



REVIEW DOCUMENT FOR GRADUATE PROGRAMS IN 2021

Organized by:

徐睿滢 Ruiying Xu
+353 857736316
ruiying.xu@ucdconnect.ie

孟子茉Zimo Meng
+353 873982916
zimo.meng@ucdconnect.ie

Data Preparation

Missing Value

- **数值缺失机制：**

1.完全随机缺失(MCAR)：缺失数据与该变量的真实值无关，与其他变量的数值也无关。

举例：一位老师抱着批改完的卷子走在路上，不小心摔倒丢失了几张卷子，因此有几位同学没有成绩。这种成绩缺失不是因为成绩这个变量本身高或低而丢失的，而是随机丢失的；也与性别等无关，不会出现男生卷子丢失概率高，女生卷子丢失概率低的问题。

2.条件随机缺失(MAR)：缺失数据与其他变量有关。

举例：我们的目标是要统计一个班学生的基本信息，包括名字、性别、身高、体重等。而此时如果某一学生的体重这一变量缺失，这一事件最可能发生在哪些人身上呢？一般来说，是女生。因此体重缺失与已知变量性别相关，这就叫做条件随机缺失。

3.非随机缺失(NNAR)：缺失数据依赖于该变量本身。

举例：通常在收集数据时收入一栏很容易缺失，发生这种情况的原因可能是填写人收入过高或过低。因此收入缺失与填写人本身收入有关，这就叫做非随机缺失。

- **处理方法：**

- 1. **删除**

主要有简单删除法和权重法。简单删除法是对缺失值进行处理的最原始方法。

- (1) **简单删除法**

此方法将存在缺失值的数据条目（对象，元组，记录）进行删除。这种方法简单易行，在对象有多个属性缺失值、被删除的含缺失值的对象与信息表中的数据量相比非常小的情况下是非常有效的。然而，这种方法却有很大的局限性。它是以减少历史数据来换取信息的完备，会造成资源的大量浪费，丢弃了大量隐藏在这些对象中的信息。在信息表中本来包含的对象很少的情况下，删除少量对象就足以严重影响到信息表信息的客观性和结果的正确性；当每个属性空值的百分比变化很大时，它的性能非常差。

- (2) **权重法**

当缺失值的类型为非完全随机缺失的时候，通过对完整的数据加权来减小偏差。把数据不完全的个案标记后，将完整的数据个案赋予不同的权重，个案的权重可以通过logistic或probit回归求得。如果解释变量中存在对权重估计起决定性因素的变量，那

么这种方法可以有效减小偏差。如果解释变量和权重并不相关，它并不能减小偏差。对于存在多个属性缺失的情况，就需要对不同属性的缺失组合赋不同的权重，这将大大增加计算的难度，降低预测的准确性，这时权重法并不理想。

2. 填补

1) 均值填充

将信息表中的属性分为数值属性和非数值属性来分别进行处理。如果空值是数值型的，就根据该属性在其他所有对象的取值的平均值来填充该缺失的属性值；如果空值是非数值型的，就根据统计学中的众数原理，用该属性在其他所有对象的取值次数最多的值(即出现频率最高的值)来补齐该缺失的属性值。

2) 特殊值填充 (Treating Missing Attribute values as Special values)

将空值作为一种特殊的属性值来处理，它不同于其他的任何属性值。如所有的空值都用“unknown”填充。这样将形成另一个有趣的概念，可能导致严重的数据偏离，一般不推荐使用。

3) 同类均值插补：

用层次聚类模型预测缺失变量的类型，再以该类型的均值插补。假设 $X=(X_1, X_2 \dots X_p)$ 为信息完全的变量，Y为存在缺失值的变量，那么首先对X或其子集行聚类，然后按缺失个案所属类来插补不同类的均值。**如果在以后统计分析中还需以引入的解释变量和Y做分析，那么这种插补方法将在模型中引入自相关，给分析造成障碍。**

4) 同类均值随机插补：

在利用上述方法进行均值插补时，会存在一个问题：但凡是同类样本，对该变量的预测值都是相同的。为了解决这一问题，在进行数值插补时，我们可以生成正态随机数，其中均值为该类样本均值，方差为该类样本方差。

5) 热卡填充 (就近补齐)

对于一个包含空值的对象，就近补齐在完整数据中找到一个与它最相似的对象，然后用这个相似对象的值来进行填充。不同的问题可能会选用不同的标准来对相似进行判定。该方法概念上很简单，且利用了数据间的关系来进行空值估计。这个方法的缺点在于难以定义相似标准，主观因素较多。

6) K近邻

先根据欧式距离或相关分析来确定距离缺失数据最近的K个样本，将这K个值加权平均来估计该样本的缺失数据。

7) 极大似然估计

在缺失类型为随机缺失的条件下，假设模型对于完整的样本是正确的，那么通过观测数据的边际分布可以对未知参数进行极大似然估计。这种方法也被称为忽略缺失值的极大似然估计，对于极大似然的参数估计实际中常采用的计算方法是期望值最大化 (Expectation Maximization, EM)。该方法比删除个案和单值插补更有吸引力，它一个重要前提：**适用于大样本**。有效样本的数量足够以保证ML估计值是渐近无偏的并服从正态分布。但是这种方法可能会陷入局部极值，收敛速度也不是很快，并且计算很复杂。

8) 回归

基于完整的数据集，建立回归方程（模型）。对于包含空值的对象，将已知属性值代入方程来估计未知属性值，以此估计值来进行填充。当变量不是线性相关或预测变量高度相关时会导致有偏差的估计。

9) 多重插补

多值插补的思想来源于贝叶斯估计，认为待插补的值是随机的，它的值来自于已观测到的值。具体实践上通常是估计出待插补的值，然后再加上不同的噪声，形成多组可供选择的插补值。根据某种选择依据，选取最合适的选择值。

多重插补方法分为三个步骤：①为每个空值产生一套可能的插补值，这些值反映了无响应模型的不确定性；每个值都可以被用来插补数据集中的缺失值，产生若干个完整数据集合。②每个插补数据集合都用针对完整数据集的统计方法进行统计分析。③对来自各个插补数据集的结果，根据评分函数进行选择，产生最终的插补值。

假设一组数据，包括三个变量Y1, Y2, Y3，它们的联合分布为正态分布，将这组数据处理成三组，A组保持原始数据，B组仅缺失Y3，C组缺失Y1和Y2。在多值插补时，对A组将不进行任何处理，对B组产生Y3的一组估计值（作Y3关于Y1, Y2的回归），对C组作产生Y1和Y2的一组成对估计值（作Y1, Y2关于Y3的回归）。

当用多值插补时，对A组将不进行处理，对B、C组将完整的样本随机抽取形成为m组（m为可选择的m组插补值），每组个案数只要能够有效估计参数就可以了。对存在缺失值的属性的分布作出估计，然后基于这m组观测值，对于这m组样本分别产生关于参数的m组估计值，给出相应的预测即，这时采用的估计方法为极大似然法，在计算机中具体的实现算法为期望最大化法（EM）。对B组估计出一组Y3的值，对C将利用Y1, Y2, Y3它们的联合分布为正态分布这一前提，估计出一组(Y1, Y2)。

上例中假定了Y1, Y2, Y3的联合分布为正态分布。这个假设是人为的，但是已经通过验证（Graham和Schafer于1999），非正态联合分布的变量，在这个假定下仍然可以估计到很接近真实值的结果。

多重插补和贝叶斯估计的思想是一致的，但是多重插补弥补了贝叶斯估计的几个不足。

(1)贝叶斯估计以极大似然的方法估计，极大似然的方法要求模型的形式必须准确，如果参数形式不正确，将得到错误得结论，即先验分布将影响后验分布的准确性。而多重插补所依据的是大样本渐近完整的数据的理论，在数据挖掘中的数据量都很大，先验分布将极小的影响结果，所以先验分布的对结果的影响不大。

(2)贝叶斯估计仅要求知道未知参数的先验分布，没有利用与参数的关系。而多重插补对参数的联合分布作出了估计，利用了参数间的相互关系。

表 3.4 各种插补效果的比较

插补方法	优点	缺点	适用环境
类均值插补	简单易行；被插补的值比较稳定	不能反映缺失值变异性；低估了资料变异	低缺失率首选
类随机插补	能体现数据变异性	依赖于观测值；稳健性差	低缺失率
回归插补	方差估计较好	稳健性依赖于辅助变量；抽样误差不易控制	变量间相关性强
EM 插补	利用充分，考虑了缺失值的不确定性	计算复杂	高缺失率
多重插补 (MCMC 法)	利用充分，考虑了缺失值的不确定性	计算复杂	高缺失率首选

5. 缺失值怎么处理？异常值怎么判断？

连续型我答用均值填充，又问离散型怎么弄，想了半天没想到，面试官提示介绍下极大似然估计法，最后说用众数去填充。

异常值看数据分布，均值、标准差，画图，箱线图判断等

可以回到

Outlier

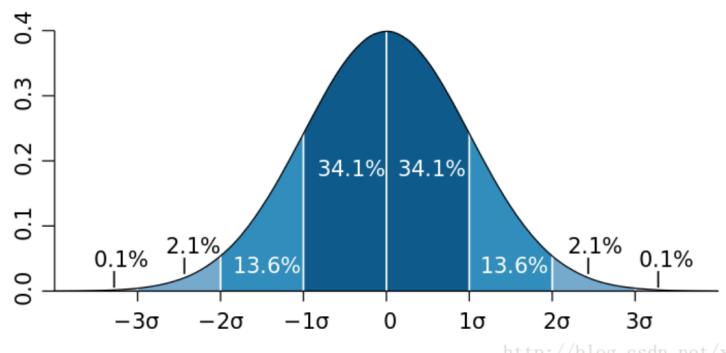
1. 简单统计分析：

对属性值进行一个描述性的统计，从而查看哪些值是不合理的。

2. 3σ 原则

当数据服从正态分布：

根据正态分布的定义可知，距离平均值 3σ 之外的概率为 $P(|x-\mu|>3\sigma) \leq 0.003$ ，这属于极小概率事件，在默认情况下我们可以认定，距离超过平均值 3σ 的样本是不存在的。因此，当样本距离平均值大于 3σ ，则认定该样本为异常值。

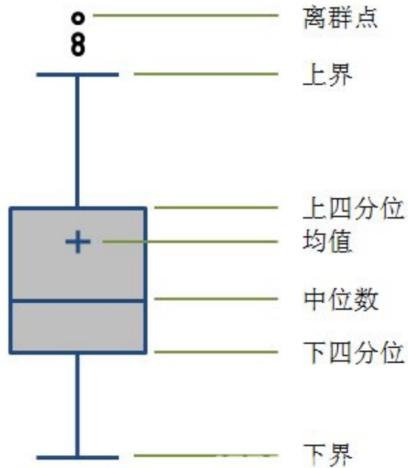


当数据不服从正态分布：

当数据不服从正态分布，可以通过远离平均距离多少倍的标准差来判定，多少倍的取值需要根据经验和实际情况来决定。

3. 箱型图分析

箱型图提供了一个识别异常值的标准，即大于或小于箱型图设定的上下界的数值即为异常值，箱型图如下图所示：



首先我们定义上四分位和下四分位。

上四分位我们设为 U , 表示的是所有样本中只有 $1/4$ 的数值大于 U

同理, 下四分位我们设为 L , 表示的是所有样本中只有 $1/4$ 的数值小于 L

那么, 上下界又是什么呢?

我们设上四分位与下四分位的差值为IQR, 即: $IQR=U-L$

那么, 上界为 $U+1.5IQR$, 下界为: $L - 1.5IQR$

箱型图选取异常值比较客观, 在识别异常值方面有一定的优越性。

异常值的处理方法常用有四种:

1. 删除含有异常值的记录
2. 将异常值视为缺失值, 交给缺失值处理方法来处理
3. 用平均值来修正
4. 不处理

需要强调的是, 如何判定和处理异常值, 需要结合实际。

样本不平衡问题

<https://zhuanlan.zhihu.com/p/84322912>

<https://zhuanlan.zhihu.com/p/56882616>

样本类别不均衡可能导致过拟合。

对模型训练和模型评估有何影响和危害

样本类别不均衡将导致样本量少的分类所包含的特征过少, 并很难从中提取规律; 即使得到分类模型, 也容易产生过度依赖与有限的数据样本而导致过拟合问题, 当模型应用到新的数据上时, 模型的预测效果也会很差, 关注的评估指标表现也会不理想。

传统的学习方法以降低**总体分类准确度ACC (accuracy)**为目标, 将所有样本一视同仁, 同等对待, 造成了分类器在多数类的分类精度较高而在少数类的分类精度很低。

需要注意的是, 尽管少数类的样本个数更少, 表示的质量也更差, 但其通常会携带更重要的信息, 因此一般我们更关注模型正确分类少数类样本的能力。

机器学习模型都有一个待优化的损失函数，以我们最常用最简单的二元分类器逻辑回归为例，其损失函数如下所示：

$$\begin{aligned} J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right] \end{aligned}$$

逻辑回归以优化总体的精度为目标，不同类别的误分类情况产生的误差是相同的。

理论学习的解决方法

1、数据层面

- 1) 考虑对大类下的样本（超过1万、十万甚至更多）进行欠采样，即删除部分样本。
- 2) 考虑对小类下的样本（不足1万甚至更少）进行过采样，即添加部分样本的副本。

这样的直接删除和复制，实际上是对数据所提供信息的损耗。为了解决这个问题，我们可以人工产生数据样本。

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。该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。具体可以参见原始论文。这里有SMOTE算法的多个不同语言的实现版本：

Python: UnbalancedDataset模块提供了SMOTE算法的多种不同实现版本，以及多种重采样算法。

R: DMwR package。

Weka: SMOTE supervised filter。

除常见的采样方法外，以下2种方法也可做尝试：

- 1) 分治ensemble：将大类中样本聚类到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。
- 2) 分层级ensemble：使用原始数据集训练第一个学习器L1；将L1错分的数据集作为新的数据集训练L2；将L1和L2分类结果不一致的数据作为数据集训练L3；最后测试集上将三个分类器的结果汇总（结合这三个分类器，采用投票的方式来决定分类结果，因此只有当L2与L3都分类为false时，最终结果才为false，否则true。）

2、算法层面

- 转化为异常值识别问题

我们可以从不同于分类的角度去解决数据不均衡性问题，我们可以把那些小类的样本作为异常点(outliers)，因此该问题便转化为异常点检测(anomaly detection)与变化趋势检测问题(change detection)。

异常点检测即是对那些罕见事件进行识别。如通过机器的部件的振动识别机器故障，又如通过系统调用序列识别恶意程序。这些事件相对于正常情况是很少见的。

变化趋势检测类似于异常点检测，不同在于其通过检测不寻常的变化趋势来识别。如通过观察用户模式或银行交易来检测用户行为的不寻常改变。

将小类样本作为异常点这种思维的转变，可以帮助考虑新的方法去分离或分类样本。这两种方法从不同的角度去思考，让你尝试新的方法去解决问题。

- 尝试不同的分类算法

强烈建议不要对待每一个分类都使用自己喜欢而熟悉的分类算法。应该使用不同的算法对其进行比较，因为不同的算法使用于不同的任务与数据。具体可以参见“Why you should be Spot-Checking Algorithms on your Machine Learning Problems”。

决策树往往在类别不均衡数据上表现不错。它使用基于类变量的划分规则去创建分类树，因此可以强制地将不同类别的样本分开。目前流行的决策树算法有：C4.5、C5.0、CART和Random Forest等。

- 对小类错分进行加权惩罚

对分类器的小类样本数据增加权值，降低大类样本的权值（这种方法其实是产生了新的数据分布，即产生了新的数据集，译者注），从而使得分类器将重点集中在小类样本本身上。一个具体做法就是，在训练分类器时，若分类器将小类样本分错时额外增加分类器一个小类样本分错代价，这个额外的代价可以使得分类器更加“关心”小类样本。如penalized-SVM和penalized-LDA算法。

对小样本进行过采样（例如含L倍的重复数据），其实在计算小样本错分cost functions时会累加L倍的惩罚分数。

- 从重构分类器的角度出发

仔细对你的问题进行分析与挖掘，是否可以将你的问题划分成多个更小的问题，而这些小问题更容易解决。你可以从这篇文章In classification, how do you handle an unbalanced training set?中得到灵感。例如：

将你的大类压缩成小类；

使用One Class分类器（将小类作为异常点）；

使用集成方式，训练多个分类器，然后联合这些分类器进行分类；

将二分类问题改成多分类问题

- 在深度学习中，有哪些解决样本不平衡的方法？

深度学习同样属于机器学习中的一种典型方法，所以在机器学习中适用的方法在深度学习中同样适用。比如说：扩大数据集、类别均衡采样、人工产生数据样本，添加少类别样本的来loss惩罚项等。

深度学习中的三种方法：类别均衡采样、OHEM和focal loss

<https://zhuanlan.zhihu.com/p/56882616>

TIPS

1) 、更有效的方法：使用代价函数学习得到每个类的权值，大类的权值小，小类的权值大。刚开始，可以设置每个类别的权值与样本个数比例的倒数，然后可以使用过采样进行调优。

2) 、赋予小类样本更高的训练权值，或者对小类进行过采样。某些时候，高不平衡性下仍然可以得到效果较好的训练结果。我认为对于某些评价指标是有意义的，如AUC。

3) 、在非极端条件下直接cost-sensitive基本上可以得到不错的表现。大部分欠采样的策略都是直觉决定的，很容易丢掉一些有用的信息。但是重采样+集成的方法效果会好很多，即便是基于随机欠采样的方法效果也都还不错。在数据集噪声较小的情况下，推荐BalanceCascade，它可以用较少的基分类器数量得到较好的表现。噪声大的情况下直接re-sample+Bagging会更鲁棒一些。当然也有cost-sensitive+集成的方法。

4) 、algorithm-level methods 此类方法通常通过改变已有的学习算法来使其倾向于重视少数类数据的模式，这一过程可通过加入数据权重(e.g. boosting)或者对误分类数据的惩罚(cost sensitive learning)来使学习算法在不平衡数据上工作。

5) 、 algorithm-level methods 该类方法主要通过对已有的平衡学习算法做出适应性的修改，来修正其对多数类的分类偏好，该类方法最广为人知的一个分支即代价敏感学习(cost sensitive learning) [2]。顾名思义即对不同类别的误分类样本施加不同的罚项(penalty)，以二分类问题为例，我们会增大将Min误分类为Maj的penalty (i.e. 将Min分类为Maj会贡献更多的Loss) 来使模型的输出更加倾向于少数类。

6) 、如果实际数据已经类别极度不平衡了，比如正负样本比例1: 10000，那么一切的处理方法都无法挽救[苦笑表情]。

记住，其实并不知道哪种方法最适合你的任务与数据，你可以使用一些启发式规则或经验去选择某一个较优算法。当然最好的方法测试每一种算法，然后选择最好的方法。最重要的是，从点滴开始做起，根据自己现有的知识，并不断学习去一步步完善。

你不必成为一个精通所有算法的算法奇才或者一个建立准确而可靠的处理数据不平衡的模型的统计学家，你只需要根据你的问题的实际情况从上述算法或方法中去选择一种或两种方法去使用。希望上述的某些方法能够解决你的问题。例如使用准确率P值，召回率R值，以及auc、ks评价指标或重采样算法速度快并且有效。

Classification

Linear Regression to Logistic Regression

<https://zhuanlan.zhihu.com/p/39363869>

Linear Regression

思路

线性回归是利用数理统计中回归分析，这是一个参数模型，它假设因变量和自变量之间是线性关系的，在已知模型格式（线性方程）的情况下，简化为求最优参数的问题，（误差e为误差服从均值为0的正态分布）

找到使得cost function（一般是最小二乘）最小的参数，这个参数可以通过矩阵推导出来，同时在数据越来越多，计算变得复杂的情况下，可以使用梯度下降找到结果

求解参数推导

线性回归是一个参数模型，在已知模型格式的情况下，可以简化成求最优参数

> Function form:

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

(James et al., 2014, §2.1.1).

> Cost function: 最小二乘法

$$J(w) = \frac{1}{2m} \sum_{i=1}^m (f(x_i) - y_i)^2$$

简化为求解 $J(w)$ 的最小值时的参数

继续简化为求 $J'(w) = 0$ 时的参数 w 时

矩阵方法: <https://zhuanlan.zhihu.com/p/66192625>

概率方法: <https://zhuanlan.zhihu.com/p/95398010>

<https://zhuanlan.zhihu.com/p/25434586>

> 但是单纯最小二乘法，数据量少时会产生过拟合，为了防止：1.添加数据量 2.提取特征PCA

3. 正则化

正则化求解推导: <https://zhuanlan.zhihu.com/p/95398010>

L1正则 \Rightarrow Lasso Regression

L2正则 \Rightarrow Ridge Regression

<https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>

优缺点

线性回归鉴于其简单易用（训练快），易理解（可以根据系数给出每个变量的理解和解释），所以得到了很广泛的应用，在某些场景下或许有优于其它复杂方法的表现。

但是真实模型与线性模型差距较远，则拟合差

另一方面，线性回归所能够模拟的关系其实远不止线性关系。线性回归中的“线性”指的是系数的线性，而通过对特征的非线性变换，以及广义线性模型的推广，输出和特征之间的函数关系可以是高度非线性的。另一方面，也是更为重要的一点，线性模型的易解释性使得它在物理学、经济学、商学等领域中占据了难以取代的地位。

笔试/面试题整理

- 线性回归中如何判断哪个变量对因变量有影响？要看什么值？
参数 w

- 线性回归中的置信区间

https://www.sohu.com/a/200734936_655370

<https://zhuanlan.zhihu.com/p/49221740>

- 线性回归分析中的残差 (Residuals)

首先，理解以下什么是残差，然后，然后介绍残差服从的概率分布。

残差：在回归分析中，测定值与按回归方程预测的值之差，即预测值和真实值之差。

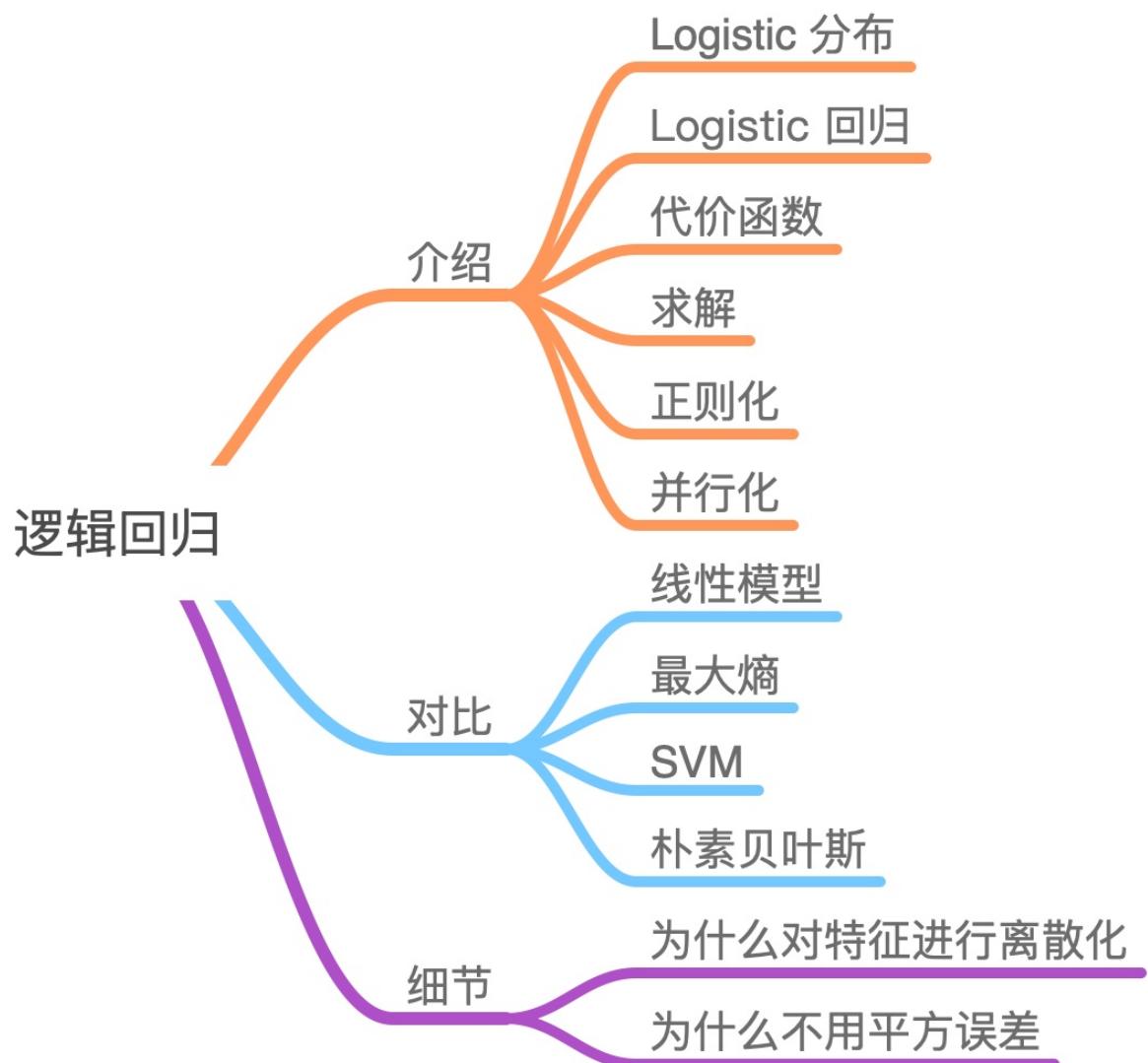
线性回归分析中，目标是残差最小化。残差平方和是关于参数的函数（可通过数学推导得到）。

为了求残差极小值，令残差关于参数的偏导数为零，会得到残差和为零（偏导=0，计算得到），即残差均值为零。

总结：线性回归中的残差服从均值（期望）为0的高斯分布（正态分布）。

- L1正则为什么可以把系数压缩成0，坐标回归的具体实现细节?
L1正则化可以实现稀疏（即截断），使训练得到的权重为0；
L1正则会产生稀疏解，即不相关的特征对应的权重为0，就相当于降低了维度。但是L1的求解复杂度要高于L2，并且L1更为流行
正则化就是对loss进行惩罚（加了正则化项之后，使loss不可能为0,lambda越大惩罚越大-->lambda较小时，约束小，可能仍存在过拟合；太大时，使loss值集中于正则化的值上）
正则化使用方法：L1/L2/L1+L2
- LR在特征较多时可以进行怎样的优化？-->L1正则有特征选择的作用
如果是离线的话，L1正则可以有稀疏解，batch大点应该也有帮助，在线的解决思路有ftrl,rds,robots,还有阿里的mlr。当然还可以用gbdt,fm,ffm做一些特性选择和组合应该也有效果。

Logistic Regression





<https://zhuanlan.zhihu.com/p/28408516>

超全总结: <https://zhuanlan.zhihu.com/p/100763009>

总结: <https://zhuanlan.zhihu.com/p/74874291>

思路(Sigmoid函数)

当我们想将线性回归应用到分类问题，考虑到 $\omega^T x + b$ 取值是连续的，因此它不能拟合离散变量（比如二分类问题，将X对应的y分为类别0和类别1），那连续的结果能拟合成什么呢，概率是连续的。所以我们可以找到一个联系函数（在原来的线性方程的基础上，加一层激活函数），将X映射到 $y \in \{0, 1\}$ 。单位跃迁函数可以满足这个要求，

$$p(y = 1|x) = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases}, \quad z = \omega^T x + b$$

为了能准确求出一个最优参数，我们需要的激活函数必须可微，也就是说是连续的，对数几率函数是一个常用的替代函数：

$$y = \frac{1}{1 + e^{-(\omega^T x + b)}}$$

这一层激活函数，它找到了分类概率 $P(Y=1)$ 与输入向量 x 的直接关系，然后通过比较概率值来判断类别。这个过程类似感知机。

也就是说，输出 $Y=1$ 的对数几率是由输入 x 的线性函数表示的模型，这就是逻辑回归模型。当 $w^T x + b$ 的值越接近正无穷， $P(Y=1|x)$ 概率值也就越接近 1。因此逻辑回归的思路是，先拟合决策边界(不局限于线性，还可以是多项式)，再建立这个边界与分类的概率联系，从而得到了二分类情况下的概率。

通过上述推导我们可以看到 Logistic 回归实际上是使用线性回归模型的预测值逼近分类任务真实标记的对数几率，其优点有：

直接对分类的概率建模，无需实现假设数据分布，从而避免了假设分布不准确带来的问题；

不仅可预测出类别，还能得到该预测的概率，这对一些利用概率辅助决策的任务很有用；

对数几率函数是任意阶可导的凸函数，有许多数值优化算法都可以求出最优解。

选择0.5作为阈值是一个一般的做法，实际应用时特定的情况可以选择不同阈值，如果对正例的判别准确性要求高，可以选择阈值大一些，对正例的召回要求高，则可以选择阈值小一些。

参数求解就是求 w

这个参数决定了决策边界 Decision Boundary (决策面): 线性决策面 ($-3 + x_1 + x_2 = 0$

) , 非线性决策面 ($-1 + x_1^2 + x_2^2 = 0$)

在逻辑回归中, 决策边界由 $\theta^T x = 0$ 定义。

与线性回归不同的是, 逻辑回归由于其联系函数的选择, 它的参数估计方法不再使用最小二乘法, 而是极大似然法 (Maximum Likelihood Method) 或交叉熵法。最小二乘法是最小化预测和实际之间的欧氏距离, 极大似然法的思想也是如出一辙的, 但是它是通过最大化预测属于实际的概率来最小化预测和实际之间的“距离”。详细推导涉及凸优化理论, 梯度下降法, 牛顿法等。

Cost Function

<https://zhuanlan.zhihu.com/p/38853901>

极大似然法 -- 二分类

因为在二分类问题中, 标签不是1就是0, 我们使用伯努利分布, 即给定 x 后 y 符合伯努利分布, 公式

$$P(y_i | \mathbf{x}_i) = p^{y_i} (1 - p)^{1-y_i}$$

我们采集到了一个样本 (\mathbf{x}_i, y_i) 。对这个样本, 它的标签是 y_i 的概率是

$p^{y_i} (1 - p)^{1-y_i}$ 。 (当 $y=1$, 结果是 p ; 当 $y=0$, 结果是 $1-p$)。

如果我们采集到了一组数据一共 N 个,

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3) \dots (\mathbf{x}_N, y_N)\}$, 这个合成在一起的合事件发生的总概率怎么求呢? 其实就是将每一个样本发生的概率相乘就可以了, 即采集到这组样本的概率:

$$P_{\text{总}} = P(y_1 | \mathbf{x}_1) P(y_2 | \mathbf{x}_2) P(y_3 | \mathbf{x}_3) \dots P(y_N | \mathbf{x}_N)$$

$$= \prod_{n=1}^N p^{y_n} (1 - p)^{1-y_n}$$

$P_{\text{总}}$ 是一个函数, 并且未知的量只有 w (在 p 里面)。

由于连乘很复杂, 我们通过两边取对数来把连乘变成连加的形式, 即:

$$\begin{aligned}
F(\mathbf{w}) &= \ln(P_{\text{总}}) = \ln\left(\prod_{n=1}^N p^{y_n} (1-p)^{1-y_n}\right) \\
&= \sum_{n=1}^N \ln(p^{y_n} (1-p)^{1-y_n}) \\
&= \sum_{n=1}^N (y_n \ln(p) + (1-y_n) \ln(1-p))
\end{aligned}$$

$$p = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

其中，

这个函数 $F(\mathbf{w})$ 又叫做它的损失函数。损失函数可以理解成衡量我们当前的模型的输出结果，跟实际的输出结果之间的差距的一种函数。这里的损失函数的值等于事件发生的总概率，我们希望它越大越好。但是跟损失的含义有点儿违背，因此也可以在前面取个负号。

最大似然估计，就是让这个cost最小， $F(\mathbf{w})$ 最大，即观察到这些数据的概率最大

$$\mathbf{w}^* = \arg \max_w F(\mathbf{w}) = -\arg \min_w F(\mathbf{w})$$

我们使用梯度下降法来推导

$$\begin{aligned}
\nabla F(\mathbf{w}) &= \nabla \left(\sum_{n=1}^N (y_n \ln(p) + (1-y_n) \ln(1-p)) \right) \\
&= \sum (y_n \ln'(p) + (1-y_n) \ln'(1-p)) \\
&= \sum \left((y_n \frac{1}{p} p') + (1-y_n) \frac{1}{1-p} (1-p)' \right) \\
&= \sum (y_n (1-p) \mathbf{x}_n - (1-y_n) p \mathbf{x}_n) \\
&= \sum_{n=1}^N (y_n - p) \mathbf{x}_n
\end{aligned}$$

它是如此简洁优雅，这就是我们选取sigmoid函数的原因之一。当然我们也能够把p再展开，即：

$$\nabla F(\mathbf{w}) = \sum_{n=1}^N \left(y_n - \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}_n}} \right) \mathbf{x}_n$$

梯度下降中的梯度指的是代价函数对各个参数的偏导数，偏导数的方向决定了在学习过程中参数下降的方向，学习率（通常用 α 表示）决定了每步变化的步长，有了导数和学习率就可以使用梯度下降算法（Gradient Descent Algorithm）更新参数了，即求解使 $J(\theta)$ 最小的参数 θ ：

$$\theta_j = \theta_j - \alpha \left(\frac{\partial}{\partial \theta_j} J(\theta) \right) = \theta_j - \alpha \left(\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right)$$

梯度下降法(Gradient Descent)，可以用来解决这个问题。核心思想就是先随便初始化一个 \mathbf{w}_0 ，然后给定一个步长 η ，通过不断地修改 $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$ ，从而最后靠近到达取得最大值的点，即不断进行下面的迭代过程，直到达到指定次数，或者梯度等于0为止。

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \nabla F(\mathbf{w})$$

PS.

