同的组合，对组合进行评价，再与其他的组合进行比较。这样就将子集的选择看作是一个是一个优化问题

两个特征进行组合的方法是，把每个特征交叉组合？（待验证）

16.常用的数据预处理方法有哪些？

在sklearn中的使用具体可以参考：

<https://blog.csdn.net/q383700092/article/details/54571887>

其实现在**标准化（正则化）**和**归一化**都会混用，常见的标准化方法为z-score标准化（或者去除均值和标准差缩放）公式为：(X-mean)/std  计算时对每个属性/每列分别进行。将数据按照属性（按列进行）减去其均值，并处以其标准差。**得到的结果是，对于每个属性/每列来说所有数据都聚集在0附近，标准差为1。**经过处理后的数据均值为0，标准差为1。**均值为0,标准差为1的正态分布称为标准正态分布**

对应sklearn中方法为：

from sklearn import preprocessing

X\_scaled = preprocessing.scale(X)

#处理后数据的均值和方差

>>> X\_scaled.mean(axis=0)

array([ 0.,  0.,  0.])

>>> X\_scaled.std(axis=0)

array([ 1.,  1.,  1.])

以及范围缩放标准化，常用的缩放范围是（0-1）缩放，这时又对应了归一化。

对应sklearn中方法：

data\_scaler=preprocessing.MinMaxScaler(feature\_range(0,1))

data\_scaler.fit\_transform(data)

那么标准化与归一化有哪些好处，如何用？

如何用？**在涉及到计算点与点之间的距离时，使用归一化或标准化都会对最后的结果有所提升，甚至会有质的区别。**那在归一化与标准化之间应该如何选择呢？根据上一节我们看到，如果把所有维度的变量一视同仁，在最后计算距离中发挥相同的作用应该选择标准化，如果想保留原始数据中由标准差所反映的潜在权重关系应该选择归一化。另外，标准化更适合现代嘈杂大数据场景。

归一化的好处：**一是为了后面数据处理的方便，把不同量纲的东西放在同一量纲下比较，即把不同来源的数据统一到一个参考系下，这样比较起来才有意义。**简单的举个例子：一张表有两个变量，一个是体重kg，一个是身高cm。假设一般情况下体重这个变量均值为60（kg），身高均值为170（cm）。1，这两个变量对应的单位不一样，同样是100，对于身高来说很矮，但对于体重来说已经是超重了。另外，单位越小，数值越大，对结果的影响也越大，譬如170cm=1700mm。 简单讲，归一化的目的是可以用数值来直接进行比较，如果不归一化由于变量特性不同，同样加10，代表的意义不一样。

二是，**保正程序运行时收敛加快，大部分模型归一化后收敛速度会加快。**例如，下面的例子，房间数和面积数不在一个量纲上，面积数值太小，房间数太大，成椭圆状，按照梯度收敛速度会慢，理想的是数据类似圆圈的形状，经过有限几个步骤则收敛了。

正则化不具体说见：<http://www.cnblogs.com/chaosimple/p/4153167.html>

**独热编码（one-hot Encoder()具体看24条）**

在基于距离的模型中，对于离散型特征，应该采用one-hot编码方式，而且对于很多神经网络模型，能够很好地处理稀疏特征的情况。

**对于基于树的模型中处理离散型特征不需要采用one-hot编码方式。**

from sklearn import preprocessing

enc = preprocessing.OneHotEncoder()

enc.fit([[0,0,3],[1,1,0],[0,2,1],[1,0,2]])

array = enc.transform([[0,1,3]]).toarray()

print array

17.基于基尼不纯度的CART决策树的缺点

使用基于不纯度的方法的时候，要记住：1、这种方法存在[偏向](http://link.springer.com/article/10.1186/1471-2105-8-25" \o ")，对具有更多类别的变量会更有利；2、对于存在关联的多个特征，其中任意一个都可以作为指示器（优秀的特征），并且一旦某个特征被选择之后，其他特征的重要度就会急剧下降，因为不纯度已经被选中的那个特征降下来了，其他的特征就很难再降低那么多不纯度了，这样一来，只有先被选中的那个特征重要度很高，其他的关联特征重要度往往较低。在理解数据时，这就会造成误解，导致错误的认为先被选中的特征是很重要的，而其余的特征是不重要的，但实际上这些特征对响应变量的作用确实非常接近的（这跟Lasso是很像的）。

1. 何为稀疏线性关系？

稀疏线性关系的意思就是绝大多数的特征和样本输出没有关系，线性拟合后这些特征维度的系数会全部为0，只有少量和输出相关的特征的回归系数不为0，这就是“稀疏线性关系”。

何为鲁棒性、归纳偏好和奥卡姆剃刀原理？

鲁棒性一般指模型的健壮性稳定性。

**归纳偏好：**机器学习算法在学习过程中对某种类型假设的偏好称为『归纳偏好』**。任何一个有效的**机器学习算法必有其归纳的偏好，否则它将被假设空间中看似在训练集上『等效』的假设所迷惑，而无法产生确定的学习结果。例如在分类问题中，如果随机抽选训练集上等效的假设（可以认为所有的正反例并没有区别），那么它的分类结果其实是不确定的，这要根据它所选取的样本来决定，这样的学习显然是没有意义的。

**奥卡姆剃刀原理：当两个假说具有完全相同的解释力和预测力时，我们以那个较为简单的假说作为讨论依据。**剃刀原则不是一个理论而是一个原理，它的目的是为了精简抽象实体。大部分情况下，应用奥卡姆剃刀原理是合适的；但是这不代表奥卡姆剃刀就是正确的。**剃刀原则并不是一种定理，而是一种思维方式**，**做决策树分析的时候，采用9个属性的预测性能和5个属性的预测性能是相似的，那么我们就会选择5个属性来预测。**

**什么是无偏估计？协方差？**

无偏估计是用[样本](https://baike.baidu.com/item/%E6%A0%B7%E6%9C%AC/19974592" \t "https://baike.baidu.com/item/%E6%97%A0%E5%81%8F%E4%BC%B0%E8%AE%A1/_blank)统计量来估计总体参数时的一种无偏推断。估计量的[数学期望](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6%E6%9C%9F%E6%9C%9B" \t "https://baike.baidu.com/item/%E6%97%A0%E5%81%8F%E4%BC%B0%E8%AE%A1/_blank)等于被估计参数的[真实值](https://baike.baidu.com/item/%E7%9C%9F%E5%AE%9E%E5%80%BC" \t "https://baike.baidu.com/item/%E6%97%A0%E5%81%8F%E4%BC%B0%E8%AE%A1/_blank)，则称此此估计量为被估计参数的无偏估计，即具有无偏性，是一种用于评价估计量优良性的准则。无偏估计的意义是：在多次重复下，它们的平均数接近所估计的参数真值

**举例理解：**

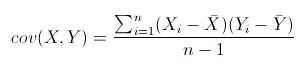
比如我要对某个学校一个年级的上千个学生估计他们的平均水平（真实值，上帝才知道的数字），那么我决定抽样来计算。

我抽出一个10个人的样本，可以计算出一个均值。那么如果我下次重新抽样，抽到的10个人可能就不一样了，那么这个从样本里面计算出来的均值可能就变了，对不对？

因为这个均值是随着我抽样变化的，而我抽出哪10个人来计算这个数字是随机的，那么这个均值也是随机的。但是这个均值也会服从一个规律（一个分布），那就是如果我抽很多次样本，计算出很多个这样的均值，这么多均值们的平均数应该接近上帝才知道的真实平均水平。

如果你能理解“样本均值”其实也是一个随机变量，那么就可以理解为这个随机变量的期望是真实值，所以无偏（这是无偏的定义）；而它又是一个随机变量，只是估计而不精确地等于，所以是无偏估计量。

**协方差：**

均值，标准差，方差这些都是对一维数据的度量，那么协方差是对多维数据的度量，它代表两个特征之间的关联程度

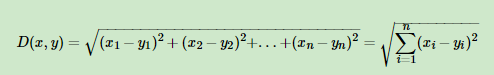
协方差的结果有什么意义呢？如果结果为正值，则说明两者是正相关的，结果为负值就说明负相关的，协方差多了就是协方差矩阵。

19.KNN详细解析

KNN做回归和分类的主要区别在于最后做预测时候的决策方式不同。KNN做分类预测时，一般是选择多数表决法，即训练集里和预测的样本特征最近的K个样本，预测为里面有最多类别数的类别。而KNN做回归时，一般是选择平均法，即最近的K个样本的样本输出的平均值作为回归预测值。

**KNN的三要素：**k值的选取，距离度量的方式和分类决策规则

可以通过交叉验证选择一个合适的k值。对于距离的度量，我们最常用的是欧式距离，即对于两个n维向量x和y，两者的欧式距离定义为：



**KNN的实现方式：**

1. 蛮力实现：即为计算预测样本和训练样本中所有点的距离，选择距离最小的前K个元素。利用投票表决法确定预测样本的类别。（或利用均值法确定预测样本的值）

其优点是实现简单，但是其缺点是当数据过大时计算量太大。

1. KD树

KD树算法没有一开始就尝试对测试样本分类，而是先对训练集建模，建立的模型就是KD树，建好了模型再对测试集做预测。

KD树算法包括三个步骤：建树，搜索最近邻，预测

**建树的过程如下：**假设训练样本有n个特征，分别计算这n个特征的方差，选择方差最大的第k个特征作为KD树的树根。然后再选取第k个特征所有样本点的中值（中位数），对于第K个特征的特征值小于中值的样本归为左子树，对于第K个特征的特征值大于等于中值的样本归为右子树。然后对左右子树按此方法循环建立KD树。

**搜索最近邻的过程如下：**当我们生成KD树以后，就可以去预测测试集里面的样本目标点了。对于一个目标点，我们首先在KD树里面找到包含目标点的叶子节点。以目标点为圆心，以目标点到叶子节点样本实例的距离为半径，得到一个超球体，最近邻的点一定在这个超球体内部。然后返回叶子节点的父节点，检查另一个子节点包含的超矩形体是否和超球体相交，如果相交就到这个子节点寻找是否有更加近的近邻,有的话就更新最近邻。如果不相交那就简单了，我们直接返回父节点的父节点，在另一个子树继续搜索最近邻。当回溯到根节点时，算法结束，此时保存的最近邻节点就是最终的最近邻。

**KD树的预测步骤如下：**在KD树搜索最近邻的基础上，我们选择到了第一个最近邻样本，就把它置为已选。在第二轮中，我们忽略置为已选的样本，重新选择最近邻，这样跑k次，就得到了目标的K个最近邻，然后根据多数表决法，如果是KNN分类，预测为K个最近邻里面有最多类别数的类别。如果是KNN回归，用K个最近邻样本输出的平均值作为回归预测值。

**其优点是大大降低了计算时间，和超球体不相交的点不再参与计算。**

**KNN算法的优缺点分析：**

**优点：**

第一、简单好用，容易理解，精度高，理论成熟，既可以用来做分类也可以用来做回归；第二、对异常值不敏感

**缺点：**

1. 计算复杂性高；空间复杂性高；第二、样本不平衡会造成很大影响。第三、最大的缺点是无法给出数据的内在含义。

**sklearn中的应用：**

from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(n\_neighbors = 15)

clf.fit(X, Y)

需要注意的是algorithm参数，它是代表算法的选择，有auto(会自动选择合适的算法，但是当数据大时训练时间会很长)‘brute’对应第一种蛮力实现，‘kd\_tree’对应第二种KD树实现，‘ball\_tree’对应第三种的球树实现。需要注意的是，如果输入样本特征是稀疏的时候，无论我们选择哪种算法，最后scikit-learn都会去用蛮力实现‘brute’。个人的经验，如果样本少特征也少，使用默认的 ‘auto’就够了。 如果数据量很大或者特征也很多，用"auto"建树时间会很长，效率不高，建议选择KD树实现‘kd\_tree’，此时如果发现‘kd\_tree’速度比较慢或者已经知道样本分布不是很均匀时，可以尝试用‘ball\_tree’。

还有距离度量metric参数，选用标准化欧式距离seuclidean

获得与测试样本最近的k个点以及距离：

samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .5]]

from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n\_neighbors=1)

neigh.fit(samples)

print(neigh.kneighbors([[1., 1., 1.]]))