逻辑回归的损失函数L是一个连续的凸函数（conveniently convex）。这样的函数的特征是，它只会有一个全局最优的点，不存在局部最优。对于GD跟SGD最大的潜在问题就是它们可能会陷入局部最优。然而这个问题在逻辑回归里面就不存在了，因为它的损失函数的良好特性，导致它不会有好几个局部最优。当我们的GD跟SGD收敛以后，我们得到的极值点一定就是全局最优的点，因此我们可以放心地用GD跟SGD来求解。

<https://zhuanlan.zhihu.com/p/44591359>

交叉熵法

最大似然估计所依据的伯努利分布只对于二分类，如果我们想要分析多分类问题，可以引入交叉熵

同时sigmoid函数只能将y值映射到0, 1之间，同样不能代表多分类问题，这时我们将sigmoid函数generalize到softmax函数（sigmoid函数的一般形式）

！记住二分类的交叉熵cost function与最大似然估计一样，同理，二分类的softmax与sigmoid相同

在Logistic regression二分类问题中，我们可以使用sigmoid函数将输入映射到[0, 1]区间中，从而得到属于某个类别的概率。将这个问题进行泛化，推广到多分类问题中，我们可以使用softmax函数，对输出的值归一化为概率值。

为了解决多分类问题，我们训练K个独立的线性模型，每个模型的结果在一起为一个向量，Softmax把分类输出的向量标准化成概率分布，cross-entropy（交叉熵）刻画预测分类和真实结果之间的相似度。

softmax函数公式为：

$$\text{softmax}(x_i) = e^{x_i} / \sum_j e^{x_j}$$

Softmax回归处理多分类问题，我们假设函数 $h_\theta(x)$ 形式如下：

$$h_{\theta}(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)}; \theta) \\ p(y^{(i)} = 2|x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

其中 $\theta_1, \theta_2, \dots, \theta_k \in \Re^{n+1}$ 是模型的参数。请注意 $\frac{1}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}}$ 这一项对概率分布进行归一化，使得所有概率之和为 1。

其中对于每个分类：

$$P(Y_i = c) = \frac{e^{\mathbf{w}_c^T \mathbf{x}_i}}{\sum_{k=1}^K e^{\mathbf{w}_k^T \mathbf{x}_i}}, c = 1, \dots, K$$

一般将参数 θ_j 写出矩阵形式：

$$\theta = \begin{bmatrix} | & | & | & | \\ \theta_1 & \theta_2 & \cdots & \theta_K \\ | & | & | & | \end{bmatrix}$$

其中，用softmax函数得到每个分类对应的概率之后，分类结果为 $\text{argmax}(y)$ ，即概率最大的分类结果对应的种类。

我们需要损失函数返回预测结果与真实结果之间的差距，并进行梯度下降进行参数更新。实际上，softmax回归更多地是用在神经网络输出层的后面，得到损失函数后进行反向传播更新。

此时的损失函数为交叉熵

$$\text{CrossEntropy} = - \sum_{i=1}^m y_i \log(p_i)$$

其中 p_i 是softmax回归求得的结果， y_i 是实际的类别。可以看出，交叉熵损失函数只计算真实类别对应的预测概率的损失，而不考虑其他的预测概率损失（非真实分类对应的 y_i 为0）。

对比二分类的logistic回归和多分类的softmax回归，可以发现，logistic回归是softmax回归的特例，即只有两种情况时，分类概率为 p 和 $1 - p$ 。

然后计算损失函数的偏导函数，得到：

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^m [x^{(i)} (\mathbb{1}\{y^{(i)} = j\} - p(y^{(i)} = j | x^{(i)}; \theta))]$$

初始化参数 θ 之后，我们重复：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

接下来 $J(\theta)$ 对 θ_j 求偏导，上文已经知晓sigmoid函数具有很好的求导特性，这里：

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$h'_\theta(x) = h_\theta(x)(1 - h_\theta(x))$$

又有：

$$(log x)' = \frac{1}{x}$$

故：

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{\partial}{\partial \theta_j} \frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \\ &= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \frac{1}{h_\theta(x^{(i)})} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))(-x_j^{(i)}) + (1 - y^{(i)}) \frac{1}{1 - h_\theta(x^{(i)})} (1 - h_\theta(x^{(i)}))h_\theta(x^{(i)})x_j^{(i)}] \\ &= -\frac{1}{m} \sum_{i=1}^m [-y^{(i)}(1 - h_\theta(x^{(i)}))x_j^{(i)} + (1 - y^{(i)})h_\theta(x^{(i)})x_j^{(i)}] \\ &= \frac{1}{m} \sum_{i=1}^m [y^{(i)} - y^{(i)}h_\theta(x^{(i)}) - h_\theta(x^{(i)}) + y^{(i)}h_\theta(x^{(i)})]x_j^{(i)} \\ &= \frac{1}{m} \sum_{i=1}^m [y^{(i)} - h_\theta(x^{(i)})]x_j^{(i)} \end{aligned}$$

所以参数更新：

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m [y^{(i)} - h_\theta(x^{(i)})]x_j^{(i)}$$

极大似然与交叉熵

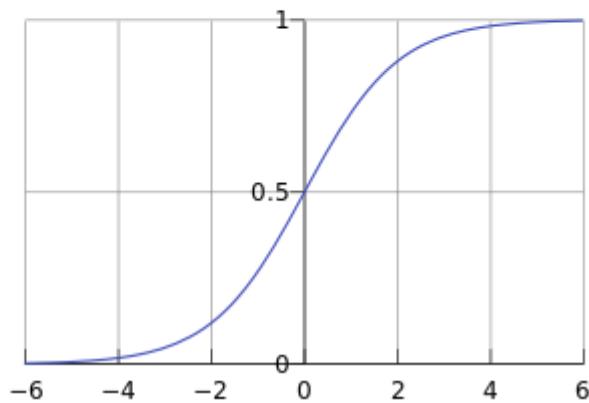
https://blog.csdn.net/diligent_321/article/details/53115369

Sigmoid / Softmax

Sigmoid函数(逻辑函数) 二分类

- $$g(z) = \frac{1}{1 + e^{-z}}$$

其函数曲线如下：



sigmoid函数是一个S形的曲线，它的取值在[0, 1]之间，在远离0的地方函数的值会很快接近0或者1。它的这个特性对于解决二分类问题十分重要。

<https://blog.csdn.net/zhihengqianjun/article/details/75303820>

Softmax函数 多分类

在Logistic regression二分类问题中，我们可以使用sigmoid函数将输入映射到[0, 1]区间中，从而得到属于某个类别的概率。将这个问题进行泛化，推广到多分类问题中，我们可以使用softmax函数，对输出的值归一化为概率值。

softmax把分类输出标准化成概率分布，cross-entropy (交叉熵) 刻画预测分类和真实结果之间的相似度。

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

<https://zhuanlan.zhihu.com/p/27223959>

重要：https://blog.csdn.net/Daniel_djf/article/details/41901063

<https://blog.nex3z.com/2017/05/02/sigmoid-%E5%87%BD%E6%95%B0%E5%92%8C-softmax-%E5%87%BD%E6%95%B0%E7%9A%84%E5%8C%BA%E5%88%AB%E5%92%8C%E5%85%B3%E7%B3%BB/>

Sigmoid 和 Softmax 是在逻辑回归和神经网络中常用的两个函数，初学时经常会对二者的差异和应用场景产生疑惑。

$$S(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid 函数形式为：

Sigmoid 是一个可微的有界函数，在各点均有非负的导数。当 $x \rightarrow \infty$ 时， $S(x) \rightarrow 1$ ；当 $x \rightarrow -\infty$ 时， $S(x) \rightarrow 0$ 。常用于二元分类 (Binary Classification) 问题，以及神经网络的激活函数 (Activation Function)（把线性的输入转换为非线性的输出）。

Softmax 函数形式为：

$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}, j = 1, 2, \dots, K$$

对于一个长度为 K 的任意实数矢量，Softmax 可以把它压缩为一个长度为 K 的、取值在 $(0, 1)$ 区间的实数矢量，且矢量中各元素之和为 1。它在多元分类 (Multiclass Classification) 和神经网络中也有很多应用。Softmax 不同于普通的 max 函数：max 函数只输出最大的那个值，而 Softmax 则确保较小的值也有较小的概率，不会被直接舍弃掉，是一个比较“Soft”的“max”。

在二元分类的情况下，对于 Sigmoid，有：

$$p(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$p(y = 0|x) = 1 - p(y = 1|x) = \frac{e^{-\theta^T x}}{1 + e^{-\theta^T x}}$$

而对 $K=2$ 的 Softmax，有：

$$p(y = 1|x) = \frac{e^{\theta_1^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{1}{1 + e^{(\theta_0^T - \theta_1^T)x}} = \frac{1}{1 + e^{-\beta x}}$$

$$p(y = 0|x) = \frac{e^{\theta_0^T x}}{e^{\theta_0^T x} + e^{\theta_1^T x}} = \frac{e^{(\theta_0^T - \theta_1^T)x}}{1 + e^{(\theta_0^T - \theta_1^T)x}} = \frac{e^{-\beta x}}{1 + e^{-\beta x}}$$

其中

$$\beta = -(\theta_0^T - \theta_1^T)$$

可见在二元分类的情况下，Softmax 退化为了 Sigmoid。

正则化处理过拟合

过拟合 (Overfitting) 是机器学习中常见的问题，模型在训练集上拟合过度导致不能泛化到测试集。如果一个模型过拟合，我们说它具有高方差(high variance)。同样的，模型也有可能欠拟合(Underfitting)，这意味着我们的模型甚至不能拟合训练集，更不用说去泛化测试集，它具有高偏差(high bias)。

这里我们讨论简单些的正则化，L2正则：

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

其中 $i = 1, 2, 3 \dots m$ 为 m 个样本， $j = 1, 2, 3 \dots n$ 为 n 个特征。 λ 为正则化参数，除以2是为了求导可以约去。

因为我们未对 θ_0 正则化，L2正则化后的梯度下降参数更新分两种情况：

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \\ \theta_j &:= \theta_j - \alpha \frac{1}{m} \left[\sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} + \lambda \theta_j \right]\end{aligned}$$

补充知识

激活函数都有什么，他们有什么特点？

<https://www.zhihu.com/question/67366051>

Sigmoid存在的问题：梯度消失、其输出不是关于原点中心对称的（训练数据不关于原点对称时，收敛速度非常慢à输入中心对称，得到的输出中心对称时，收敛速度会非常快）、计算耗时

Tanh激活函数存在的问题：梯度消失、计算耗时，但是其输出是中心对称的

$$\text{lg}(x) = \frac{1}{1 + e^{-x}} \quad \text{and} \quad \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}}.$$

ReLU：其输出不关于原点对称；反向传播时，输入神经元小于0时，会有梯度消失问题；当 $x=0$ 时，该点梯度不存在（未定义）；

ReLU失活 (dead RELU) 原因：权重初始化不当、初始学习率设置的非常大

Maxout：根据设置的 K 值，相应的增大了神经元的参数个数

Xavier权重初始化方法：对每个神经元的输入开根号

Name	Plot	Equation	Derivative (with respect to x)	Range
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	{0, 1}
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}^{[1]}$	$f'(x) = f(x)(1 - f(x))$	(0, 1)
Tanh		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	(-1, 1)
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$(-\frac{\pi}{2}, \frac{\pi}{2})$
ElliotSig ^{[9][10][11]} Softsign ^{[12][13]}		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$	(-1, 1)
Inverse square root unit (ISRU) ^[14]		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	$f'(x) = \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3$	$(-\frac{1}{\sqrt{\alpha}}, \frac{1}{\sqrt{\alpha}})$
Inverse square root linear unit (ISRLU) ^[14]		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \left(\frac{1}{\sqrt{1 + \alpha x^2}}\right)^3 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\frac{1}{\sqrt{\alpha}}, \infty)$
Square Nonlinearity (SQNL) ^[11]		$f(x) = \begin{cases} 1 & : x > 2.0 \\ x - \frac{x^2}{4} & : 0 \leq x \leq 2.0 \\ x + \frac{x^2}{4} & : -2.0 \leq x < 0 \\ -1 & : x < -2.0 \end{cases}$	$f'(x) = 1 \mp \frac{x}{2}$	(-1, 1)
Rectified linear unit (ReLU) ^[15]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$
Bipolar rectified linear unit (BReLU) ^[16]		$f(x_i) = \begin{cases} \text{ReLU}(x_i) & \text{if } i \bmod 2 = 0 \\ -\text{ReLU}(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$f'(x_i) = \begin{cases} \text{ReLU}'(x_i) & \text{if } i \bmod 2 = 0 \\ -\text{ReLU}'(-x_i) & \text{if } i \bmod 2 \neq 0 \end{cases}$	$(-\infty, \infty)$
Leaky rectified linear unit (Leaky ReLU) ^[17]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Parametric rectified linear unit (PReLU) ^[18]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)^{[2]}$
Randomized leaky rectified linear unit (RReLU) ^[19]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$
Exponential linear unit (ELU) ^[20]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	知乎 (Elu)

模型的cost function

在利用深度学习模型解决有监督问题时，比如分类、回归、去噪等，我们一般的思路如下：

1. 信息流forward propagation, 直到输出端；
2. 定义损失函数 $L(x, y | \theta)$ ；
3. 误差信号back propagation。采用数学理论中的“链式法则”，求 $L(x, y | \theta)$ 关于参数 θ 的梯度；
4. 利用最优化方法（比如随机梯度下降法），进行参数更新；
5. 重复步骤3、4，直到收敛为止；

下面我们就来理清他们

在第2步中，我们通常会见到多种损失函数的定义方法，常见的有均方误差（error of mean square）、最大似然误差（maximum likelihood estimate）、最大后验概率（maximum posterior probability）、交叉熵损失函数（cross entropy loss），的区别和联系。一般地，一个机器学习模型选择哪种损失函数，是凭借经验而定的，没有什么特定的标准。具体来说，

(1) 均方误差是一种较早的损失函数定义方法，它衡量的是两个分布对应维度的差异性之和。说点题外话，与之非常接近的一种相似性度量标准“余弦角”，则衡量的是两个分布整体的相似性，也即把两个向量分别作为一个整体，计算出的夹角作为其相似性大小的判断依据，读者可以认真体会这两种相似性判断标准的差异；

(2) 最大似然误差是从概率的角度，求解出能完美拟合训练样例的模型参数theta，使得概率 $p(y | x, \theta)$ 最大化；

(3) 最大化后验概率，即使得概率 $p(\theta | x, y)$ 最大化，实际上也等价于带正则化项的最大似然概率（详细的数学推导可以参见Bishop 的Pattern Recognition And Machine Learning），它考虑了先验信息，通过对参数值的大小进行约束来防止“过拟合”；

(4) 交叉熵损失函数，衡量的是两个分布 p 、 q 的相似性。在给定集合上两个分布 p 和 q 的cross entropy定义如下：

$$H(p, q) = \mathbb{E}_p[-\log q] = H(p) + D_{KL}(p||q),$$

其中， $H(p)$ 是 p 的熵， $D_{KL}(p||q)$ 表示KL-divergence。对于离散化的分布 p 和 q ，

$$H(p, q) = - \sum p(x) \log q(x).$$

在机器学习应用中， p 一般表示样例的标签的真实分布，为确定值，故最小化交叉熵和最小化KL-divergence是等价的，只不过之间相差了一个常数。

值得一提的是，在分类问题中，交叉熵的本质就是似然函数的最大化。证明如下，

记带标签的样例为 (x, y) ，其中 x 表示输入特征向量， $y=[y_1, y_2, \dots, y_c]$ 表示真实标签的one-hot表示， $y=[y_1, y_2, \dots, y_c]$ 表示模型输出的分布， c 表示样例输出的类别数，那么，

(1) 对于二分类问题， $p(x)=[1, 0]$ ， $q(x)=[y_1, y_2]$ ， $y_1=p(y=1|x)$ 表示模型输出为真的概率，交叉熵 $H(p, q)=-(1*y_1+0*y_2)=-y_1$ ，显然此时交叉熵的最小化等价于似然函数的最大化；

(2) 对于多分类问题，假设 $p(x)=[0, 0, 0, \dots, 1, 0, 0]$ ， $q(x)=[y_1, y_2, y_3, \dots, y_k, y_{(k+1)}, y_{(k+2)}]$ ，即表示真实样例标签为第 k 类， $y_k=p(y=k|x)$ 表示模型输出为第 k 类的概率，交叉熵 $H(p, q)=-(0*y_1+0*y_2+0*y_3+\dots+1*y_k+0*y_{(k+1)}+0*y_{(k+2)})=-y_k$ ，此时同上。

优缺点

通过上述推导我们可以看到 Logistic 回归实际上是使用线性回归模型的预测值逼近分类任务真实标记的对数几率，其优点有

- 逻辑回归的可解释性很强

在模型训练完成之后，我们获得了一组n维的权重向量 w 跟偏差 b。

对于权重向量 w ，它的每一个维度的值，代表了这个维度的特征对于最终分类结果的贡献大小。假如这个维度是正，说明这个特征对于结果是有正向的贡献，那么它的值越大，说明这个特征对于分类为正起到的作用越重要。

对于偏差b (Bias)，一定程度代表了正负两个类别的判定的容易程度。假如b是0，那么正负类别是均匀的。如果b大于0，说明它更容易被分为正类，反之亦然。

根据逻辑回归里的权重向量在每个特征上面的大小，就能够对于每个特征的重要程度有一个量化的清楚的认识，这就是为什么说逻辑回归模型有着很强的解释性的原因。

- 直接对分类的概率建模，无需实现假设数据分布，从而避免了假设分布不准确带来的问题；
- 不仅可预测出类别，还能得到该预测的概率，这对一些利用概率辅助决策的任务很有用；
- 对数几率函数是任意阶可导的凸函数，有许多数值优化算法都可以求出最优解。
- 逻辑回归训练速度很快，可用于工业级别的数据，也可以在使用其他准确率更高的算法之前先用逻辑回归计算出baseline，查看下当前的数据在算法上的表现，以判断是否还要继续进行数据清洗和特征工程。可用于概率预测，也可用于分类；对于数据中小噪声的鲁棒性很好。

缺点：对数据特征间的独立性要求较高；不适用于features和label为非线性关系的数据中；当特征空间很大、特征有缺失时，逻辑回归的性能不是很好。

笔试/面试题整理

- 请简述逻辑回归与线性回归的区别与联系

区别：

线性回归假设变量服从正态分布，逻辑回归假设变量服从伯努利分布；

线性回归优化的目标是均方差函数，逻辑回归优化的目标是似然函数；

线性回归中因变量与自变量呈现线性关系，而逻辑回归并没有要求；

线性回归解决的是回归问题，取值范围是实数域，逻辑回归还可以解决分类问题，其取值范围是[0,1]；

参数估计上都是使用极大似然估计的方法估计参数，但是线性回归的损失函数是均方差，逻辑回归是交叉熵；

联系：

两者都是线性模型，线性回归属于普通线性回归模型，逻辑回归属于广义线性模型
逻辑回归是线性回归套上了一个Sigmoid函数。

逻辑回归的cost function

- LR为什么用sigmoid函数，这个函数有什么优点和缺点？为什么不用其他函数？（sigmoid是伯努利分布的指数族形式）
我们希望将y值作为概率训练，sigmoid函数正好符合概率在0, 1之间的需求
sigmoid函数求导非常简单，易求得最优参数
逻辑回归认为函数其概率服从伯努利分布，将其写成指数族分布的形式，Sigmoid函数的本质是伯努利分布的后验概率（推导：
https://blog.csdn.net/yileahcwks/article/details/84335939?utm_medium=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase
）

于是有：

$$\ln \frac{y}{1-y} = w^T x + b$$

我们将 y 视为 x 为正例的概率，则 1-y 为 x 为其反例的概率。两者的比值称为几率 (odds)，指该事件发生与不发生的概率比值，若事件发生的概率为 p。则对数几率：

$$\ln(odds) = \ln \frac{y}{1-y}$$

将 y 视为类后验概率估计，重写公式有：

$$w^T x + b = \ln \frac{P(Y=1|x)}{1-P(Y=1|x)}$$
$$P(Y=1|x) = \frac{1}{1+e^{-(w^T x+b)}} \quad \text{知乎 @ashergaga}$$

也就是说，输出 Y=1 的对数几率是由输入 x 的线性函数表示的模型，这就是逻辑回归模型。

- LR服从什么分布？
线性回归假设响应变量服从正态分布，逻辑回归假设响应变量服从伯努利分布
线性回归要求自变量与因变量呈线性关系，而逻辑回归没有要求
伯努利分布，只有0, 1两种结果，所以是在二分类问题中会用到，也就是之前提到的分类问题；

而高斯分布，是我们在求解最小化损失函数的时候，当时用最小二乘法表示，是因为假设误差函数满足高斯分布的时候的最大似然函数中部分的 log 形式

之前有点疑惑，感觉有点混淆模型和分布的概念

现在感觉，分布是对于结果集，或者是训练集的一个描述，应该是根据分布的情况选取相应的模型

- 交叉熵与最大似然估计

我理解为从不同角度，但是最终结果相同

最大似然是多个概率相乘，把这个连乘取对数，就会得到[真实值(0,1)乘log]的连加，如果是多分类问题，可以理解成[0,0,1,0,0]，即只取第三个y值，然后再连家
交叉熵的公式正巧就是这样的

- 什么是代价函数？怎么选择的Loss等等问题（为什么是它？）

逻辑回归模型的数学形式确定后，剩下就是如何去求解模型中的参数。一般求参数就是代价函数最低的参数。

概况来讲，任何能够衡量模型预测出来的值 $h(\theta) = w^T x$ 与真实值 y 之间的差异的函数都可以叫做代价函数 $C(\theta)$ ，如果有多个样本，则可以将所有代价函数的取值求均值，记做 $J(\theta)$ 。因此很容易就可以得出以下关于代价函数的性质：

- 选择代价函数时，最好挑选对参数 θ 可微的函数（全微分存在，偏导数一定存在）
- 对于每种算法来说，代价函数不是唯一的；
- 代价函数是参数 θ 的函数；
- 总的代价函数 $J(\theta)$ 可以用来评价模型的好坏，代价函数越小说明模型和参数越符合训练样本 (x, y) ；
- $J(\theta)$ 是一个标量；

在优化参数 θ 的过程中，最常用的方法是梯度下降，这里的梯度就是代价函数 $J(\theta)$ 对 $\theta_1, \theta_2, \dots, \theta_n$ 的偏导数。由于需要求偏导，我们可以得到另一个关于代价函数的性质：
：选择代价函数时，最好挑选对参数 θ 可微的函数（全微分存在，偏导数一定存在）

代价函数的常见形式

经过上面的描述，一个好的代价函数需要满足两个最基本的要求：能够评价模型的准确性，对参数 θ 可微。

<1>. 在线性回归中，最常用的是均方误差(Mean squared error)，即

- $$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$
- m ：训练样本的个数；
- $h_\theta(x)$ ：用参数 θ 和 x 预测出来的 y 值；
- y ：原训练样本中的 y 值，也就是标准答案；
- 上角标 (i) ：第 i 个样本。

<2>. 在逻辑回归中，最常用的是代价函数是交叉熵(Cross Entropy)，交叉熵是一个常见的代价函数，在神经网络中也会用到。下面是《神经网络与深度学习》一书对交叉熵的解释：

交叉熵是对「出乎意料」（译者注：原文使用surprise）的度量。神经元的目标是去计算函数 $x \rightarrow y = y(x)$ 。但是我们让它取而代之计算函数 $x \rightarrow a = a(x)$ 。假设我们把 a 当作 y 等于 1 的概率， $1-a$ 是 y 等于 0 的概率。那么，交叉熵衡量的是我们在知道 y 的真实值时的平均「出乎意料」程度。当输出是我们期望的值，我们的「出乎意料」程度比较低；当输出不是我们期望的，我们的「出乎意料」程度就比较高。

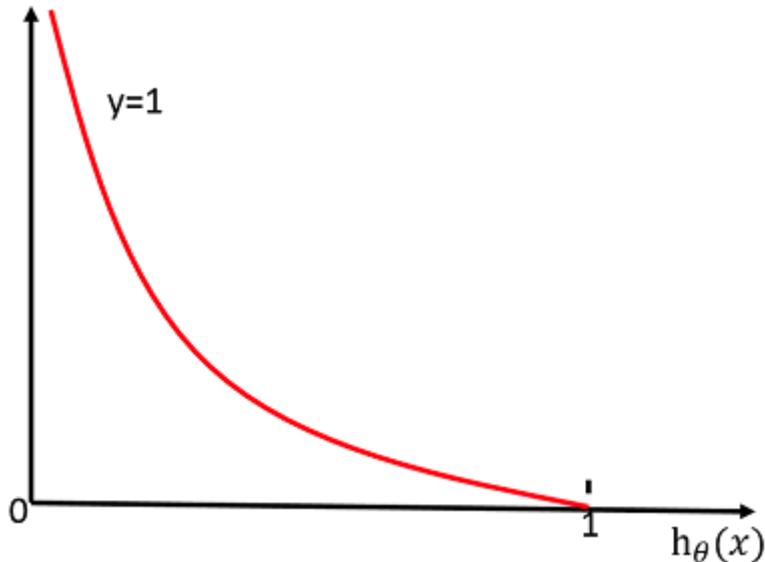
$$C(\theta) = y \log h_\theta(x) + (1 - y) \log(1 - h_\theta(x))$$

由于我们只有两分类，所以 y 只有0, 1两个结果，所以 $C(\theta)$ 等价于

$$C(\theta) = \begin{cases} -\log(h_\theta(x)), & y=1 \\ -\log(1-h_\theta(x)), & y=0 \end{cases}, \text{ where } h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

当 $y = 1$ 时，

$$C(\theta) = -\log(h_\theta(x))$$

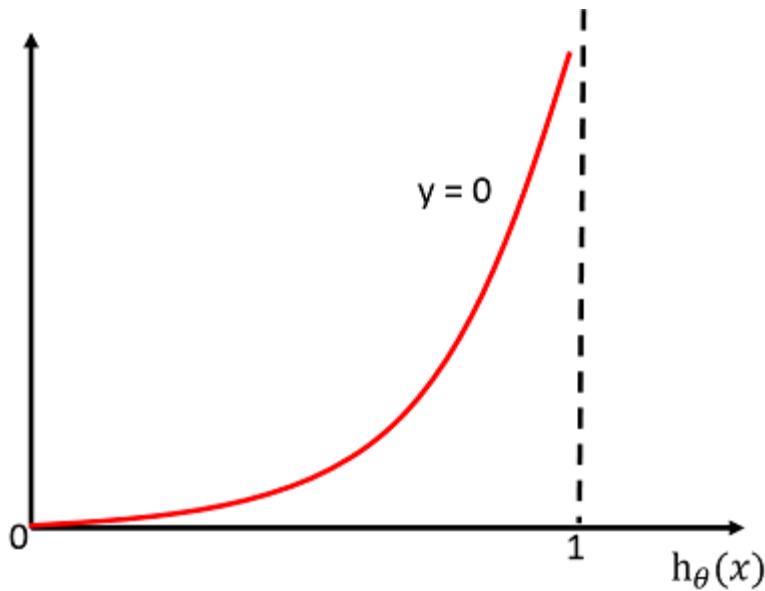


从图中可以看出， $y = 1$ ，当预测值 $h_\theta(x) = 1$ 时，可以看出代价

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ 函数 $C(\theta)$ 的值为0，这正是我们希望的。

如果预测值 $h_\theta(x) = 1$ 即 $P(y = 1|x; \theta) = 0$ ，意思是预测 $y = 1$ 的概率为0，但是事实上 $y = 1$ ，因此代价函数 $C(\theta) = \infty$ 相当于给学习算法一个惩罚。

同理，我们也可以画出当 $y = 0$ 时，函数 $C(\theta)$ 的图像：



$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m (y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

交叉熵作为损失函数还有一个好处是使用sigmoid函数在梯度下降时能避免均方误差损失函数学习速率降低的问题，因为学习速率可以被输出的误差所控制。

p:真实样本分布，服从参数为p的0-1分布，即 $X \sim B(1, p)$

q:待估计的模型，服从参数为q的0-1分布，即 $X \sim B(1, q)$

两者的交叉熵为：

$$CEH(p, q)$$

$$= - \sum_{x \in \mathcal{X}} \mathbf{p}(\mathbf{x}) \log \mathbf{q}(\mathbf{x})$$

$$= -[P_p(x=1) \log P_q(x=1) + P_p(x=0) \log P_q(x=0)]$$

$$= -[p \log q + (1-p) \log(1-q)]$$

$$= -[\mathbf{y} \log \mathbf{h}_\theta(\mathbf{x}) + (1-\mathbf{y}) \log(1 - \mathbf{h}_\theta(\mathbf{x}))]$$

对所有训练样本取均值得：

$$-\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

这个结果与通过最大似然估计方法求出来的结果一致。

交叉熵作为损失函数还有一个好处是使用sigmoid函数在梯度下降时能避免均方误差损失函数学习速率降低的问题，因为学习速率可以被输出的误差所控制。

- 交叉熵和最大似然估计，作为cost function的区别？

给定一堆数据，假如我们知道它是从某一种分布中随机取出来的，可是我们并不知道这个分布具体的参数，即“模型已定，参数未知”。例如，我们知道这个分布是正态分布，但是不知道均值和方差；或者是二项分布，但是不知道均值。最大似然估计（MLE, Maximum Likelihood

Estimation）就可以用来估计模型的参数。MLE的目标是找出一组参数，使得模型产生出观测数据的概率最大。

我们知道每次抛硬币都是一次二项分布，设正面朝上的概率是，那么似然函数为：

$$p(\mathbf{X}; \mu) = \prod_{i=1}^n p(x_i; \mu) = \prod_{i=1}^n \mu^{x_i} (1 - \mu)^{1-x_i}$$

为了求导方便，一般对目标取log。所以最优化对似然函数等同于最优化对数似然函数：

$$p(\mathbf{X}; \mu) = \prod_{i=1}^n p(x_i; \mu) = \prod_{i=1}^n \mu^{x_i} (1 - \mu)^{1-x_i}$$

对于二分类，利用极大似然估计求解估计中利用对数求解，最后与交叉熵形式与意义不谋而合。

- 逻辑回归的输入特征为离散量，为什么？

在工业界，很少直接将连续值作为特征喂给逻辑回归模型，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：

- 稀疏向量内积乘法运算速度快，计算结果方便存储，容易scalable（扩展）。
- 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。
- 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。
- 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。
- 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。

李沐少帅指出，模型是使用离散特征还是连续特征，其实是一个“海量离散特征+简单模型”同“少量连续特征+复杂模型”的权衡。既可以离散化用线性模型，也可以用连续特征加深度学习。就看是喜欢折腾特征还是折腾模型了。通常来说，前者容易，而且可以n个人一起并行做，有成功经验；后者目前看很赞，能走多远还须拭目以待。

大概的理解：

- 1) 计算简单
- 2) 简化模型
- 3) 增强模型的泛化能力，不易受噪声的影响

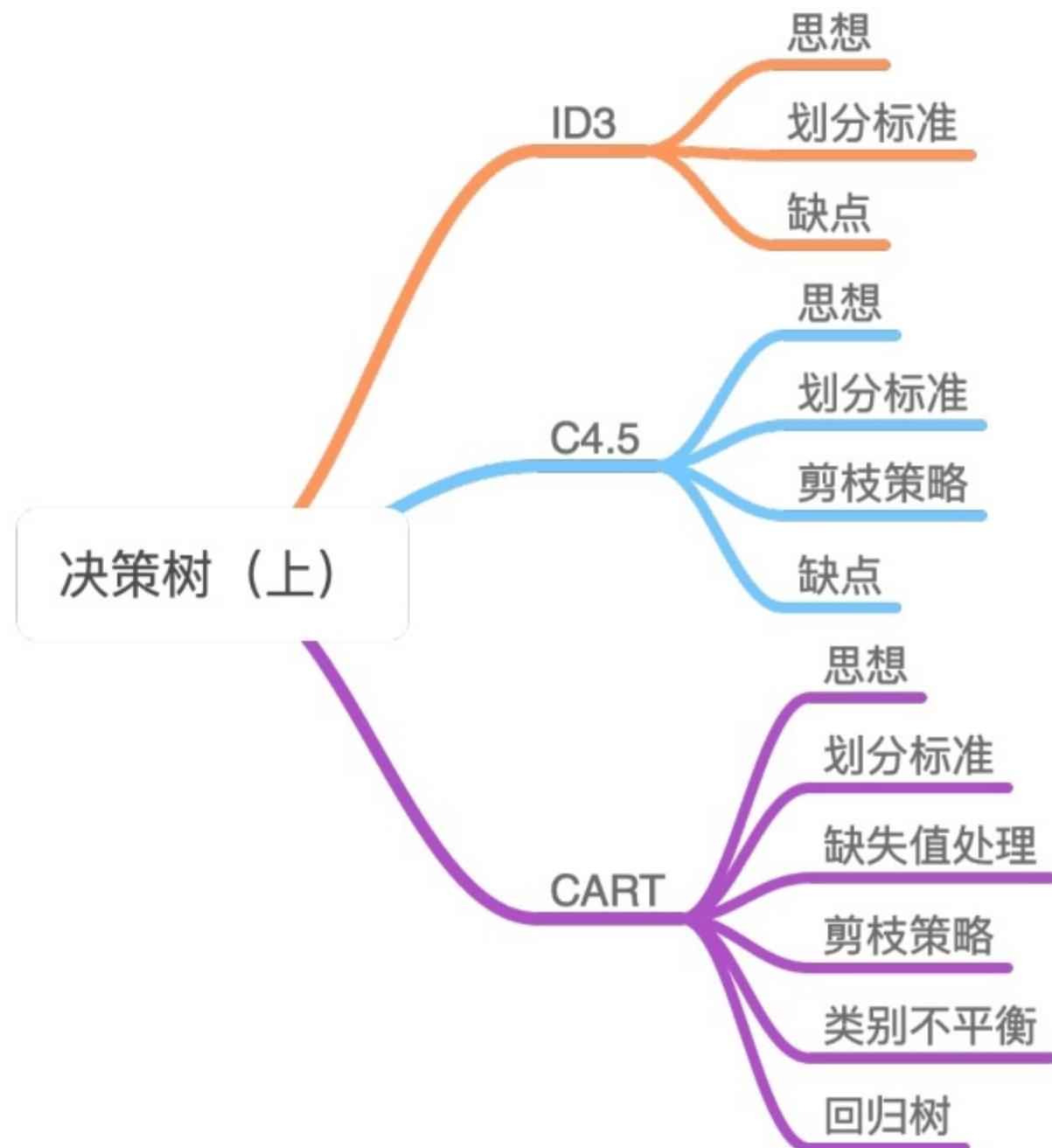
KNN

Tree-Based Algorithms

Decision Tree

三种决策树的回归原理

算法	支持模型	树结构	特征选择	连续值处理	缺失值处理	剪枝
ID3	分类	多叉树	信息增益	不支持	不支持	不支持
C4.5	分类	多叉树	信息增益比	支持	支持	支持
CART	分类, 回归	二叉树	基尼系数, 均方差	支持	支持	支持



<https://zhuanlan.zhihu.com/p/48274203>

<https://zhuanlan.zhihu.com/p/85731206>

ID3决策树

思路

建立在奥卡姆的剃刀的基础上，同时越小型的树，越稳定。所以在表现相似的情况下，我们选择更简单的树

从信息论中的知识：信息熵越大，从而样本纯度越低。ID3的核心思想就是以信息增益来度量特征选择，即选择信息增益最大的特征进行分裂。算法采用自上而下的贪婪搜索遍历可能的决策树空间。

其大致步骤为：

1. 初始化特征集合和数据集合；
2. 计算数据集合信息熵和所有特征的条件熵，选择信息增益最大的特征作为当前决策节点；
3. 更新数据集合和特征集合（删除上一步使用的特征，并按照特征值来划分不同分支的数据集合）；
4. 重复 2, 3 两步，若子集值包含单一特征，则为分支叶子节点。

信息增益表示得知特征A的信息，使得样本不确定性减少的程度。

参数计算

数据集的信息熵

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

针对某个特征 A，对于数据集 D 的条件熵 $H(D|A)$ 为：

$$\begin{aligned} H(D|A) &= \sum_{i=1}^n \frac{|D_i|}{|D|} H(D_i) \\ &= - \sum_{i=1}^n \frac{|D_i|}{|D|} \left(\sum_{k=1}^K \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|} \right) \end{aligned}$$

其中 D_i 表示 D 中特征 A 取第 i 个值的样本子集， D_{ik} 表示 D_i 中属于第 k 类的样本子集。

信息增益 = 信息熵 - 条件熵：

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

信息增益越大表示使用特征 A 来划分所获得的“纯度提升越大”。

优缺点

1. ID3 没有剪枝策略，容易过拟合；

2. 信息增益准则对可取值数目较多的特征有所偏好，类似“编号”的特征其信息增益接近于1；---> C4.5树（可解决所有缺点）
3. 只能用于处理离散分布的特征；
4. 没有考虑缺失值。

C4.5回归树 信息增益比

思路

C4.5 算法最大的特点是克服了 ID3 对特征数目的偏重这一缺点，引入信息增益率来作为分类标准。

C4.5 相对于 ID3 的缺点对应有以下改进方式：

- 引入悲观剪枝策略进行后剪枝；
- 引入信息增益率作为划分标准；
- 将连续特征离散化，假设n个样本的连续特征A有m个取值，C4.5将其排序并取相邻两样本值的平均数共m-1个划分点，分别计算以该划分点作为二元分类点时的信息增益，并选择信息增益最大的点作为该连续特征的二元离散分类点；
- 对于缺失值的处理可以分为两个子问题：

问题一：在特征值缺失的情况下进行划分特征的选择？（即如何计算特征的信息增益率）

问题二：选定该划分特征，对于缺失该特征值的样本如何处理？（即到底把这个样本划分到哪个结点里）

第一步，计算所有特征的信息增益或者信息增益率的时候，假设数据集一共10000个样本，特征A中缺失了5000个，则无视缺失值，在剩下的5000个特征中计算信息增益（或者信息增益率），最后乘以0.5，思想就是缺失值多的特征通过这种降低权重的方式来体现信息的缺失；

第二部，如果运气不好，正好这个A特征乘0.5之后得到的信息增益或者增益率还是最大的，那么就像西瓜书中提到的那样，存在缺失值的样板按照比例进入分裂之后的新的分支，假设根据特征A分裂得到两个新的分支，一个分支有2000个样本，一个分支有3000个样本，则按照比例2000个缺失值和3000个缺失值样本分别进入两个分支。（实际上ID3我没有明确去查是否采用了这种处理方法，c4.5是采用了这种，不过问题不大，应用在ID3上也没啥毛病。。。另外原始ID3无法处理连续值和缺失值）

参数推导

利用信息增益率可以克服信息增益Entropy bias的缺点，其公式为

$$Gain_{ratio}(D, A) = \frac{Gain(D, A)}{H_A(D)}$$

$$H_A(D) = - \sum_{i=1}^n \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$$

$H_A(D)$ 称为特征A的固有值。

这里需要注意，信息增益率对可取值较少的特征有所偏好（分母越小，整体越大），因此C4.5并不是直接用增益率最大的特征进行划分，而是使用一个启发式方法：先从候选划分特征中找到信息增益高于平均值的特征，再从中选择增益率最高的。

剪枝策略

预剪枝：

停止分裂的时机：

- 节点内数据样本低于某一阈值；
- 所有节点特征都已分裂；
- 节点划分前准确率比划分后准确率高。

预剪枝不仅可以降低过拟合的风险而且还可以减少训练时间，但另一方面它是基于“贪心”策略，会带来欠拟合风险。

后剪枝：

在已经生成的决策树上进行剪枝，从而得到简化版的剪枝决策树。

C4.5 采用的悲观剪枝方法，用递归的方式从低往上针对每一个非叶子节点，评估用一个最佳叶子节点去代替这棵子树是否有益。如果剪枝后与剪枝前相比其错误率是保持或者下降，则这棵树就可以被替换掉。**C4.5 通过训练数据集上的错误分类数量来估算未知样本上的错误率。**

悲观剪枝：<https://www.cnblogs.com/zhangchaoyang/articles/2842490.html>

后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。但同时其训练时间会大的多。

优缺点

缺点：

剪枝策略可以再优化；

- C4.5 用的是多叉树，用二叉树效率更高；
- C4.5 只能用于分类；
- C4.5 使用的熵模型拥有大量耗时的对数运算，连续值还有排序运算；
- C4.5 在构造树的过程中，对数值属性值需要按照其大小进行排序，从中选择一个分割点，所以只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时，程序无法运行。

CART树

Classification and regression tree

ID3 和 C4.5 虽然在对训练样本集的学习中可以尽可能多地挖掘信息，但是其生成的决策树分支、规模都比较大，CART 算法的二分法可以简化决策树的规模，提高生成决策树的效率。

思路

CART 包含的基本过程有分裂，剪枝和树选择。

- 分裂：分裂过程是一个二叉递归划分过程，其输入和预测特征既可以是连续型的也可以是离散型的，CART 没有停止准则，会一直生长下去；
CART 算法的基本步骤可以简述如下。首先，遍历当前数据集中所有特征的所有取值，寻找一个最优的变量-取值组合作为分裂点（选取规则请见下节介绍），划分出两个特征空间。然后，分别对位于不同空间的数据子集重复前述过程，直至满足某终止条件（例如：当前空间内的样本量低于特定阈值，或当前空间分裂后某子空间中的样本量低于特定阈值，或分裂已达一定深度等）。
划分举例：<https://blog.csdn.net/ACdreamers/article/details/44664481>
- 剪枝：采用代价复杂度剪枝，从最大树开始，每次选择训练数据熵对整体性能贡献最小的那个分裂节点作为下一个剪枝对象，直到只剩下根节点。CART 会产生一系列嵌套的剪枝树，可以证明，使式某个 α 中取得最小值的子树就位于其中。因此，最后只需通过比较该系列子树的值，即可得到目标子树。需要从中选出一颗最优的决策树；
- 树选择：用单独的测试集评估每棵剪枝树的预测性能（也可以用交叉验证）。超参数的最优值可以通过交叉验证等重抽样方法确定。
- CART 在 C4.5 的基础上进行了很多提升。
 1. C4.5 为多叉树，运算速度慢，CART 为二叉树，运算速度快；
 2. C4.5 只能分类，CART 既可以分类也可以回归；
 3. CART 使用 Gini 系数作为变量的不纯度量，减少了大量的对数运算；
 4. CART 采用代理测试来估计缺失值，而 C4.5 以不同概率划分到不同节点中；
 5. CART 采用“基于代价复杂度剪枝”方法进行剪枝，而 C4.5 采用悲观剪枝方法。

划分标准(Gini系数，而非上面的信息增益或信息增益率)

熵模型拥有大量耗时的对数运算，基尼指数在简化模型的同时还保留了熵模型的优点。**基尼指数代表了模型的不纯度，基尼系数越小，不纯度越低，特征越好。**这和信息增益（率）正好相反。

$$\begin{aligned} Gini(D) &= \sum_{k=1}^K \frac{|C_k|}{|D|} \left(1 - \frac{|C_k|}{|D|}\right) \\ &= 1 - \sum_{k=1}^K \left(\frac{|C_k|}{|D|}\right)^2 Gini(D|A) \\ &= \sum_{i=1}^n \frac{|D_i|}{|D|} Gini(D_i) \end{aligned}$$

其中 k 代表类别。

基尼指数反映了从数据集中随机抽取两个样本，其类别标记不一致的概率。因此基尼指数越小，则数据集纯度越高。基尼指数偏向于特征值较多的特征，类似信息增益。基尼指数可以用来度量任何不均匀分布，是介于 0~1 之间的数，0 是完全相等，1 是完全不相等，

此外，当 CART 为二分类，其表达式为：

$$Gini(D|A) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

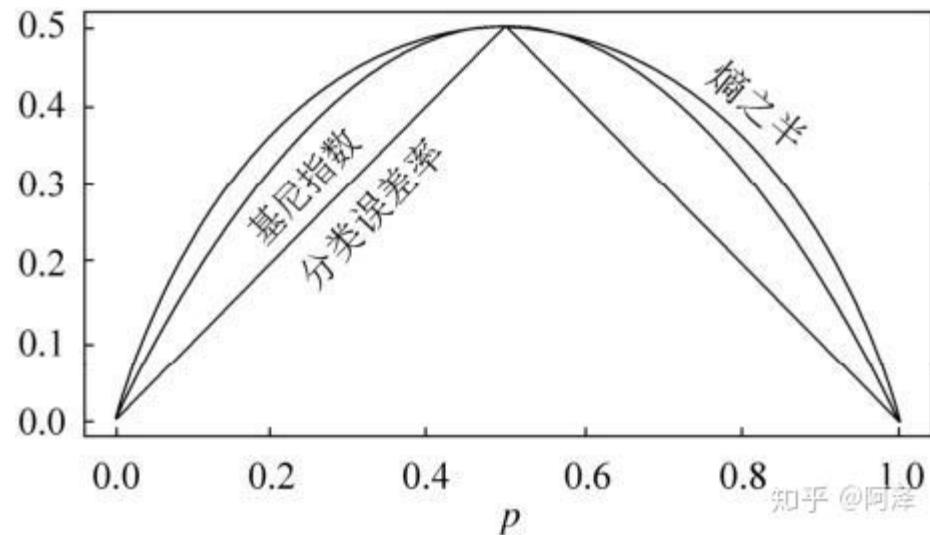
我们可以看到在平方运算和二分类的情况下，其运算更加简单。当然其性能也与熵模型非常接近。

那么问题来了：基尼指数与熵模型性能接近，但到底与熵模型的差距有多大呢？

我们知道 $\ln(x) = -1 + x + o(x)$ ，所以

$$\begin{aligned} H(X) &= -\sum_{k=1}^K p_k \ln p_k \\ &\approx \sum_{k=1}^K p_k (1 - p_k) \end{aligned}$$

我们可以看到，基尼指数可以理解为熵模型的一阶泰勒展开。这边在放上一张很经典的图：



Target feature是连续变量时的回归树

在处理回归问题时，CART 算法使用均方误差 MSE(Mean Squared Error) 来衡量节点的不纯度；MSE 越大，不纯度越高。CART 算法的优化目标就是降低节点的不纯度。具体地说，若将某

个父节点记为 R_k ，再记其分裂后的两个子节点为 R_{kl} 和 R_{kr} ；则该算法所选择的分裂点需使下式取得最小值：

$$MSE(R_{kl}) \times \frac{N_{kl}}{N_k} + MSE(R_{kr}) \times \frac{N_{kr}}{N_k}$$

其中， N_k 、 N_{kl} 和 N_{kr} 分别代表节点 R_k 、 R_{kl} 和 R_{kr} 中的观察单位数量。

缺失值处理：

上文说到，模型对于缺失值的处理会分为两个子问题：

1. 如何在特征值缺失的情况下进行划分特征的选择？
2. 选定该划分特征，模型对于缺失该特征值的样本该进行怎样处理？

cart用的是surrogate splits（替代划分）的方式来处理缺失值：

对于问题 1，CART 一开始严格要求分裂特征评估时只能使用在该特征上没有缺失值的那部分数据，在后续版本中，CART 算法使用了一种惩罚机制来抑制提升值，从而反映出缺失值的影响（例如，如果一个特征在节点的 20% 的记录是缺失的，那么这个特征就会减少 20% 或者其他数值）。

对于问题 2，CART 算法的机制是为树的每个节点都找到代理分裂器。总的原则就是使其分叉的效果与最佳分叉属性相似，即分叉的误差最小。首先，如果某个存在缺失值的特征恰好是当前的分裂增益最大的特征，那么我们需要遍历剩余的特征，剩余的特征中如果有也存在缺失值的特征，那么这些特征忽略，仅仅在完全没有缺失值的特征上进行选择，我们选择其中能够与最佳增益的缺失特征分裂之后增益最接近的特征进行分裂。

如果我们事先设置了一定的标准仅仅选择差异性在一定范围内的特征作为代理特征进行分裂而导致了没有特征和最佳缺失特征的差异性满足要求，或者所有特征都存在缺失值的情况下，缺失样本默认进入个数最大的叶子节点。

cart的代理损失非常的麻烦，主要是费时间，因为计算要涉及到和其它所有 特征进行比较从而找出某个完整的特征来作为代替分裂，可想而知计算量之大，考虑到目前涉及到树基本是在集成的框架下应用的，所以我们常用xgb、lgb、cab等等都没有采纳计算起来这么麻烦的分类方法，一方面太耗费时间，性能会变得很渣，一方面在集成的框架下影响并不是特别大

剪枝策略

剪枝举例：<https://www.cnblogs.com/chenhuabin/p/11774926.html>

采用一种“基于代价复杂度的剪枝”方法进行后剪枝，这种方法会生成一系列树，每个树都是通过将前面的树的某个或某些子树替换成一个叶节点而得到的，这一系列树中的最后一棵树仅含一个用来预测类别的叶节点。然后用一种成本复杂度的度量准则来判断哪棵子树应该被一个预测类别值的叶节点所代替。这种方法需要使用一个单独的测试数据集来评估所有的树，根据它们在测试数据集上的分类性能选出最佳的树。

我们来看具体看一下代价复杂度剪枝算法：

首先我们将最大树称为 T_0 ，我们希望减少树的大小来防止过拟合，但又担心去掉节点后预测误差会增大，所以我们定义了一个损失函数来达到这两个变量之间的平衡。损失函数定义如下：

$$C_\alpha(T) = C(T) + \alpha|T|$$

T 为任意子树， $C(T)$ 为预测误差， $|T|$ 为子树 T 的叶子节点个数， α 是参数， $C(T)$ 衡量训练数据的拟合程度， $|T|$ 衡量树的复杂度， α 权衡拟合程度与树的复杂度。

其中 α 值增大，就是给 $|T|$ 值加砝码，就是更看重复杂度。很容易想到的是，如果剪掉后和没剪时的损失函数一样或者差别不大的话，那当然是剪掉好了，只留下一个点，就能代表一个树杈，这样树就被简化了。

那么如何找到合适的 α 来使得复杂度和拟合度达到最好的平衡点呢，最好的办法就是另 α 从 0 取到正无穷，对于每一个固定的 α ，我们都可以找到使得 $C_\alpha(T)$ 最小的最优子树 $T(\alpha)$ 。当 α 很小的时候， T_0 是最优子树；当 α 最大时，单独的根节点是这样的最优子树。随着 α 增大，我们可以得到一个这样的子树序列： $T_0, T_1, T_2, T_3, \dots, T_n$ ，这里的子树 T_{i+1} 生成是根据前一个子树 T_i 剪掉某一个内部节点生成的。

Breiman 证明：将 α 从小增大， $0 = \alpha_0 < \alpha_1 < \dots < \alpha_n < \infty$ ，在每个区间 $[\alpha_i, \alpha_{i+1})$ 中，子树 T_i 是这个区间里最优的。

这是代价复杂度剪枝的核心思想。

我们每次剪枝都是针对某个非叶节点，其他节点不变，所以我们只需要计算该节点剪枝前和剪枝后的损失函数即可。

对于任意内部节点 t ，剪枝前的状态，有 $|T_t|$ 个叶子节点，预测误差是 $C(T_t)$ ；剪枝后的状态：只有本身一个叶子节点，预测误差是 $C(t)$ 。

因此剪枝前以 t 节点为根节点的子树的损失函数是：

$$C_\alpha(T) = C(T_t) + \alpha|T|$$

剪枝后的损失函数是

$$C_\alpha(t) = C(t) + \alpha$$

通过 Breiman 证明我们知道一定存在一个 α 使得 $C_\alpha(T) = C_\alpha(t)$ ，使得这个值为：

$$\alpha = \frac{C(t) - C(T_t)}{|T_t| - 1}$$

此时， T_t 和 t 的损失函数相等，而 t 的节点少，那么保留 t 就可以了， T_t 就可以剪掉了。所以每个最优子树对应的是一个区间，在这个区间内都是最优的。

然后我们对 T_i 中的每个内部节点 t 都计算：

$$g(t) = \frac{C(t) - C(T_t)}{|T_t| - 1}$$

$g(t)$ 表示阈值，故我们每次都会减去最小的 T_t 。

对于类别不平衡：

CART 的一大优势在于：无论训练数据集有多失衡，它都可以将其自动消除不需要建模人员采取其他操作。

CART 使用了一种先验机制，其作用相当于对类别进行加权。这种先验机制嵌入于 CART 算法判断分裂优劣的运算里，在 CART 默认的分类模式中，总是要计算每个节点关于根节点的类别频率的比值，这就相当于对数据自动重加权，对类别进行均衡。

对于一个二分类问题，节点 node 被分成类别 1 当且仅当：

$$\frac{N_1(node)}{N_1(root)} > \frac{N_0(node)}{N_0(root)}$$

比如二分类，根节点属于 1 类和 0 类的分别有 20 和 80 个。在子节点上有 30 个样本，其中属于 1 类和 0 类的分别是 10 和 20 个。如果 $10/20 > 20/80$ ，该节点就属于 1 类。

通过这种计算方式就无需管理数据真实的类别分布。假设有 K 个目标类别，就可以确保根节点中每个类别的概率都是 $1/K$ 。这种默认的模式被称为“先验相等”。

先验设置和加权不同之处在于先验不影响每个节点中的各类别样本的数量或者份额。先验影响的是每个节点的类别赋值和树生长过程中分裂的选择。

笔试/面试题总结

Ensemble集成算法

集成算法 ensemble：以非常简单的 classifier 为元素（不用神经网络等复杂的）

Bagging: 随机森林(OOB1/3(1/3 证明计算)self-testing 自动 cross-validation) 减小方差 variance

Boosting: AdaBoost (α 数值手推/证明下界无穷小), RegionBoost, xgBoost, GBDT 减小 bias

Random Forest

集成学习 (ensemble) 思想是为了解决单个模型或者某一组参数的模型所固有的缺陷，从而整合起更多的模型，取长补短，避免局限性。随机森林就是集成学习思想下的产物，将许多棵决策树整合成森林，并合起来用来预测最终结果。

- **自助法 (bootstrap)**

自助法顾名思义，是这样一种方法：即从样本自身中再生成很多可用的同等规模的新样本，从自己中产生和自己类似的，所以叫做自助，即不借助其他样本数据。自助法的具体含义如下：

如果我们有个大小为N的样本，我们希望从中得到m个大小为N的样本用来训练。那么我们可以这样做：首先，在N个样本里随机抽出一个样本x₁，然后记下来，放回去，再抽出一个x₂，…，这样重复N次，即可得到N的新样本，这个新样本里可能有重复的。重复m次，就得到了m个这样的样本。实际上就是一个有放回的随机抽样问题。每一个样本在每一次抽的时候有同样的概率（1/N）被抽中。

这个方法在样本比较小的时候很有用，比如我们的样本很小，但是我们希望留出一部分用来做验证，那如果传统方法做train-validation的分割的话，样本就更小了，bias会更大，这是不希望的。而自助法不会降低训练样本的规模，又能留出验证集（因为训练集有重复的，但是这种重复又是随机的），因此有一定的优势。

Self-testing 趋近于1/3

$$1 - \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right)^n$$

- **Bagging**

bagging的名称来源于（Bootstrap AGGREGATING），意思是自助抽样集成，这种方法将训练集分成m个新的训练集，然后在每个新训练集上构建一个模型，各自不相干，最后预测时我们将这个m个模型的结果进行整合，得到最终结果。整合方式就是：分类问题用majority voting，回归用均值。

算法过程如下：

- 1) 从原始样本集中抽取训练集。每轮从原始样本集中使用Bootstraping的方法抽取n个训练样本（在训练集中，有些样本可能被多次抽取到，而有些样本可能一次都没有被抽中）。共进行k轮抽取，得到k个训练集。（k个训练集之间是相互独立的）
- 2) 每次使用一个训练集得到一个模型，k个训练集共得到k个模型。（注：这里并没有具体的分类算法或回归方法，我们可以根据具体问题采用不同的分类或回归方法，如决策树、感知器等）
- 3) 对分类问题：将上步得到的k个模型采用投票的方式得到分类结果；对回归问题，计算上述模型的均值作为最后的结果。（所有模型的重要性相同）

- **步骤：**

- 1) 假如有N个样本，则有放回的随机选择N个样本（每次随机选择一个样本，然后返回继续选择）。这选择好了的N个样本用来训练一个决策树，作为决策树根节点处的样本。
- 2) 当每个样本有M个属性时，在决策树的每个节点需要分裂时，随机从这M个属性中选取出m个属性，满足条件m << M。然后从这m个属性中采用某种策略（比如说信息增益）来选择1个属性作为该节点的分裂属性。
- 3) 决策树形成过程中每个节点都要按照步骤2来分裂（很容易理解，如果下一次该节点选出来的那一个属性是刚刚其父节点分裂时用过的属性，则该节点已经达到了叶子节点，无须继续分裂了）。一直到不能够再分裂为止。注意整个决策树形成过程中没有进行剪枝。

按照步骤1~3建立大量的决策树，这样就构成了随机森林了。

- **优点：**

- 1) 表现性能好，与其他算法相比有着很大优势。
- 2) 随机森林能处理很高维度的数据（也就是很多特征的数据），并且不用做特征选择。
- 3) 在训练完之后，随机森林能给出哪些特征比较重要。
- 4) 训练速度快。

5) 在训练过程中，能够检测到feature之间的影响。

- 6) 对于不平衡数据集来说，随机森林可以平衡误差。当存在分类不平衡的情况时，随机森林能提供平衡数据集误差的有效方法。
- 7) 如果有很大一部分的特征遗失，用RF算法仍然可以维持准确度。
- 8) 随机森林算法有很强的抗干扰能力（具体体现在6,7点）。所以当数据存在大量的数据缺失，用RF也是不错的。
- 9) 随机森林抗过拟合能力比较强（虽然理论上说随机森林不会产生过拟合现象，但是在现实中噪声是不能忽略的，增加树虽然能够减小过拟合，但没有办法完全消除过拟合，无论怎么增加树都不行，再说树的数目也不可能无限增加的。）
- 10) 随机森林能够解决分类与回归两种类型的问题，并在这两方面都有相当好的估计表现。（虽然RF能做回归问题，但通常都用RF来解决分类问题）。
- 11) 在创建随机森林时候，对generalization error(泛化误差)使用的是无偏估计模型，泛化能力强。

- 缺点：

- 1) 随机森林在解决回归问题时，并没有像它在分类中表现的那么好，这是因为它并不能给出一个连续的输出。当进行回归时，随机森林不能够做出超越训练集数据范围的预测，这可能导致在某些特定噪声的数据进行建模时出现过度拟合。（PS:随机森林已经被证明在某些噪音较大的分类或者回归问题上回过拟合）。
- 2) 对于许多统计建模者来说，随机森林给人的感觉就像一个黑盒子，你无法控制模型内部的运行。只能在不同的参数和随机种子之间进行尝试。
- 3) 可能有很多相似的决策树，掩盖了真实的结果。
- 4) 对于小数据或者低维数据（特征较少的数据），可能不能产生很好的分类。（处理高维数据，处理特征遗失数据，处理不平衡数据是随机森林的长处）。
- 5) 执行数据虽然比boosting等快，但比单只决策树慢多了。

- 为什么随机森林可以很好解决过拟合问题：

主要依靠了其中两个随机过程，即产生决策树的样本是随机生成，构建决策树的特征值是随机选取。当随机森林产生的树的数目趋近无穷的时候，理论上根据大数定理可以证明训练误差与测试误差是收敛到一起的。

当然实际过程中，由于不可能产生无穷的决策树，模型参数的设置问题会影响在相同运行时间内拟合结果的过拟合程度的不同。但总而言之，调整参数后，随机森林可以有效的降低过拟合的程度。

- 为什么不能用全样本去训练m棵决策树：

全样本训练忽视了**局部样本**的规律，对于模型的泛化能力是有害的。

GBDT/XGBoost

<https://zhuanlan.zhihu.com/p/29765582>

GBDT 的全称是 Gradient Boosting Decision Tree，梯度提升决策树。

Boosting、bagging和stacking是集成学习的三种主要方法。不同于bagging方法，boosting方法通过分步迭代（stage-wise）的方式来构建模型，在迭代的每一步构建的弱学习器都是为了弥补已有模型的不足。Boosting族算法的著名代表是AdaBoost，AdaBoost算法通过给已有模型预测错误的样本更高的权重，使得先前的学习器做错的训练样本在后续受到更多的关注的方式来弥补已有模型的不足。与AdaBoost算法不同，梯度提升方法在迭代的每一步构建一个能

够沿着梯度最陡的方向降低损失 (steepest-descent) 的学习器来弥补已有模型的不足。经典的AdaBoost算法只能处理采用指数损失函数的二分类学习任务[2]，而梯度提升方法通过设置不同的可微损失函数可以处理各类学习任务（多分类、回归、Ranking等），应用范围大大扩展。另一方面，AdaBoost算法对异常点（outlier）比较敏感，而梯度提升算法通过引入bagging思想、加入正则项（xgboost）等方法能够有效地抵御训练数据中的噪音，具有更好的健壮性。这也是为什么梯度提升算法（尤其是采用决策树作为弱学习器的GBDT算法）如此流行的原因，有种观点认为GBDT是性能最好的机器学习算法，这当然有点过于激进又固步自封的味道，但通常各类机器学习算法比赛的赢家们都非常青睐GBDT算法，由此可见该算法的实力不可小觑。

基于梯度提升算法的学习器叫做 GBM(Gradient Boosting Machine)。理论上，GBM 可以选择各种不同的学习算法作为基学习器。GBDT 实际上是 GBM 的一种情况。

为什么梯度提升方法倾向于选择决策树作为基学习器呢？（也就是 GB 为什么要和 DT 结合，形成 GBDT）决策树可以认为是 if-then 规则的集合，易于理解，可解释性强，预测速度快。同时，决策树算法相比于其他的算法需要更少的特征工程，比如可以不用做特征标准化，可以很好的处理字段缺失的数据，也可以不用关心特征间是否相互依赖等。决策树能够自动组合多个特征。不过，单独使用决策树算法时，有容易过拟合缺点。所幸的是，通过各种方法，抑制决策树的复杂性，降低单颗决策树的拟合能力，再通过梯度提升的方法集成多个决策树，最终能够很好的解决过拟合的问题。由此可见，梯度提升方法和决策树学习算法可以互相取长补短，是一对完美的搭档。

至于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。现在主流的GBDT算法实现中这些方法基本上都有实现，因此GBDT算法的超参数还是比较多的，应用过程中需要精心调参，并用交叉验证的方法选择最佳参数。

GBDT 中的决策树是回归树，预测结果是一个数值，在点击率预测方面常用 GBDT，例如用户点击某个内容的概率。

假设现在你有样本集 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ ，然后你用一个模型，如

$F(x)$ 去拟合这些数据，使得这批样本的平方损失函数（即 $\frac{1}{2} \sum_0^n (y_i - F(x_i))^2$ ）最小。但是你发现虽然模型的拟合效果很好，但仍然有一些差距，比如预测值 $F(x_1) = 0.8$ ，而真实值 $y_1 = 0.9$ ， $F(x_2) = 1.4$ ， $y_2 = 1.3$ 等等。另外你不允许更改原来模型 $F(x)$ 的参数，那么你有什么办法进一步来提高模型的拟合能力呢。

既然不能更改原来模型的参数，那么意味着必须在原来模型的基础之上做改善，那么直观的做法就是建立一个新的模型 $f(x)$ 来拟合 $F(x)$ 未完全拟合真实样本的残差，即 $y - F(X)$ 。所以对于每个样本来说，拟合的样本集就变成了：
 $(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots (x_n, y_n - F(x_n))$

在第一部分， $y_i - F(x_i)$ 被称为残差，这一部分也就是前一模型 ($F(x_i)$) 未能完全拟合的部分，所以交给新的模型来完成。

我们知道gbdt的全称是Gradient Boosting Decision Tree，其中gradient被称为梯度，更一般的理解，可以认为是一阶导，那么这里的残差与梯度是什么关系呢。在第一部分，我们提到

了一个叫做平方损失函数的东西，具体形式可以写成 $\frac{1}{2} \sum_0^n (y_i - F(x_i))^2$ ，熟悉其他算法的原理应该知道，这个损失函数主要针对回归类型的问题，分类则是用熵值类的损失函数。具体到平方损失函数的式子，你可能已经发现它的一阶导其实就是残差的形式，所以基于残差的gbdt是一种特殊的gbdt模型，它的损失函数是平方损失函数，常用来处理回归类的问题。具体形式可以如下表示：

$$\text{损失函数: } L(y, F(x)) = \frac{1}{2}(y - F(x))^2$$

$$\text{所以我们想最小化 } J = \frac{1}{2} \sum_0^n (y_i - F(x_i))^2$$

损失函数的一阶导：

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

正好残差就是负梯度：

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

GBDT 算法的每一步在生成决策树时只需要拟合前面的模型的残差。

损失函数

理论上，损失函数可以为任何函数，只要你认为这个式子的结果可以用来表示预测值与真实值的差距

在这一部分，我们要讨论的是上一节中提到的MSE误差平方和是否是一个好的损失函数选择：不是，因为它对于异常值过于敏感
所以一般回归类的损失函数会用绝对损失或者huber损失函数来代替平方损失函数：

- ▶ Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- ▶ Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

y_i	0.5	1.2	2	5*
$F(x_i)$	0.6	1.4	1.5	1.7
Square loss	0.005	0.02	0.125	5.445
Absolute loss	0.1	0.2	0.5	3.3
Huber loss($\delta = 0.5$)	0.005	0.02	0.125	1.525

目标函数Function Form

<https://zhuanlan.zhihu.com/p/29765582>

如前面所述，gbdt模型可以认为是是由k个基模型组成的一个加法运算式：

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), f_k \in F$$

那么一般化的损失函数是预测值 \hat{y} 与 真实值 y 之间的关系，如我们前面的平方损失函数，那么对于n个样本来说，则可以写成：

$$L = \sum_{i=1}^n l(y_i, \hat{y}_i)$$

更一般的，我们知道一个好的模型，在偏差和方差上有一个较好的平衡，而算法的损失函数正是代表了模型的偏差面，最小化损失函数，就相当于最小化模型的偏差，但同时我们也需要兼顾模型的方差，所以目标函数还包括抑制模型复杂度的正则项，到这里我们的目标函数就可以看成是损失函数及复杂度的最小化，公式为：

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

其中 Ω 代表了基模型的复杂度，若基模型是树模型，则树的深度、叶子节点数等指标可以反应树的复杂程度。

对于Boosting来说，它采用的是前向优化算法，即从前往后，逐渐建立基模型来优化逼近目标函数，具体过程如下：

$$\begin{aligned}
\hat{y}_i^0 &= 0 \\
\hat{y}_i^1 &= f_1(x_i) = \hat{y}_i^0 + f_1(x_i) \\
\hat{y}_i^2 &= f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i) \\
&\dots \\
\hat{y}_i^t &= \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i)
\end{aligned}$$

那么，在每一步，如何学习一个新的模型呢，答案的关键还是在于gbdt的目标函数上，即新模型的加入总是以优化目标函数为目的的。

我们以第t步的模型拟合为例，在这一步，模型对第 i 个样本 x_i 的预测为：

其中 $f_t(x_i)$ 就是我们这次需要加入的新模型，即需要拟合的模型，此时，目标函数就可以写成：

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^t) + \sum_{i=1}^t \Omega(f_i) \\
&= \sum_{i=1}^n l(y_i, \hat{y}_i^{t-1} + f_t(x_i)) + \Omega(f_t) + constant
\end{aligned} \tag{1}$$

即此时最优化目标函数，就相当于求得了 $f_t(x_i)$ 。

Q：问题到现在已经很简单了，只要先创建一个模型，然后把 y 替换成残差，重新训练就可以了，为什么还要梯度下降求下一个模型参数呢？

A：如果不考虑正则化。训练样本只有一个的时候是 $\hat{y} = y$ 损失最小，这时候用什么损失函数都会得到类似的结果。但是训练样本多的时候，不同的损失函数的最优 \hat{y}_i 是不一样的，如果这个时候拟合 $y - \hat{y}_i$ ，得到的就只是 MSE 损失函数的最优解，而不一定是所想要的损失函数的最优解。另外在有正则项的情况下就不再是 $\hat{y} = y$ 时损失函数最小了，所以我们需要计算损失函数的梯度，而不能直接使用分模型来拟合残差。

2.1, 原始loss计算量更大，二阶近似计算量小，还有分割点的近似计算。

2.2 二阶近似解更稳定。

http://sofasofa.io/forum_main_post.php?postid=1004392

使用梯度找到下一个模型的参数 (原始GBDT与XGBoost的最大区别：一阶导与二阶导)

我们知道泰勒公式中，若 Δx 很小时，我们只保留二阶导是合理的：

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$$

函数为损失函数，把 \hat{y}_i^{t-1} 看成是的 x , $f_t(x_i)$ 看成是 Δx 。

目标函数 (cost function + 复杂度正则) 可以写成：

$$Obj^{(t)} = \sum_{i=1}^n \left[l(y_i, \hat{y}_i^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant \quad (3)$$

其中 g_i 为损失函数的一阶导, h_i 为损失函数的二阶导, 注意这里的导是对 \hat{y}_i^{t-1} 求导。

$$\sum_{i=1}^n (y_i - (\hat{y}_i^{t-1} + f_t(x_i)))^2$$

我们以 平方损失函数为例，则

$$g_i = \partial_{\hat{y}^{t-1}} (\hat{y}^{t-1} - y_i)^2 = 2(\hat{y}^{t-1} - y_i)$$

$$h_i = \partial_{\hat{y}^{t-1}}^2 (\hat{y}^{t-1} - y_i)^2 = 2$$

由于在第t步 \hat{y}_i^{t-1} 其实是一个已知的值，所以 $l(y_i, \hat{y}_i^{t-1})$ 是一个常数，其对函数优化不会产生影响，因此，目标函数可以写成：

$$Obj^{(t)} \approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \quad (4)$$

所以我么只要求出每一步损失函数的一阶和二阶导的值 (由于前一步的 \hat{y}^{t-1} 是已知的，所以这两个值就是常数) 代入上述目标函数，然后最优化目标函数，就可以得到每一步的 $f(x)$ ，最后根据加法模型得到一个整体模型。