(array([[ 0.5]]), array([[2]]...))

输出结果的意思是样本中的第三个点与测试样本的距离最近，距离为0.5

尽管数据不平衡会对其造成很大影响但是并没有进行相应处理，所以应该自己处理。不存在过拟合问题。牵扯到距离计算数据标准化在参数中处理

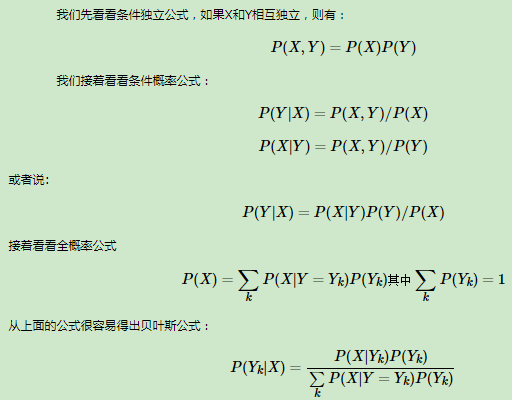
20.朴素贝叶斯详细解析

朴素贝叶斯是生成方法，也就是直接找出特征输出Y和特征X的联合分布P(X,Y),然后用

P(Y|X)=P(X,Y)/P(X)

得出。朴素贝叶斯很直观，计算量也不大，在很多领域有广泛的应用

**相应的统计学知识：**



P(某特征下是某类别的概率)=(P(这一类别中这一特征的概率)P(这一类别的概率))/P(这一特征的概率)

案例分析：常用于垃圾邮件分类

1. 读取训练数据和其对应的类别。
2. 根据所有训练数据建立自己的词袋（把训练集中出现的所有不重复单词组成一个列表）
3. 为了概率的计算把单词向量化（对每个邮件建立一个和词袋等长的全0列表，把该邮件出现的单词在词袋中对应位置元素置为1）
4. 计算词袋中每个单词在各类别中出现的概率，计算各类别的概率
5. 输入测试集，根据公式预测是各个类别的概率，判定其为最大概率的那一类别。

**该算法的优缺点分析：**

**优点：**

第一、发源于古典数学理论，有稳定的分类效率。第二、对小规模的数据表现很好，能够处理多分类任务，适合增量式训练。第三、对缺失数据不太敏感，算法也比较简单。

**缺点：**

第一、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。第二、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。第三、对于数据不平衡问题效果差。

对应的sklearn中的用法：

在scikit-learn中，一共有3个朴素贝叶斯的分类算法类。分别是GaussianNB，MultinomialNB和BernoulliNB。其中GaussianNB就是先验为高斯分布的朴素贝叶斯，MultinomialNB就是先验为多项式分布的朴素贝叶斯，而BernoulliNB就是先验为伯努利分布的朴素贝叶斯。

from sklearn.naive\_bayes import GaussianNB

clf = GaussianNB()

#拟合数据

clf.fit(X, Y)

没有距离计算问题且只是个数的计算特征的值用不到所以不用标准化，不会过拟合问题产生，只是数据不平衡会对其造成很大影响，sklearn类库中没有相应处理要自己处理。

21.线性回归的详细解析

线性回归特征的值会参与到计算，为了使每个特征具有相同的重要性，要先对数据进行标准化处理（sklearn中好像自动处理了没有自己实验过）。线性回归会牵扯到过拟合问题，具体怎么处理看下面具体算法的分析。因为是回归没有类别不平衡问题。

回归的目的是预测数值型数据的目标值。目标值的计算是通过一个线性方程得到的，这个方程称为回归方程，各未知量（特征）前的系数为回归系数，求这些系数的过程就是回归。

对于普通线性回归使用的损失函数一般为平方误差。把其用最小二乘法进行优化得到的关于系数w求导所得到的矩阵形式的表达式求得的w便为最优解了。（在sklearn中LinearRegression和Ridge使用的优化算法都是最小二乘法）

对应的sklearn的应用：

from sklearn.linear\_model import LinearRegression

linreg = LinearRegression()

linreg.fit(X\_train, y\_train)

拟合完毕后，我们看看我们的需要的模型系数结果：

print linreg.intercept\_

print linreg.coef\_

输出如下：

[ 447.06297099]

[[-1.97376045 -0.23229086 0.0693515 -0.15806957]]

这样我们就得到了在步骤1里面需要求得的5个值。也就是说PE和其他4个变量的关系如下：

PE=447.06297099−1.97376045∗AT−0.23229086∗V+0.0693515∗AP−0.15806957∗RH

普通的线性回归没有考虑过拟合问题

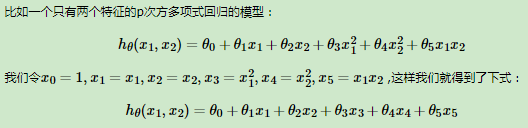
**局部加权线性回归：**

线性回归求得的是具有最小均方误差的无偏估计，有可能出现欠拟合的问题，所以有些方法允许在估计中引入一些偏差，从而降低预测的均方误差。**局部加权线性回归它为待遇测的点其附近的每个点赋予一定的权重。它利用高斯核对附近的点赋予更高的权重。**

**线性回归的推广：**

1. 若未知数的形式不为1次方为2次方以上，则为多项式回归

多项式回归可以通过向高维映射转换为线性回归问题进行处理。



就是说，对于二维的不是线性的数据，我们将其映射到了五维以后，就变成了线性的数据。

第二、广义线性回归：如我们的输出Y不满足和X的线性关系，但是lnY和X满足线性关系，模型函数如下：lnY=Xθ

**线性回归的正则化：**

为了防止模型的过拟合，我们在建立线性模型的时候经常需要加入正则化项。一般有L1正则化和L2正则化。

**线性回归的L1正则化通常称为Lasso回归**，它和一般线性回归的区别是在损失函数上增加了一个L1正则化的项，L1正则化的项有一个常数系数α来调节损失函数的均方差项和正则化项的权重，具体Lasso回归的损失函数表达式如下：

J(θ)=12n(Xθ−Y)T(Xθ−Y)+α||θ||1

　　　　其中n为样本个数，α为常数系数，需要进行调优。||θ||1为L1范数。

 　Lasso回归可以使得一些特征的系数变小，甚至还是一些绝对值较小的系数直接变为0。增强模型的泛化能力。

Lasso回归的求解办法一般有坐标轴下降法（coordinate descent）和最小角回归法（ Least Angle Regression）在**sklearn中Lasso类采用的是坐标轴下降法，后面讲到的LassoLars类采用的是最小角回归法**

Lasso回归的使用场景：一般来说，对于高维的特征数据，尤其线性关系是稀疏的，我们会采用Lasso回归。或者是要在一堆特征里面找出主要的特征，那么Lasso回归更是首选了。但是Lasso类需要自己对α调优，所以不是Lasso回归的首选，一般用到的是下一节要讲的LassoCV类。

sklearn中LassoCV的使用：之所以用它而不是Lasso因为它对正则项的权重系数alpha提供了交叉验证选优的方法。

from sklearn.linear\_model import LassoCV

lassocv = LassoCV(alphas=[0.01, 0.1, 0.5, 1, 3, 5, 7, 10, 20, 100])

lassocv .fit(X\_train, y\_train)

lassocv .alpha\_

**线性回归的L2正则化通常称为Ridge回归**，它和一般线性回归的区别是在损失函数上增加了一个L2正则化的项，和Lasso回归的区别是Ridge回归的正则化项是L2范数，而Lasso回归的正则化项是L1范数。具体Ridge回归的损失函数表达式如下：

　　　　 J(θ)=12(Xθ−Y)T(Xθ−Y)+12α||θ||2

　　　　其中α为常数系数，需要进行调优。||θ||2为L2范数。

　　　Ridge回归在不抛弃任何一个特征的情况下，缩小了回归系数，使得模型相对而言比较的稳定，但和Lasso回归比，这会使得模型的特征留的特别多，模型解释性差。

 　　　 Ridge回归的求解比较简单，一般用最小二乘法。这里给出用最小二乘法的矩阵推导形式，和普通线性回归类似。

　　　令J(θ)的导数为0，得到下式：

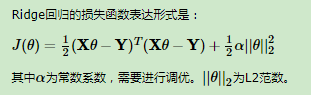
　　　　XT(Xθ−Y)+αθ=0XT(Xθ−Y)+αθ=0

　　　　整理即可得到最后的θ的结果：

　　　　θ=(XTX+αE)−1XTYθ=(XTX+αE)−1XTY

 　　　其中E为单位矩阵。

sklearn中Ridge回归的使用：



通过Ridge回归的损失函数表达式可以看到，α越大，那么正则项惩罚的就越厉害，得到回归系数α就越小，最终趋近与0。而如果α越小，即正则化项越小，那么回归系数α就越来越接近于普通的线性回归系数。

**使用场景：**

一般来说，只要我们觉得数据有线性关系，用LinearRegression类拟合的不是特别好，**需要正则化，可以考虑用RidgeCV类。**不是为了学习的话就不用Ridge类。为什么这里只是考虑用RidgeCV类呢？因为线性回归正则化有很多的变种，Ridge只是其中的一种。所以可能需要比选。**如果输入特征的维度很高，而且是稀疏线性关系的话，RidgeCV类就不合适了。这时应该主要考虑下面几节要讲到的Lasso回归类家族**

from sklearn.linear\_model import Ridge

ridge = Ridge(alpha=1)

ridge.fit(X\_train, y\_train)

训练完了，可以看看模型参数是多少:

print ridge.coef\_

print ridge.intercept\_ #常数项

输出结果如下：

[[-1.97373209 -0.2323016 0.06935852 -0.15806479]]

[ 447.05552892]

上面对模型的训练并没有结束，因为根据公式可知alpha是人为设定的而这里设置为了1，应该对其进行优化。

from sklearn.linear\_model import RidgeCV

ridgecv = RidgeCV(alphas=[0.01, 0.1, 0.5, 1, 3, 5, 7, 10, 20, 100])

ridgecv.fit(X\_train, y\_train)

ridgecv.alpha\_

输出结果为：7.0，说明在我们给定的这组超参数中， 7是最优的α值

22.Logistic回归详解

之所以用梯度上升是由极大似然法推出的

求极大似然函数估计值的一般步骤：

（1） 写出[似然函数](https://baike.baidu.com/item/%E4%BC%BC%E7%84%B6%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E6%9E%81%E5%A4%A7%E4%BC%BC%E7%84%B6%E4%BC%B0%E8%AE%A1/_blank)；

（2） 对似然函数取对数，并整理；

（3） 求[导数](https://baike.baidu.com/item/%E5%AF%BC%E6%95%B0" \t "https://baike.baidu.com/item/%E6%9E%81%E5%A4%A7%E4%BC%BC%E7%84%B6%E4%BC%B0%E8%AE%A1/_blank)；

（4） 解似然方程 。

由此求出的权重w便为最优的。

**对于数据的形式：**

理论上，Logistic回归中的自变量可以是任何形式，定量资料和定性资料均可。但我觉得在数据分析时更倾向于自变量以分类的形式进入模型，因为这样更方便解释。

例如体重，如果直接进行分析，结果提示的是每增加1Kg发生某病的危险。而现实中多数疾病可能对体重增加1Kg不敏感，或者我们医务人员不关心增加1Kg所发生的变化，而关注的是胖子是不是比瘦子有更高的发病风险。So，很多情况下将连续自变量转化为分类变量可能会有更合理的结果解释。

对于数据预处理问题应该先进行异常值检测缺失值处理等，样本数据并不需要进行标准化处理，因为没有距离计算不同大小数据会有不同权重，一些连续数据最好离散化操作。

因为是分类所以就牵扯到类别不均衡的问题，可通过该方法的参数进行调节，设置class\_weight=”balanced”即可

Logistic回归是一种分类算法，既可以处理二分类也可以处理多分类，一般把其用于二分类的分类器。Logistic本质上是一个基于条件概率的判别模型(DiscriminativeModel)。利用了Sigmoid函数值域在[0,1]这个特性。利用了线性回归的原理，只不过线性回归最后求得的是数值型的结果，而Logistic回归最后会利用sigmoid函数把求得的结果转化为0-1之间的值，把大于0.5的判定为1，小于0.5的判定为0。Logistic回归的损失函数用最大似然估计构造，优化应该用梯度递增算法，不断迭代求得的权重系数w便为最优结果。