<https://blog.csdn.net/yangxudong/article/details/53872141>

用决策树来表示上一步的目标函数

上述是GBM的思想，现在我们来思考当 $f(x)$ 是决策树时，来怎么做呢。
其实就是来找到决策树的 $f(x)$ 与损失函数

$f(x)$ 变为 $w_{q(x)}$ ，这里的 $q(x)$ 代表了每个样本在哪个叶子结点上，而 w_q 则代表了哪个叶子结点取什么 w 值。

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

如果决策树的复杂度可以由正则项来定义
的复杂度由生成的树的叶子节点数量和叶子节点对应的值向量的L2范数决定。

我们假设 $I_j = \{i | q(x_i) = j\}$ 为第 j 个叶子节点的样本集合，则等式4根据上面的一些变换可以写成：

$$\begin{aligned} Obj^{(t)} &\approx \sum_{i=1}^n \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned} \quad (5)$$

即我们之前样本的集合，现在都改写成叶子结点的集合，由于一个叶子结点有多个样本存在，

因此才有了 $\sum_{i \in I_j} g_i$ 和 $\sum_{i \in I_j} h_i$ 这两项。

定义 $G_j = \sum_{i \in I_j} g_i$ ， $H_j = \sum_{i \in I_j} h_i$ ，则等式5可以写成：

$$Obj^{(t)} = \sum_{j=1}^T \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T$$

如果树的结构是固定的，即 q 是确定的，或者说我们已经知道了每个叶子结点有哪些样本，所以 G_j 和 H_j 是确定的，但 w 不确定（ w 其实就是我们需要预测的值），那么令目标函数一阶导为0，则可以求得叶子结点 j 对应的值：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad (6)$$

目标函数的值可以化简为：

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \quad (7)$$

如何最优化目标函数

那么对于单棵决策树，一种理想的优化状态就是枚举所有可能的树结构，因此过程如下：

- a、首先枚举所有可能的树结构，即 Q ；
- b、计算每种树结构下的目标函数值，即等式7的值；
- c、取目标函数最小（大）值为最佳的数结构，根据等式6求得每个叶子节点的 [公式] 取值，即样本的预测值。

但上面的方法肯定是不可行的，因为树的结构千千万，所以一般用贪心策略来优化：

- a、从深度为0的树开始，对每个叶节点枚举所有的可用特征
- b、针对每个特征，把属于该节点的训练样本根据该特征值升序排列，通过线性扫描的方式来决定该特征的最佳分裂点，并记录该特征的最大收益（采用最佳分裂点时的收益）
- c、选择收益最大的特征作为分裂特征，用该特征的最佳分裂点作为分裂位置，把该节点生长出左右两个新的叶节点，并为每个新节点关联对应的样本集
- d、回到第1步，递归执行到满足特定条件为止

那么如何计算上面的收益呢，很简单，仍然紧扣目标函数就可以了。假设我们在某一节点上二

分裂成两个节点，分别是左 (L) 右 (R)，则分裂前的目标函数是 $-\frac{1}{2} \left[\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] + \gamma$ ，分裂后则是 $-\frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} \right] + 2\gamma$ ，则对于目标函数来说，分裂后的收益是（这里假设是最小化目标函数，所以用分裂前-分裂后）：

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \quad (8)$$

等式8计算出来的收益，也是作为变量重要度输出的重要依据。

所以gbdt的算法可以总结为：

- a、算法在拟合的每一步都新生成一颗决策树；
- b、在拟合这棵树之前，需要计算损失函数在每个样本上的一阶导和二阶导，即 g_i 和 h_i ；
- c、通过上面的贪心策略生成一颗树，计算每个叶子结点的 G_j 和 H_j ，利用等式6计算预测值 w ；
- d、把新生成的决策树 $f_t(x)$ 加入 $\hat{y}_i^t = \hat{y}_i^{t-1} + \epsilon f_t(x_i)$ ，其中 ϵ 为学习率，主要为了抑制模型的过拟合。

Lightgbm

传统的boosting算法（如GBDT和XGBoost）已经有相当好的效率，但是在如今的大样本和高维度的环境下，传统的boosting似乎在效率和可扩展性上不能满足现在的需求了，主要的原因就是传统的boosting算法需要对每一个特征都要扫描所有的样本点来选择最好的切分点，这是非常的耗时。为了解决这种在大样本高纬度数据的环境下耗时的问题，Lightgbm使用了如下两种解决办法：一是GOSS（Gradient-based One-Side Sampling，基于梯度的单边采样），不是使用所用的样本点来计算梯度，而是对样本进行采样来计算梯度；二是EFB（Exclusive Feature Bundling，互斥特征捆绑），这里不是使用所有的特征来进行扫描获得最佳的切分点，而是将某些特征进行捆绑在一起降低特征的维度，是寻找最佳切分点的消耗减少。这样大大的降低的处理样本的时间复杂度，但在精度上，通过大量的实验证明，在某些数据集上使用Lightgbm并不损失精度，甚至有时还会提升精度。

- Gradient-based One-Side Sampling (GOSS)

GOSS（基于梯度的单边采样）方法的主要思想就是，**梯度大的样本点**在信息增益的计算上扮演着主要的作用，也就是说这些梯度大的样本点会贡献更多的信息增益，因此为了保持信息增益评估的精度，当我们对样本进行下采样的时候保留这些梯度大的样本点，而对于梯度小的样本点按比例进行随机采样即可。

在AdaBoost算法中，我们在每次迭代时更加注重上一次错分的样本点，也就是上一次错分的样本点的权重增大，而在GBDT中并没有本地的权重来实现这样的过程，所以在AdaBoost中提出的采样模型不能应用在GBDT中。但是，每个样本的梯度对采样提供了非常有用的信息。也就是说，如果一个样本点的梯度小，那么该样本点的训练误差就小并且已经经过了很好的训练。一个直接的办法就是直接抛弃梯度小的样本点，但是这样做的话会改变数据的分布和损失学习的模型精度。GOSS的提出就是为了避免这两个问题的发生。

算法：

输入：训练数据，迭代步数d，大梯度数据的采样率a，小梯度数据的采样率b，损失函数和弱学习器的类型（一般为决策树）；

输出：训练好的强学习器；

- (1) 根据样本点的梯度的绝对值对它们进行降序排序；
- (2) 对排序后的结果选取前 $a * 100\%$ 的样本生成一个大梯度样本点的子集；
- (3) 对剩下的样本集合 $(1-a) * 100\%$ 的样本，随机的选取 $b * (1-a) * 100\%$ 个样本点，生成一个小梯度样本点的集合；
- (4) 将大梯度样本和采样的小梯度样本合并；
- (5) 将小梯度样本乘上一个权重系数 $(1 - a) / b$ ；
- (6) 使用上述的采样的样本，学习一个新的弱学习器；
- (7) 不断地重复(1) ~ (6)步骤直到达到规定的迭代次数或者收敛为止。

从上面的描述可知，当 $\alpha=0$ 时，GOSS算法退化为随机采样算法；当 $\alpha=1$ 时，GOSS算法变为采取整个样本的算法。在许多情况下，GOSS算法训练出的模型精确度要高于随机采样算法。另一方面，采样也将会增加弱学习器的多样性，从而潜在的提升了训练出的模型泛化能力。

- Exclusive Feature Bundling (EFB)

Lightgbm实现中不仅进行了数据采样，也进行了特征抽样，使得模型的训练速度进一步的减少。但是该特征抽样又与一般的特征抽样有所不同，是将互斥特征绑定在一起从而减少特征维度。主要思想就是，通常在实际应用中高纬度的数据往往都是稀疏数据（如one-hot编码），这使我们有可能设计一种几乎无损的方法来减少有效特征的数量。尤其，在稀疏特征空间中许多特征都是互斥的（例如，很少同时出现非0值）。这就使我们可以安全的将互斥特征绑定在一起形成一个特征，从而减少特征维度。但是怎样的将互斥特征绑定在一起了？Lightgbm作者使用的是基于直方图（histograms）的方法。

算法：

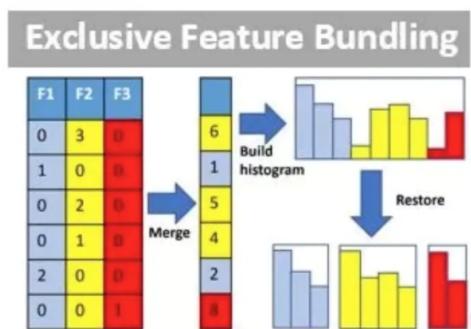
输入：特征F，最大冲突数K，图G；

输出：特征捆绑集合bundles；

(1) 构造一个边带有权重的图，其权值对应于特征之间的总冲突；

(2) 通过特征在图中的度来降序排序特征；

(3) 检查有序列表中的每个特征，并将其分配给具有小冲突的现有bundling（由每个绑定的最大冲突率控制），或创建新bundling。



对于具有高维稀疏特征的数据，很多特征是互斥的（即多个特征之间最多只有一个特征的取值为非0），EFB通过捆绑多个互斥特征形成一个“大特征”，从而大大减少特征的数量，相当于是一种降维的方法。

直方图算法的基本思想是先把连续的特征值离散化成k个整数，同时构造一个宽度为k的直方图。在遍历数据的时候，根据离散化后的值作为索引在直方图中累积统计量，当遍历一次数据后，直方图累积了需要的统计量，然后根据直方图的离散值，遍历寻找最优的分割点（减少遍历）。

<https://zhuanlan.zhihu.com/p/35155992>

笔试/面试题整理

- Xgboost与lightgbm的区别和适用场景

(1) xgboost采用的是level-wise的分裂策略，而lightGBM采用了leaf-wise的策略，区别是xgboost对每一层所有节点做无差别分裂，可能有些节点的增益非常小，对结果影响不大，但是xgboost也进行了分裂，带来了不必要的开销。leaf-wise的做法是在当前所有叶子节点中选择分裂收益最大的节点进行分裂，如此递归进行，很明显leaf-wise这种做法容易过拟合，因为容易陷入比较高的深度中，因此需要对最大深度做限制，从而避免过拟合。

(2) lightgbm使用了基于histogram的决策树算法，这一点不同与xgboost中的exact算法，histogram算法在内存和计算代价上都有不小优势。1) 内存上优势：很明显，直方图算法的内存消耗为($\#data * \#features * 1\text{Bytes}$)(因为对特征分桶后只需保存特征离散化之后的值)，而xgboost的exact算法内存消耗为： $(2 * \#data * \#features * 4\text{Bytes})$ ，因为xgboost既要保存原始feature的值，也要保存这个值的顺序索引，这些值需要32位的浮点数来保存。2) 计算上的优势，预排序算法在选择好分裂特征计算分裂收益时需要遍历所有样本的特征值，时间为($\#data$)，而直方图算法只需要遍历桶就行了，时间为($\#bin$)

(3) 直方图做差加速，一个子节点的直方图可以通过父节点的直方图减去兄弟节点的直方图得到，从而加速计算。

(4) lightgbm支持直接输入categorical 的feature，在对离散特征分裂时，每个取值都当作一个桶，分裂时的增益算的是“是否属于某个category”的gain。类似于one-hot编码。

(5) xgboost在每一层都动态构建直方图，因为xgboost的直方图算法不是针对某个特定的feature，而是所有feature共享一个直方图(每个样本的权重是二阶导)，所以每一层都要重新构建直方图，而lightgbm中对每个特征都有一个直方图，所以构建一次直方图就够了。

其适用场景根据实际项目和两种算法的优点进行选择。

- XGBoost对比GBDT有什么优势

GBDT 的缺点很明显，Boost 是一个串行过程，不好并行化，而且计算复杂度高，同时不太适合高维稀疏特征；

XGBoost的优点

实现了分裂点寻找近似算法。

利用了特征的稀疏性。

1. 传统GBDT以CART作为基分类器，xgboost还支持线性分类器，这个时候xgboost相当于带L1和L2正则化项的逻辑斯蒂回归（分类问题）或者线性回归（回归问题）。
2. 传统GBDT在优化时只用到一阶导数信息，xgboost则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。顺便提一下，xgboost工具支持自定义代价函数，只要函数可一阶和二阶求导。

3. xgboost在代价函数里加入了正则项，用于控制模型的复杂度。正则项里包含了树的叶子节点个数、每个叶子节点上输出的score的L2模的平方和。从 Bias-variance tradeoff角度来讲，正则项降低了模型的variance，使学习出来的模型更加简单，防止过拟合，这也是xgboost优于传统GBDT的一个特性。
4. Shrinkage（缩减），相当于学习速率（xgboost中的eta）。xgboost在进行完一次迭代后，会将叶子节点的权重乘上该系数，主要是为了削弱每棵树的影响，让后面有更大的学习空间。实际应用中，一般把eta设置得小一点，然后迭代次数设置得大一点。（补充：传统GBDT的实现也有学习速率）
5. 列抽样（column subsampling）。xgboost借鉴了随机森林的做法，支持列抽样，不仅能降低过拟合，还能减少计算，这也是xgboost异于传统gbdt的一个特性。
6. 对缺失值的处理。对于特征的值有缺失的样本，xgboost可以自动学习出它的分裂方向。
7. xgboost工具支持并行。boosting不是一种串行的结构吗？怎么并行的？注意xgboost的并行不是tree粒度的并行，xgboost也是一次迭代完才能进行下一次迭代的（第t次迭代的代价函数里包含了前面t-1次迭代的预测值）。xgboost的并行是在特征粒度上的。我们知道，决策树的学习最耗时的一个步骤就是对特征的值进行排序（因为要确定最佳分割点），xgboost在训练之前，预先对数据进行了排序，然后保存为block结构，后面的迭代中重复地使用这个结构，大大减小计算量。这个block结构也使得并行成为了可能，在进行节点的分裂时，需要计算每个特征的增益，最终选增益最大的那个特征去做分裂，那么各个特征的增益计算就可以开多线程进行。
8. 可并行的近似直方图算法。树节点在进行分裂时，我们需要计算每个特征的每个分割点对应的增益，即用贪心法枚举所有可能的分割点。当数据无法一次载入内存或者在分布式情况下，贪心算法效率就会变得很低，所以xgboost还提出了一种可并行的近似直方图算法，用于高效地生成候选的分割点。基于分布式通信框架 rabbit，可以运行在 MPI 和 yarn 上。（最新已经不基于 rabbit 了）实现做了面向体系结构的优化，针对 cache 和内存做了性能优化。

聚类分析 Clustering

聚类分析定义：

聚类分析是根据在数据中发现的描述对象及其关系的信息，将数据对象分组。目的是，组内的对象相互之间是相似的（相关的），而不同组中的对象是不同的（不相关的）。组内相似性越大，组间差距越大，说明聚类效果越好。

聚类效果的好坏依赖于两个因素：1) 衡量距离的方法；2) 聚类算法

K-means

K-mean是一种无监督的聚类算法，也就是说并没有告诉训练算法某一个数据属于哪个类别。

- 算法：
 - 1) 选择k个初始质心，初始质心可以随机选择，每个质心为一类。
 - 2) 计算数据集样本中其它的点到质心的距离（欧式距离或余弦相似度），然后选取最近质心的类别作为自己的类别。

- 3) 重新计算每个类的质心，所谓质心就是一个类中的所有观测的平均向量（这里称为向量，是因为每一个观测都包含很多变量，所以我们把一个观测视为一个多维向量，维数由变量数决定）。
- 4) 重复2)和3)，直到达到某一阀值时停止。这个阀值可以是迭代的轮数，也可以是当质心不发生改变的时候或者质心变化的幅度小于某一个值的时候停止迭代。
- Pseudocode:

```

输入: 样本集  $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ ;
聚类簇数  $k$ .
过程:
1: 从  $D$  中随机选择  $k$  个样本作为初始均值向量  $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k\}$ 
2: repeat
3:   令  $C_i = \emptyset (1 \leq i \leq k)$ 
4:   for  $j = 1, 2, \dots, m$  do
5:     计算样本  $\mathbf{x}_j$  与各均值向量  $\boldsymbol{\mu}_i (1 \leq i \leq k)$  的距离:  $d_{ji} = \|\mathbf{x}_j - \boldsymbol{\mu}_i\|_2$ ;
6:     根据距离最近的均值向量确定  $\mathbf{x}_j$  的簇标记:  $\lambda_j = \arg \min_{i \in \{1, 2, \dots, k\}} d_{ji}$ ;
7:     将样本  $\mathbf{x}_j$  划入相应的簇:  $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$ ;
8:   end for
9:   for  $i = 1, 2, \dots, k$  do
10:    计算新均值向量:  $\boldsymbol{\mu}'_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$ ;
11:    if  $\boldsymbol{\mu}'_i \neq \boldsymbol{\mu}_i$  then
12:      将当前均值向量  $\boldsymbol{\mu}_i$  更新为  $\boldsymbol{\mu}'_i$ 
13:    else
14:      保持当前均值向量不变
15:    end if
16:  end for
17: until 当前均值向量均未更新
输出: 簇划分  $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ 

```

- 算法公式化解释:

记 k 个簇中心分别为 $u_1, u_2, u_3, \dots, u_k$, 每个簇的样本数目为 N_1, N_2, \dots, N_k 。

使用平方误差做为误差函数, 得:

$$J(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \dots, \boldsymbol{\mu}_k) = \frac{1}{2} \sum_{j=1}^K \sum_{i=1}^{N_j} (\mathbf{x}_i - \boldsymbol{\mu}_j)^2$$

将该函数做为目标函数, 求解该函数的最小值。可以使用梯度下降法求, 该函数为凸函数, 驻点为:

$$\frac{\partial J}{\partial \boldsymbol{\mu}_j} = -2 \sum_{i=1}^{N_j} (\mathbf{x}_i - \boldsymbol{\mu}_j) \xrightarrow{\text{令}} 0 \Rightarrow \boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \mathbf{x}_i$$

可以看到, 要想使损失函数最小, 聚类中心要为各簇中样本点的平均值。由此可以看出, K-means算法在每次迭代更新时使用各簇中样本点的平均值为聚类中心是有道理的。

- K值的确定:
- 1) 手肘法 (elbow method)

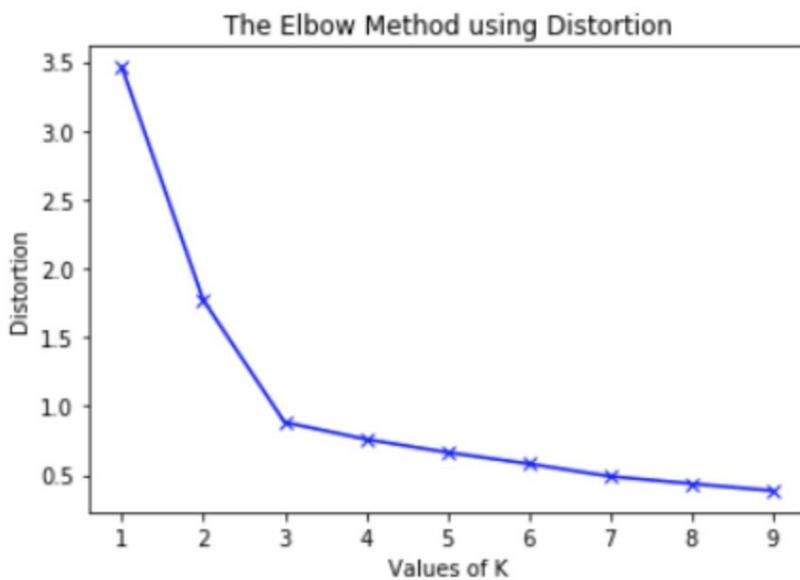
手肘法的核心指标是SSE(sum of the squared errors, 误差平方和),

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

http://blog.csdn.net/qq_15738501

其中, C_i 是第*i*个簇, p 是 C_i 中的样本点, m_i 是 C_i 的质心 (C_i 中所有样本的均值), SSE是所有样本的聚类误差, 代表了聚类效果的好坏。

手肘法的核心思想是: 随着聚类数k的增大, 样本划分会更加精细, 每个簇的聚合程度会逐渐提高, 那么误差平方和SSE自然会逐渐变小。并且, 当k小于真实聚类数时, 由于k的增大会大幅增加每个簇的聚合程度, 故SSE的下降幅度会很大, 而当k到达真实聚类数时, 再增加k所得到的聚合程度回报会迅速变小, 所以SSE的下降幅度会骤减, 然后随着k值的继续增大而趋于平缓, 也就是说SSE和k的关系图是一个手肘的形状, 而这个肘部对应的k值就是数据的真实聚类数。当然, 这也是该方法被称为手肘法的原因。



2) 轮廓系数法

该方法的核心指标是轮廓系数（Silhouette Coefficient），某个样本点 X_i 的轮廓系数定义如下：

$$S = \frac{b - a}{\max(a, b)}$$

其中， a 是 X_i 与同簇的其他样本的平均距离，称为凝聚度， b 是 X_i 与最近簇中所有样本的平均距离，称为分离度。而最近簇的定义是

$$C_j = \arg \min_{C_k} \frac{1}{n} \sum_{p \in C_k} |p - X_i|^2$$

其中 p 是某个簇 C_k 中的样本。事实上，简单点讲，就是用 X_i 到某个簇所有样本平均距离作为衡量该点到该簇的距离后，选择离 X_i 最近的一个簇作为最近簇。

求出所有样本的轮廓系数后再求平均值就得到了平均轮廓系数。平均轮廓系数的取值范围为 $[-1, 1]$ ，且簇内样本的距离越近，簇间样本距离越远，平均轮廓系数越大，聚类效果越好。那么，很自然地，平均轮廓系数最大的 k 便是最佳聚类数。

3) Calinski-Harabasz准则

$$VRC_k = \frac{SSB}{SSW} * (N - k) / (k - 1)$$

其中 SSB 是类间方差， $SSB = \sum_{i=1}^k n_i ||m_i - m||^2$ ， m 为所有点的中心点， m_i 为某类的中心点；

SSW 是类内方差， $SSW = \sum_{i=1}^k \sum_{x \in c_i} ||x - m_i||^2$ ；

$(N-k)/(k-1)$ 是复杂度；

VRC_k 比率越大，数据分离度越大。

- 为什么不能使用曼哈顿距离？

欧式距离：

$$d_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

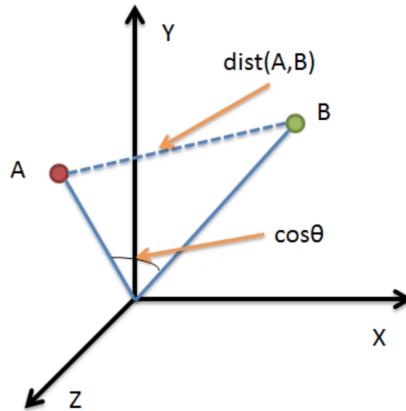
曼哈顿距离：

$$d_{12} = |x_1 - x_2| + |y_1 - y_2|$$

- 余弦相似度

余弦相似度用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小。相比距离度量，余弦相似度更加注重两个向量在方向上的差异，而非距离或长度上。下图表示余弦相似度的余弦是哪个角的余弦， A , B 是三维空间中的两个向量，这两个点

与三维空间原点连线形成的角，如果角度越小，说明这两个向量在方向上越接近，在聚类时就归成一类：



看一个例子：歌手大赛，三个评委给三个歌手打分，第一个评委的打分 (10, 8, 9) 第二个评委的打分 (4, 3, 2)，第三个评委的打分 (8, 9, 10)

如果采用余弦相似度来看每个评委的差异，虽然每个评委对同一个选手的评分不一样，但第一、第二两个评委对这四位歌手实力的排序是一样的，只是第二个评委对满分有更高的评判标准，说明第一、第二个评委对音乐的品味上是一致的。

因此，用余弦相似度来看，第一、第二个评委为一类人，第三个评委为另外一类。

如果采用欧氏距离，第一和第三个评委的欧氏距离更近，就分成一类人了，但其实不太合理，因为他们对于四位选手的排名都是完全颠倒的。

总之，如果注重数值本身的差异，就应该用欧氏距离，如果注重的是上例中的这种的差异，就要用余弦相似度来计算。

- 标准化问题：

如果选用了欧式距离计算距离，变量的数值差别很大，就需要进行数据的标准化。举例而言，如果x都是1-10之间的数，y都是1000以上的数，那么，在计算距离的时候Y起到的作用就比X大很多，X对于距离的影响几乎可以忽略，就会产生问题。进行数据标准化，即将数据按比例缩放，使之落入一个特定区间。去除数据的单位限制，将其转化为无量纲的纯数值，便于不同单位或量级的指标能够进行计算和比较。

标准化方法最常用的有两种：

min-max标准化（离差标准化）：对原始数据进行线性变换，使结果落到【0, 1】区间，转换方法为 $X' = (X - \text{min}) / (\text{max} - \text{min})$ ，其中max为样本数据最大值，min为样本数据最小值。

z-score标准化（标准差标准化）：处理后的数据符合标准正态分布（均值为0，方差为1），转换公式：X减去均值，再除以标准差。

- 关于离群值的处理：

离群值就是远离整体的，非常异常、非常特殊的数据点，在聚类之前应该将这些“极大”“极小”之类的离群数据都去掉，否则会对于聚类的结果有影响。但是，离群值往往自身就很有分析的价值，可以把离群值单独作为一类来分析。

- 适用范围及缺陷：

当潜在的簇形状是凸面的，簇与簇之间区别较明显，且簇大小相近时，其聚类结果较理想。对于处理大数据集合，该算法非常高效，且伸缩性较好。

缺陷：

1. 只能用于数值型数据，无法用于类别数据

如果是类别型变量，可以使用k-modes算法

- 算法步骤：

01. 随机选取k个初始中心点；
02. 针对数据集中的每个样本点，计算样本点与k个中心点的距离（这边计算的是汉明距离，两个等长字符串在对应位置上不同字符的数目），将样本点划分到离它最近的中心点所对应的类别中；
03. 类别划分完成后，重新确定类别的中心点，将类别中所有样本各特征的众数作为新的中心点对应特征的取值，即该类中所有样本的众心；
04. 重复步骤2 3，直到总距离（各个簇中样本与各自簇中心距离之和）不再降低，返回最后的聚类结果。

→ 效率：

时间复杂度低于k-means和k-medoids

2. 对初始聚类中心敏感

改进：

K-means++

→ 算法思想：

初始的聚类中心之间的相互距离要尽可能的远。

→ 算法步骤：

步骤一：随机选取一个样本作为第一个聚类中心 c_1 ；

步骤二：计算每个样本与当前已有类聚中心最短距离（即与最近一个聚类中心的距离），用 $D(x)$ 表示；接着计算每个样本被选为下一个聚类中心的概率

$\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$ 这个值越大，表示被选取作为聚类中心的概率较大；最后，用轮盘法选出下一个聚类中心；

步骤三：重复步骤二，直到选出 k 个聚类中心。

轮盘法：在 $[0, 1]$ 之间取一个随机数 R 。然后用 R 减 $P(A)$ ，如果减去后的结果小于等于 0 就选 A 作为下一个点，如果减去后还大于 0，就继续再减去 $P(B)$ ，直到减去后的结果小于等于 0。用最后减去时的那个概率值对应的点作为下一个点。

→ 效率：

K-means++ 能显著的改善分类结果的最终误差。

尽管计算初始点时花费了额外的时间，但是在迭代过程中，k-mean 本身能快速收敛，因此算法实际上降低了计算时间。

二分k均值(bisecting k-means)算法

→ 算法思想：

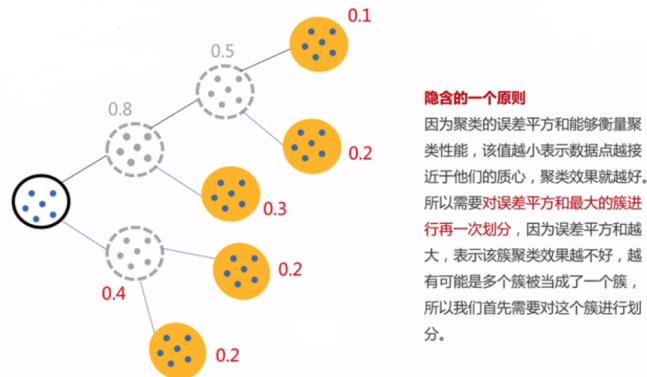
首先将所有点作为一个簇，然后将该簇一分为二。之后选择能最大程度降低聚类代价函数（也就是误差平方和）的簇划分为两个簇。以此进行下去，直到簇的数目等于用户给定的数目 k 为止。

→ 算法步骤：

```

1 | 初始化簇表，使之包含由所有的点组成的簇。
2 | repeat
3 |   从簇表中取出一个簇。
4 |   {对选定的簇进行多次二分试验}
5 |   for i=1 to 试验次数 do
6 |     使用基本k均值，二分选定的簇。
7 |   endfor
8 |   从二分试验中选择具有最小误差的两个簇。
9 |   将这两个簇添加到簇表中。
10| until 簇表中包含k个簇

```



→ 效率：

可以加速k-means算法的执行速度，因为它的相似度计算少了并且不受初始化问题的影响，因为这里不存在随机点的选取，且每一步都保证了误差最小。

3. 对于不是凸的数据集比较难收敛

改进：

基于密度的聚类算法更加适合。

DESCAN算法

4. 如果各隐含类别的数据不平衡，比如各隐含类别的数据量严重失衡，或者各隐含类别的方差不同，则聚类效果不佳。
5. 采用迭代方法，得到的结果只是局部最优。
6. 对噪音和异常点比较的敏感

k-means选取质点是对某类簇中所有的样本点维度求平均值，即获得该类簇质点的维度。当聚类的样本点中有“噪声”（离群点）时，在计算类簇质点的过程中会受到噪声异常维度的干扰，造成所得质点和实际质点位置偏差过大，从而使类簇发生“畸变”。举例：类簇C1中已经包含点A(1,1)、B(2,2)、C(1,2)、D(2,1)，假设N(100,100)为异常点，当它纳入类簇C1时，计算质点 $\text{Centroid}((1+2+1+2+100)/5, (1+2+2+1+100)/5) = \text{centroid}(21, 21)$ ，此时可能造成了类簇C1质点的偏移，在下一轮迭代重新划分样本点的时候，将大量不属于类簇C1的样本点纳入，因此得到不准确的聚类结果。

改进：

LOF算法

→ 算法思想：

通过比较每个点p和其邻域点的密度来判断该点是否为异常点，如果点p的密度越低，越可能被认定是异常点。

→ 相关概念：

$d(p,o)$: 两点p和o之间的距离

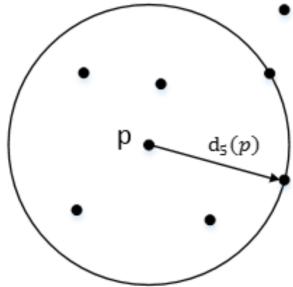
k -distance: 第K距离

对于点p的第k距离 $d_k(p)$ 定义如下:

$d_k(p) = d(p, o)$, 并且满足:

- a) 在集合中至少有不包括p在内的k个点 $o \in C\{x \neq p\}$, 满足 $d(p, o) \leq d(p, o)$;
- b) 在集合中最多有不包括p在内的 $k - 1$ 个点 $o \in C\{x \neq p\}$, 满足 $d(p, o) < d(p, o)$

p的第k距离, 也就是距离p第k远的点的距离, 不包括p, 如图3。



k-distance neighborhood of p: 第k距离邻域

点p的第k距离邻域 $N_k(p)$, 就是p的第k距离即以内的所有点, 包括第k距离。

因此p的第k邻域点的个数 $|N_k(p)| \geq k$ 。

reach-distance: 可达距离

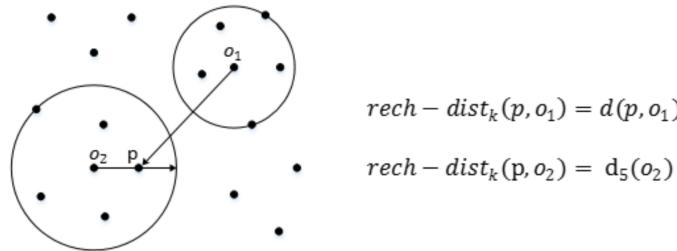
点o到点p的第k可达距离定义为:

$$\text{reach-distance}_k(p, o) = \max\{k - \text{distance}(o), d(p, o)\}$$

也就是说, 点o到点p的第k可达距离, 至少是o的第k距离, 或者为o、p间的真实距离。

这也意味着, 离点o最近的k个点, o到它们的可达距离被认为相等, 且都等于 $d_k(o)$ 。

如图4, o_1 到p的第5可达距离为 $d(p, o_1)$, o_2 到p的第5可达距离为 $d_5(o_2)$ 。



local reachability density: 局部可达密度

点p的局部可达密度表示为:

$$\text{lrd}_k(p) = 1 / \left(\frac{\sum_{o \in N_k(p)} \text{reach-dist}_k(p, o)}{|N_k(p)|} \right)$$

表示点p的第k邻域内点到p的平均可达距离的倒数。

注意, 是p的邻域点 $N_k(p)$ 到p的可达距离, 不是p到 $N_k(p)$ 的可达距离, 一定要弄清楚关系。并且, 如果有重复点, 那么分母的可达距离之和有可能为0, 则会导致lrd变为无限大, 下面还会继续提到这一点。

这个值的含义可以这样理解, 首先这代表一个密度, 密度越高, 我们认为越可能属于同一簇, 密度越低, 越可能是离群点。如果p和周围邻域点是同一簇, 那么可达距离越可能为较小的 $d_k(o)$, 导致可达距离之和较小, 密度值较高; 如果p和周围邻居点较远, 那么可达距离可能都会取较大值 $d(p, o)$, 导致密度较小, 越可能是离群点。

local outlier factor: 局部离群因子

点p的局部离群因子表示为：

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{ldk(o)}{ldk(p)}}{|N_k(p)|} = \frac{\sum_{o \in N_k(p)} lrd_k(o)}{|N_k(p)|} / lrd_k(p)$$

表示点p的邻域点 $N_k(p)$ 的局部可达密度与点p的局部可达密度之比的平均数。

如果这个比值越接近1，说明p的其邻域点密度差不多，p可能和邻域同属一族；如果这个比值越小于1，说明p的密度高于其邻域点密度，p为密集点；如果这个比值大于1，说明p的密度小于其邻域点密度，p越可能是异常点。

但该算法除了要事先确定簇数K和对初始聚类中心敏感外，经常以局部最优结束，同时对“噪声”和孤立点敏感，并且该方法不适于发现非凸面形状的簇或大小差别很大的簇。

k中心点算法 (k-medoids)

→ 算法思想：

每次迭代后的质点都是从聚类的样本点中选取，而选取的标准就是当该样本点成为新的质点后能提高类簇的聚类质量，使得类簇更紧凑。该算法使用绝对误差标准来定义一个类簇的紧凑程度。

$$E = \sum_1^k \sum_{p \in C_j} |p - o_j|$$

(p是空间中的样本点，Oj是类簇Cj的质点)

如果某样本点成为质点后，绝对误差能小于原质点所造成的绝对误差，那么K中心点算法认为该样本点是可以取代原质点的，在一次迭代重计算类簇质点的时候，我们选择绝对误差最小的那个样本点成为新的质点。

Eg：样本点A → E1=10

样本点B → E2=11

样本点C → E3=12

原质点O → E4=13，那我们选举A作为类簇的新质点。

与K-means算法一样，K-medoids也是采用欧几里得距离来衡量某个样本点到底是属于哪个类簇。终止条件是，当所有的类簇的质点都不在发生变化时，即认为聚类结束。

→ 效率：

虽然可以改善k-means对噪声敏感的问题，但是算法的时间复杂度上升为 $O(k(n-k)^2)$

● 笔试、面试题整理

- 1) K-means算法的时间空间复杂度（数据挖掘算法岗）
- 2) 对于k-means算法，除了随机选择质心，还有什么方法可以确定初始的聚类中心（数据挖掘算法岗）
- 3) k-means 和 k-means ++ 的区别
- 4) 如何优化k-means
- 5) k-means算法思想：

对于给定的样本集，在最小化误差函数的基础上将数据划分为预定的类数K。让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的

EM聚类（最大期望算法）

- 最大似然

极大似然估计，是一种概率论在统计学的应用，它是参数估计的方法之一。说的是已知某个随机样本满足某种概率分布，但是其中具体的参数不清楚，参数估计就是通过若干次试验，观察其结果，利用结果推出参数的大概值。最大似然估计是建立在这样的思想上：已知某个参数能使这个样本出现的概率最大，所以干脆把这个参数作为估计的真实值。

在学校那么多男生（身高）中，我们独立地按照概率密度 $p(x|\theta)$ 抽取100了个（身高），组成样本集 X ，我们想通过样本集 X 来估计出未知参数 θ 。这里概率密度 $p(x|\theta)$ 我们知道了是高斯分布 $N(\mu, \sigma^2)$ 的形式，其中的未知参数是 $\theta = [\mu, \sigma^2]$ 。抽到的样本集是 $X = \{x_1, x_2, \dots, x_N\}$ ，其中 x_i 表示抽到的第 i 个人的身高，这里 N 就是 100，表示抽到的样本个数。

由于每个样本都是独立地从 $p(x|\theta)$ 中抽取的，抽到男生 A（的身高）的概率是 $p(x_A|\theta)$ ，抽到男生 B 的概率是 $p(x_B|\theta)$ ，因为他们是独立的，所以同时抽到男生 A 和男生 B 的概率是 $p(x_A|\theta) * p(x_B|\theta)$ ，同理，同时抽到这 100 个男生的概率就是他们各自概率的乘积。

$$L(\theta) = L(x_1, \dots, x_n; \theta) = \prod_{i=1}^n p(x_i; \theta), \theta \in \Theta.$$

这个函数放映的是在不同的参数 θ 取值下，取得当前这个样本集的可能性，因此称为参数 θ 相对于样本集 X 的似然函数（likelihood function）。记为 $L(\theta)$ 。

因此，我们就只需要找到一个参数 θ ，其对应的似然函数 $L(\theta)$ 最大，也就是说抽到这 100 个男生（的身高）概率最大。这个叫做 θ 的最大似然估计量，记为：

$$\hat{\theta} = \arg \max l(\theta)$$

为了便于分析，我们定义了对数似然函数，将其变为相加的

$$H(\theta) = \ln L(\theta) = \ln \prod_{i=1}^n p(x_i; \theta) = \sum_{i=1}^n \ln p(x_i; \theta)$$

求最大似然函数估计值的一般步骤：

- (1) 写出似然函数；
- (2) 对似然函数取对数，并整理；
- (3) 求导数，令导数为 0，得到似然方程；
- (4) 解似然方程，得到的参数即为所求；

- EM 算法

简介：

期望最大算法是一种从不完全数据或有数据丢失的数据集（存在隐含变量）中求解概率模型参数的最大似然估计方法。

具体而言，

如果抽取得到的每个样本都不知道是从哪个分布抽取的，这个时候，对于每一个样本或者你抽取到的人，就有两个东西需要猜测或者估计的了，一是这个人是男的还是女

的？二是男生和女生对应的身高的高斯分布的参数是多少？（鸡生蛋还是蛋生鸡问题）

假设我们想估计知道A和B两个参数，在开始状态下二者都是未知的，但如果知道了A的信息就可以得到B的信息，反过来知道了B也就得到了A。可以考虑首先赋予A某种初值，以此得到B的估计值，然后从B的当前值出发，重新估计A的取值，这个过程一直持续到收敛为止。

步骤：

先给这个分布设置一个初始值，然后求这个隐含变量的期望，当成是这个隐含变量的已知值，然后用最大似然求解那个分布的参数，那假设这个参数比之前的那个随机的参数要好，它更能表达真实的分布，那么我们再通过这个参数确定的分布去求这个隐含变量的期望，然后再最大化，得到另一个更优的参数，……迭代。

算法流程：

初始化分布参数 θ

重复以下步骤直到收敛：

E步骤：根据参数初始值或上一次迭代的模型参数来计算出隐性变量的后验概率，其实就是隐性变量的期望。作为隐藏变量的现估计值：

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)}; \theta).$$

M步骤：将似然函数最大化以获得新的参数值：

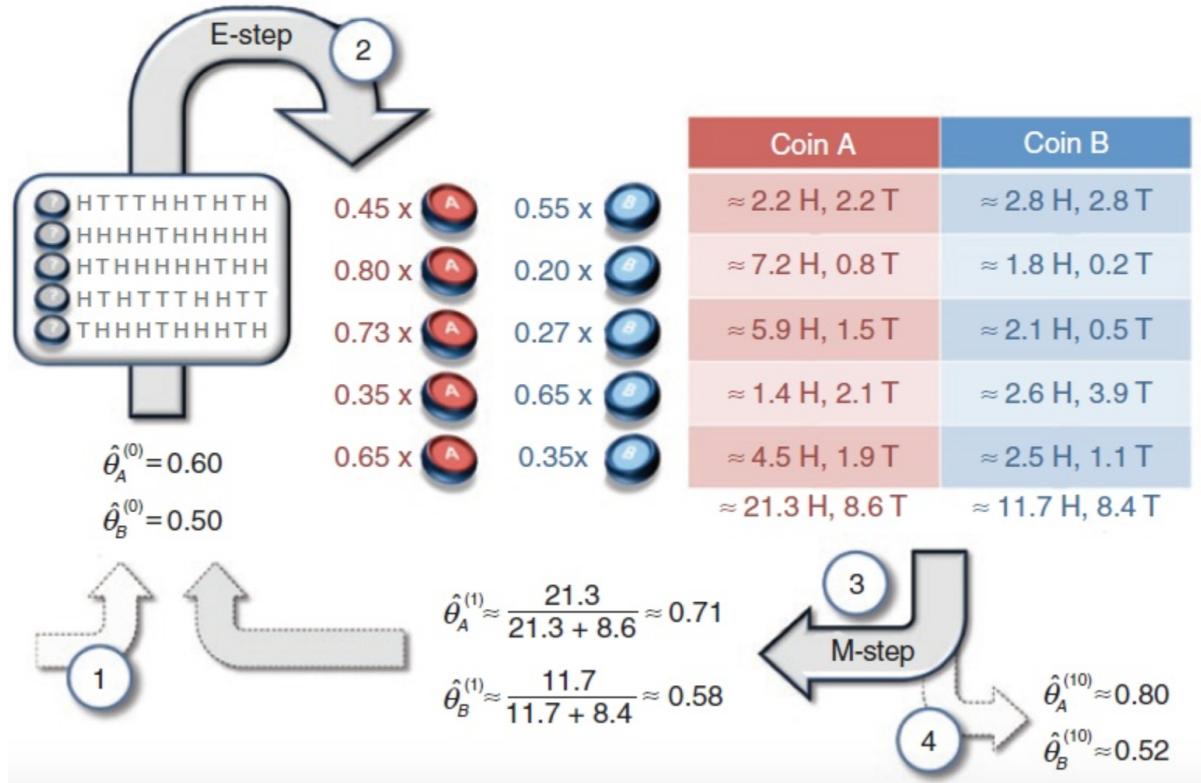
$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

案例：

假设有两枚硬币A、B，以相同的概率随机选择一个硬币，进行如下的掷硬币实验：共做5次实验，每次实验独立的掷十次，结果如图中a所示，例如某次实验产生了H、T、T、T、T、H、H、T、H、T、H（H代表正面朝上）。a是在知道每次选择的是A还是B的情况下进行，b是在不知道选择的是A还是B的情况下进行，问如何估计两个硬币正面出现的概率？

a Maximum likelihood	Coin A	Coin B
B H T T T H H T H T H		5 H, 5 T
A H H H H T H H H H H	9 H, 1 T	
A H T H H H H H T H H	8 H, 2 T	
B H T H T T T H H T T		4 H, 6 T
A T H H H T H H H T H	7 H, 3 T	
5 sets, 10 tosses per set		24 H, 6 T 9 H, 11 T
		$\hat{\theta}_A = \frac{24}{24 + 6} = 0.80$
		$\hat{\theta}_B = \frac{9}{9 + 11} = 0.45$

b Expectation maximization



CASE a

已知每个实验选择的是硬币A 还是硬币 B， 重点是如何计算输出的概率分布，这其实也是极大似然求导所得。

$$\begin{aligned} \underset{\theta}{\operatorname{argmax}} \log P(Y|\theta) &= \log((\theta_B^5(1-\theta_B)^5)(\theta_A^9(1-\theta_A))(\theta_A^8(1-\theta_A)^2)(\theta_B^4(1-\theta_B)^6)(\theta_A^7(1-\theta_A)^3)) \\ &= \log[(\theta_A^{24}(1-\theta_A)^6)(\theta_B^9(1-\theta_B)^{11})] \end{aligned}$$

上面这个式子求导之后发现，5次实验中A正面向上的次数再除以总次数作为即为 $\hat{\theta}_A$ ， 5次实验中B正面向上的次数再除以总次数作为即为 $\hat{\theta}_B$ ， 即：

$$\hat{\theta}_A = \frac{24}{24+6} = 0.80$$

$$\hat{\theta}_B = \frac{9}{9+11} = 0.45$$

CASE b

由于并不知道选择的是硬币 A 还是硬币 B，因此采用EM算法。

E步：初始化 $\hat{\theta}_A^{(0)} = 0.60$ 和 $\hat{\theta}_B^{(0)} = 0.50$ ，计算每个实验中选择的硬币是 A 和 B 的概率，例如第一个实验中选择 A 的概率为：

$$P(z = A|y_1, \theta) = \frac{P(z = A, y_1|\theta)}{P(z = A, y_1|\theta) + P(z = B, y_1|\theta)} = \frac{(0.6)^5 * (0.4)^5}{(0.6)^5 * (0.4)^5 + (0.5)^{10}} = 0.45$$

$$P(z = B|y_1, \theta) = 1 - P(z = A|y_1, \theta) = 0.55$$

计算出每个实验为硬币 A 和硬币 B 的概率，然后进行加权求和。

M步：求出似然函数下界 $Q(\theta, \theta^i)$ ， y_j 代表第 j 次实验正面朝上的个数， μ_j 代表第 j 次实验选择硬币 A 的概率， $1 - \mu_j$ 代表第 j 次实验选择硬币 B 的概率。

$$\begin{aligned} Q(\theta, \theta^i) &= \sum_{j=1}^5 \sum_z P(z|y_j, \theta^i) \log P(y_j, z|\theta) \\ &= \sum_{j=1}^5 \mu_j \log(\theta_A^{y_j} (1 - \theta_A)^{10-y_j}) + (1 - \mu_j) \log(\theta_B^{y_j} (1 - \theta_B)^{10-y_j}) \end{aligned}$$

针对L函数求导来对参数求导，例如对 θ_A 求导：

$$\begin{aligned} \frac{\partial Q}{\partial \theta_A} &= \mu_1 \left(\frac{y_1}{\theta_A} - \frac{10 - y_1}{1 - \theta_A} \right) + \dots + \mu_5 \left(\frac{y_5}{\theta_A} - \frac{10 - y_5}{1 - \theta_A} \right) = \mu_1 \left(\frac{y_1 - 10\theta_A}{\theta_A(1 - \theta_A)} \right) + \dots + \mu_5 \left(\frac{y_5 - 10\theta_A}{\theta_A(1 - \theta_A)} \right) \\ &= \frac{\sum_{j=1}^5 \mu_j y_j - \sum_{j=1}^5 10\mu_j \theta_A}{\theta_A(1 - \theta_A)} \end{aligned}$$

求导等于 0 之后就可得到图中的第一次迭代之后的参数值：

$$\hat{\theta}_A^{(1)} = 0.71$$

$$\hat{\theta}_B^{(1)} = 0.58$$

当然，基于Case a 我们也可以用一种更简单的方法求得：

$$\hat{\theta}_A^{(1)} = \frac{21.3}{21.3 + 8.6} = 0.71$$

$$\hat{\theta}_B^{(1)} = \frac{11.7}{11.7 + 8.4} = 0.58$$

第二轮迭代：基于第一轮EM计算好的 $\hat{\theta}_A^{(1)}, \hat{\theta}_B^{(1)}$ ，进行第二轮 EM，计算每个实验中选择的硬币是 A 和 B 的概率（E步），然后在计算M步，如此继续迭代……迭代十步之后
 $\hat{\theta}_A^{(10)} = 0.8, \hat{\theta}_B^{(10)} = 0.52$

- 迭代一定会收敛，但不一定会收敛到真实的参数值，因为可能会陷入局部最优。所以 EM算法的结果很受初始值的影响。
- 应用：
k-means聚类的迭代算法实际上就是EM算法的体现。在k-means中的隐含变量是每个样本所属类别。每次确认中心点后重新标记是E步，根据标记重新求中心点是M

GMM (参数模型聚类)

<https://zhuanlan.zhihu.com/p/50686800>

层次聚类 (Hierarchical Clustering)

与k-means相比，层次聚类可以避免K值选择和初始聚类中心点选择的问题。
它是通过计算不同类别的相似度类创建一个有层次的嵌套的树。有两种方法，分裂法和凝聚法。分裂法是从上向下把大的类别分割；凝聚法是从下向上把小的类别聚合。

- **分裂法**

输入：样本集合D，聚类数目或者某个条件（一般是样本距离的阈值，这样就可不设置聚类数目）

输出：聚类结果

1. 将样本集中的所有的样本归为一个类簇；

repeat:

2. 在同一个类簇（计为c）中计算两两样本之间的距离，找出距离最远的两个样本a,b；

3. 将样本a, b分配到不同的类簇c1和c2中；

4. 计算原类簇（c）中剩余的其他样本点和a, b的距离，若是 $dis(a) < dis(b)$ ，则将样本点归到c1中，否则归到c2中；

until：达到聚类的数目或者达到设定的条件

- **凝聚法（使用较多）**

输入：样本集合D，聚类数目或者某个条件（一般是样本距离的阈值，这样就可不设置聚类数目）

输出：聚类结果

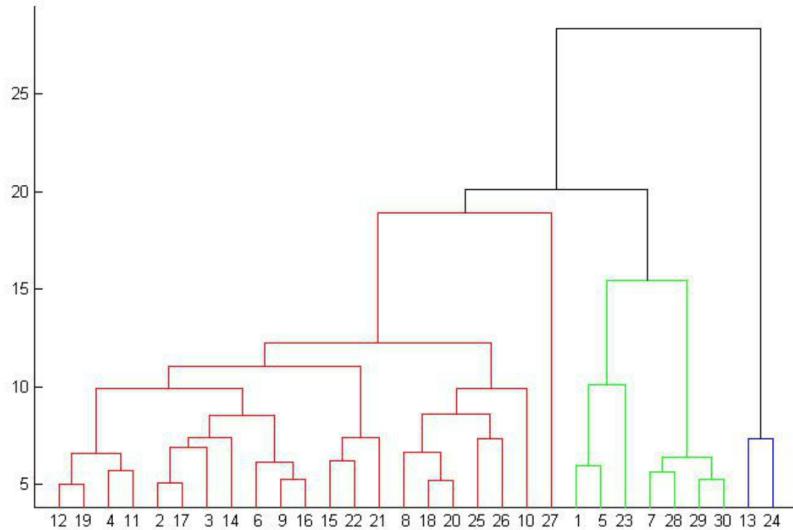
1. 将样本集中的所有的样本点都当做一个独立的类簇；

repeat:

2. 计算两两类簇之间的距离，找到距离最小的两个类簇c1和c2；

3. 合并类簇c1和c2为一个类簇；

until: 达到聚类的数目或者达到设定的条件



• 类簇间距离

- 1) Single-Linkage (最小距离) : 又叫做 nearest-neighbor , 就是取两个类中距离最近的两个样本的距离作为这两个集合的距离，也就是说，最近两个样本之间的距离越小，这两个类之间的相似度就越大。容易造成一种叫做 Chaining 的效果，两个 cluster 明显从“大局”上离得比较远，但是由于其中个别的点距离比较近就被合并了，并且这样合并之后 Chaining 效应会进一步扩大，最后会得到比较松散的 cluster 。
- 2) Complete-Linkage (最大距离) : 这个则完全是 Single Linkage 的反面极端，取两个集合中距离最远的两个点的距离作为两个集合的距离。其效果也是刚好相反的，限制非常大，两个 cluster 即使已经很接近了，但是只要有不配合的点存在，就顽固到底，老死不相合并，也是不太好的办法。这两种相似度的定义方法的共同问题就是指考虑了某个有特点的数据，而没有考虑类内数据的整体特点。
以上两种趋向于对离群点或噪声数据过分敏感。
- 3) Average-linkage (类平均距离) : 这种方法就是把两个集合中的点两两的距离全部放在一起求一个平均值，相对也能得到合适一点的结果。average-linkage 的一个变种就是取两两距离的中值，与取均值相比更加能够解除个别偏离样本对结果的干扰。
- 4) 均值距离：两个簇的平均值作为中心点，取这两个均值之间的距离作为两个簇的距离。

使用均值距离和平均距离是对最小和最大距离之间的一种折中方法，而且可以克服离群点敏感性问题。尽管均值距离计算简单，但是平均距离也有它的优势，因为它既能处理数值数据又能处理分类数据。

5)

- **优缺点：**

优点：

- 1) 一次性地得到了整个聚类的过程，只要得到聚类树，想要分多少个cluster都可以直接根据树结构来得到结果，改变 cluster数目不需要再次计算数据点的归属。
- 2) 距离和规则的相似度容易定义，限制少。
- 3) 可以发现类的层次关系
- 4) 可以聚类成其他形状

缺点：

- 1) 计算复杂度太高
- 2) 容易聚类成链状

Model Evaluation

像前面提到的类别不平衡问题，准确度这个评价指标在类别不均衡的分类任务中并不能work，甚至进行误导（分类器不work，但是从这个指标来看，该分类器有着很好的评价指标得分）。

这里推荐指标：

混淆矩阵(Confusion Matrix)：使用一个表格对分类器所预测的类别与其真实的类别的样本统计，分别为：TP、FN、FP与TN。

精确度(Precision)

召回率(Recall)

F1得分(F1 Score)：精确度与找召回率的加权平均。

Kappa (Cohen kappa)

ROC曲线(ROC Curves)：见Assessing and Comparing Classifier Performance with ROC Curves

Cross-Validation

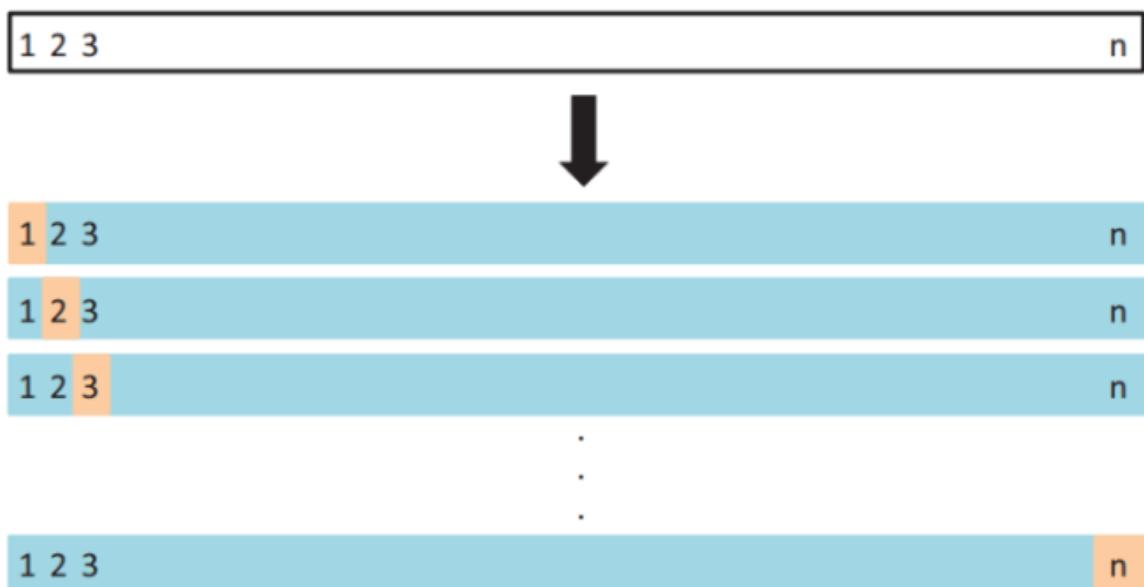
普通的训练集，测试集划分有两个问题，首先最终模型与参数的选取将极大程度依赖于你对训练集和测试集的划分方法，在不同的划分方法下，test MSE的变动是很大的，而且对应的最优degree也不一样。所以如果我们的训练集和测试集的划分方法不够好，很有可能无法选择到最好的模型与参数。另一个问题是，该方法只用了部分数据进行模型的训练，当用于模型训练的数据量越大时，训练出来的模型通常效果会越好。所以训练集和测试集的划分意味着我们无法充分利用我们手头已有的数据，所以得到的模型效果也会受到一定的影响。

交叉验证 (Cross validation)解决了这种问题，交叉验证用于防止模型过于复杂而引起的过拟合。有时亦称循环估计，是一种统计学上将数据样本切割成较小子集的实用方法。于是可以先在一个子集上做分析，而其它子集则用来做后续对此分析的确认及验证。一开始的子集被称为训练集。

但是请注意，这里调的参数不是指算法中的“参数”比如线性模型的权重，而是一些模型特有的**超参数**，比如SVM的核函数选择、集成模型比如随机森林的最小分割子树之类的。

即交叉验证并不是为了去获取一个模型，他的主要目的是为了去评估这个模型的一些特性，（这也是我看了统计学习导论之后，慢慢在脑子里形成的一个认识。）所以说，你这个最后要选取的模型，这些个都不是，而应该是去采用全部数据来训练出来的模型。

LOOCV (Leave-one-out cross-validation)



如上图所示，假设我们现在有n个数据组成的数据集，那么LOOCV的方法就是每次取出一个数据作为测试集的唯一元素，而其他n-1个数据都作为训练集用于训练模型和调参。结果就是我们最终训练了n个模型，每次都能得到一个MSE。而计算最终test MSE则就是将这n个MSE取平均。

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \text{MSE}_i.$$

比起test set approach，LOOCV有很多优点。首先它不受测试集和训练集划分方法的影响，因为每一个数据都单独的做过测试集。同时，其用了n-1个数据训练模型，也几乎用到了所有的数据，保证了模型的bias更小。不过LOOCV的缺点也很明显，那就是计算量过于大，是test set approach耗时的n-1倍。

为了解决计算成本太大的弊端，又有人提供了下面的式子，使得LOOCV计算成本和只训练一个模型一样快。

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2$$

其中 \hat{y}_i 表示第*i*个拟合值，而 h_i 则表示leverage。关于 h_i 的计算方法详见线性回归的部分

K-fold Cross Validation

另外一种折中的办法叫做K折交叉验证，和LOOCV的不同在于，我们每次的测试集将不再只包含一个数据，而是多个，具体数目将根据K的选取决定。比如，如果K=5，那么我们利用五折交叉验证的步骤就是：

1. 将所有数据集分成5份
2. 不重复地每次取其中一份做测试集，用其他四份做训练集训练模型，之后计算该模型在测试集上的 MSE_i
3. 将5次的 MSE_i 取平均得到最后的MSE

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k MSE_i.$$

不难理解，其实LOOCV是一种特殊的K-fold Cross Validation (K=N)。

Bias-Variance Trade-Off for k-Fold Cross-Validation

最后，我们要说说K的选取。事实上，和开头给出的文章里的部分内容一样，K的选取是一个 Bias和Variance的trade-off。

K越大，每次投入的训练集的数据越多，模型的Bias越小。但是K越大，又意味着每一次选取的训练集之前的相关性越大（考虑最极端的例子，当k=N，也就是在LOOCV里，每次的训练数据几乎是一样的）。而这种大相关性会导致最终的test error具有更大的Variance。

一般来说，根据经验我们一般选择k=5或10。

Cross-Validation on Classification Problems

上面我们讲的都是回归问题，所以用MSE来衡量test error。如果是分类问题，那么我们可以用以下式子来衡量Cross-Validation的test error：

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n Err_i$$

其中 Err_i 表示的是第*i*个模型在第*i*组测试集上的分类错误的个数。

样本量较少时的交叉验证

数据量较少时可以用LOOCV。

此外还有一种比较特殊的交叉验证方式，也是用于样本量少的时候。叫做自助法(bootstrapping)。比如我们有m个样本(m较小)，每次在这m个样本中随机采集一个样本，放入训练集，采样完后把样本放回。这样重复采集m次，我们得到m个样本组成的训练集。当然，这m个样本中很有可能有重复的样本数据。同时，用原始的m个样本做测试集。这样接着进行交叉验证。由于我们的训练集有重复数据，这会改变数据的分布，因而训练结果会有估计偏差，因此，此种方法不是很常用，除非数据量真的很少，比如小于20个。

混淆矩阵

		Actual class	
		positive class	negative class
Predicted class	positive class	True Positive(TP)	False Positive(FP)
	negative class	False Negative(FN)	True Negative(TN)

在sklearn中，二分类问题下的混淆矩阵需要分别将表1中的predicted class和Actual class对调，将横纵坐标的positive class和negative class都分别对调，再重新计算混淆矩阵。

通过混淆矩阵，我们可以很直观地看清一个模型在各个类别(positive和negative)上分类的情况。

表2：TP、FP、FN、TN

TP	真实类别为positive，模型预测的类别也为positive
FP	预测为positive，但真实类别为negative，真实类别和预测类别不一致
FN	预测为negative，但真实类别为positive，真实类别和预测类别不一致
TN	真实类别为negative，模型预测的类别也为negative

TP、FP、TN、FN，第二个字母表示样本被预测的类别，第一个字母表示样本的预测类别与真实类别是否一致。

准确率Accuracy/FP rate/TP rate

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{\text{all data}}$$

在数据集**不平衡**时，准确率将不能很好地表示模型的性能。可能会存在准确率很高，而少数类样本全分错的情况，此时应选择其它模型评价指标。

True Positive Rate (真正率, TPR) 或灵敏度 (sensitivity)

$$TPR = TP / (TP + FN)$$

正样本预测结果数 / 正样本实际数

True Negative Rate (真负率, TNR) 或特指度 (specificity)

$$TNR = TN / (TN + FP)$$

负样本预测结果数 / 负样本实际数

False Positive Rate (假正率, FPR)

$$FPR = FP / (FP + TN)$$

被预测为正的负样本结果数 / 负样本实际数

False Negative Rate (假负率, FNR)

$$FNR = FN / (TP + FN)$$

被预测为负的正样本结果数 / 正样本实际数

Average Class Accuracy

TP rate与TN rate的平均值

不平衡问题对这个值的影响较大，可以改用几何平均值

F1值/F β 值

精确率（查准率）和召回率（查全率）

$$\text{precision} = \frac{TP}{TP + FP} = \frac{TP}{\text{预测为positive的样本}}$$

$$\text{recall} = \frac{TP}{TP + FN} = \frac{TP}{\text{真实为positive的样本}}$$

positive class的精确率表示在预测为positive的样本中真实类别为positive的样本所占比例；精准率低表示被预测为True的人中有多少个真的True。如果这个数值高，则表示被预测为True的样本基本都真的是True。

positive class的召回率表示在真实为positive的样本中模型成功预测出的样本所占比例。召回率高表示，真的高的样本，基本都被预测出来了。所以对于一些代价敏感的分类问题，如疾病验证，则希望召回率高。

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F1值就是精确率和召回率的调和平均值，F1值认为精确率和召回率一样重要。

$$F_\beta = \frac{1 + \beta^2}{\frac{1}{\text{precision}} + \frac{\beta^2}{\text{recall}}} = \frac{(1 + \beta^2) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

在 $\beta=1$ 时， F_β 就是 F_1 值，此时 F_β 认为精确率和召回率一样重要；当 $\beta>1$ 时， F_β 认为召回率更重要；当 $0<\beta<1$ 时， F_β 认为精确率更重要。除了 F_1 值之外，常用的还有 F_2 和 $F_{0.5}$ 。

ROC曲线及其AUC值

ROC描述了所有可能threshold阈值（0到1）的情况

ROC曲线的纵坐标**True Positive Rate (TPR)** 在数值上就等于positive class的recall，记作recall_{positive}，横坐标**False Positive Rate (FPR)** 在数值上等于(1 - negative class的recall)，记作(1 - recall_{negative})如下所示：

$$\begin{aligned} \text{TPR} &= \frac{TP}{TP + FN} \\ &= \text{recall}_{\text{positive}} \end{aligned}$$

$$\begin{aligned} \text{FPR} &= \frac{FP}{FP + TN} = \frac{FP + TN - TN}{FP + TN} \\ &= 1 - \frac{TN}{FP + TN} \\ &= 1 - \text{recall}_{\text{negative}} \end{aligned}$$

通过对分类阈值 θ （默认0.5）从大到小或者从小到大依次取值，我们可以得到很多组TPR和FPR的值，将其在图像中依次画出就可以得到一条ROC曲线，阈值 θ 取值范围为[0,1]。

ROC曲线在图像上越接近左上角(0,1)模型越好，即ROC曲线下面与横轴和直线FPR = 1围成的面积（AUC值）越大越好。直观上理解，纵坐标TPR就是recall_{positive}值，横坐标FPR就是(1 - recall_{negative})，前者越大越好，后者整体越小越好，在图像上表示就是曲线越接近左上角(0,1)坐标越好。

要知道哪个模型更好，则需要计算每条曲线的AUC值，一般认为AUC值越大越好。AUC值由定义通过计算ROC曲线、横轴和直线 $FPR = 1$ 三者围成的面积即可得到。

多分类的ROC

- 1.如果已经掌握了二分类的套路的话，你可以把多分类分成n个二分类来解决。
- 2.如a、b、c三分类问题我可以转化为a与其他类别、b与其他类别、c与其他类别。
- 3.如果想要一个总体评估效果的话再把他们加和就可以了 你也可以得到一个混淆矩阵。同理roc曲线你也可以画出来。
另外多分类问题最好按照第二点分别都看看，因为有可能你总体的效果好，仅仅是因为有一部分类别预测的很准，另外一些类别根本分不出来，个人觉得还是很有必要的。

可解释性

<https://www.jiqizhixin.com/articles/2019-10-30-9>

泛化Generalization & 稳定性Stability

在模型算法选择环节：选择稳定性较好的模型，例如LR、随机森林、xgboost这类泛化能力较好的模型。值得注意的是，模型的区分能力和稳定性是一对矛盾体，过分追求稳定性，一定会牺牲一部分区分度。如何在两者中平衡，需要在特定业务环境下，做出合理的决定。另：经过实践可知。使用传统LR评分模型的稳定性优于其他算法。这是由于LR是每个特征的线性组合，它的稳定性依据多个特征，而xgb的稳定性更多是取决于前几颗树的稳定性，如果第一颗树的特征出现问题，其它都受到影响。

Deep Learning

补充知识点

Vapnik–Chervonenkis (VC) dimension

<https://sunoonlee.github.io/2017/07/generalization-error-bound/>

泛化能力是机器学习的关键. 我们利用数据来训练模型, 目的不是得到最小的训练误差, 而是让模型在一般情形下表现较好.

为了衡量模型表现, 我们需要定义误差的度量 (Error Measure). 跟常说的损失函数/目标函数等名词也是一个意思. 我们按照 Learning From Data 里的方式定义三种误差:

- E_{in} : in-sample error, 训练误差
- E_{out} : out-of-sample error, 训练集外的误差, 是在整个输入空间上的误差期望.
- $E_{out} - E_{in}$: 泛化误差, 表示由训练集泛化至训练集外的过程中产生的误差. 不过, 也有人用“泛化误差”表示 E_{out} .

我们的目标是让 E_{out} 尽量小, 这样模型在遇到新鲜数据时才能有好的表现, 这样的模型才是有用的. 但直接优化 E_{out} 不现实, 而 E_{in} 的优化相对容易, 那么在此基础上, 我们只需要使 E_{out} 尽量接近 E_{in} , 也就是尽量减小泛化误差.

泛化误差能否达到足够小, 决定了机器学习的可行性. 统计学习理论对这个问题给出了一些解答, 证明这种可行性在概率意义上成立.

Hoeffding 不等式

先来回顾一下大数定律: 样本数量越多, 随机变量的样本均值就越接近期望值. 统计学习理论利用了大数定律的一种特殊形式: Hoeffding 不等式.

$$P[|E_{out} - E_{in}| > \epsilon] \leq 2e^{-2\epsilon^2 N}, \forall \epsilon > 0$$

就是说, 当数据量 N 足够大时, 泛化误差与训练集误差有很大概率会非常接近. 听起来很美, 是不是泛化问题就这样解决了呢?

可惜不然. 以上不等式只适用于事先确定的单一假设函数. 而在机器学习过程中, 最终的那个假设函数 g 是从一个比较大的集合里选出来的. 如果集合大小为 M , 对集合内不同的假设函数取 union bound, 那么不等式就变成了:

$$P[|E_{out}(g) - E_{in}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}, \forall \epsilon > 0$$

多了一个 M , 结果就完全变味了. 要知道, 一般的机器学习模型, M 都是无穷大, 因此不等式右边会很大, 作为一个概率的限值已经没有意义了.

“有效”假设函数

幸运的是, 上面这个 M 取得很保守, 实际情况没有这么坏. 在无穷多的假设函数里, “有效”数量往往是有限的. 这种“有效性”表现在假设函数作用于数据集时的效果. 当多个假设函数在数据集上得到的分类结果相同时 (比如, 对三个数据点, 分类结果都是“正正负”), 有效数量为 1. (这样一个“有效假设函数”叫做 dichotomy.)