CTR（广告点击预估）一般用LR

CTR中对LR所用的特征一般都要离散化，为什么？

普遍而言噪声很大的环境中，离散化可以降低特征中包含的噪声，提升特征的表达能力。

在工业界，很少直接将连续值作为逻辑回归模型的特征输入，而是将连续特征离散化为一系列0、1特征交给逻辑回归模型，这样做的优势有以下几点：

0. 离散特征的增加和减少都很容易，易于模型的快速迭代；

1. 稀疏向量内积乘法运算速度快，计算结果方便存储，容易扩展；

2. 离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰；

3. 逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；

4. 离散化后可以进行特征交叉，由M+N个变量变为M\*N个变量，进一步引入非线性，提升表达能力；

5. 特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问；

6. 特征离散化以后，起到了简化了逻辑回归模型的作用，降低了模型过拟合的风险。

具体参考：<https://www.zhihu.com/question/31989952>

****在广告LR模型中，为什么要做特征组合？****

在业界，LR模型之所以很受欢迎，主要是因为LR模型本质是对数线性模型，实现简单，易于并行，大规模扩展方便，迭代速度快，同时使用的特征比较好解释，预测输出在0与1之间契合概率模型。（模型的可解释性举例，比如A-B的权重比较大，A代表用户，B代表物品，那么可以认为A是对B比较感兴趣的）但是，线性模型对于非线性关系缺乏准确刻画，特征组合正好可以加入非线性表达，增强模型的表达能力。另外，广告LR中，基本特征可以认为是用于全局建模，组合特征更加精细，是个性化建模，因为在这种大规模离散LR中，单对全局建模会对部分用户有偏，对每一用户建模又数据不足易过拟合同时带来模型数量爆炸，所以基本特征+组合特征兼顾了全局和个性化。比如特征向量中，有用户A，B，C，物品E,F,G。基本的特征A,B.C.E.F.G对应的权重，对应的是每个对象的偏置权重，但如果A偏好E,B偏好F，那么组合特征A-E,B-F就是对用户的个性进行建模，组合特征A-E,B-F的权重就是代表A对E的喜好，和B-F的喜好。

连续特征离散化

什么是连续属性离散化？连续属性离散化的目的，具体参考文档《连续值的离散化》

离散特征的交叉

为什么要进行特征交叉？因为特征交叉可以让线性模型学习到非线性特征。特征交叉就按笛卡尔积理解好了，假设集合A={a, b}，集合B={0, 1, 2}，则两个集合的笛卡尔积为{(a, 0), (a, 1), (a, 2), (b, 0), (b, 1), (b, 2)}

# 对于特征离散化，特征交叉，连续特征离散化非常经典的解释

参考：

http://lib.csdn.net/article/machinelearning/61015?knId=46

23.决策树算法详解

sklearn中对决策树的构造是基于CART的，决策树不需要对数据预处理，而对于过拟合以及数据不平衡问题的处理方法都在最后决策树的优缺点中介绍了。

决策树是一种贪心算法，树的构建只是使每次划分最优，不一定能达到全局最优。

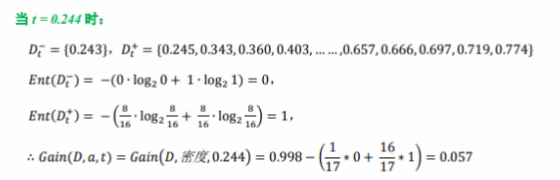
信息熵：熵度量了事物的不确定性，越不确定的事物，它的熵就越大。

信息增益：它度量了在知道当前特征之后类别的不确定性所减少的程度。信息增益越大不确定性减少的程度越大，对类别的确定越有利。

**ID3:**ID3算法就是用信息增益大小来判断当前节点应该用什么特征来构建决策树，用计算出的信息增益最大的特征来建立决策树的当前节点

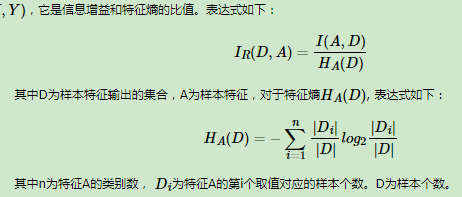
**ID3存在的不足：**第一、没有考虑连续特征，比如长度，密度都是连续值，无法在ID3运用。这大大限制了ID3的用途。第二、ID3采用信息增益大的特征优先建立决策树的节点。但是在相同条件下，取值比较多的特征比取值少的特征信息增益大。（对于信息增益作为评价标准偏向于选择较多的特征，其实是很简单的。从信息增益的定义我们知道，它度量了输出在知道特征以后不确定性减少程度。当特征类别数较多时候，这个不确定性减少程度一般会多一些，因为此时特征类别数较多，每个类别里的样本数量较少，样本更容易被划分的散落到各个特征类别，即样本不确定性变小。）第三、ID3算法对于缺失值的情况没有做考虑。第四、没有考虑过拟合的问题。

**C4.5:**C4.5是在ID3的不足之处上进行改进得来的。**第一、对于连续值的处理。**例如某特征具备m个连续值，先对这m个连续值进行排序，然后取这m个值两两之间的中值作为这一特征的离散取值，这样就把连续型特征转换为离散型特征了。在根据划分出的取值进行条件熵的计算时，小于该值的当做一类，大于该值的当做另一类，同时计算完条件熵之后该属性后面还可以参与子节点的产生选择过程。



对于连续值的处理具体参考：<https://blog.csdn.net/u012328159/article/details/79396893>

1. 引入信息增益比大小来判断当前节点应该用什么特征来构建决策树，用计算出的信息增益比最大的特征来建立决策树的当前节点。如下为信息增益比的计算公式：



具体计算参考：

https://blog.csdn.net/zjsghww/article/details/51638126

由公式可得特征数越多的特征对应的特征熵越大(特征熵是衡量特征本身的不确定性程度)，它作为分母，可以校正信息增益容易偏向于取值较多的特征的问题。

1. 提供了缺失值处理。第四、提供了剪枝操作来解决决策树的过拟合问题。

**C4.5处理连续性数值和离散型数值最大的区别在于，对于连续型数值以两两中值作为离散型取值，在处理完之后它还可以参与子节点的产生选择过程。**

**C4.5算法的缺点**：

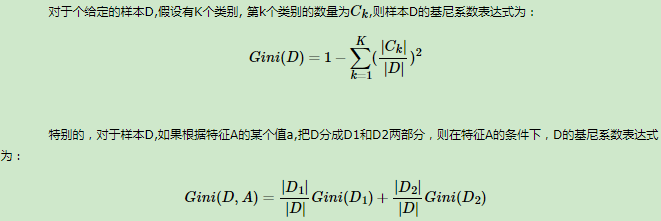
C4.5由于仍然使用了熵模型，里面有大量的耗时的对数运算,如果是连续值还有大量的排序运算。

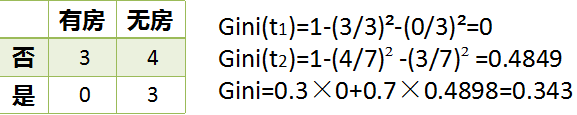
C4.5只能用于分类。且使用了模型较为复杂的多叉树。

**CART：**它既可以用于分类也可用于回归，是sklearn中的决策树所使用的方法。

**CART对于连续值的处理**和C4.5思路一样把连续型数值转换为离散型的，但是在判断当前节点应该用什么特征来构建决策树时它选择的度量方式是基尼系数（基尼系数代表了模型的不纯度）基尼系数越小，则不纯度越低，特征越好。**CART对于离散值得处理与C4.5也有不同之处**。对于CART分类树离散值的处理问题，采用的思路是不停的二分离散特征。回忆下ID3或者C4.5，如果某个特征A被选取建立决策树节点，如果它有A1,A2,A3三种类别，我们会在决策树上一下建立一个三叉的节点。这样导致决策树是多叉树。但是CART分类树使用的方法不同，他采用的是不停的二分，还是这个例子，CART分类树会考虑把A分成{A1}和{A2,A3},{A2}和{A1,A3},{A3}和{A1,A2}三种情况，找到基尼系数最小的组合，比如{A2}和{A1,A3},然后建立二叉树节点，一个节点是A2对应的样本，另一个节点是{A1,A3}对应的节点。同时，由于这次没有把特征A的取值完全分开，后面我们还有机会在子节点继续选择到特征A来划分A1和A3。这和ID3或者C4.5不同，在ID3或者C4.5的一棵子树中，离散特征只会参与一次节点的建立。

Gini系数的计算：





**CART对于离散值与连续值的处理：**

对于CART分类树连续值的处理问题，其思想和C4.5是相同的，都是将连续的特征离散化。唯一的区别在于在选择划分点时的度量方式不同，C4.5使用的是信息增益比，则CART分类树使用的是基尼系数。

对于CART分类树离散值的处理问题，采用的思路是不停的二分离散特征。

回忆下ID3或者C4.5，如果某个特征A被选取建立决策树节点，如果它有A1,A2,A3三种类别，我们会在决策树上一下建立一个三叉的节点。这样导致决策树是多叉树。但是CART分类树使用的方法不同，他采用的是不停的二分，还是这个例子，CART分类树会考虑把A分成{A1}和{A2,A3},{A2}和{A1,A3},{A3}和{A1,A2}三种情况，找到基尼系数最小的组合，比如{A2}和{A1,A3},然后建立二叉树节点，一个节点是A2对应的样本，另一个节点是{A1,A3}对应的节点。同时，由于这次没有把特征A的取值完全分开，后面我们还有机会在子节点继续选择到特征A来划分A1和A3。这和ID3或者C4.5不同，在ID3或者C4.5的一棵子树中，离散特征只会参与一次节点的建立。

**CART解决回归问题：**

首先，我们要明白，什么是回归树，什么是分类树。两者的区别在于样本输出，如果样本输出是离散值，那么这是一颗分类树。如果果样本输出是连续值，那么那么这是一颗回归树。

除了概念的不同，CART回归树和CART分类树的建立和预测的区别主要有下面两点：

1)连续值的处理方法不同

2)决策树建立后做预测的方式不同。

　　对于连续值的处理，我们知道CART分类树采用的是用基尼系数的大小来度量特征的各个划分点的优劣情况。这比较适合分类模型，但是对于回归模型，我们使用了常见的和方差的度量方式，CART回归树的度量目标是，对于任意划分特征A，对应的任意划分点s，把特征划分成的数据集D1和D2，求出使D1和D2各自集合的均方差最小，同时D1和D2的均方差之和最小所对应的特征和特征值划分点。

对于决策树建立后做预测的方式，上面讲到了CART分类树采用叶子节点里概率最大的类别作为当前节点的预测类别。而回归树输出不是类别，它采用的是用最终叶子的均值或者中位数来预测输出结果。

　　除了上面提到了以外，CART回归树和CART分类树的建立算法和预测没有什么区别。

**CART树剪枝策略：**

CART回归树和CART分类树的剪枝策略除了在度量损失的时候一个使用均方差，一个使用基尼系数，算法基本完全一样，这里我们一起来讲。

　　由于决策时算法很容易对训练集过拟合，而导致泛化能力差，为了解决这个问题，我们需要对CART树进行剪枝，即类似于线性回归的正则化，来增加决策树的泛化能力。但是，有很多的剪枝方法，我们应该这么选择呢？CART采用的办法是后剪枝法，即先生成决策树，然后产生所有可能的剪枝后的CART树，然后使用交叉验证来检验各种剪枝的效果，选择泛化能力最好的剪枝策略。

　　也就是说，CART树的剪枝算法可以概括为两步，第一步是从原始决策树生成各种剪枝效果的决策树，第二部是用交叉验证来检验剪枝后的预测能力，选择泛化预测能力最好的剪枝后的数作为最终的CART树。