(以下都是就二分类问题而言) 假设函数“有效”数量的上限是 $2N$, 达到这个上限时, 意味着假设函数集合 H 能够穷尽 N 个数据点分类的所有可能性 (这时可以说 H shatter 了数据点).

VC 维

为了抽象出一个表达模型复杂度的量, V 同学和 C 同学一起提出了 VC 维 (VC dimension), 即: 一个假设函数集合 H 最多能 shatter 多少数据点.

以二维平面的感知机为例: 对三个不共线的点, 它总是能给出所有的分类可能, 但对四个点就办不到了. 因此二维感知机的 VC 维是 3. 更一般地, d 维感知机的 VC 维等于 $d+1$. VC 维可以看做对模型有效参数数量或自由度的一种度量.

泛化误差上界

借助“有效”假设函数和 VC 维的概念, 经过一些推导, 可以大幅减小前面不等式右端的概率限值. 选定一个 tolerance level δ , 则下式以不小于 $1-\delta$ 的概率成立:

$$E_{out} \leq E_{in} + \Omega(N, H, \delta)$$

其中,

- $\Omega(N, H, \delta) = \sqrt{\frac{8}{N} \ln \frac{4m_H(2N)}{\delta}}$.
- $m_H(N)$ 是“增长函数”, 表示任选 N 个点时, H 能得到的最大的 dichotomy 数量.

由于增长函数一般不太容易计算, 可以进一步用 VC 维表示它的上界:

$$m_H(N) \leq N^{d_{vc}} + 1$$

$$\text{于是, } \Omega(N, H, \delta) \leq \sqrt{\frac{8}{N} \ln \frac{4((2N)^{d_{vc}} + 1)}{\delta}}$$

以上就是历经千辛万苦得到的一个泛化误差上界, 这个结论非常宝贵, 因为它证明了机器学习在理论上的可行性. 不过, 因为这个上界具有广泛的一般性, 推导过程中多次放大, 最终得到的上界非常松, 实际应用的意义主要在其相对值而非绝对值.

可以看出影响泛化误差的两个因素: 数据量和模型复杂度. 数据量不必多说, 自然多多益善. 模型复杂度的影响更微妙一些: 当复杂度增加时, 一般 E_{in} 会减小, 而泛化误差 Ω 项会增大; 为了得到最优的 E_{out} , 需要两者之间达到一种平衡.

Bias-Variance Tradeoff

<http://work.caltech.edu/library/081.html>

在上一小节, 我们求出了最小误差的上界, 在这里我要提一下决策树ID3算法, 不仅由于剃刀原理, 而且由于我们想减小泛化误差, 所以在 E_{in} 相同时, 我们选用复杂度低的模型, 减小泛化误差.

泛化误差又由偏差、方差、噪声三部分构成。

泛化误差: E_{out} 的公式为

$$R_{exp} = E[(y_i - \hat{y})^2]$$

偏差: 度量了模型的期望预测和真实结果的偏离程度, 刻画了模型本身的拟合能力。

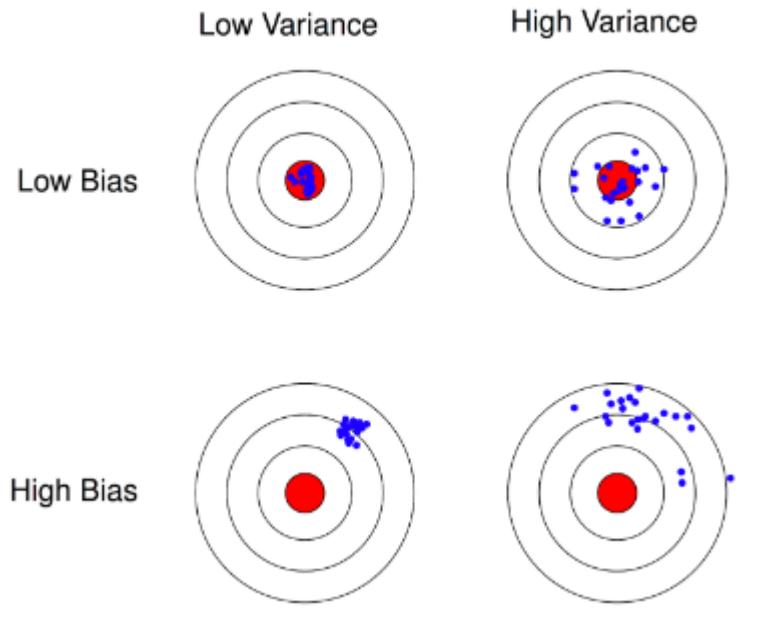
$$bias^2(x) = (\bar{y} - y)^2$$

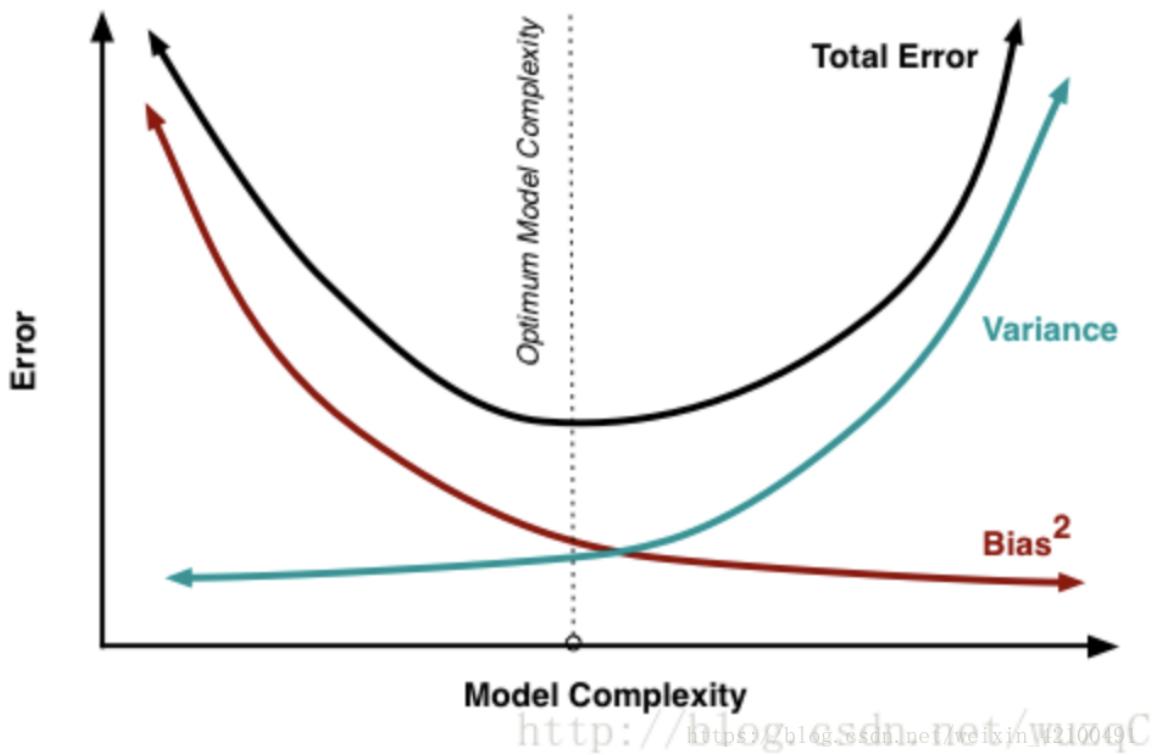
方差：度量了同样大小的训练集的变动所导致的学习性能的变化，即刻画了数据扰动所造成的影响。

$$var(x) = E[(y_i - \bar{y})^2]$$

噪声：表达了当前任务上任何模型所能达到的期望泛化误差的下界，刻画了学习问题本身的难度。

$$noise^2 = (y - y')^2$$





证明：blog.csdn.net/SYSU_BOND/article/details/79933418

$$R_{exp} = var(x) + bias^2(x) + noise^2$$

方差反映了函数 $f(X)$ 对于训练样本 X 的敏感程度，较低的方差则意味着模型对于不同样本的输出更加稳定，即在相同的采样样本下其输出的变动更小。而偏差则反映了分类器对输入样本的估计期望与真实结果的接近程度。因此对于某一模型的期望泛化误差其由偏差、方差共同决定。如果我们期望模型对样本的预测误差越小，则要求模型的偏差越小，然而较小的偏差将导致模型的方差增加，这也就造成了bias-variance trade-off。

对于给定的偏差其方差将随着训练样本的增加而减小。因此在大量样本的前提下，学习任务即转化为使模型的偏差较小，在这样的策略下某些算法取得了不错的效果。然而对于一些分类问题的模型，该策略并不管用如朴素贝叶斯和最近邻方法，其在较高的偏差下仍然有很强的竞争力。

那么我们如何缓解这一矛盾呢？一般的做法是采用Automatic\Data Driven的方法（剪枝、正则等等）。此外我们还可以增加样本数目（理论泛化误差的上界与样本数目有关，样本数目越多则泛化误差越小，当样本数目趋于无穷时，误差也将趋于0），以及选择合适的样本（不同的算法对于样本分布存在不同的偏好）缓解偏差-方差窘境（然而这种方法在现实问题中很难被采用）。

笔试/面试题

1. 过拟合是什么原因造成的，有哪几个方面？

模型层面，样本不均衡，维度过多，数据样本太少

正则化

规则化就是向你的模型加入某些规则，加入先验，缩小解空间，减小求出错误解的可能性。你要把你的知识数学化告诉这个模型，对代价函数来说，就是加入对模型“长相”的惩罚。规则化就是说给需要训练的目标函数加上一些规则（限制），让他们不要自我膨胀。防止了过拟合。正则化大概有两个功能：1，从模型修正上看，起了一个trade-off作用，用于平衡学习过程中两个基本量，名字诸如bias-variance、拟合能力-泛化能力、损失函数-推广能力、经验风险-结构风险等等；2，从模型求解上看，正则化提供了一种唯一解的可能，众所周知，光用最小二乘拟合可能出现无数组解，加个L1或L2正则化项能有唯一解，即不稳定性。

<https://www.zhihu.com/question/20924039>

L1 和 L2 是最常见的正则化类型，它们会向代价函数中添加一个正则项，更新通用的代价函数。代价函数=损失（比如二元交叉熵）+正则项由于添加了正则项，权重矩阵的值会下降，因为它是假设神经网络具有更小的权重矩阵，从而使模型更简单。因此，这样也会大幅降低过拟合。不过，L1 和 L2 中的正则项有所不同。

在 L2 中，我们有：

$$\text{Cost function} = \text{Loss} + \frac{\lambda}{2m} * \sum \|w\|^2$$

在这里， λ 就是正则化参数。它是一个超参数，优化它的值会让模型性能更好。L2 正则化又被称为权值衰减，因为它会迫使权值衰减到 0（但不会真得等于 0）推导为什么。