**CART的缺点：**

1）大家应该有注意到，无论是ID3, C4.5还是CART,在做特征选择的时候都是选择最优的一个特征来做分类决策，但是大多数，分类决策不应该是由某一个特征决定的，而是应该由一组特征决定的。这样得到的决策树更加准确。这个决策树叫做多变量决策树(multi-variate decision tree)。**在选择最优特征的时候，多变量决策树不是选择某一个最优特征，而是选择最优的一个特征线性组合来做决策。这个算法的代表是OC1**，这里不多介绍。

　　2）如果样本发生一点点的改动（**这里指的是数据分布的波动，而决策树有很好的容错性是指特征存在异常值并不会给最终的预测结果造成很大的影响。**），就会导致树结构的剧烈改变。这个可以通过集成学习里面的随机森林之类的方法解决。

**三种算法的比较：**



**出了以上三种方法能用于构造决策树以外还有C5.0和OC1**

**决策树的优缺点分析：**

首先我们看看决策树算法的优点：

　　1）简单直观，生成的决策树很直观。

　　2）基本不需要预处理，不需要提前归一化，处理缺失值。（只有在牵扯到距离计算时才会归一化，而决策树中没有距离的计算，且自有处理缺失值的方法）

　　3）使用决策树预测的代价是O(log2m)。 m为样本数。

　　4）既可以处理离散值也可以处理连续值。很多算法只是专注于离散值或者连续值。

　　5）可以处理多维度输出的分类问题。

　　6）相比于神经网络之类的黑盒分类模型，决策树在逻辑上可以得到很好的解释

　　7）可以交叉验证的剪枝来选择模型，从而提高泛化能力。

　　8）对于异常点的容错能力好，健壮性高。**是指特征存在异常值并不会给最终的预测结果造成很大的影响**

我们再看看决策树算法的缺点:

　　1）**决策树算法非常容易过拟合，导致泛化能力不强。可以通过设置节点最少样本数量和限制决策树深度来改进**。

　　2）决策树会因为样本发生一点点的改动，就会导致树结构的剧烈改变。这个可以通过集成学习之类的方法解决。

　　3）寻找最优的决策树是一个NP难的问题，我们一般是通过启发式方法，容易陷入局部最优。可以通过集成学习之类的方法来改善。

　　4）有些比较复杂的关系，决策树很难学习，比如异或。这个就没有办法了，一般这种关系可以换神经网络分类方法来解决。

　　5）**如果某些特征的样本比例过大，生成决策树容易偏向于这些特征。这个可以通过调节样本权重来改善。**

具体的使用参照：<http://www.cnblogs.com/pinard/p/6056319.html>

**树的集成学习详解**

集成学习可以用于分类问题集成，回归问题集成，特征选取集成，异常点检测集成等等。集成学习是以若干个弱的个体学习器为基础集成一个强学习器。因此这就牵扯到了如何得到若干个个体学习器，一般有两种方法:第一、所有的个体学习器都是一个种类的，或者说是同质的。比如都是决策树个体学习器，或者都是神经网络个体学习器。第二种是所有的个体学习器不全是一个种类的，或者说是异质的。比如我们有一个分类问题，对训练集采用支持向量机个体学习器，逻辑回归个体学习器和朴素贝叶斯个体学习器来学习，再通过某种结合策略来确定最终的分类强学习器。

目前来说，**同质个体学习器的应用是最广泛的，一般我们常说的集成学习的方法都是指的同质个体学习器。**而同质个体学习器使用最多的模型是CART决策树和神经网络。同质个体学习器按照个体学习器之间是否存在依赖关系可以分为两类，第一个是个体学习器之间存在强依赖关系，一系列个体学习器基本都需要串行生成，代表算法是boosting系列算法，第二个是个体学习器之间不存在强依赖关系，一系列个体学习器可以并行生成，代表算法是bagging和随机森林（Random Forest）系列算法。

**集成学习的结合策略：**

1. 平均法

对于数值类的回归预测问题，通常使用的结合策略是平均法，也就是说，对于若干个弱学习器的输出进行平均得到最终的预测输出。

最简单的平均是算术平均，也就是说最终预测是：



如果每个个体学习器有一个权重w，则最终预测是：



其中wi是个体学习器hi的权重，通常有：



1. 投票法

对于分类问题的预测，我们通常使用的是投票法。最简单的投票法是相对多数投票法，也就是我们常说的少数服从多数，如果不止一个类别获得最高票，则随机选择一个做最终类别。

稍微复杂的投票法是绝对多数投票法，也就是我们常说的要票过半数。在相对多数投票法的基础上，不光要求获得最高票，还要求票过半数。否则会拒绝预测。

　　更加复杂的是加权投票法，和加权平均法一样，每个弱学习器的分类票数要乘以一个权重，最终将各个类别的加权票数求和，最大的值对应的类别为最终类别。

1. 学习法

上两节的方法都是对弱学习器的结果做平均或者投票，相对比较简单，但是可能学习误差较大，于是就有了学习法这种方法，对于学习法，代表方法是stacking，当使用stacking的结合策略时， 我们不是对弱学习器的结果做简单的逻辑处理，而是再加上一层学习器，也就是说，我们将训练集弱学习器的学习结果作为输入，将训练集的输出作为输出，重新训练一个学习器来得到最终结果。

　　在这种情况下，我们将弱学习器称为初级学习器，将用于结合的学习器称为次级学习器。对于测试集，我们首先用初级学习器预测一次，得到次级学习器的输入样本，再用次级学习器预测一次，得到最终的预测结果。

**Boosting及其常用算法：**

注意：对于Adaboost和GBDT这两个基于Boosting的树集成算法是不可以并行的，因为后面的每一棵树的构建都依赖于前一棵树，但是在做预测时，因为模型都是建立好的，所以可以并行。

Boosting算法的工作机制是首先从训练集用初始权重训练出一个弱学习器1，根据弱学习的学习误差率表现来更新训练样本的权重，使得之前弱学习器1学习误差率高的训练样本点的权重变高，使得这些误差率高的点在后面的弱学习器2中得到更多的重视。然后基于调整权重后的训练集来训练弱学习器2.，如此重复进行，直到弱学习器数达到事先指定的数目T，最终将这T个弱学习器通过集合策略进行整合，得到最终的强学习器。

Boosting中的分类器的权重不相等，每个权重代表的是其对应分类器在上一轮迭代中的成功度。

对于基于boosting的算法都需要解决四个问题：

1. 如何计算学习误差率e?
2. 如何得到弱学习器权重系数α?
3. 如何更新样本权重D?
4. 使用何种结合策略？
5. Adaboost（自适应boosting）

Adaboost既可以用作分类，也可以用作回归。Adaboost的基学习器可以随便选择，但是一般用CART决策树与CART回归树的，所以它具备了决策树的优势，对异常值不敏感，不用对数据进行预处理。对于类别不均衡问题sklearn在决策树中给了处理方法，对于过拟合问题也给了相应参数进行处理。

Adaboost是模型为加法模型，学习算法为前向分步学习算法，损失函数为指数函数的分类问题。

**Adaboost的分类问题：**

Adaboost的二元分类问题，对于其运行过程如下：

训练集中的每个样本赋予一个权重，这些权重构成了向量D。一开始这些权重都初始化成相等的值（1/样本的总个数）首先在训练数据上训练出一个弱分类器并计算该分类器的错误率，然后在同一数据集上再次训练弱分类器。在分类器的第二次训练中，将会重新调整每个样本的权重，其中那些第一次分对的样本权重会降低，而第一次分错的样本权重将会提高。为了从所有弱分类器中得到最终的分类结果，AdaBoost为每个弱分类器都分配了一个权重值alpha，这些alpha值是基于每个弱分类器的错误率计算的。

错误率=（未正确分类的样本数目）/所有样本数目

对于二分类而言Alpha的计算公式如下：

为什么这样计算每个弱分类器的权重系数，简单的来说从公式可以看出，对于错误率越大的分类器而言其权重系数将会越小。计算完弱分类器的权重系数之后应该对样本的权重向量D进行更新

具体计算公式见P118.Adaboost分类采用的是加权平均法作为集合策略。再结合sign()进行分类

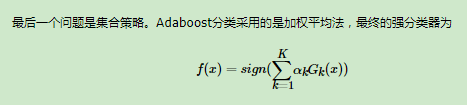
Adaboost的多元分类问题：

对于Adaboost多元分类算法，其实原理和二元分类类似，最主要区别在弱分类器的系数上。比如Adaboost SAMME算法，它的弱分类器的系数



其中R为类别数。从上式可以看出，如果是二元分类，R=2，则上式和我们的二元分类算法中的弱分类器的系数一致。

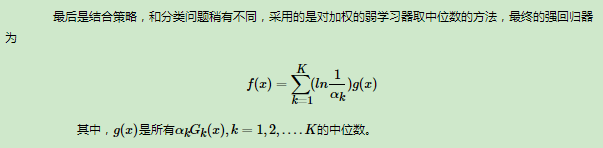
对于分类采用的集合策略，



把多个若分类器的分类结果进行加权求和然后利用符号函数进行分类。比如第一个弱分类器分为1，其权重为0.3，第二个分为-1，其权重为0.2，进行相加。

**Adaboost的回归问题：**

具体解析见：<http://www.cnblogs.com/pinard/p/6133937.html>



**Adaboost的优缺点分析：**

Adaboost的主要优点有：

第一、Adaboost作为分类器时，分类精度很高。第二、在Adaboost的框架下，可以使用各种回归分类模型来构建弱学习器，非常灵活。第三、作为简单的二元分类器时，构造简单，结果可理解。第四、不容易发生过拟合。

Adaboost的主要缺点有：

对异常样本敏感，异常样本在迭代中可能会获得较高的权重，影响最终的强学习器的预测准确性。

在sklearn中使用见：http://www.cnblogs.com/pinard/p/6136914.html

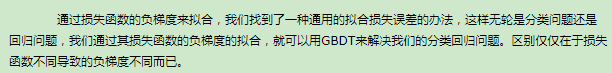
1. GBDT(梯度提升决策树)

它是基于CART的，所以分类回归用CART的。

不断的进行迭代，使得得到的每个弱学习器能够使损失函数最小。对于损失误差的衡量用负梯度法。

GBDT也是迭代，使用了前向分布算法，但是弱学习器限定了只能使用CART回归树模型，同时迭代思路和Adaboost也有所不同。

在GBDT的迭代中，假设我们前一轮迭代得到的强学习器是ft−1(x), 损失函数是L(y,ft−1(x)), 我们本轮迭代的目标是找到一个CART回归树模型的弱学习器ht(x)，让本轮的损失L(y,ft(x)=L(y,ft−1(x)+ht(x))最小。也就是说，本轮迭代找到决策树，要让样本的损失尽量变得更小。



GBDT的思想可以用一个通俗的例子解释，假如有个人30岁，我们首先用20岁去拟合，发现损失有10岁，这时我们用6岁去拟合剩下的损失，发现差距还有4岁，第三轮我们用3岁拟合剩下的差距，差距就只有一岁了。如果我们的迭代轮数还没有完，可以继续迭代下面，每一轮迭代，拟合的岁数误差都会减小。

在sklearn中的具体应用：

<http://www.cnblogs.com/pinard/p/6143927.html>

也是基于CART决策树的集成学习算法，因此不必考虑数据处理问题。样本不均衡问题可用CART弱学习器中的参数控制。只需考虑过拟合问题，除了在弱分类器中可控制外，还可通过learning\_rate:以及subsample这两个参数进行调节。除此之外还需注意的是二分类与多分类问题，只需将loss参数的值设为默认的"deviance"。它对二元分类和多元分类各自都有比较好的优化。

Bagging及其常用算法：

Bagging的算法原理和 boosting不同，它的弱学习器之间没有依赖关系，可以并行生成。bagging的个体弱学习器的训练集是通过随机采样得到的。通过T次的随机采样，我们就可以得到T个采样集，对于这T个采样集，我们可以分别独立的训练出T个弱学习器，再对这T个弱学习器通过集合策略来得到最终的强学习器。

　　对于这里的随机采样有必要做进一步的介绍，这里一般采用的是自助采样法（Bootstap sampling）,即对于m个样本的原始训练集，我们每次先随机采集一个样本放入采样集，接着把该样本放回，也就是说下次采样时该样本仍有可能被采集到，这样采集m次，最终可以得到m个样本的采样集，由于是随机采样，这样每次的采样集是和原始训练集不同的，和**其他采样集也是不同的**，这样得到多个不同的弱学习器。

**随机森林**

首先，RF使用了CART决策树作为弱学习器，第二，在使用决策树的基础上，RF对决策树的建立做了改进，对于普通的决策树，我们会在节点上所有的n个样本特征中选择一个最优的特征来做决策树的左右子树划分，但是RF通过随机选择节点上的一部分样本特征，这个数字小于n，假设为nsub，然后在这些随机选择的nsub个样本特征中，选择一个最优的特征来做决策树的左右子树划分。这样进一步增强了模型的泛化能力。（即为随机森林选择CART决策树作为弱分类器，但是在建树时不是用全部的样本特征，而是选择一部分样本特征再从这些选择出的样本特征中选取一个最优特征。）

随机森林的推广：

1. extra trees是RF的一个变种, 原理几乎和RF一模一样。但是它是利用样本的全部特征，同时对于划分特征的选择不是选择最优的，而是随机选择一个样本特征进行划分。
2. Totally Random Trees Embedding(以下简称 TRTE)是一种非监督学习的数据转化方法。它将低维的数据集映射到高维，从而让映射到高维的数据更好的运用于分类回归模型。
3. Isolation Forest（孤立森林）（以下简称IForest）是一种异常点检测的方法。它也使用了类似于RF的方法来检测异常点。详细分析下面条目有。

**kmeans聚类算法**

传统的kmeans思路分析：

根据经验确定应该把数据聚类的类的个数N,随机的从数据集中选取N个点作为初始质心，遍历数据集中的每个点，把各个点归为与其距离最近的质心那一类（这里样本的各个特征应该同等重要，所以应该把数据标准化）。把所有数据聚为N类之后，选取每一类各个样本特征的平均值作为新的质心。**注意这里的质心的选择是随机的，会导致聚类的效果不太好且运行时间也会过长。**

**k-medoids算法**

它和kmeans聚类算法的区别在于最后质心的更新上，聚类好之后质心不是像kmeans那样更新为各个点的均值，而是遍历该类别中的每一个点计算离其它点的距离之和，选取值最小的那个作为质心。

**二分kmeans聚类算法**

Kmeans算法中质心的选择至关重要，但是算法中质心却是随机选择的，这难免会使得模型的准确度下降。为了弱化初始质心对模型的影响，出现了二分kmeans聚类算法。

**思路：**首先将所有点作为一个簇，然后将该簇一分为二。之后选择能最大程度降低聚类代价函数（也就是误差平方和）的簇划分为两个簇。以此进行下去，直到簇的数目等于用户给定的数目k为止

**伪代码如下：**

将所有数据点看成一个簇   
当簇数目小于k时   
对每一个簇   
计算总误差   
在给定的簇上面进行k-均值聚类（k=2）   
计算将该簇一分为二后的总误差   
选择使得误差最小的那个簇进行划分操作

**对于选择质心方法的优化kmeans++思路解析：**

总体思路与kmeans相同只不过对于质心的选择方法进行了优化。对于质心不再是全部都随机进行选择。确定思路如下：第一、先从样本集中随机的选取一个样本点作为质心。第二、对于数据集中的每一个数据，计算它与已选择的所有质心离其最近的质心之间的距离组成集合D。第三、从D中选择最大的值作为新增的质心。第四、重复第二第三步骤N-1次直到选择出N个所有质心。

**对于距离计算进行优化的elkan K-Means思路解析：**

在传统的K-Means算法中，我们在每轮迭代时，要计算所有的样本点到所有的质心的距离，这样会比较的耗时。那么，对于距离的计算有没有能够简化的地方呢？elkan K-Means算法就是从这块入手加以改进。它的目标是减少不必要的距离的计算。那么哪些距离不需要计算呢？

　　　　elkan K-Means利用了两边之和大于等于第三边,以及两边之差小于第三边的三角形性质，来减少距离的计算。

　　第一种规律是对于一个样本点x和两个质心μj1,μj2。如果我们预先计算出了这两个质心之间的距离)D(j1,j2)，则如果计算发现2D(x,j1)≤D(j1,j2),我们立即就可以知道D(x,j1)≤D(x,j2)。此时我们不需要再计算D(x,j2),也就是说省了一步距离计算。

　　第二种规律是对于一个样本点x和两个质心μj1,μj2。我们可以得到D(x,j2)≥max{0,D(x,j1)−D(j1,j2)}。这个从三角形的性质也很容易得到。

　　利用上边的两个规律，elkan K-Means比起传统的K-Means迭代速度有很大的提高。但是如果我们的样本的特征是稀疏的，有缺失值的话，这个方法就不使用了，此时某些距离无法计算，则不能使用该算法。

**大样本优化Mini Batch K-Means以及kmeans的优缺点具体见：**

<http://www.cnblogs.com/pinard/p/6164214.html>

**GMM（高斯混合模型聚类算法）**

顾名思义，Gaussian Mixture Model ，就是假设数据服从 Mixture Gaussian Distribution ，换句话说，数据可以看作是从数个 Gaussian Distribution 中生成出来的。

每个 GMM 由 K 个 Gaussian 分布组成，每个 Gaussian 称为一个“Component”，这些 Component 线性加成在一起就组成了 GMM 的概率密度函数：

我们使用k个多元高斯分布的混合高斯分布GMM来对数据进行聚类，其中每一个分布代表一个数据簇。首先，随机选择k个对象代表各个簇的均值（中心），猜测每一个簇的协方差矩阵，并假定初始状态 时每个簇的概率相等； 然后，根据多元高斯密度函数求出每一个对象属于每一个簇的概率，并求出数据的似然函数值；最后，根据每一个数据点属于每一个簇的概率，来更新每一个簇的均值，协方差矩阵，和每一个簇的概率。不断迭代以上两步，直到算法收敛。 这时我们根据每一个对象属于每一个簇的概率，将对象指派的概率最高的簇中。

关键部分就是EM算法部分。

算法中只知道每个向量的坐标，要将这些向量聚类为k个gauss分布中，但这k个cluster的高斯分布的参数未知，当然另一个未知量是各个向量所归属的类别。

首先初始化各个gauss分布的参数

E-step：

           根据这k个gauss分布的参数求得各个向量归属各个cluster的概率矩阵

M-step：

           根据E-step概率矩阵更新gauss分布参数

以上两步交替进行，直到收敛。

GMM与kmeans的区别和联系

参考：https://www.cnblogs.com/zjutzz/p/5083623.html

**BIRCH聚类算法**

BIRCH算法比较适合于数据量大，类别数K也比较多的情况。它运行速度很快，只需要单遍扫描数据集就能进行聚类，具体在sklearn中的使用见：<http://www.cnblogs.com/pinard/p/6200579.html>

推荐首选的谱聚类聚类算法

它的主要思想是把所有的数据看做空间中的点，这些点之间可以用边连接起来。距离较远的两个点之间的边权重值较低，而距离较近的两个点之间的边权重值较高，通过对所有数据点组成的图进行切图，让切图后不同的子图间边权重和尽可能的低，而子图内的边权重和尽可能的高，从而达到聚类的目的。

在sklearn中的应用见：<http://www.cnblogs.com/pinard/p/6235920.html>

支持向量机（SVM）的简单分析应用

**SVM数学知识具体参考：**

**https://blog.csdn.net/zhangping1987/article/details/21931663**

**数学知识补充**

对于线性可分的超平面

既然能线性可分，那么就有超平面IMG_256(向量化表示IMG_257)将这数据集分开，使得一侧是“+1”类，另一侧是“-1类”。

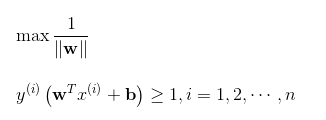
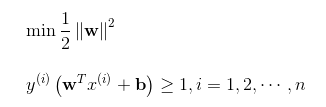
****第一个知识点：已知超平面IMG_256和数据集IMG_257，哪个一点离这条超平面最近，答案：哪一个点使得IMG_258最小，哪一点离这个超平面最近****

第二个知识点：****已知超平面IMG_259和点IMG_260，IMG_261，那么这个点到这个超平面的距离为：IMG_262,向量表示法为IMG_263****

**对于SVM的描述**

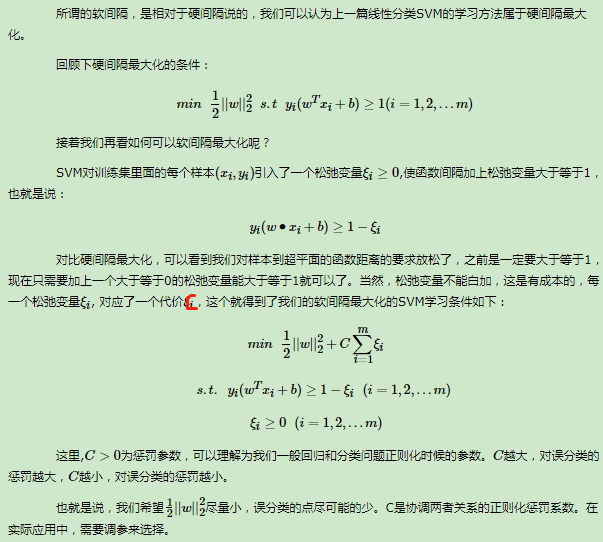
SVM分为线性可分支持向量机，线性向量机和非线性支持向量机。当训练数据可分时，通过硬间隔最大化，学习一个线性的分类器，即线性可分支持向量机，又称为硬间隔支持向量机，当训练数据近似线性可分时，通过软间隔最大化，也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向量机，当训练数据线性不可分时，通过使用核技巧及软间隔最大化，学习非线性支持向量机。

当输入空间为欧式空间或离散集合、特征空间为希尔伯特空间时，核函数表示将输入从输入空间映射到特征空间得到的特征向量之间的内积，通过使用核函数可以学习非线性支持向量机，等价于隐式地在高维的特征空间中学习线性支持向量机，这样的方法称为核技巧。

**线性可分支持向量机**又称为硬间隔支持向量机（一般处理的是二分类问题，）它处理的是数据集线性可分的情况，既然线性可分则必然存在超平面能够把数据分为正负两类，但是这种超平面会有很多，这就需要考虑选取出能使得训练出来的模型泛化能力最强的超平面。也就是说对于误分类点的容错性更好。这时又需要考虑哪些点最容易被误分类，显然离超平面越近的点越容易被误分类，如果离超平面最近的点都能被正确分类那么远的点就必然会被正确分类。因此所选的超平面应该尽量使得数据集离超平面的最小距离最大化，这样模型才拥有最好的泛化能力。离超平面最小的距离为IMG_256故则有，最终表示为，这是硬间隔支持向量机用以代表线性可分的支持向量机模型。其中y为类别标签。离超平面最近的正负类中的这些点被称为支持向量。分离超平面为wTx+b=0，所有的样本不光可以被超平面分开，还和超平面保持一定的函数距离（下图函数距离为1），那么这样的分类超平面是比感知机的分类超平面优的。可以证明，这样的超平面只有一个。和超平面平行的保持一定的函数距离的这两个超平面对应的向量，我们定义为支持向量，支持向量所在的超平面定义为wTx+b=-1和wTx+b=1.注意这里为什么是+-1？实际上，支持向量只是到中间那个平面距离最近的点。wx+b=y，其中y只用比非支持向量的y要小就行了。而题主说为什么y是1，因为如果把超平面wx+b=y这个式子的每一项除以y，那么虽然值都变了，但平面还是这个平面，也就是说有一系列的w和b表示这个平面，那我们需要固定某一个值。在svm的求解过程中，需固定y=1，然后转化成二次规划去求解。就是说，直接固定支持向量到分类超平面的距离为1。因为如果不是+-1的话，假设为某个数r,左右两边同除以r,右边就为+-1了,然后左边的结果用w和b替代。所以到底是正负几并不重要。