在 L1 中，我们有：

$$\text{Cost function} = \text{Loss} + \frac{\lambda}{2m} * \sum \|w\|$$

在这里，我们惩罚权重的绝对值。不像 L2，这里的权重可能会被降为 0。所以，如果我们想压缩模型，这会非常有用。除此之外的情况下，我们通常优先选用 L2 正则化。

在 Keras 中，我们可以通过使用 regularizers 在任何一层上直接应用正则化。

下面是向网络中致密层应用 L2 正则化的代码示例：

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01))
```

注意：这里的值 0.01 是正则化参数（即 λ ）的值，我们需要对其进一步优化。我们可以用“网格搜索法”对其优化。

同样地，我们也可以应用 L1 正则化。我们后面会通过研究案例进一步查看详情。

似然函数/最大似然估计

高维度稀疏矩阵

This effect was called the curse of dimensionality by Bellman in 1961 and the name has stuck.

2.4. The curse of dimensionality. Suppose that each component x_i of the input vector $x = (x_1, \dots, x_n)$ has k possible values. We might, for example, divide each dimension into k cells or buckets². Then there are k^n possible values or cells in which x might lie: a number which rises exponentially³ with the number n of dimensions. If we are to accurately model the mapping f we would need a data point in each cell, so the amount of training data also grows exponentially with the number of dimensions. This effect was called the *curse of dimensionality* by Bellman in 1961 and the name has stuck.

(The dimension of the output vector $y = (y_1, \dots, y_m)$ should also be taken into account: a more correct value above is k^{n+m} ; however, as y is often 1-dimensional, we have omitted this.)

If, as in practice, the amount of data is limited, then we will have a very sparse distribution of data in high dimensional spaces, which greatly limits some approaches' ability to represent the function g . Other approaches, such as Monte Carlo techniques or Multi Layer Perceptrons (see below) are less susceptible to this.

Another implication of the sparse distribution of data in high dimensional spaces is that the nearest neighbours of a given data item may actually be quite far away, and so methods — such as the k -nearest neighbour classification method or k NN — which use nearest neighbours may not work well.

相对熵，交叉熵cost function，softmax激活函数关系？

相对熵 ID3信息增益 各种熵

<https://charlesliuyx.github.io/2017/09/11/%E4%BB%80%E4%B9%88%E6%98%AF%E4%BF%A1%E6%81%AF%E7%86%B5%E3%80%81%E4%BA%A4%E5%8F%89%E7%86%B5%E5%92%8C%E7%9B%B8%E5%AF%B9%E7%86%B5/>

交叉熵用来比较两个向量的相似度

对应的标签和预测值

*	猫	青蛙	老鼠
Label	0	1	0
Pred	0.3	0.6	0.1

那么

$$\begin{aligned} loss &= -(0 \times \log(0.3) + 1 \times \log(0.6) + 0 \times \log(0.1)) \\ &= -\log(0.6) \end{aligned}$$

对应一个batch的loss就是

$$loss = -\frac{1}{m} \sum_{j=1}^m \sum_{i=1}^n y_{ji} \log(\hat{y}_{ji})$$

信息熵用来衡量某一个feature的不确定程度，相对熵用来衡量在一个descriptive feature下，target feature的不确定程度。权重×这个类别的target feature的信息熵
交叉熵和相对熵都是对于两个概率分布的，条件熵是对于一个条件下另一个的（如上图，没法说0猫的情况下预测是什么样的）。

最大似然与交叉熵，二分的交叉熵与最大似然估计一样

<https://blog.csdn.net/u012436149/article/details/78006552>

交叉熵 cost function 交叉熵在单分类问题上基本是标配的方法

$$loss = - \sum_{i=1}^n y_i \log(\hat{y}_i)$$

对应的标签和预测值

*	猫	青蛙	老鼠
Label	0	1	0
Pred	0.3	0.6	0.1

那么

$$\begin{aligned} loss &= -(0 \times \log(0.3) + 1 \times \log(0.6) + 0 \times \log(0.1)) \\ &= -\log(0.6) \end{aligned}$$

对应一个batch的loss就是

$$loss = -\frac{1}{m} \sum_{j=1}^m \sum_{i=1}^n y_{ji} \log(\hat{y}_{ji})$$

m为当前batch的样本数

多分类

<https://blog.csdn.net/tsyccnh/article/details/79163834>

统计



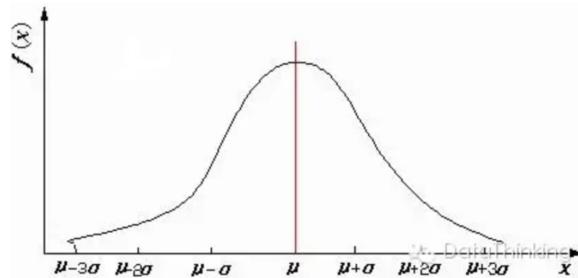
统计基础

分布

- 正态分布

正态分布又名高斯分布，若随机变量 X 服从一个数学期望为 μ ，方差为 σ^2 的高斯分布，记为 (μ, σ^2) 。其概率密度函数为正态分布的期望值 μ 决定了其位置，其标准差 σ 决定了分布的幅度。我们通常所说的标准正态分布是 $\mu = 1$, $\sigma = 1$ 的正态分布。

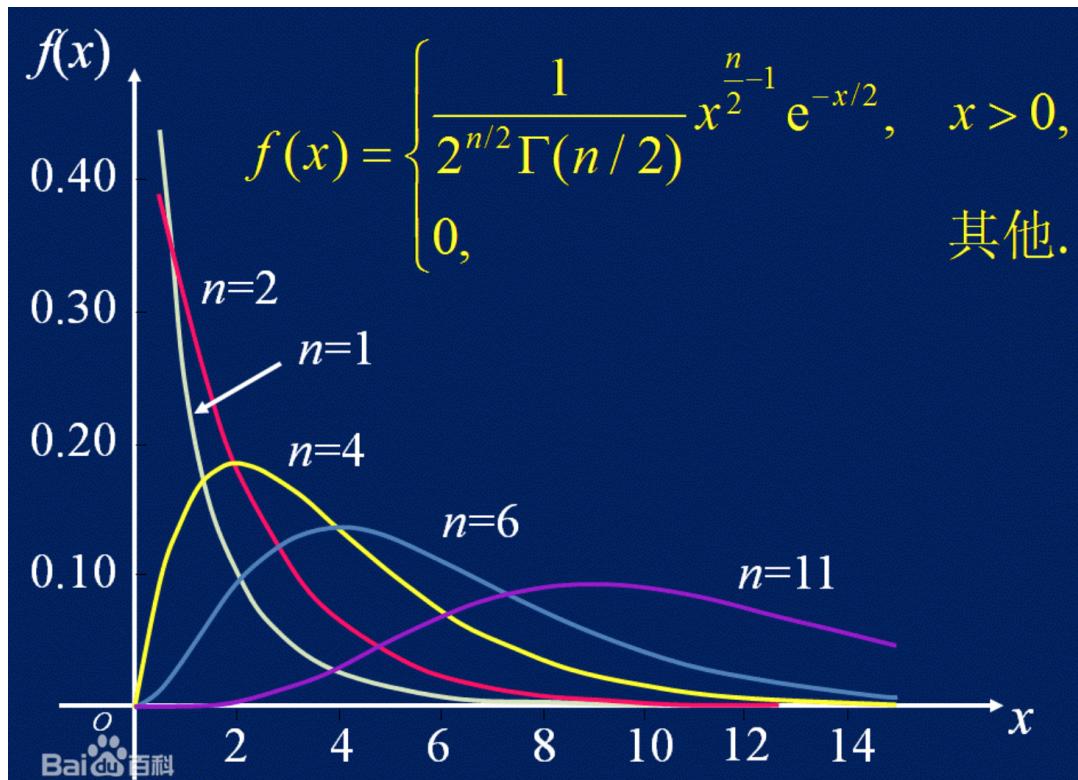
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$



- 卡方分布

设 X_1, X_2, \dots, X_n i.i.d. $\sim N(0, 1)$, 令 $X = \sum_{i=1}^n X_i^2$, 则称 X 是自由度为 n 的 χ^2 变量, 其分布称为自由度为 n 的 χ^2 分布, 记为 $X \sim \chi_n^2$.

<https://blog.csdn.net/anshiyuan/article/details/8007000>



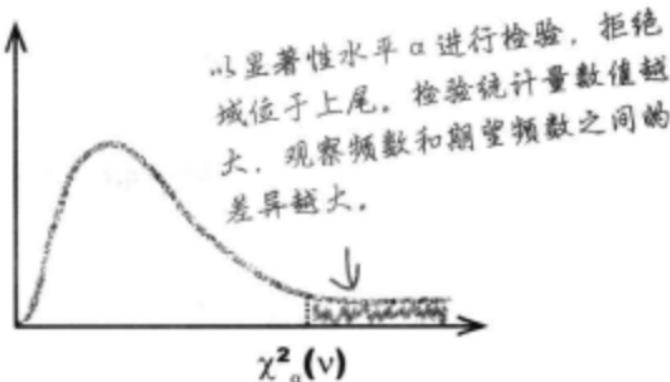
卡方分布的用途:

检查实际结果与期望结果之间何时存在显著差异。

- 1、检验拟合优度：也就是说可以检验一组给定数据与指定分布的吻合程度。
- 2、检验两个变量的独立性：通过这个方法检查变量之间是否存在某种关系。

步骤：

- 1、确定要进行检验的假设 (H_0) 及其备择假设 H_1 .
- 2、求出期望 E 和自由度 V .
- 3、确定用于做决策的拒绝域 (右尾) .
- 4、计算检验统计量.
- 5、查看检验统计量是否在拒绝域内.
- 6、做出决策



计算统计量步骤： (期望频数总和与观察频数总和相等)

1、得出相应的观察频数和期望频数

检验拟合优度：

期望频数=观察频数之和*每种结果的概率

X	-2	23	48	73	98
$P(X = x)$	0.977	0.008	0.008	0.006	0.001

检验变量间独立性：

期望概率求解步骤：

1、算出赌局结果和庄家频数以及各项总和，如下表称为列联表

	庄家A	庄家B	庄家C	合计	赢局次数
庄家A合计	43	49	22	114	
平	8	2	5	15	
赔	47	44	30	121	
合计	98	95	57	250	

↖ 总和

2、算出庄家A的赢局期望。

a、求出赢局概率： $P(\text{赢})=\text{赢局合计}/\text{总和}$

b、庄家A坐庄概率： $P(A)=\text{合计A}/\text{总和}$

c、假设庄家A和赌局结果独立，其坐庄出现赢局概率： $P(A \text{ 坐庄赢局})=P(\text{赢}) \times P(A)$

d、赢局的期望频数=总和* $P(A \text{ 坐庄赢局})$

即：

$$\begin{aligned} \text{期望频数} &= \frac{\text{总和}}{\text{总和}} \times \frac{\text{赢局合计}}{\text{总和}} \times \frac{A \text{ 总计}}{\text{总和}} \\ &= \frac{\text{赢局合计} \times A \text{ 总计}}{\text{总和}} \end{aligned}$$

推广：期望频数= 行合计 \times 列合计 / 总和

2、利用卡方公式计算检验统计量：(O代表观察期望，E代表期望频数)

$$\chi^2 = \sum (O - E)^2 / E$$

自由度计算：

自由度： 取值不受限制的变量的个数

检验拟合优度：类别数减1

检验独立性： $(a-1) * (b-1)$, a,b对应两个检验条件对应的分类数

- T 分布

设随机变量 $X \sim N(0, 1)$, $Y \sim \chi^2_n$, 且 X 和 Y 独立, 则称

$$T = \frac{X}{\sqrt{Y/n}}$$

为自由度为 n 的 t 变量, 其分布称为自由度为 n 的 t 分布, 记为

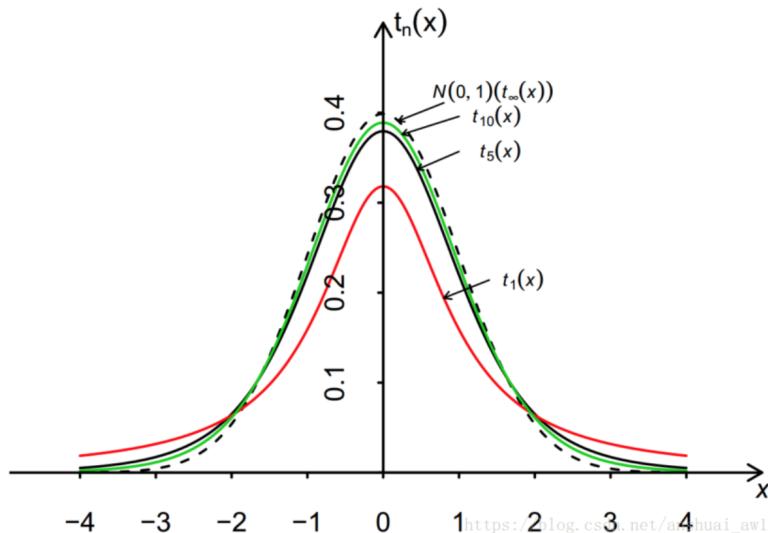
$$T \sim t_n.$$

https://blog.csdn.net/anshuai_awl

设随机变量 $T \sim t_n$, 则其密度函数为

$$t_n(x) = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})\sqrt{n\pi}} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}}, \quad -\infty < x < \infty \quad (4.2)$$

https://blog.csdn.net/anshuai_awl



https://2log.cs3.net/a4huai_awl

单总体t检验

检验一个样本平均数与一个已知的总体平均数差异是否显著。

适用条件: 总体服从正态分布; 样本量小于30

T统计量:

$$t = \frac{\bar{x} - \mu}{S/\sqrt{n}} \sim t(n-1)$$

\bar{x} ——样本均值

μ ——总体均值

S ——样本标准差

n ——样本容量

双总体t检验:

检验两个样本各自所代表的总体的均值差异是否显著, 包括独立样本t检验和配对样本t检验。

◆ 独立样本t检验

检验两个独立样本所代表的总体均值差异是否显著。

适用条件：

1. 两样本均来自于正态总体
2. 两样本相互独立
3. 满足方差齐性（两总体方差相等）

POOLED T-TEST

1. X normally distributed variable
2. Independent simple random samples
3. Unknown population standard deviations
4. Population standard deviations are equal

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_{\bar{x}_1 - \bar{x}_2})}{(s_{\bar{x}_1 - \bar{x}_2})}$$

$$\mu_{\bar{x}_1 - \bar{x}_2} = \mu_1 - \mu_2$$

$$s_{\bar{x}_1 - \bar{x}_2} = s_p \sqrt{(1/n_1) + (1/n_2)}$$

$$df = n_1 + n_2 - 2$$

NON-POOLED T-TEST

1. X normally distributed variable
2. Independent simple random samples
3. Unknown population deviations
4. Population standard deviations are different

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_{\bar{x}_1 - \bar{x}_2})}{(s_{\bar{x}_1 - \bar{x}_2})}$$

$$\mu_{\bar{x}_1 - \bar{x}_2} = \mu_1 - \mu_2$$

$$s_{\bar{x}_1 - \bar{x}_2} = \sqrt{(s_1^2/n_1) + (s_2^2/n_2)}$$

$$df = [\dots]$$

❖ 配对样本t检验

检验两个配对样本所代表的总体均值差异是否显著。

配对样本主要包含以下两种情形：

1. 同源配对，也就是同质的对象分别接受两种不同的处理。例如：为了验证某种记忆方法对改善儿童对词汇的记忆是否有效，先随机抽取40名学生，再随机分为两组。一组使用该训练方法，一组不使用，三个月后对这两组的学生进行词汇测验，得到数据。问该训练方法是否对提高词汇记忆量有效？
2. 自身配对

某组同质对象接受两种不同的处理。例如：某公司推广了一种新的促销方式，实施前和实施后分别统计了员工的业务量，得到数据。试问这种促销方式是否有效？

适用条件：

每对数据的差值必须服从正态分布

统计量：

$$t = \frac{\bar{x}_d}{S_d / \sqrt{n}}$$

两配对样本对应元素做差后形成的新样本

\bar{x}_d ——新样本均值

S_d ——新样本标准差

n ——新样本容量

- F 分布

设随机变量 $X \sim \chi_m^2$, $Y \sim \chi_n^2$, 且 X 和 Y 独立, 则称

$$F = \frac{X/m}{Y/n}$$

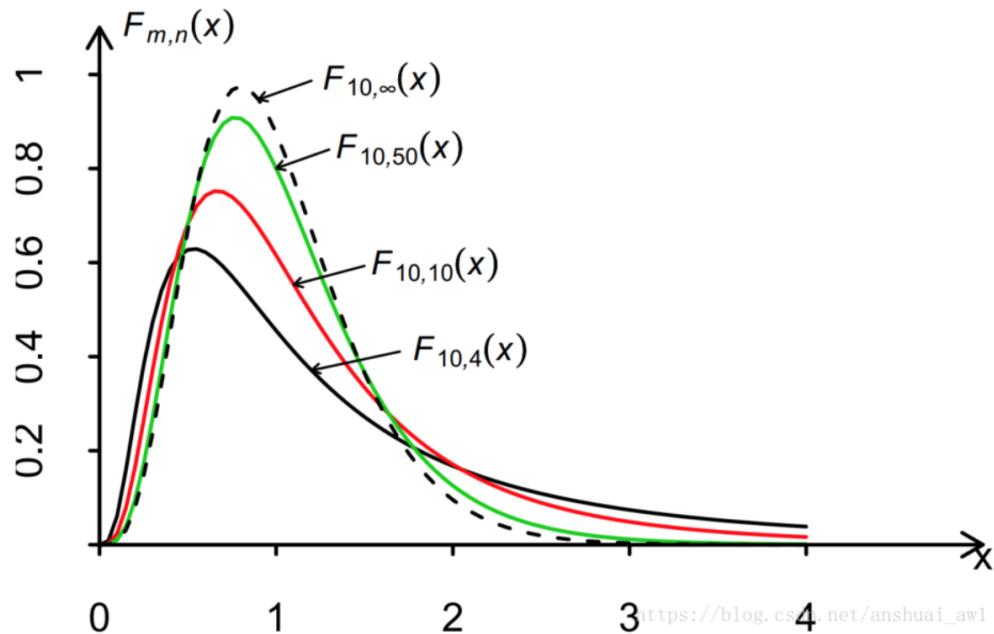
为自由度分别是 m 和 n 的 F 变量, 其分布称为自由度分别是 m 和 n 的 F 分布, 记为 $F \sim F_{m,n}$

若随机变量 $Z \sim F_{m,n}$, 则其密度函数为

$$f_{m,n}(x) = \begin{cases} \frac{\Gamma(\frac{m+n}{2})}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})} m^{\frac{m}{2}} n^{\frac{n}{2}} x^{\frac{m}{2}-1} (n+mx)^{-\frac{m+n}{2}}, & x > 0, \\ 0, & \text{其它.} \end{cases}$$

https://blog.csdn.net/anshuai_awl (4.3)

自由度为 m, n 的 F 分布的密度函数如下图:



- 二项分布

二项分布的基本描述:

在概率论和统计学里面, 带有参数 n 和 p 的二项分布表示的是 n 次独立试验的成功次数的概率分布。在每次独立试验中只有取两个值, 表示成功的值的概率为 p , 那么表示试验不成功的概率为 $1-p$ 。这样一种判断成功和失败的二值试验又叫做伯努利试验。特殊地, 当 $n=1$ 的时候, 我们把二项分布称为伯努利分布。

二项分布频繁地用于对以下描述的一种实验进行建模：从总数量大小为N的两个事物中进行n次放回抽样，以某一事物为基准，计算成功抽取这个事物的次数的概率。要注意的是必须进行的是放回抽样，对于不放回抽样我们一般用超几何分布来对这样的实验进行建模。

二项分布的概率质量函数：

一般来说，如果一个随机变量X满足二项分布的话，那么它一定有一个参数 $n \in \mathbb{N}$ 且还有一个参数 $p \in [0,1]$ 。这样的话，我们可以把关于X的二项分布写成 $X \sim B(n, p)$ 。对应的概率质量函数如下。

$$Pr(k; n, p) = \Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

这里 $k = 0, 1, 2, \dots, n$ ，并且原括号是组合的表示形式，组合的计算公式如下。

$$\binom{n}{k} = \frac{n!}{k!(n - k)!}$$

这个公式表示的从n个数中取k个数构成一个组合能有多少种不同的取法。整个二项分布我们可以描述为求n次独立的伯努利试验，成功k次的概率是多少。由于一次成功的概率是已知的，因此我们必须求出n次试验中，成功k次可能发生在哪次试验中，一共有多少种可能都要求出来，因此求的就是n取k组合数目。

The mean of a binomial random variable X , μ_X , is defined (and computed) as follows

$$\mu_X = np$$

The variance of a binomial random variable X , σ^2_X , is defined (and computed) as follows

$$\sigma^2_X = np(1 - p)$$

The standard deviation of a binomial random variable X , σ_X , is defined (and computed) as follows

$$\sigma_X = \sqrt{np(1 - p)}$$

- 泊松分布

考察一个变量是否服从泊松分布，需要满足以下条件：

X 是在一个区间(时间、空间、长度、面积、部件、整机等等)内发生特定事件的次数，可以取值为 $0, 1, 2, \dots$ ；

一个事件的发生不影响其它事件的发生，即事件独立发生；

事件的发生率是相同的，不能有些区间内发生率高一些而另一些区间低一些；

两个事件不能在同一个时刻发生；

一个区间内一个事件发生的概率与区间的大小成比例。满足以上条件，则 X 就是泊松随机变量，其分布就是泊松分布。

泊松分布的概率分布为

$$P(X = x) = \frac{\lambda^x}{x!} e^{-\lambda}$$

其中： $\lambda > 0$ 是常数，是区间事件发生率的均值。

- 指数分布

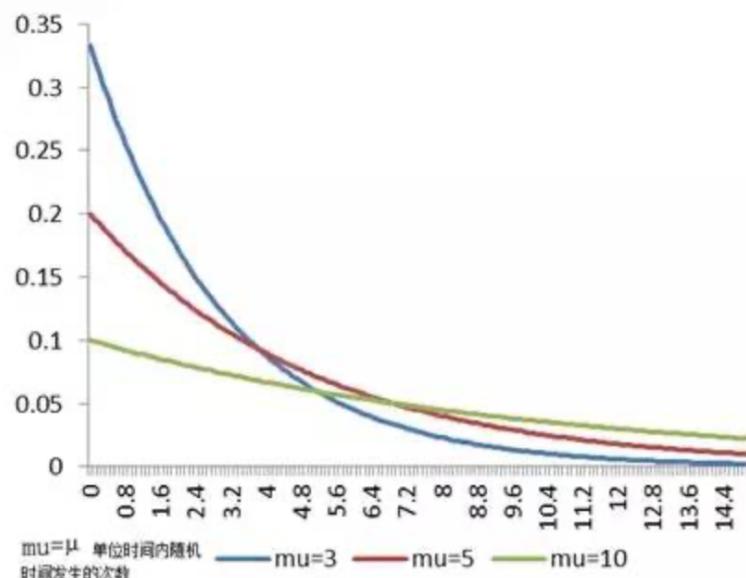
指数分布是事件的时间间隔的概率

指数分布的概率密度为：

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & x < 0, \end{cases}$$

式中： x 是给定的时间； λ 为单位时间事件发生的次数； $e = 2.71828$ 。

指数分布概率密度曲线如下图：



指数分布具有以下特征：

- (1) 随机变量X的取值范围是从0到无穷；
- (2) 极大值在 $x = 0$ 处，即 $f(x) = \lambda$ ；
- (3) 函数为右偏，且随着x的增大，曲线稳步递减；
- (4) 随机变量的期望值和方差为 $\mu = 1/\lambda$, $\sigma^2 = 1/\lambda^2$ 。

通过对概率密度函数的积分，就可以得到相应的概率，其表达式有两种

$$P(X \geq x) = e^{-\lambda x}$$

$$P(X \leq x) = 1 - e^{-\lambda x}$$

例：某电视机生产厂生产的电视机平均10年出现大的故障，且故障发生的次数服从泊松分布。（1）该电视机使用15年后还没有出现大故障的比例；（2）如果厂家想提供大故障免费维修的质量担保，但不能超过全部产量的20%，试确定提供担保的年数。

- 几何分布

满足以下四个条件：

(1) 做某事件的次数（也叫试验次数）是固定的，用n表示。（例如，抛硬币3次，求婚101次）

(2) 每一次事件都有两个可能的结果（成功，或者失败）。 （例如，求婚被接受（成功），求婚被拒绝（失败））

(3) 每一次“成功”的概率都是相等的，成功的概率用p表示。

(4) 这一点也即和二项分布的区别所在，二项分布求解的问题是成功x次的概率。而几何分布求解的问题则变成了——试验x次，才取得第一次成功的概率。举个栗子，求婚101次，第101次才被接受。的概率。

几何分布事件的概率：

$$p(x) = (1 - p)^{x-1} p$$

其中，p表示成功的概率，x表示试验的次数。

计算上面那个栗子，求婚成功的概率为0.5，求婚101次才被接受的概率为：

$$P(\text{求婚101次获得成功}) = (1-0.5)^{100} * 0.5 = 0.5^{100}$$

- 超几何分布

非常常见的一种分布，常用来表示在N个物品中有指定商品M个，不放回抽取n个，抽中指定商品的个数，即 $X \sim H(N, n, M)$ ，则抽中k件的概率为：

$$p(k) = P(X = k) = \frac{\binom{M}{k} * \binom{N-M}{n-k}}{\binom{N}{n}}$$

置信区间与假设检验

与效应量同时使用， p 值不一定是因为 r 大，还可能是因为样本量大。

类型 I 错误

如果原假设为真，但您否定它，则会犯类型 I 错误。犯类型 I 错误的概率为 α （即您为假设检验设置的显著性水平）。 α 为 0.05 表明，当您否定原假设时，您愿意接受 5% 的犯错概率。为了降低此风险，必须使用较低的 α 值。但是，使用的 alpha 值越小，在差值确实存在时检测到实际差值的可能性也越小。

类型 II 错误

如果原假设为假，但您无法否定它，则会犯类型 II 错误。犯类型 II 错误的概率为 β ， β 依赖检验功效。可以通过确保检验具有足够大的功效来降低犯类型 II 错误所带来的风险。方法是确保样本数量足够大，以便在差值确实存在时检测到实际差值。

概率论

朴素贝叶斯法

对于随机事件 A 和 B

$$P(A|B) = \frac{P(AB)}{P(B)} = \frac{P(B|A)}{P(B)} P(A)$$

$P(A)$ 为先验概率，这个公式可以理解为是在计算：当 B 条件已知时，原本的先验概率 $P(A)$ 会发生什么样的变化。

一个村子，三个小偷：A1小张，A2小英，A3小郑。事件B为村子发生失窃。已知小张去偷东西成功的概率为0，小英去偷东西成功的概率是1/2，小郑去偷东西成功的概率是1。某一天，村子一个人大喊：失窃啦！！！然后警察来了。一共有3个嫌疑人：A1小张，A2小英，A3小郑。警局已经对他们的偷窃能力有备案：小张去偷东西成功的概率为0，小英去偷东西成功的概率是1/2，小郑去偷东西成功的概率是1。试问：这三人中，与这次失窃案件有关的概率是多少。

由题目我们知道，三个人去偷东西的概率都是1/3，所以我们有：

$$P(A_1) = P(A_2) = P(A_3) = \frac{1}{3}$$

$$P(B|A_1) = 0, P(B|A_2) = \frac{1}{2}, P(B|A_3) = 1$$

求解

在偷窃发送之前，我们认为：三个人去偷窃的概率都是一样的（这是我们的主观感受）。故，我们有：

$$P(A_1) = P(A_2) = P(A_3) = \frac{1}{3}$$

$$P(B) = \frac{1}{2}$$

$$P(B|A_1) = 0, P(B|A_2) = \frac{1}{2}, P(B|A_3) = 1$$

我们需求的是: $P(A_1|B), P(A_2|B), P(A_3|B)$, 应用全概率公式, 条件概率公式与乘法公式, 有:

$$P(A_1|B) = \frac{P(A_1B)}{P(B)} = \frac{P(A_1)P(B|A_1)}{\sum_{i=1}^3 P(A_i)P(B|Ai)} = \frac{\frac{1}{3} * 0}{\frac{1}{2}} = 0$$

$$P(A_2|B) = \frac{P(A_2B)}{P(B)} = \frac{P(A_2)P(B|A_2)}{\sum_{i=1}^3 P(A_i)P(B|Ai)} = \frac{\frac{1}{3} * \frac{1}{2}}{\frac{1}{2}} = \frac{1}{3}$$

$$P(A_3|B) = \frac{P(A_3B)}{P(B)} = \frac{P(A_3)P(B|A_3)}{\sum_{i=1}^3 P(A_i)P(B|Ai)} = \frac{\frac{1}{3} * 1}{\frac{1}{2}} = \frac{2}{3}$$

所以, 最大嫌疑人是小政。

为了避免全是0的情况, 拉普拉斯平滑
其他分布的贝叶斯

https://blog.csdn.net/Mr_Robert/article/details/89923339?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-2.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-2.nonecase

优缺点

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, 能够处理多分类任务, 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。

笔试/面试题

<https://www.nowcoder.com/discuss/95737>

1. 某城市发生了一起汽车撞人逃跑事件，该城市只有两种颜色的车，蓝20%绿80%，事发时现场有一个目击者，他指证是蓝车，但是根据专家在现场分析，当时那种条件能看正确的可能性是80%，那么，肇事的车是蓝车的概率是多少？

我们的观测(evidence)B是目击者指证蓝车，需要计算肇事车是蓝车这一事件A的条件概率 $\Pr[A|B]$ 。

2.

统计学习

统计学习方法的经典研究主题包括：

- 线性回归模型
- 感知机
- k 近邻法
- 决策树
- Logistic 回归于最大熵模型
- 支持向量机
- 提升方法
- EM 算法
- 隐马尔可夫模型
- 条件随机场

无偏估计

Business

A/B test

模型

CRISP-DM

Business Understanding

分析需求又包括收集需求、分析需求、明确需求。收集需求的主要方法有：和使用对象进行访谈、市场调查、走访专家等。分析需求推荐利用思维导图：**5W2H、人货场**。

Data Understanding

保证分析中需要的数据是易获得的
埋点

Data Preparation

Modelling

Business validation

Deployment

TIPS

- 1.数据分析是要考虑成本的，可以从数据中分析出来的事情，尽量不要线下监控
- 2.时间是很重要的，周一的数据不要和周日的比，这也是ABtest一般要超过一周的原因，也要注意分析天/周/月/年循环规律（剔除节假日或其他特殊值）
- 3.不仅要做数据分析，也需要搭建数据化管理的体系

数据化管理

根据管理层次：业务指导管理、营运分析管理、经营策略管理、战略管理

根据业务逻辑：销售中的数据化管理、商品中的数据化管理、财务中的数据化管理、人事中的数据化管理、生产中的数据化管理、物流中的数据化管理

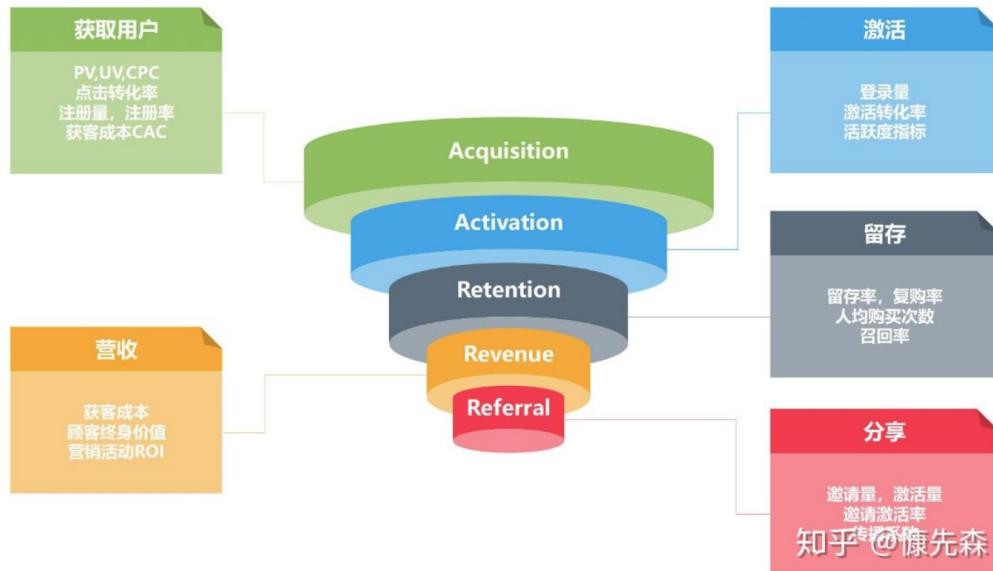
怎么理解数据分析

Descriptive, predictive, prescriptive

数据分析的价值/意义

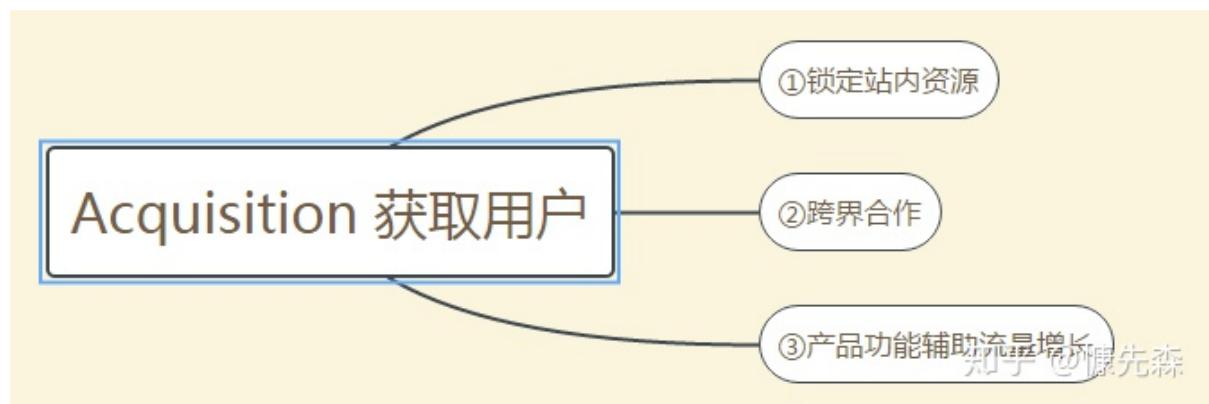
深度分析用户，来达到更高的转化率以及收益长远发展。提高决策的速度性和正确性。

AARRR模型



Acquisition：获取用户

毫无疑问，在电商营销活动中，获取用户意味着需要拓展**页面流量**，相对较大型的电商营销活动至少可以从以下三个方面获取用户（流量）：



站内资源

每个大型营销活动，平台都会锁定大量营销资源辅助活动曝光，各种的视觉、站内氛围的改造等，都是为活动造势和引流。

- 亚马逊黑色星期五
- 京东618
- 淘宝双十一
- 小红书会员日等

但是，在调配各种的站内营销资源之前，更必要的是去观察成交目标和转化率，更进一步分析**不同产品的销售问题**，并结合**每个资源位的投资回报率**，进行综合评估，从而锁定最利于活动的资源位及排期。

跨行合作

相信在重要的时间节点，不仅电商，各行各业都在做自己的营销活动，异业合作无疑是另一获取流量的渠道，通过不同行业的互补，将营销活动渗透到站外的用户群。

跨行合作的关键是：

- 我需要什么，对方能给我什么。
- 对方需要什么，我能不能给。

跨界合作无疑能为平台对外扩展，获取更多外部流量打下良好的基础，同时，在筛选不同各种优秀的平台的时候，也需要注意各平台之间的口碑和平台质量。

产品功能辅助流量增长

“周期长、品类多、阶段性主推”这往往是电商大型促销活动的特点，如何让用户在**每个时间段内都能回流**？一般来说我们通过以下方法：

- 场次预约提醒

场次预约提醒，不难理解，就是在下一场活动开始时，微信公众号或者app对用户进行消息提醒，引导用户再次回流至卖场。

- SNS后置奖品分享

SNS后置奖品分享，SNS可以吸进用户回流和裂变流量，通过送小奖品的形式引导用户参加社交活动，但多数用户都是奔着奖品去的，领完奖品之后他们对于跳转卖场的兴趣通常比较小，所以这时候我们可以将玩法前置，而奖品后置。

Activation：提高活跃度

对于不同的产品/商业模式，我们对用户的活跃度的定义不同。在电商中，活跃度是以商品成交为准，如何吸引用户下单并提高其转化率，这才是研究的重中之重。



1. 促销力度+噱头玩法

总所周知，顾客对于商品的价格，从来不是越便宜越好或越贵越好，所以，与其让顾客觉得便宜，还不如让顾客觉得“占便宜”。然而，对于现在商品满减优惠券满天飞的电商环境里面，顾客似乎已经熟视无睹，甚至有点麻木，所以这时候更需要配合一些噱头玩法，才能增强用户占便宜的心理。

新用户专属福利：开启新用户身份识别，上线仅新用户可见的超低价商品，如新人1元包邮等，可有效转化新用户。

秒杀的氛围：即“限时低价”，将其氛围做足，迫使用户快速下单。

2.精准推荐系统

推荐系统，顾名思义就是根据你浏览过的历史，推荐你喜欢内容的系统，像淘宝京东的猜你喜欢。

通过将这个功能模块加入营销活动中，可以个性化地推介用户感兴趣的的商品，增强商品转化。据估计，该模块的成交可以占到整个卖场成交的10-20%之间，不容忽视。

3.商品精细化运营

营销活动每时每刻都在消耗资源，一般情况下，重点模块的选品如果把握不准确，可以调取近期top单品，重点模块主推top单品可以有效提升该模块的转化率。

其次，货品应当是一个动态调整的过程，让用户的成交情况来决定商品的去留，这需要运营实行商品的赛马机制，做好实时数据监控，不达标的商品及时更替，也可以保证转化率的提升。

Retention：提高留存率

用户流失的分析

<https://www.zhihu.com/question/26225801>

首先采用“两层模型”分析：对用户进行细分，包括新老、渠道、活动、画像等多个维度，然后分别计算每个维度下不同用户的次日留存率。通过这种方法定位到导致留存率下降的用户群体是谁。

对于目标群体次日留存下降问题，具体情况具体分析。具体分析可以采用“内部-外部”因素考虑。

a. 内部因素分为获客（渠道质量低、活动获取非目标用户）、满足需求（新功能改动引发某类用户不满）、提活手段（签到等提活手段没达成目标、产品自然使用周期低导致上次获得的大量用户短期内不需要再使用等）；新手上手难度大、收费不合理、产品服务出现重大问题、活动质量低、缺少留存手段、用户参与度低等

b. 外部因素采用PEST分析（宏观经济环境分析），政治（政策影响）、经济（短期内主要是竞争环境，如对竞争对手的活动）、社会（舆论压力、用户生活方式变化、消费心理变化、价值观变化等偏好变化）、技术（创新解决方案的出现、分销渠道变化等）。市场、竞争对手、社会环境、节假日等

新老用户流失

新用户流失：原因可能有非目标用户（刚性流失）、产品不满足需求（自然流失）、产品难以上手（受挫流失）和竞争产品影响（市场流失）。

新用户要考虑如何在较少的数据支撑下做流失用户识别，提前防止用户流失，并如何对有效的新用户进行挽回。

老用户流失：原因可能有到达用户生命周期衰退期（自然流失）、过度拉升arpu导致低端用户驱逐（刚性流失）、社交蒸发难以满足前期用户需求（受挫流失）和竞争产品影响（市场流失）。

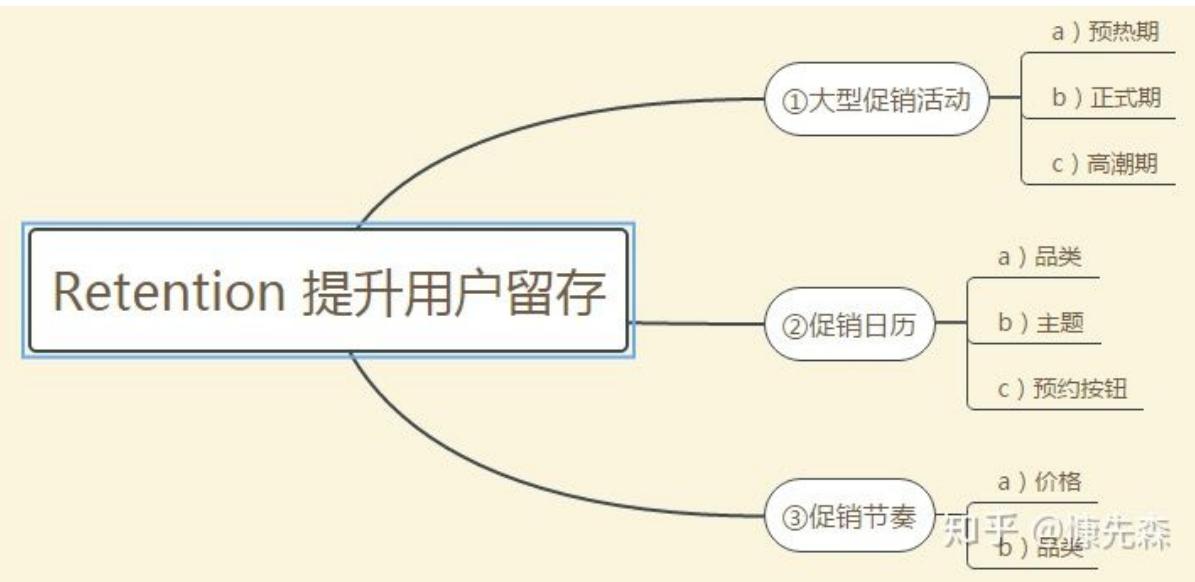
老用户有较多的数据，更容易进行流失用户识别，做好防止用户流失更重要。当用户流失后，要考虑用户生命周期剩余价值，是否需要进行挽回。

如果某天日活下降了，你会怎么分析（音乐APP）

- 将APP内每个操作步骤量化为产品中具体的页面或按钮或其他可以被统计的行为。例如引导页注册——选择歌手——选择歌曲类型——进入歌曲首页——搜索、收藏等其他页面，假如是选择歌曲类型的环节用户流失率高，那就看看是不是歌曲的类型不符合用户的喜好，或者是图片、视觉上的问题，导致用户离开。
- 将以上用户行为进行代码埋点，比如竞品、app store或者安卓市场上下载量较高排行较前的音乐APP等，通过用户行为分析用户在APP中的行为。通过埋点得出的数据可以得出用户在哪个环节的跳转和流失率，还有用户的习惯和喜好等。