**线性支持向量机，又称为软间隔支持向量机**

对于数据本是线性可分的但是存在异常点的情况，使得硬间隔支持向量机不能很好的确定划分数据的超平面，这时应该用软间隔支持向量机。



**非线性支持向量机**

1. 核函数与常用的核函数

我们遇到线性不可分的样例时，常用做法是把样例特征映射到高维空间中去(如上一节的多项式回归）但是遇到线性不可分的样例，一律映射到高维空间，那么这个维度大小是会高到令人恐怖的。此时，核函数就体现出它的价值了，核函数的价值在于它虽然也是将特征进行从低维到高维的转换，但核函数好在它在低维上进行计算，而将实质上的分类效果（利用了内积）表现在了高维上，这样避免了直接在高维空间中的复杂计算，真正解决了SVM线性不可分的问题。

1. 常用的核函数（也是sklearn中可选的核函数）

第一、线性核函数。第二、多项式核函数。第三、高斯核函数在SVM中也称为径向基核函数（Radial Basis Function,RBF），它是非线性分类SVM最主流的核函数。libsvm默认的核函数就是它。第四、Sigmoid核函数

24.对于数据中的离散型特征的处理方法有哪些？

由于sklearn在训练模型时只接收数值型的数据，因此应该把字符型型数据转换为数字型，一般采用独热编码或标记编码

具体参照：<https://blog.csdn.net/gao1440156051/article/details/55096630>

https://blog.csdn.net/u012328159/article/details/71617381

以用pandas的DataFrame（）数据为前提。

testdata = pd.DataFrame({**'pet'**: [**'cat'**, **'dog'**, **'dog'**, **'fish'**],**'age'**: [4 , 6, 3, 3],**'salary'**:[4, 5, 1, 1]})

1. 用sklearn中的labelEncoder()进行标记编码

注意对于传入的进行标记编码的值不能存在空值。

**from** sklearn.preprocessing **import** LabelEncoder

testdata.**pet**=LabelEncoder().fit\_transform(testdata.**pet**)

1. 对于字符串的数据最后转化为二值型数据

**from** sklearn.feature\_extraction **import** DictVectorizer

testdata=DictVectorizer(sparse=**False**).fit\_transform(testdata.to\_dict(orient=**'record'**))

一次性将字符型数据转换为二值型编码

1. 直接用pandas的get\_dummies(),可用且方便

testdata=pd.get\_dummies(testdata)

注意他会对所有字符型的数据进行one\_hot编码

对连续数据进行离散化处理

为什么要这样参考连续特征离散化

Titanic中的例子：

*# Mapping Fare*

dataset.loc[ dataset['Fare'] <= 7.91, 'Fare']= 0

dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1

dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare'] = 2

dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3

dataset['Fare'] = dataset['Fare'].astype(int)

*# Mapping Age*

dataset.loc[ dataset['Age'] <= 16, 'Age'] = 0

dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1

dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2

dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3

dataset.loc[ dataset['Age'] > 64, 'Age'] = 4 ;

25.何为正态分布（又称高斯分布）？为什么会在机器学习中得到广泛的应用？

正态分布（Normal distribution），也称“常态分布”，又名[高斯分布](https://baike.baidu.com/item/%E9%AB%98%E6%96%AF%E5%88%86%E5%B8%83" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)，是一个在[数学](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6/107037" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)、物理及工程等领域都非常重要的[概率](https://baike.baidu.com/item/%E6%A6%82%E7%8E%87" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)分布，正态曲线呈钟型，两头低，中间高，左右对称因其曲线呈钟形，因此人们又经常称之为[钟形曲线](https://baike.baidu.com/item/%E9%92%9F%E5%BD%A2%E6%9B%B2%E7%BA%BF" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)。

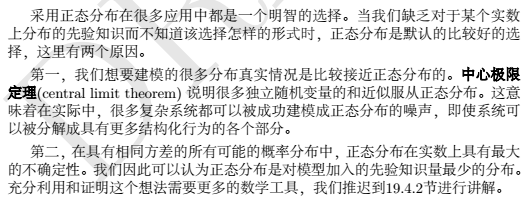
若[随机变量](https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E5%8F%98%E9%87%8F" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)X服从一个[数学期望](https://baike.baidu.com/item/%E6%95%B0%E5%AD%A6%E6%9C%9F%E6%9C%9B" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)为μ、[方差](https://baike.baidu.com/item/%E6%96%B9%E5%B7%AE" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)为σ^2的正态分布，记为N(μ，σ^2)。其[概率密度函数](https://baike.baidu.com/item/%E6%A6%82%E7%8E%87%E5%AF%86%E5%BA%A6%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)为正态分布的[期望值](https://baike.baidu.com/item/%E6%9C%9F%E6%9C%9B%E5%80%BC" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)μ决定了其位置，其[标准差](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E5%B7%AE" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)σ决定了分布的幅度。当μ = 0,σ = 1时的正态分布是[标准正态分布](https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83" \t "https://baike.baidu.com/item/%E6%AD%A3%E6%80%81%E5%88%86%E5%B8%83/_blank)。

那么为什么正态分布会在机器学习中得到广泛的应用？

机器学习基于统计学用的较多的是统计机器学习，统计学习的基础是统计，统计的基础之一就是中心极限定理，中心极限定理的结果就是高斯分布。

中心极限定理。说白了就是一大堆乱七八糟的（有界）随机变量加起来就像是高斯分布，这使得实际遇到的很多分布（如噪声等）近似服从高斯分布。

正态分布在现实中很多地方都能得到利用。



典型关联分析(Canonical Correlation Analysis，CCA)

它是挖掘数据关联关系的常用算法。比如我们拿到两组数据，第一组是人身高和体重的数据，第二组是对应的跑步能力和跳远能力的数据。那么我们能不能说这两组数据是相关的呢？CCA可以帮助我们分析这个问题。

先引入一维数据X,Y的相关系数的计算公式：



其中cov(X,Y)是X和Y的协方差，而D(X),D(Y)分别是X和Y的方差。相关系数ρ的取值为[-1,1],ρ的绝对值越接近于1，则X和Y的线性相关性越高。越接近于0，则X和Y的线性相关性越低。

**注意：**以上公式只适用于一维数据相关系数的计算，对于多维不适用应该进行变通。

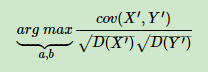
CCA使用的方法是将多维的X和Y都用线性变换为1维的X'和Y'，然后再使用相关系数来看X'和Y'的相关性。将数据从多维变到1位，也可以理解为CCA是在进行降维，将高维数据降到1维，然后再用相关系数进行相关性的分析。

CCA的算法思想：

CCA是将高维的两组数据分别降维到1维，然后用相关系数分析相关性。但是降维的标准该如何选择？回想下主成分分析PCA，降维的原则是投影方差最大；再回想下线性判别分析LDA，降维的原则是同类的投影方差小，异类间的投影方差大。CCA它选择的投影标准是降维到1维后，两组数据的相关系数最大。

假设我们的数据集是X和Y，X为n1×m的样本矩阵。Y为n2×m的样本矩阵.其中m为样本个数，而n1,n2分别为X和Y的特征维度。对于X矩阵，我们将其投影到1维，或者说进行线性表示，对应的投影向量或者说线性系数向量为a,对于Y矩阵，我们将其投影到1维，或者说进行线性表示，对应的投影向量或者说线性系数向量b, 这样X ,Y投影后得到的一维向量分别为X',Y'。我们有

进而可得CCA的优化目标是最大化ρ(X′,Y′)得到对应的投影向量a,b，即



在投影前，我们一般会把原始数据进行标准化，得到均值为0而方差为1的数据X和Y。

CCA算法小结：

CCA算法广泛的应用于数据相关度的分析，同时还是偏最小二乘法的基础。但是由于它依赖于数据的线性表示，当我们的数据无法线性表示时，CCA就无法使用，此时我们可以利用核函数的思想，将数据映射到高维后，再利用CCA的思想降维到1维，求对应的相关系数和线性关系，这个算法一般称为KCCA。

异常点检测常用的iForest(独立森林)算法

样本数据过大时推荐采用这种异常值检测方法

**原理分析：**

iForest森林也由大量的树组成。iForest中的树叫isolation tree，简称iTree。iTree树和决策树不太一样，其构建过程也比决策树简单，因为其中就是一个完全随机的过程。具体实施过程如下：  
 假设数据集有N条数据，构建一颗iTree时，从N条数据中均匀抽样(一般是无放回抽样)出ψ个样本出来，作为这颗树的训练样本。在样本中，随机选一个特征，并在这个特征的所有值范围内(最小值与最大值之间)随机选一个值，对样本进行二叉划分，将样本中小于该值的划分到节点的左边，大于等于该值的划分到节点的右边。这样得到了一个分裂条件和左、右两边的数据集，然后分别在左右两边的数据集上重复上面的过程，直接达到终止条件。终止条件有两个，一个是数据本身不可再分(只包括一个样本，或者全部样本相同)，另外一个是树的高度达到log2(ψ)。不同于决策树，iTree在算法里面已经限制了树的高度。当然不限制也可以，只是算法为了效率考虑，只需要达到log2(ψ)深度即可。  
把所有的iTree树构建好了，就可以对测数据进行预测了。预测的过程就是把测试数据在iTree树上沿对应的条件分支往下走，直到达到叶子节点，并记录这过程中经过的路径长度h(x)，即从根节点，穿过中间的节点，最后到达叶子节点，所走过的边的数量(path length)。  
最后，将h(x)带入，计算每条待测数据的异常分数(Anomaly Score)如果分数越接近1，其是异常点的可能性越高；如果分数都比0.5要小，那么基本可以确定为正常数据；如果所有分数都在0.5附近，那么数据不包含明显的异常样本。

**在sklearn中的使用：**

算法基本上不需要配置参数就可以直接使用，通常就以下几个(参数明显比随机森林简单)：  
n\_estimators: 默认为100，配置iTree树的多少

max\_samples: 默认为265，配置采样大小

max\_features: 默认为全部特征，对高维数据，可以只选取部分特征

**from** sklearn.ensemble **import** IsolationForest

ilf=IsolationForest()  
ilf.fit(X)  
s=ilf.predict(X)

返回的是只包含-1和1元素的数组，-1代表可能为异常值的点

异常点检测常用算法OneClassSVM

在样本数据较少时建议采用这种异常值检测方法

对于那些不同类别数据极度不平衡的数据可以把样本训练成一类，即为OneClassSVM，同时它还可以用于异常点检测。

在sklearn中的用法

**from** sklearn.svm **import** OneClassSVM

classer=OneClassSVM()  
classer.fit(X)  
yHat=classer.predict(X)  
print(yHat)

注意：由于其是非监督学习方法，所以不用传入类别列表。返回数据和iForest一样是只包含1,-1的列表，-1代表可能是异常点，X是训练样本。检测完后的数据筛选

list=[i **for** (i,item) **in** enumerate(yHat) **if** item==1]

X=X[list]

皮尔逊相关系数与互信息法以及组合特征

计算每一个特征与响应变量的相关性：工程上常用的手段有计算皮尔逊系数和互信息系数，皮尔逊系数只能衡量线性相关性而互信息系数能够很好地度量各种相关性，但是计算相对复杂一些，好在很多toolkit里边都包含了这个工具（如sklearn的MINE），得到相关性之后就可以排序选择特征了

通过特征组合后再来选择特征：如对用户id和用户特征最组合来获得较大的特征集再来选择特征，这种做法在推荐系统和广告系统中比较常见，这也是所谓亿级甚至十亿级特征的主要来源，原因是用户数据比较稀疏，组合特征能够同时兼顾全局模型和个性化模型

互信息法选择特征样例：

from sklearn.feature\_selection import SelectKBest

from minepy import MINE #由于MINE的设计不是函数式的，定义mic方法将其为函数式的，返回一个二元组，二元组的第2项设置成固定的P值0.5

def mic(x, y):

m = MINE()

m.compute\_score(x, y)

return (m.mic(), 0.5)

#选择K个最好的特征，返回特征选择后的数据

SelectKBest(lambda X, Y: array(map(lambda x:mic(x, Y), X.T)).T, k=2).fit\_transform(iris.data, iris.target)

sklearn.pipeline.FeatureUnion和Sklearn.pipeline.Pipeline

FeatureUnion是连接两个转换之后的结果

Pipeline是使用最终模型进行变换的流水线，按顺序应用变换列表和最终估计器。 流水线的中间步骤必须是“变换”，即它们必须实施拟合和变换方法。 最终的估算人员只需要实施合适的。管道的目的是组装几个可以一起交叉验证的步骤，同时设置不同的参数。 为此，它可以使用名称和参数名称以'\_\_'分隔来设置各个步骤的参数。

案例如下：

*#encoding=utf-8***from** sklearn.pipeline **import** Pipeline, FeatureUnion  
**from** sklearn.model\_selection **import** GridSearchCV  
**from** sklearn.svm **import** SVC  
**from** sklearn.datasets **import** load\_iris  
**from** sklearn.decomposition **import** PCA  
**from** sklearn.feature\_selection **import** SelectKBest,chi2  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** accuracy\_score  
**import** numpy **as** np  
iris = load\_iris()  
X, y = iris.data, iris.target  
*#This dataset is way to high-dimensional. Better do PCA:*pca = PCA()  
*#Maybe some original features where good, too?*selection = SelectKBest()  
*# Build estimator from PCA and Univariate selection:*combined\_features = FeatureUnion([(**"pca"**, pca), (**"univ\_select"**, selection)])  
*# Use combined features to transform dataset:*svm = SVC(kernel=**"linear"**)  
*# Do grid search over k, n\_components and C:*pipeline = Pipeline([(**"features"**, combined\_features), (**"svm"**, svm)])  
param\_grid = dict(features\_\_pca\_\_n\_components=[1, 2, 3],  
 features\_\_univ\_select\_\_k=[1, 2],  
 svm\_\_C=[0.1, 1, 10])  
grid\_search = GridSearchCV(pipeline, param\_grid=param\_grid)  
grid\_search.fit(X, y)  
print(grid\_search.best\_params\_)  
*#通过上面的得到最佳参数下面基于这些超参数进行模型构建*pca = PCA(n\_components=2)  
selection = SelectKBest(k=2)  
combined\_features = FeatureUnion([(**"pca"**, pca), (**"univ\_select"**, selection)])  
*#fit\_transform先训练数据，再转换数据最后进行合并*X\_features = combined\_features.fit\_transform(X,y)  
X\_train,X\_val,y\_train,y\_val=train\_test\_split(X\_features,y,test\_size=0.2)  
svm = SVC(kernel=**"linear"**)  
svm.fit(X\_train,y\_train)  
yHat=svm.predict(X\_val)  
print(accuracy\_score(yHat,y\_val))

**概率图模型**

概率图模型是一类用图来表达变量相关关系的概率模型，它以图为表示工具，最常见的是用一个结点表示一个或一组随机变量，结点之间的边表示变量间的概率相关关系。概率图模型可大致分为两类：第一类是使用有向无环图表示变量间的依赖关系，称为有向图模型或贝叶斯网，第二类是使用无向图表示变量间的相关关系，称为无向图模型或马尔科夫网。

**马尔科夫系列**

马尔科夫链（Markov chain）：系统下一刻的转态仅由当前状态决定，不依赖于以往的任何状态。（或者说系统的当前状态只依赖于它的前一个状态而与其它任何状态无关。）

隐马尔科夫模型（Hidden Markov Model,简称HMM）:它是结构最简单的动态贝叶斯网，这是一种著名的有向图模型，主要用于时序分析，在语音识别，自然语言处理等领域应用广泛。

什么样的问题需要HMM模型？要具备如下特征。第一：我们的问题是基于序列的，比如时间序列，或状态序列。第二：我们的问题中有两类数据，一类序列数据是可以观测到的，即观测序列。另一类数据是不能观测到的，即隐藏状态序列，简称状态序列。

隐马尔科夫模型中的变量可分为两组，第一组是状态变量，通常假定状态变量是隐藏的、不可观测的，因此状态变量亦称隐变量，另一组是观测变量，xi表示第i时刻的观测值，隐马尔科夫模型中的箭头表示变量间的依赖关系。在任一时刻，观测变量的取值近依赖于状态变量（即xt仅由yt确定）与其它状态变量及观测变量的取值无关。同时t时刻的状态yt仅依赖于t-1时刻的状态yt-1,与此前的t-2个状态无关。

**CRF条件随机场**

条件随机场(Conditional Random Fields, 以下简称CRF)是给定一组输入序列条件下另一组输出序列的条件概率分布模型，它是一种判别式无向图模型，生成式模型是直接对联合分布进行建模，而判别式模型则是对条件分布进行建模，隐马尔科夫模型和马尔科夫随机场是生成式模型，而条件随机场是生成式模型。在自然语言处理中得到了广泛应用。

**什么样的问题需要CRF？**

为了让我们的分类器表现的更好，可以在标记数据的时候，可以考虑相邻数据的标记信息。这一点，是普通的分类器难以做到的。而这一块，也是CRF比较擅长的地方。

在实际应用中，自然语言处理中的词性标注(POS Tagging)就是非常适合CRF使用的地方。词性标注的目标是给出一个句子中每个词的词性（名词，动词，形容词等）。而这些词的词性往往和上下文的词的词性有关，因此，使用CRF来处理是很适合的，当然CRF不是唯一的选择，也有很多其他的词性标注方法。

**什么是随机场？什么是马尔科夫随机场？什么是条件随机场**？

**随机场**是由若干个位置组成的整体，当给每一个位置中按照某种分布随机赋予一个值之后，其全体就叫做随机场。还是举词性标注的例子：假如我们有一个十个词形成的句子需要做词性标注。这十个词每个词的词性可以在我们已知的词性集合（名词，动词...)中去选择。当我们为每个词选择完词性后，这就形成了一个随机场。

**马尔科夫随机场（Markov Random Field，简称MRF）是随机场的特例**，它是一种典型的马尔科夫网，这是一种著名的无向图模型，图中的每一个结点表示一个或一组变量，结点之间的边表示两个变量之间的依赖关系。它假设随机场中某一个位置的赋值仅仅与和它相邻的位置的赋值有关，和与其不相邻的位置的赋值无关。继续举十个词的句子词性标注的例子：如果我们假设所有词的词性只和它相邻的词的词性有关时，这个随机场就特化成一个马尔科夫随机场。比如第三个词的词性除了与自己本身的位置有关外，只与第二个词和第四个词的词性有关。

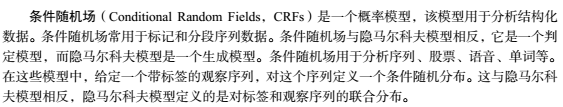
**CRF是马尔科夫随机场的特例**，它假设马尔科夫随机场中只有X和Y两种变量，X一般是给定的，而Y一般是在给定X的条件下我们的输出。这样马尔科夫随机场就特化成了条件随机场。在我们十个词的句子词性标注的例子中，X是词，Y是词性。因此，如果我们假设它是一个马尔科夫随机场，那么它也就是一个CRF。

　　　对于CRF，我们给出准确的数学语言描述：

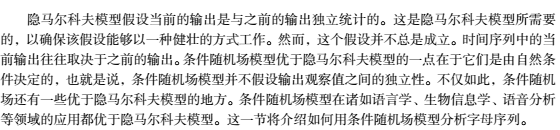
　　　设X与Y是随机变量，P(Y|X)是给定X时Y的条件概率分布，若随机变量Y构成的是一个马尔科夫随机场，则称条件概率分布P(Y|X)是条件随机场。

**MRF与CRF条件随机场的区别**

**MRF处理的是联合概率，CRF处理的是条件概率。**



HMM与CRF的比较



**采样**

采样用于解决什么问题？

从MCMC开始，前个MC指蒙特卡罗方法，后个MC指马尔科夫链

具体参考：

http://www.cnblogs.com/pinard/p/6625739.html

最早的蒙特卡罗方法都是为了求解一些不太好求解的求和或者积分问题。比如积分：

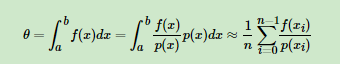
θ=∫baf(x)dxθ=∫abf(x)dx

如果我们很难求解出f(x)的原函数，那么这个积分比较难求解。当然我们可以通过蒙特卡罗方法来模拟求解近似值。

如果在(a,b)范围内数据是均匀分布的，那么我们可以取n个值用它们的均值来代表(a,b)区间上所有f(x)的值，这样我们上面的积分近似的求解为

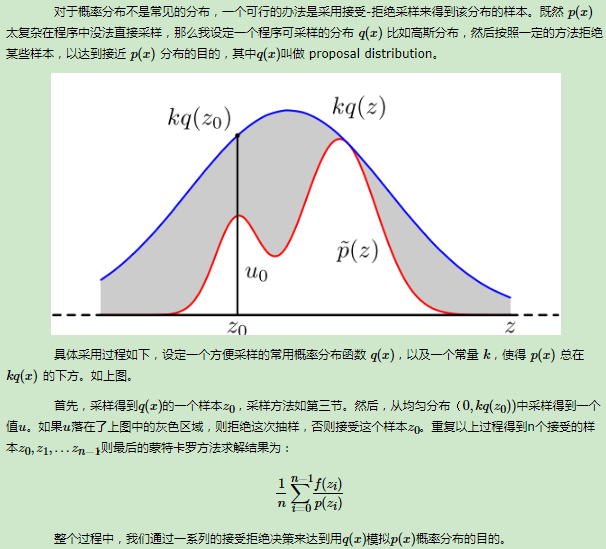
虽然上面的方法可以一定程度上求解出近似的解，但是它隐含了一个假定，即x在[a,b]之间是均匀分布的，而绝大部分情况，x在[a,b]之间不是均匀分布的。如果我们用上面的方法，则模拟求出的结果很可能和真实值相差甚远。

怎么解决这个问题呢？ 如果我们可以得到x在[a,b]的概率分布函数p(x)，那么我们的定积分求和可以这样进行：



上式最右边的这个形式就是蒙特卡罗方法的一般形式。当然这里是连续函数形式的蒙特卡罗方法，但是在离散时一样成立。

上一节我们讲到蒙特卡罗方法的关键是得到x的概率分布。不过很多时候，我们的xx的概率分布不是常见的分布，这意味着我们没法方便的得到这些非常见的概率分布的样本集。那这个问题怎么解决呢？



使用接受-拒绝采样，我们可以解决一些概率分布不是常见的分布的时候，得到其采样集并用蒙特卡罗方法求和的目的。但是接受-拒绝采样也只能部分满足我们的需求，在很多时候我们还是很难得到我们的概率分布的样本集。要想将蒙特卡罗方法作为一个通用的采样模拟求和的方法，必须解决如何方便得到各种复杂概率分布的对应的采样样本集的问题。而我们下一篇要讲到的马尔科夫链就是帮助找到这些复杂概率分布的对应的采样样本集的白衣骑士。如果假定我们可以得到我们需要采样样本的平稳分布所对应的马尔科夫链状态转移矩阵，那么我们就可以用马尔科夫链采样得到我们需要的样本集，进而进行蒙特卡罗模拟。但是一个重要的问题是，随意给定一个平稳分布π,如何得到它所对应的马尔科夫链状态转移矩阵P呢？这是个大问题。我们绕了一圈似乎还是没有解决任意概率分布采样样本集的问题。

　　幸运的是，MCMC采样通过迂回的方式解决了上面这个大问题，我们在下一篇来讨论MCMC的采样，以及它的使用改进版采样: M-H采样和Gibbs采样.

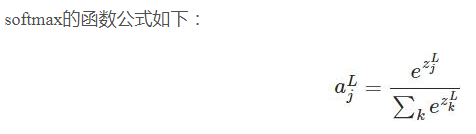
对于MCMC采样我们的目标矩阵P可以通过任意一个马尔科夫链状态转移矩阵QQ乘以α(i,j)得到。α(i,j)我们有一般称之为接受率。取值在[0,1]之间，可以理解为一个概率值。即目标矩阵P可以通过任意一个马尔科夫链状态转移矩阵Q以一定的接受率获得。但是这个采样算法还是比较难在实际中应用，为什么呢？问题在上面第三步的c步骤，接受率这儿。由于α(xt,x∗)可能非常的小，比如0.1，导致我们大部分的采样值都被拒绝转移，采样效率很低。有可能我们采样了上百万次马尔可夫链还没有收敛，也就是上面这个n1要非常非常的大，这让人难以接受，怎么办呢？这时就轮到我们的M-H采样出场了。

但是M-H采样有两个缺点：一是需要计算接受率，在高维时计算量大。并且由于接受率的原因导致算法收敛时间变长。二是有些高维数据，特征的条件概率分布好求，但是特征的联合分布不好求。因此需要一个好的方法来改进M-H采样，这就是我们下面讲到的Gibbs采样。

**有了Gibbs采样来获取概率分布的样本集，有了蒙特卡罗方法来用样本集模拟求和，他们一起就奠定了MCMC算法在大数据时代高维数据模拟求和时的作用。**

**Softmax函数以及交叉熵损失函数**

Softmax函数：



softmax函数最明显的特点在于：它把每个神经元的输入占当前层所有神经元输入之和的比值，当作该神经元的输出。这使得输出更容易被解释：神经元的输出值越大，则该神经元对应的类别是真实类别的可能性更高。

另外，softmax不仅把神经元输出构造成概率分布，而且还起到了归一化的作用，适用于很多需要进行归一化处理的分类问题。

softmax用于多分类过程中，它将多个神经元的输出，映射到（0,1）区间内，**可以看成概率来理解**，从而来进行多分类！

**交叉熵：**

在机器学习中的分类算法中，我们总是最小化交叉熵，因为交叉熵越低，就证明由算法所产生的策略最接近最优策略，也间接证明我们算法所算出的非真实分布越接近真实分布。

之所以选择交叉熵作为损失函数而不是平方差是因为用平方差作为损失函数训练过程较为缓慢。

**期望（数学期望，均值）**

数学期望又称期望、均值，对于离散型数据，x属于X，每个x发生的概率是Px则期望为x1\*Px1+...xn\*Pxn

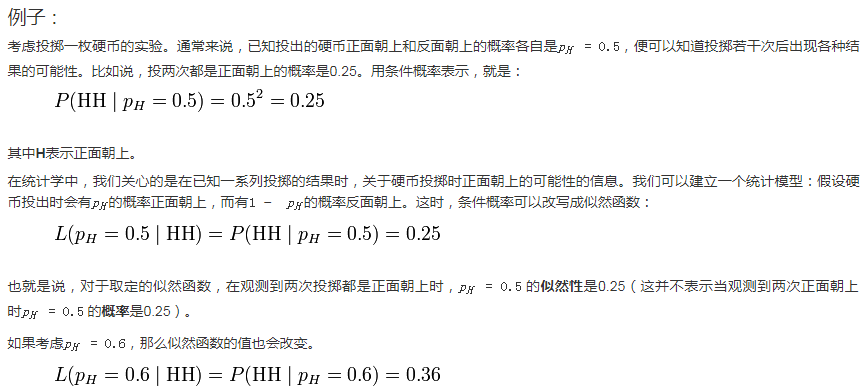
**似然函数**

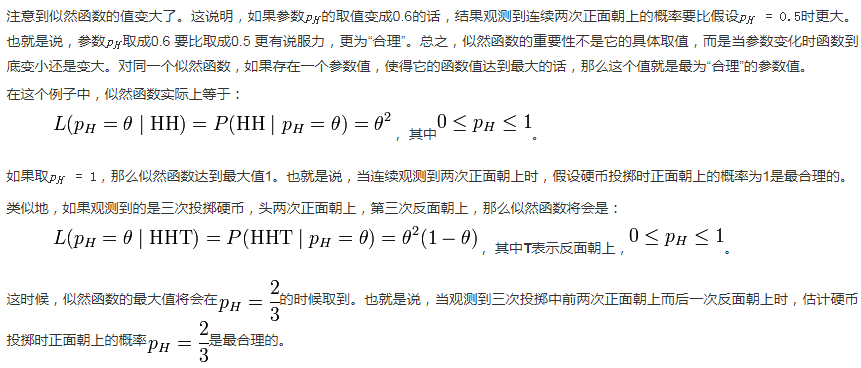
在[数理统计学](http://zh.wikipedia.org/wiki/%E6%95%B0%E7%90%86%E7%BB%9F%E8%AE%A1%E5%AD%A6" \o "数理统计学" \t "https://blog.csdn.net/lwq1026/article/details/_blank)中，似然函数是一种关于[统计模型](http://zh.wikipedia.org/w/index.php?title=%E7%BB%9F%E8%AE%A1%E6%A8%A1%E5%9E%8B&action=edit&redlink=1" \o "统计模型" \t "https://blog.csdn.net/lwq1026/article/details/_blank)中的[参数](http://zh.wikipedia.org/wiki/%E5%8F%82%E6%95%B0" \o "参数" \t "https://blog.csdn.net/lwq1026/article/details/_blank)的[函数](http://zh.wikipedia.org/wiki/%E5%87%BD%E6%95%B0" \o "函数" \t "https://blog.csdn.net/lwq1026/article/details/_blank)，表示模型参数中的似然性。

**概率和似然的区别**

[概率](http://zh.wikipedia.org/wiki/%E6%A6%82%E7%8E%87" \o "概率" \t "https://blog.csdn.net/lwq1026/article/details/_blank)用于在已知一些参数的情况下，预测接下来的观测所得到的结果，而似然性则是用于在**已知某些观测所得到的结果时，对有关事物的性质的参数进行估计。**在这种意义上，似然函数可以理解为[条件概率](http://zh.wikipedia.org/wiki/%E6%9D%A1%E4%BB%B6%E6%A6%82%E7%8E%87" \o "条件概率" \t "https://blog.csdn.net/lwq1026/article/details/_blank)的逆反。

例子：





**最大似然估计**

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。

**EM算法**

EM算法也称期望最大化（Expectation-Maximum,简称EM）算法，它是一个基础算法。

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。

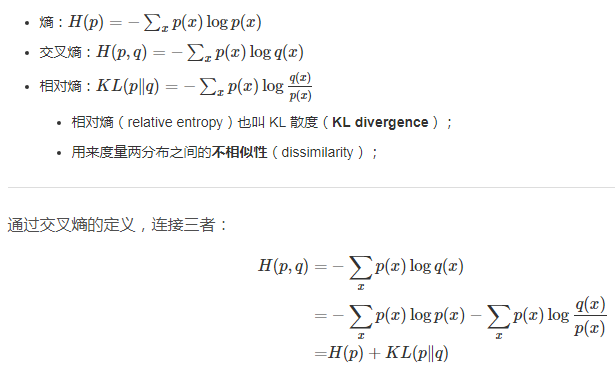
　　 但是在一些情况下，我们得到的观察数据有未观察到的隐含数据，此时我们未知的有隐含数据和模型参数，因而无法直接用极大化对数似然函数得到模型分布的参数。怎么办呢？这就是EM算法可以派上用场的地方了。

　　　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](http://www.cnblogs.com/pinard/p/6164214.html)。在K-Means聚类时，每个聚类簇的质心是隐含数据。我们会假设K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。

**熵，交叉熵，相对熵（KL散度）**



注意交叉熵和相对熵都是基于两概率分布进行衡量的，一般对目标类别标签进行独热编码，对求得的数据进行softmax转换，转换成符合概率分布的。

交叉熵作为神经网络的损失函数时，p代表正确答案，q代表预测值，通过交叉熵计算这两个概率分布之间的距离，交叉熵的值越小，两个概率分布越接近。给定两个概率分布p和q，交叉熵计算公式为：，

相似度的计算

1）利用numpy计算皮尔逊相关系数

import numpy as np

a = np.array([1,2,3])

b = np.array([2,5,8])

x = np.vstack((a,b))

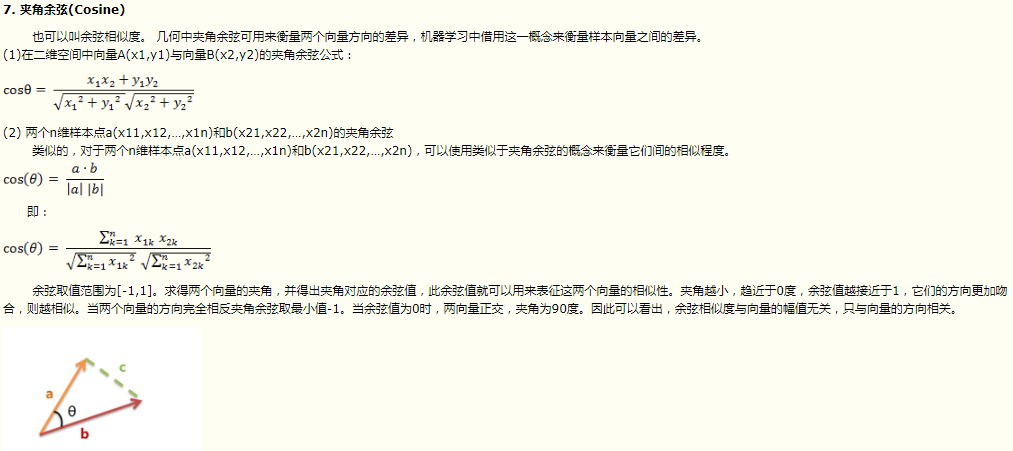
np.corrcoef(a, b)

np.corrcoef(x)

其中vstack（）是按照垂直方向水平增加维度，就是把a,b组成两行n维的数组。

Corrcoef（）返回的结果是一个两行两列数组斜对角线是两向量皮尔逊相关系数。

1. 利用scipy的距离计算库计算各种距离



例如余弦相似度的计算

from scipy.spatial.distance import pdist

X=np.vstack([x,y])

d2=1-pdist(X,'cosine')

或者d2=1-pdist([x,x],'cosine')

官方文档：<https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.spatial.distance.pdist.html>

具体参考博客：

<https://www.cnblogs.com/denny402/p/7028832.html>

1. Gram

<https://www.cnblogs.com/ljy2013/p/6425277.html>

<https://baike.baidu.com/item/N-Gram/10803752?fr=aladdin>

ROC,AUC的计算及相关概念解析

|  |  |  |
| --- | --- | --- |
| 真实情况 | 预测结果 | |
| 正例 | 反例 |
| 正例 | TP（真正例） | FN（假反例） |
| 反例 | FP（假正例） | TN（真反例） |

查准率：预测结果中预测正确的比例

真正例/（真正例+假正例）

查全率：预测的正例结果占所有正例的比率

真正例/（真正例+假反例）

真正例率（TPR）：真实的正例中，被预测正确的比例

真正例/（真正例+假反例）

假正例率（FPR）：真实的反例中，被预测错误的比例

假正例/（真反例+假正例）

AUC是ROC曲线下的面积

ROC曲线的纵轴是真正例率，横轴是假正例率

1. R曲线的纵轴是查准率，横轴是查全率

AUC在sklearn中的两种计算方法：

1. from sklearn.metrics import roc\_curve,auc

y = np.array([1, 1, 2, 2])

#注意这里的score可以是预测的0,1值也可以是预测的是正例的比率

scores = np.array([0.1, 0.4, 0.35, 0.8])

#pos\_label意思是指定类别标签2为正例

fpr, tpr, thresholds = roc\_curve(y, scores, pos\_label=2)

Print(auc(fpr, tpr))

Fpr

array([ 0. , 0.5, 0.5, 1. ])

tpr

array([ 0.5, 0.5, 1. , 1. ])

Thresholds

array([ 0.8 , 0.4 , 0.35, 0.1 ])

这里注意返回的Thresholds表示阀值，是score的降序排序

对于FPR和TPR具体的计算过程参考：

<https://blog.csdn.net/ybdesire/article/details/51999995>

这种方法是先根据ROC得到FPR和TPR，在根据FPR和TPR计算AUC

1. from sklearn.metrics import roc\_auc\_score

y\_true = np.array([0, 0, 1, 1])

y\_scores = np.array([0.1, 0.4, 0.35, 0.8])

roc\_auc\_score(y\_true, y\_scores)

这种方法直接计算出值