- 通过漏斗视图查看分析以上埋点行为事件的转化率、流失率，找到流失用户的渠道。通过观察用户的流失数据，从数据中发现和总结规律。比如是否在某个时间段（比如改版，或者是添加了某个新功能）用户开始流失、在什么节点达到最高峰，或者是集中在某一个时间点，这段时间内有什么特别的事发生？
- 分析：最基本的分析是哪一个步骤的流失率比较多，分析原因（也可能完全是猜测）是非活跃用户还是活跃用户，这些用户有没有共性或者特点，分别是什么用户（男性用户还是女性用户），有没有哪件事触发了他们的敏感点等等。
- 当然，如果要精益数据分析，还应该综合考虑流失用户来源（是通过什么途径下载到APP的）、用户分群等等。了解是什么类型的用户在流失，流失用户的有共性和特点是什么。
- 最后关键的一步就是进行验证。此时可以用Push、电话、短信、邮件、竞品对比等方式接触用户，然后思考优化方式。最后着手优化改进，并且对优化结果进行跟踪，看改进的效果如何，再综合调整运营策略。

电商的Retention

用户留存无疑是看：**留存率**，其指标之于电商就是**回访率**。电商大型营销活动持续时间都比较久，普遍在半个月左右，非常容易出现用户买一单之后就再也回不来的情况，所以这个层面需要有各种的促销活动来增大用户回访的机会。



针对大型促销活动

基本都会划分为三个阶段：

- 预热期：预约造势，通过sns、定金裂变等玩法吸引用户关注
- 正式期：前面如果证实是好的激励体系，可以让活动健康持续发展
- 高潮期：进一步引爆高潮，使用的激励方式，成长值会员体系、签到体系、积分任务体系等。

促销日历

在活动节奏做出来后，需要保持温度，让用户有持续的兴趣，这时候，可以从品类或者主题的纬度去拆分节奏，用“促销日历”类型的方式呈现给用户，并设置预约按钮，让用户定期回流。

促销节奏

就双11而言，其促销节奏分为两个维度：

- 价格
- 品类

上面两维度结合，可以以主题的形式进行包场，例如：

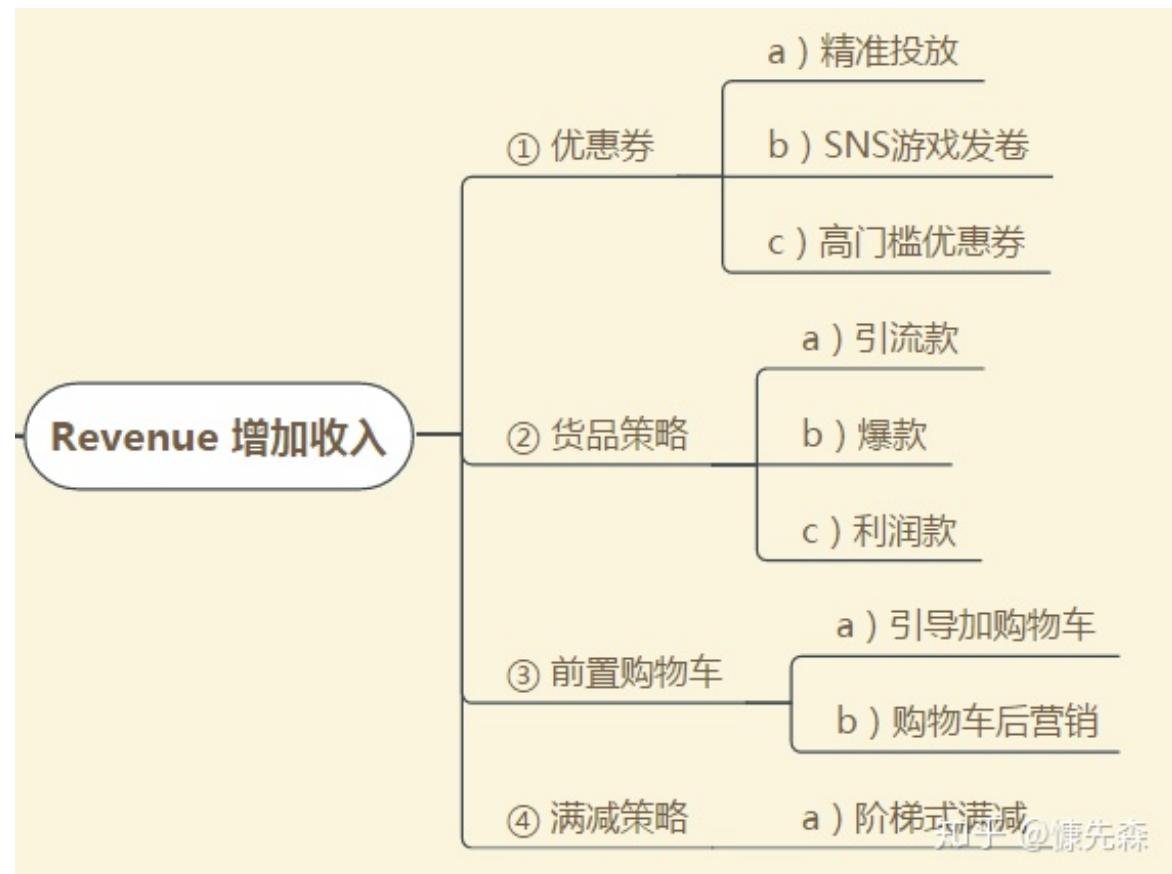
11月1日是女神日，这一天的促销利益点是全场满199-100

11月3日是超市日，这一天的利益点是两件7折

综上核心在于，用户无论在任何时间看到促销页面，都可以提前看到整个活动不同的**利益点和主推品类**，并且自主选择**预约及提醒功能**，从而这几步能大大提高顾客的回访率。

Revenue: 获取收入

对于收入这个维度，电商除了有产品的价格利润（如客单价，成本，税率，物流等）的考虑外，在运营段对收入的把控也是必不可少的，对推广费用的把控，各种优惠券的和优惠策略的精准投放，把有限的推广资金用在刀刃上。



1. 优惠券

优惠券主要可以通过这三种布局，从而可以实行收入的增加外，还能有效提高回访率：

精准投放：根据用户购买属性圈定偏好品类，定向发放优惠券。（基本电商都有实现）

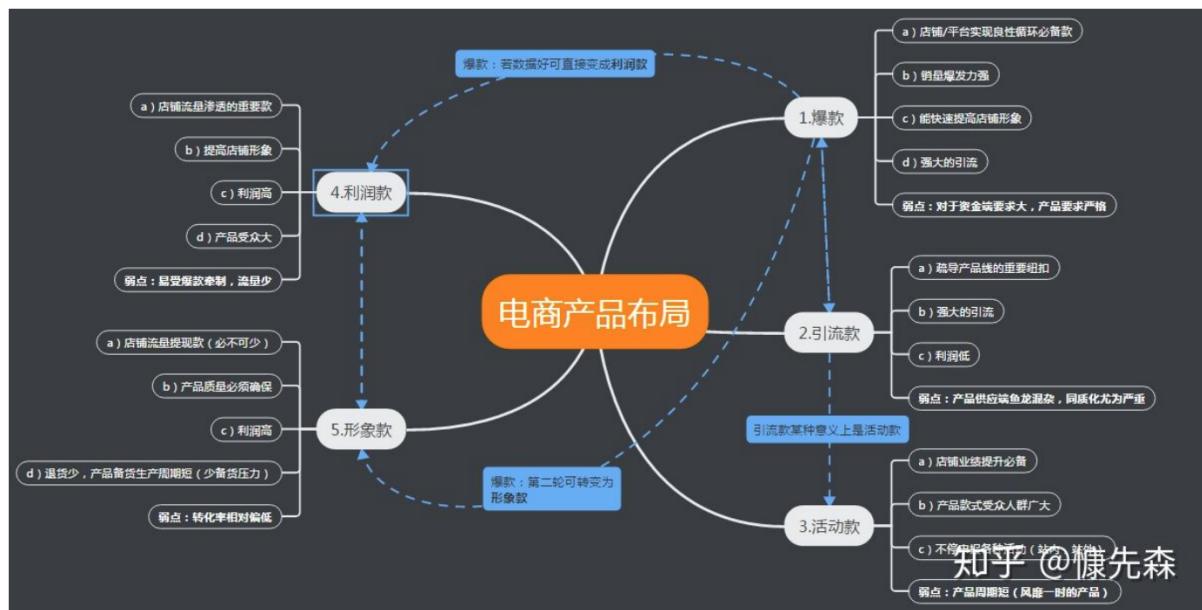
SNS游戏发卷：在优惠券使用日期开始前推送信息，让用户及时跳转券购页形成精准转化。

高门槛优惠券：券的使用门槛需要比平台/品类客单价高一点点。

例如：某品类的客单价为50元，那么10元面额的优惠券，使用门槛可以设置为60元左右。

2. 货品策略

相信大家对电商货品的布局不陌生，从引流款，爆款，利润款，品牌款到形象款，不同的产品定位决定着不同的价值和作用。



除了爆款能充当良性循环，引流款打开流量入口，输出低价流量给利润款，从利润款中获取大部分的利益，再逐步树立品牌款和形象款，打造独有的口碑标签。

另外，细分类带Tag的形象款需要**多SKU但少库存**，以满足客户对于品牌的长尾需求和供应。

3. 前置购物车

引导顾客把喜爱的商品加入购物车，商家可以对被加购物车的商品进行进一步营销，以打破用户购买决策的最后一道防线。

促成转化

精细化运营

例如，在活动开始前加购物车，该商品在活动时就有减价权益，活动生效当天引导用户回流兑换减价权益。另外，购物车里的商品能积累非常重要的用户信息，无论是引导用户选择“降价提醒”，还是给用户推送同类商品，都可以进一步提升用户的转化率。

4. 满减策略

这个策略更着重于引导顾客多购买别的商品。

普通促销做法：全场商品5折

阶梯促销满减：199减100；399减200；599减300

上述做法，实际都是五折，但用户会因为加车商品比199多了几十元，而去凑399的满减门槛；而每满199减100也是同理，凑超了一个199，就会不自觉想去凑下一个199，从而提升了成交。

Refer: 自传播

衡量用户传播的指标是**K因子**

$K = \text{用户向他的朋友发出邀请的数量} * \text{收到邀请的人转化为新用户的转化率}$

即：若平均每个用户会向20个朋友发出邀请，而平均的转化率为10%的话，

$$K = 20 * 10\% = 2$$

易见，当 $K>1$ 时，用户就会相应增长。若 $K<1$ ，那么用户到某个规模时就会停止增长。

如何提高K因子？

- 用户传播的次数
- 被传方的转化率

因此，“**分享机制**”在这里充当相当总要的角色，可以通过一系列的奖励机制刺激用户分享和传播，帮助活动或者商品获得**更多的曝光**。

而且因为用户多是分享在社交媒体，熟人之间的信任关系会让活动获取更多的新用户，如老带新、团长免单等。为了在引导用户传播的同时提升被传方的转化率，应当考虑双方均获得奖励。

活动举例：

用户通过拉好友成团购买某款商品，按照成团时间排序，奖励耗时最短的前三名团长。
活动发布后，分享率提升了28%，开团率更是翻了一倍，但相应的成团率却降低了20%，原因是大多数人都希望自己开团去拿奖励，不愿意参加别人的团。
后续可根据活动数据进行了相应的调整，改成团长与团员均可获得奖励，解决了成团率过低的情况，并且将分享订单转化率拉升了18%。

电商分析

分析方向和目的

通常电商分析会包含销售、流量&搜索、用户、商品、以及促销&优惠券等维度，基于本淘宝用户行为数据包的具体情况，会从流量、用户、商品三方面进行分析，探索用户行为规律，为用户和产品运营提供更精准的策略，从而提高GMV：



流量分析：

PV(Page View, 浏览量)，是指在一个统计周期内，浏览页面的数之和。
UV(Unique Visitor, 访客数)，是指在一个统计周期内，访问网站的人数之和。

跳出率:“跳出”是指在这件宝贝浏览的过程中，直接跳到你的下一个宝贝去浏览。“跳出率”等于跳出量/访问总量。

跳失率=只有点击行为的用户数/总用户数

用户分析:

AARRR在电商: <https://zhuanlan.zhihu.com/p/54251292>

用户行为变化趋势: pv代表浏览; fav代表收藏; cart代表加购; buy代表购买

行为漏斗分析: 行为漏斗转化分析有两种分析维度, 一种按行为计数, 偏重看有多少次浏览行为、购买行为等以及对应的转化率; 一种按uv(独立访客)计数, 偏重看有多少用户浏览、加购并转化发生了购买行为。实际工作中根据业务需求从不同的维度进行分析。

按行为计算---->用户购买流程: 浏览 (pv) -收藏 (fav) -购买 (buy)或者浏览 (pv) -加购 (cart) -购买 (buy)。采用漏斗模型对数据进行转化率分析

pv	buy	cart	fav
89405	2028	5636	2883
购买转化率	收藏转化率		加购转化率
2.3%	3.2%	6.3%	

按UV (独立访客) 计算---->

behaviour_type	uv
buy	1913
cart	4728
fav	2281
pv	28302
购买转化率	收藏转化率
6.8%	8.1%
	加购转化率
	16.7%

复购:

复购率: 有复购行为的用户数 / 有购买行为的用户数

留存:

firstday	day1	day2	day3	day4	day5	day6	day7	day8
2017-11-25	30.6%	28.1%	25.2%	25.5%	25.9%	27.5%	32.0%	31.4%
2017-11-26	25.3%	23.6%	23.7%	24.1%	23.3%	29.4%	30.2%	0.0%
2017-11-27	25.0%	24.6%	22.7%	23.9%	27.0%	29.0%	0.0%	0.0%
2017-11-28	23.5%	21.7%	23.2%	26.3%	25.7%	0.0%	0.0%	0.0%
2017-11-29	22.2%	20.6%	26.6%	24.8%	0.0%	0.0%	0.0%	0.0%
2017-11-30	21.2%	24.2%	24.1%	0.0%	0.0%	0.0%	0.0%	0.0%
2017-12-01	28.0%	24.9%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2017-12-02	24.8%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
2017-12-03	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%

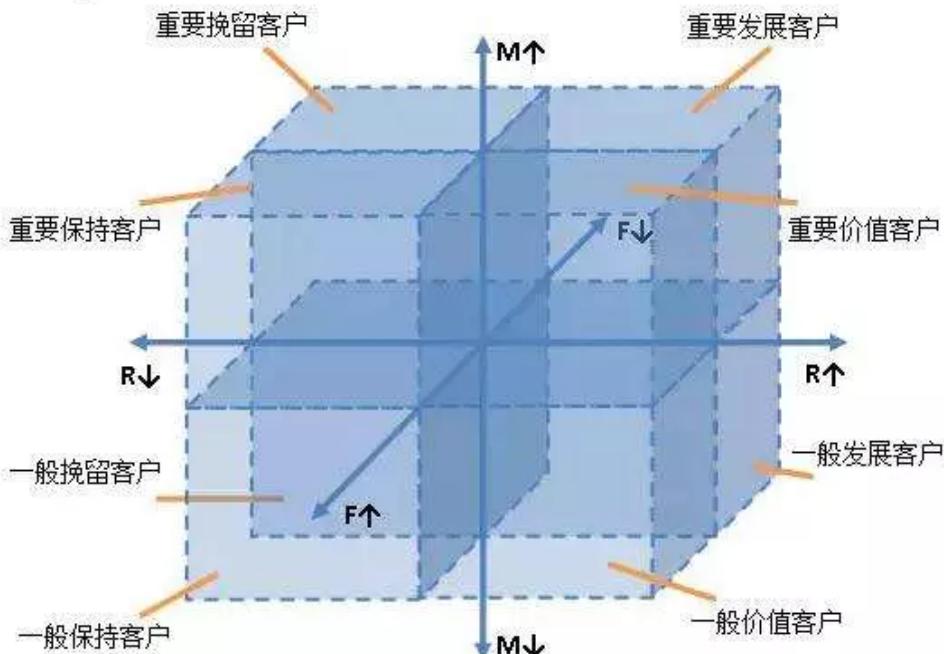
SQL计算方法:

获取每个用户的使用日期与第一次使用日期的差值

计算每日的留存用户数量

RFM用户价值模型分析:

RMF模型，利用R最近一次消费时间，M总消费金额，F消费频次，将用户划分成多个群体。



客户分类	客户数
高价值客户	79
深耕客户	1063
挽留客户	747
挽回客户	24

商品分析：

按类目分析：e.g Top 20 购买量/浏览量/购买转化率类目

按商品分析：e.g Top 20 购买量/浏览量商品

分析结果：

流量：

1. pv与uv的每日变化趋势大致相同，工作日访问量（11.27-12.1）维持在低值，周末较高。而12月第一个周末相较于11月最后一个周末的pv与uv均有较大幅度地提升，研究可发现双12的各项预热活动带来了该周末的访问量增长。
2. pv与uv的小时变化中，凌晨1-6点是用户访问的低值时段，晚上20-22时用户访问的活跃时段，早上6-10时和晚饭后18点-21点是用户访问淘宝app的增长迅速时段，符合人的正常作息。建议可在用户活跃时段推送新品及促销活动，提高购买率；若有活动日，则可在**用户活跃时段和增长迅速时段分时段投放活动信息和优惠券**。
3. 同时流量分析通常与营销活动、广告投放等结合起来评估站内站外活动和投放的效果

用户

1. 用户行为：四类用户行为在12月第一个周末迎来了峰值，访问量和加购量在周日继续上涨，但相应收藏量有所下降，需结合具体的活动情况来分析数据波动原因。
2. 从行为纬度看，浏览到加购的转化率为9.5%，从浏览到购买的转化率为2.3%；从独立访客纬度看，大约有6.8%的活跃用户在统计期间内有购买行为。需结合具体类目商品

转化率benchmark进行评估分析。若转化率低于正常水平，则可通过用户行为途径寻找原因，比如PV转化率低，则可从商品价格、页面信息等因素入手深挖原因。

3. 该周期内复购率为5.4%，较为正常。通常时间周期越长，复购率越高，在实际业务中，更常看季复购率、半年复购率甚至年复购率。不同类目的复购率也会有差异。
4. 留存率：以11.25日为首日统计日，则次日留存率为30.6%，三日留存率为25.2%，七日留存率为32%。双12活动预热使得用户在周末的留存率有所上涨。
5. 根据R, F数据将用户划分为四类等级，可针对每个等级进行精准化营销

商品分析：

1. 部分购买量高的类目/商品浏览量低，而部分浏览量高的类目/商品转化率低，需结合具体商品/类目的特性进行分析。
2. 对于热销的类目/商品，可多推出一些和该类目/商品相关的其他类目/商品捆绑交叉销售，提高销量
3. 若有新商品以及参与活动的引流商品等，也需要单独看此类商品表现
4. 运用相关分析，根据客户历史数据进行精准营销

面经

1. 业务题：

- (1) 你的一个朋友经营一个网店，以前100个人浏览会有50个人下单，最近客服咨询量提升到200，但下单却下降到40，请你帮她分析原因。
- (2) 你的朋友说最近咨询的人变多了，但是他做的是小本生意，没有能力再去请多一个客服，所以导致自己回复用户速度下降，很多人就走了没有下单，请你帮他想一下解决方案
- (3) 你的朋友按照你的建议去做，生意变好了，他赚钱了，然后他一下子请了5个客服，请你设计指标去考核客服的KPI。

- 在模型的训练迭代中，怎么评估效果？

在训练迭代中，记录每次迭代的损失函数下降，k-means算法的损失函数下降很快，表示这个算法收敛快，一般会在5, 6步之内收敛。另外Adaboost可以确保损失函数收敛至0（可推导证明），但神经网络可能会在训练中途停滞不前。

- CNN中，如何减少参数？

- 对于同分布的弱分类器，求分类器均值化之后的分布的均值跟方差

在数学与统计学中，大数定律又称大数法则、大数律，是描述相当多次数重复实验的结果的定律。根据这个定律知道，样本数量越多，则其算术平均值就有越高的概率接近期望值。

样本均值等于总体均值，样本方差等于

当样本量N逐渐趋于无穷大时，N个抽样样本的均值的频数逐渐趋于正态分布，其对原总体的分布不做任何要求，意味着无论总体是什么分布，其抽样样本的均值的频数的分布都随着抽样数的增多而趋于正态分布，如上图，这个正态分布的u会越来越逼近总体均值，并且其方差满足 a^2/n ，a为总体的标准差，注意抽样样本要多次抽取，一个容量为N的抽样样本是无法构成分布的。

- 常见分类模型的优缺点，适用场景及如何选型
- 数据挖掘各种算法，以及各种场景下的解决方案

我们接触过的数据挖掘算法基本分为分类，回归，聚类，和规则提取四大类。分类的目的是通过分类模型，将数据映射到某个特定的类别中。目前常用的分类模型有逻辑回归，决策树，集成；举例：商品推荐（精准营销），市场细分，客户获取（找出购买倾向性较高的客户进行深度跟踪营销，二八原则），流失用户行为/时间预测，信用评分（京东打白条），欺诈侦测。回归是研究一组变量和另一组之间的关系。聚类是按照数据的相似性和差异性将一组数据分为几类。组内数据相似性高，不同组之间相似性低。举例：市场细分，用户细分。关联规则算法用于提取隐藏在数据项之间的关联或相互关系，即可以根据一个数据项的出现推导出其他数据项的出现。举例：交叉销售（商品推荐）

数据挖掘主题可以归纳为营销、信用与违规识别

<https://blog.csdn.net/j2laYU7Y/article/details/81571506>

- 深度学习和普通机器学习有什么区别？

深度学习是机器学习的一个分支。机器学习是让机器自动从大量数据中识别模式，包含众多的学习方法去进行分类，回归，聚类，和规则提取。但是对机器学习来说，特征提取并不简单。特征工程往往需要人工投入大量时间去研究和调整，就好像原本应该机器解决的问题，却需要人一直在旁边搀扶。深度学习便是使用了神经网络的学习方法解决特征提取问题的一个机器学习分支。它可以自动学习特征和任务之间的关联，还能从简单特征中提取复杂的特征。与机器学习相比，深度学习比较适合处理大规模数据，由于计算量很大，它更依赖于高端硬件设施，并且运行时间较长。并且，对于深度学习来说，我们无法理解它内部的规则。

- 为什么要做数据归一化？怎么归一化？

数据归一化可以提高模型的收敛速度，涉及到距离计算的算法时可以提升模型的精度，深度学习时数据归一化可防止模型梯度爆炸。

min-max

$$x^* = \frac{x - \min}{\max - \min}$$

使输出的值在 (0, 1) 之间

缩放紧跟最大、最小值的差别有关

与标准化比较

Z-score：

那个值减平均数除以标准差

缩放与每个值都有关

在分类、聚类算法中，需要使用距离来度量相似性的时候、或者使用PCA技术进行降维的时候，Z-score standardization表现更好。

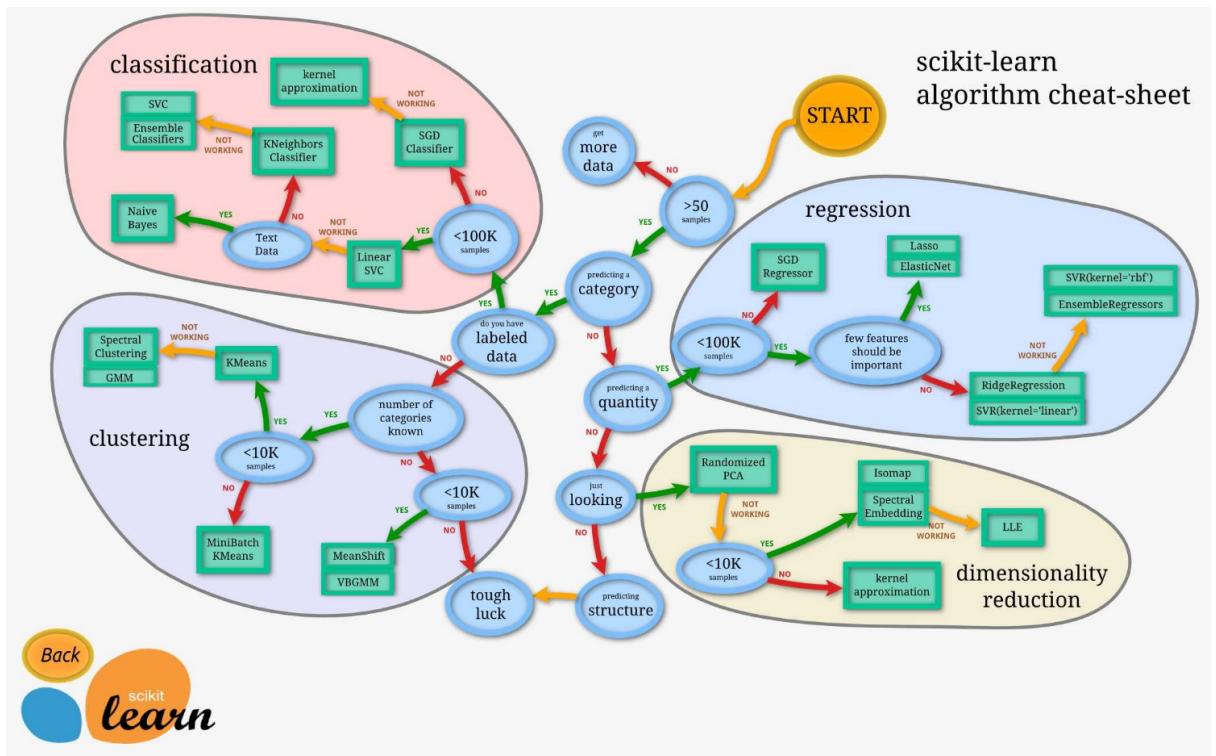
如果对输出结果的范围有要求，用归一化；如果数据较为稳定，不存在极端的最大最小值用归一化；如果数据存在异常值和较多噪音，用标准化。

- 分类模型和回归模型的区别

分类和回归的区别在于输出变量的类型。

定量输出称为回归，或者说是连续变量预测；

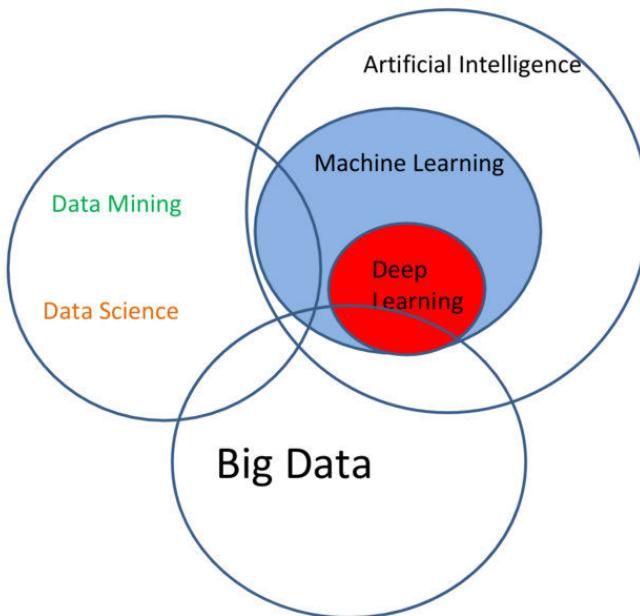
定性输出称为分类，或者说是离散变量预测。



- 分类模型可以做回归分析吗？反过来可以吗？

回归模型可以解决分类问题，实际上分类基本上都是用“回归模型”解决的，只是假设的模型不同(损失函数不一样)，因为不能把分类标签当回归问题的输出来解决。比如逻辑回归的损失函数是用的交叉熵，交叉熵是分类模型的常用损失函数。而逻辑回归的基础线性回归，损失函数就是最小二乘MSE，回归模型的常用损失函数。又如，CART树的分类的损失函数是基尼系数，而回归树的损失函数是最小二乘。

- 怎么理解数据挖掘与机器学习的关系



根据汤姆·米切尔 (Tom Mitchell) 在他关于这一主题的开创性著作中所说，机器学习 «与如何构建随经验而自动改善的计算机程序有关。» 机器学习本质上是跨学科的，并且采用了计算机科学、统计学和人工智能等领域的技术。机器学习研究的主要产物是算法，它可以根据经验促进这种自动改进，这些算法可以应用于各种不同的领域。机器学习是数据科学的核心方面。我在下面对数据科学一词进行了详细介绍，但是如果您认为其目标是从数据中提取洞察力（数据挖掘），那么机器学习是使该过程自动化的引擎。机器学习与经典统计有很多共同之处，因为它使用样本进行推断和归纳。统计数据更着重于描述性内容（尽管它可以（通过推断可以预测），机器学习对描述性的关注很少，并且仅将其用作中间步骤才能进行预测。

而在数据挖掘中，重点是算法的应用，而不是算法本身。我们可以如下定义机器学习和数据挖掘之间的关系：数据挖掘是一个过程，在此过程中，机器学习算法被用作工具来提取数据集中保存的潜在有价值的模式。更需要掌握业务，比如数据库等，这些都是机器学习不考虑的。

<http://innovaleur.com/the-data-science-puzzle-explained/>

- KNN算法

KNN算法的思想总结一下：就是在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前K（K的选择非常重要，通常不大于20）个数据，则该测试数据对应的类别就是K个数据中出现次数最多的那个分类，其算法的描述为：

- 1) 计算测试数据与各个训练数据之间的距离；
- 2) 按照距离的递增关系进行排序；
- 3) 选取距离最小的K个点；
- 4) 确定前K个点所在类别的出现频率；
- 5) 返回前K个点中出现频率最高的类别作为测试数据的预测分类。

- 有哪些常见的分类器，简单介绍下原理

常见的分类器有朴素贝叶斯，逻辑回归，决策树和一些集成模型。在这里面朴素贝叶斯因为基于统计并且比较简单，所以可以应用于非常大的数据集且不用担心过拟合。但是因为有朴素的特征相互独立的假设，所以在一些特征相关性强的样本上效果差。逻辑回归可以引入正则防止过拟合，逻辑回归比较适合工业界大数据的分类模型，要求精准时可以使用解析解，要求时间效率时可以使用随机梯度下降法来迭代。逻辑回归解释性很强，可以根据每个特征的参数值判断特征的重要程度。一般可以先做一个逻辑回归的模型看效果，并且把它的结果作为之后复杂模型的底线。但是逻辑回归对数据特征间的独立性要求较高；不适用于features和label为非线性关系的数据中；当特征空间很大、特征有缺失时，逻辑回归的性能不是很好。决策树是一种对数据要求比较低的模型，除了最初的ID3，其他的树模型的数据可以缺失，可以非线性。并且它的if-then规则可解释性非常好。ID3和CART偏向特征值较多的属性。集成模型我们用的比较多的就是随机森林，xgboost和xgbost基础上的改进的lightgbm。随机森林使用bootstrap的方法抽取样本，再抽取特征，形成多个模型，对结果进行投票或取平均，随机森林不需要自定测试集，没有过拟合的困扰，是一种方差很低的稳定的模型。

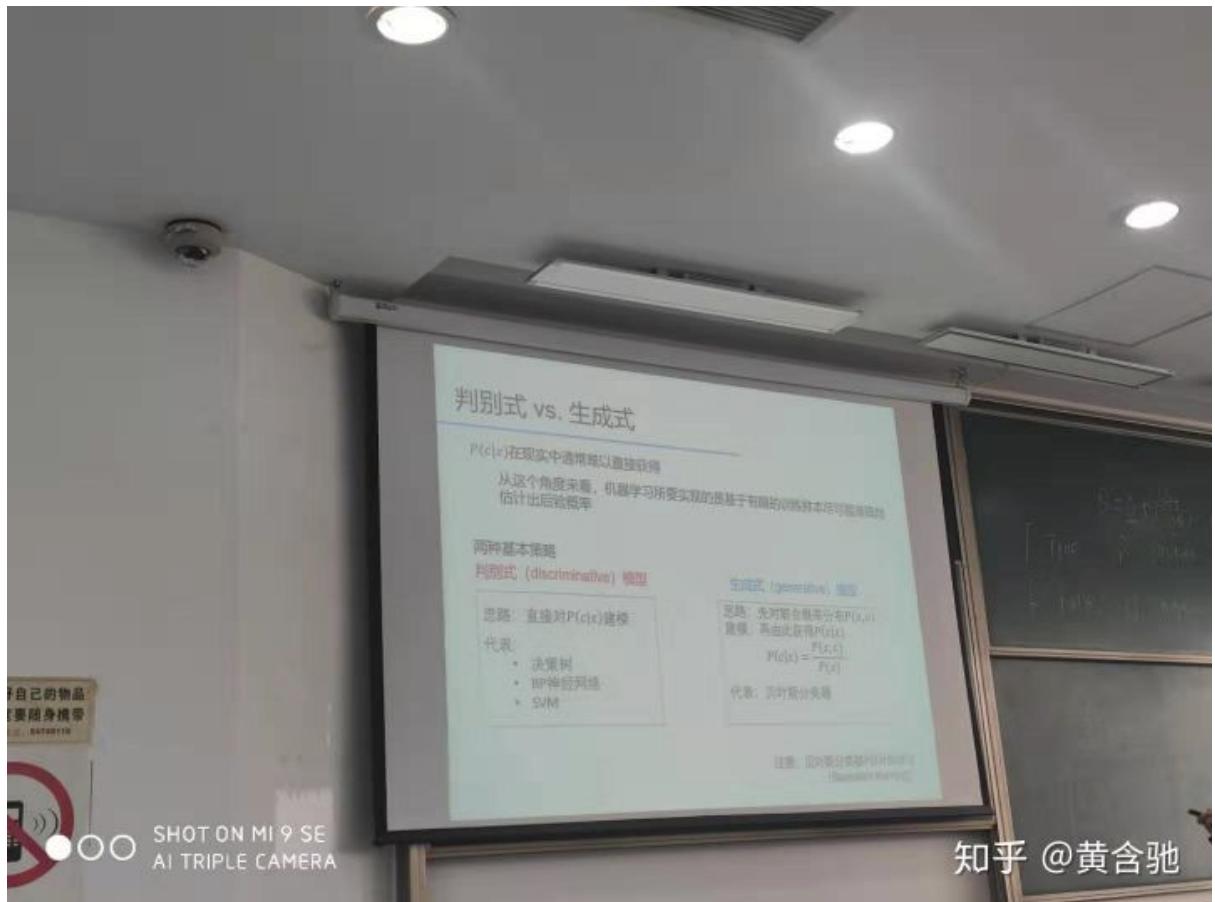
xgboost是当前来说最好的模型之一，很多算法竞赛中会用到，使用的串行的弱分类器，每一个分类器都在拟合前面分类器的残差。

- 判别模型，生成模型

总结发现：生成和判别的区别在于拟合 $p(y|x)$ 还是拟合 $p(x|y)$

拟合 $p(y|x)$ ，是拟合从果到因的关系，这种拟合出来的模型叫判别模型。

拟合 $p(x|y)$ ，是拟合从因到果的关系，这种拟合出来的模型叫生成模型。

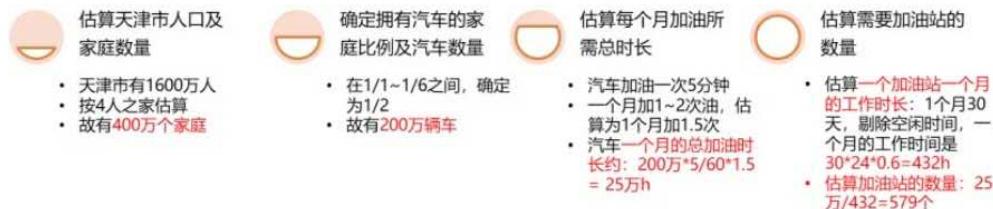


Business练习题

- 费米估算法

估计天津市加油站的数量

估算天津市加油站的数量



上海有多少理发店

估算上海市人口数量：2500万

每个人理发的频率：2个月1次

每个理发店每天接待的客人：50人

有0.8万理发店

星巴克一年的营业额

一杯价格：40

卖出频率：1分钟卖出一杯

一天卖出杯数：720杯

一家店一年营业额： $720 \times 40 \times 365 = 1051\text{万}$

星巴克店数：4000

总营业额： $1051 \times 4000 = 4,248,000,000$

- 京东分为不同会员，如何从各个方面提高下单率？

漏斗模型：浏览 - 收藏/加购 - 购买

列出每个步骤的影响因子和权重，按照权重针对每个影响因子优化。

- 假设五月份第四周天猫女装类的GMV出现整体下滑，你觉得可以怎么分析？
gmv即Gross Merchandise Volume,主要是指网站的成交金额,而这里的成交金额包括：付款金额和未付款。

使用假设检验检测是真实下降还是随机波动

对于产品的理解：服饰非常受季节性的影响

<http://www.woshipm.com/data-analysis/1628977.html>

1) 首先要定位到现象真正发生的位置，到底是谁的销售额变低了？这里划分的维度有

a. 用户（画像、来源地区、新老、渠道等）

b. 产品或栏目

c. 访问时段

2) 定位到发生未知后，进行问题拆解，关注目标群体中哪个指标下降导致网站销售额下降：

a. 销售额=入站流量 * 下单率 * 客单价

b. 入站流量 = Σ 各来源流量 * 转化率

c. 下单率 = 页面访问量 * 转化率

d. 客单价 = 商品数量 * 商品价格

3) 确定问题源头后，对问题原因进行分析，如采用内外部框架：

a. 内部：网站改版、产品更新、广告投放

- b. 外部：用户偏好变化、媒体报道、经济环境、竞品行为等
- 影响外卖平台APP的指标有哪些
 1. 常规数据指标的监测，不在话下。如用户量，新用户量，UGC量（社交产品），销量，付费量，推广期间的各种数据等等。这些是最基础也是最基本，同时也是boss们最关注的指标。你接手这项工作的时候第一任务就是把这些数据梳理好。
 2. 渠道分析，或者说流量分析。对于一个在上升期得APP来说，你们会花资源去引流量、去别的渠道拉用户。这时候就需要监测各个渠道的好坏，哪个效果好，哪个单价便宜，这都是需要渠道数据监测来完成。当然，你还需要跟踪监测不同渠道用户的后续表现，给每个渠道的用户进行打分，让BOSS知道哪个渠道值得投，哪个渠道是垃圾。同时也可以监测iPhone和Android用户的质量区别，一般来说，iphone用户质量要略高于android用户。当然，有多余精力的话还可以监测不同机型之间用户的表现区别。总之就是在不同的维度上监测不同用户的表现。
 3. 用户的核心转化率。想想你的APP的核心功能是什么，然后去监测这个核心功能的转化率。在游戏APP里可能叫付费率，在电商APP里可能叫购买率。不同的行业都有相应不同的转化率，你可以将自己的产品和行业平均进行对比，看看自己的产品在行业中所处的地位。同时，通过长期的监测，你还可以更具这项数据评判APP不同版本的好坏。
 4. 用户使用时长的监测。一方面，这是一个监测用户活跃度的非常好的指标。用户使用时间长就意味这活跃度高，反之亦然。另一方面，想一想你的APP在设计的时候，当初预计一个正常的用户每天会用多少时间，上线后用户真正用的时间是否和你的预计相同？如果这里面有很大的偏差，就说明用户对APP的认知和你当时设想是有不同的。这个时候你就需要想想如何来调整你的产品，去迎合用户的认知。（这里说一个题外话，个人认为在对产品做修改的时候一定是想办法去迎合用户，而不是想办法改变用户让用户去适应产品。这里以微博作为例子，用户一直把微博看做是一款传媒产品，一款信息交流工具。而微博一直想把它打造成一个综合社交平台，推出了微博会员，用户推荐，各种私信评论规则等，后台事实证明这一切都没有改变用户对微博的认知，微博所作的一切都是无效的。所以当你苦恼于为什么用户没有按照我的设想要用产品的时候，一定要想着我该怎样变才能迎合用户的需求，而不是去想我该怎样变才能让用户认可产品的设计？）
 5. 用户流失情况。一方面需要监测用户的流失率，比如新用户进来后，第一、三、七、三十天还在使用产品的有多少人。流失率的变化可以直观的反应APP再朝好的方向发展还是不好的方向发展。行业中也有一些平均水平指标，你可以参考这些指标评判自己APP的好坏。另一方面需要找到用户流失的地方，看看用户在哪些地方流失了，然后有的放矢，进行相应的改动。如果有能力的话，建模将用户流失的各种情况都刻画出来，这样在产品的后续改动中就更加游刃有余了。
 6. 活跃用户动态。密切关注APP活跃用户的动态，倾听他们的声音。一旦发现异常立马组织人员商讨对策。活跃用户（或者说核心用户）是APP最宝贵的资源，关注他们的一举一动，这个重要性不需要多说了吧。
 7. 用户特征描述。这点和指标关系不大，有点建模的意思了。将用户的各个指标特征进行描述，越详细越好。如性别，年龄，地域，手机型号，网络型号，职业收入，兴

趣爱好等等。这些数据平时没什么用，但对于产品人员来说，有时候会给他们很大的灵感。如果可能的话，还可以分以下维度：如活跃用户的特征是什么样的，较沉默的用户的特征是怎样的，流失用户的特征是怎样的。

8. 用户生命周期的监测。这个是专门针对那些社交、游戏类的APP来说的。当你的APP上线一段时间后（6-12个月），你可以回头看看一个正常的用户，完整的体验你的APP的流程是怎样的，大概需要多少时间。根据这个数据再结合一些其它数据可以大致的估算下你的产品能够到怎样的规模，让你的BOSS们知道这款产品最终能发展成什么样。当然这个很难，产品的发展受到太多因素的影响，光靠你一个数据分析师来预测显然是不那么靠谱的。

- 淘宝和京东有什么区别，优缺点？

淘宝和京东，一个是最大的平台电商，一个是最大的自营电商。淘宝的客户是商家，京东的客户是用户。淘宝的盈利方式是广告，618活动。京东是赚差价（通过自有仓储和物流压缩成本）。

1.淘宝

优点：价格低；种类众多，覆盖面广，无所不有；社区互动性更强；背靠阿里系生态圈；

缺点：质量参差不齐；

2.京东

优点：物流快；售后服务好；搜索可以过滤筛选有优惠活动的商品；3C数码家电产品更让人信赖；提供白条支付方式；

缺点：品类丰富度较低商品选择相对较少；

- 你觉得拼多多未来的商业体量能超过京东吗？

9月中旬，中国互联网企业最新市值排行榜显示：阿里巴巴依旧第一，拼多多市值暴增为332亿美元，京东则为：402亿美元。此时，拼多多、京东双方相差70亿美元；后续一段时间内，由于刘强东出现负面，导致京东股价下跌，低值时期为372亿美元，而拼多多，可谓是在骂声中意外的表现更加坚强，市值居然达到了332亿美，与京东的差距已经无限接近，后期竟一段时间内市值超过了京东。

不能，不同领域-科技，拼多多的质量，砍价引起不满。

能，拼多多价格便宜亲民，用户群体覆盖广-搞广告副业。

- 如果你是淘宝的运营者，客服近期接到很多反馈说发货时间太久，你要怎么解决这个问题？

1.确定是否真实存在这个问题：

确定发货时间太久占总客户比例。

2.如果真的存在

找到发货太慢的具体原因，仓库，商家，快递。

- 便利店选址

预算：自身的产品的供应链：商圈店，社区店还是加盟店

盈利情况：竞品、人流量、客流质量

费米估计营业额：人流量*人流质量*平均每人利润

- 在一个5km的商圈内，大中小三种超市，一家中型超市发现自己长期利润很低，建议。

确认是自己的问题还是商圈的问题：调查其他超市

如果是自己的问题：品类价格选址差异分析，明确自身主要目标客户群体，减少与其他超市的竞争。

提升建议：推广活动，积分体系，首单优惠，老顾客优惠。完善网络下单，送货的功能。

- 一家卖火腿肠的店铺，经过一段时间销售，积累了很多客户评价。

从哪些角度分析这些评价呢？

怎么样分析能让结果更加落地呢？

评价行为本身代表了客户的参与度，宏观上可以计算品牌整体参与度（评价人数/购买人数），和行业不同品牌做对比，验证我们的客户评价参与度、好评占比处于行业什么位置。

业务方关注的点可以划分为：物流，包装，日期，口感，价格，品牌认同等几大方面，各个方面可以找关键词来判定正负情感倾向占比，找到负面评价占比高的方面重点改善。

- 衡量产品改版的效果

新功能是否受欢迎：功能活跃比 = 新功能用户数 / 同期活跃用户数

对产品流程转化率是否有提升（漏斗模型）

对产品整体留存的影响

用户究竟如何使用新功能

- 恶意刷单检测

商家特征：商家历史销量、信用、产品类别、发货快递公司等

用户行为特征：用户信用、下单量、转化率、下单路径、浏览店铺行为、支付账号

环境特征（主要是避免机器刷单）：地区、ip、手机型号等

异常检测：ip地址经常变动、经常清空cookie信息、账号近期交易成功率上升等

评论文本检测：刷单的评论文本可能套路较为一致，计算与已标注评论文本的相似度作为特征

图片相似度检测：同理，刷单可能重复利用图片进行评论

- 评价一个你常用的手机APP，优缺点，如何改进

- B2B与B2C的商业模式有什么区别？

1) 产品特征：B2B产品主要以工业品和服务为主，会有标准化产品，但更多会需要根据客户需求定制（比如软件行业的二次开发、OEM服务等），产品用于主要用于企业大型生产或者体系建设等；B2C产品主要以标准化的消费品和服务为主，可批量生产，产品用途主要用于个人使用或者家庭等日常消费；

(2) 定价特征：B2B以竞争性投标为主，通常需要买卖双方通过协商判定来确定，强调用户的成本分析；B2C的定价主要体现卖方的意愿；

(3) 渠道特征：B2B主要以直销渠道为主，直接；B2C一般通过中间商销售，间接渠道，比如日常消费品企业不会自建线下门店，而是选择投放大型商超；

(4) 销售方式：B2B通常会搭建销售团队，注重销售人员的专业度，广告等宣传方式一般作为销售线索收集手段；B2C强调广告，注重知名度和美誉度，人员促销为辅；

(5) 购买关系：B2B交易复杂，通常购买量大，为理性和专业性采购，需要通过合同等契约来约定和固化关系；B2C为简单交易，购买量小，基于个人感性采购，无需契约；

(6) 决策特征：B2B的决策链较长，一般是团队决策，受到多个部门人员影响，但程序明确；B2C的决策为个人决策，无程序且模糊

(7) 品牌定位：B2B侧重产品功能和厂商的服务能力，企业客户更看重公司品牌，B2C侧重产品或服务的品牌影响力与价格高低，消费者更多的关注产品品牌。

- 内容电商是指用户通过观看短视频、优质文章而引发的购物行为。请你讲讲内容电商为什么兴起？以及内容电商与传统电商相比有什么优势劣势？

用户容易被优质内容吸引。吸引过后自媒体等内容产出者需要将庞大的客户流变现。
内容电商就是变现的一种方式。

优缺点：

非常适合单价高、非刚需的产品推广。用户粘性大。目标用户群体明确。

长时间用户积累。转化效果不稳定。

- 列举出三个社交电商。且解释拼多多有什么优点。社交电商有什么优点？

拼多多、小红书、淘宝直播、抖音直播。

拼多多：低价。社交自传播能力强。

社交电商：社群聚集、共性明显转化率高。价格便宜-拼团。给了很多小商家促销渠道。

- 基于微信生态的电商，你觉得它们会给阿里京东造成什么影响？

随着物联网的到来，品牌社群将扮演重要角色，用户消费特征是情景感知，品牌要跟用户产生情感交流，形成社群共创、价值分享。各大平台需要根据自身特色，找准切入点，不断创新玩法，实现用户裂变，而这所有一切的前提，是用户体验，不能再像传统电商那样粗暴卖货。

- 如果让你提升一款社交产品的活跃度，包括说用户关系的建立及互动，你想怎么做呢？

问题分析：结合产品类型（图片社交、熟人社交、陌生人社交），注意排除周期性变化，考虑产品和用户的生命周期。分析现有用户，找到不活跃的用户群体特点。

解决方法：好友推荐、提高粘性。

活动活跃：不管是什么类型的产品，活动都能起到一定的活跃作用。

·话题推荐：根据用户的行为记录，推荐用户可能感兴趣的话题，例如知乎、微博。

·签到打卡：刺激用户的挑战心理，坚持登录使用，例如薄荷阅读、keep。

·消息提醒：社交产品的消息提醒主要有两种，关系提醒和话题提醒。

·用户成长体系：根据用户登录等行为划分用户等级，满足用户的虚荣心，例如QQ。

·用户激励体系：主要包括了积分激励，徽章等虚拟物品激励，例如京东，github。

效果评估：在执行活跃度提升方案前，我们就要设定合理的评估标准和数据指标，以便于对提升效果进行评估。

在方案执行期间，及时通过数据评估方案效果，根据数据反馈来改进优化方案。”

- 定位小红书的消费者画像。对于这些目标群体列举你会采取的运营方式。

个人信息：性别、年龄、职业、收入、地区、婚育状况、爱好

产品方面信息：关注话题（品类、品牌、价格区间）、登陆时间（频率）、活跃度（评论互动）、购买情况

用户属性：性别分析、语言分析、省份分布、城市分布、终端分析、机型分析
粉丝活跃度：互动人数/粉丝总数，可以定期群发开放性的问题、带有互动性的消息或者通过活动的参与情况判断粉丝活跃度，粉丝活跃度反映了粉丝的质量是否优良
找到目标群体经常出没的其他APP或影视节目。
定向推荐。
推出拼购活动，多人购买优惠。

业务体系指标

<http://www.woshipm.com/pd/3086815.html>
<http://www.woshipm.com/operate/3280175.html>

为什么需要搭建指标体系

对于互联网产品数据分析师来说，搭建指标体系可以很好的梳理业务关系，提高问题分析效率。

我认为指标体系的主要目的有：

- 确立关键指标（北极星指标），统一各团队的努力方向；
- 搭建产品数据监控指标看板，方便综合分析各项指标，及时发现业务问题；
- 成熟的指标体系可以高效地为各部门各团队提供数据支持。

如何搭建

Step1 选取关键指标（北极星指标）

北极星指标衡量商业价值，哪一个或者几个指标最能反映产品创造的价值。

一般从两个角度思考：什么指标最能体现给用户产生的价值？什么指标最能体现给公司带来的收益？

关键指标一般要求简单直观，便于大家朝着其努力。

- 对于社交类产品来说，DAU很关键，因为产品核心是人们通过这款产品沟通交流；
- 但对于求职类产品，关键指标应该是活跃有效简历数和发布职位公司数，只有求职者和机会数量可观的情况下，才更加有可能在平台解决大家的职业发展问题；
- 对于电商类产品，GMV很重要，意味着在平台成交的金额大小；
- 对于视频类产品，目的是为了帮助用户杀时间，那么关键指标是观看时长。

Step2 从AARRR模型出发搭建产品基本指标体系

AARRR是Acquisition、Activation、Retention、Revenue、Refer这个五个单词的缩写，分别对应用户生命周期中的5个重要环节。

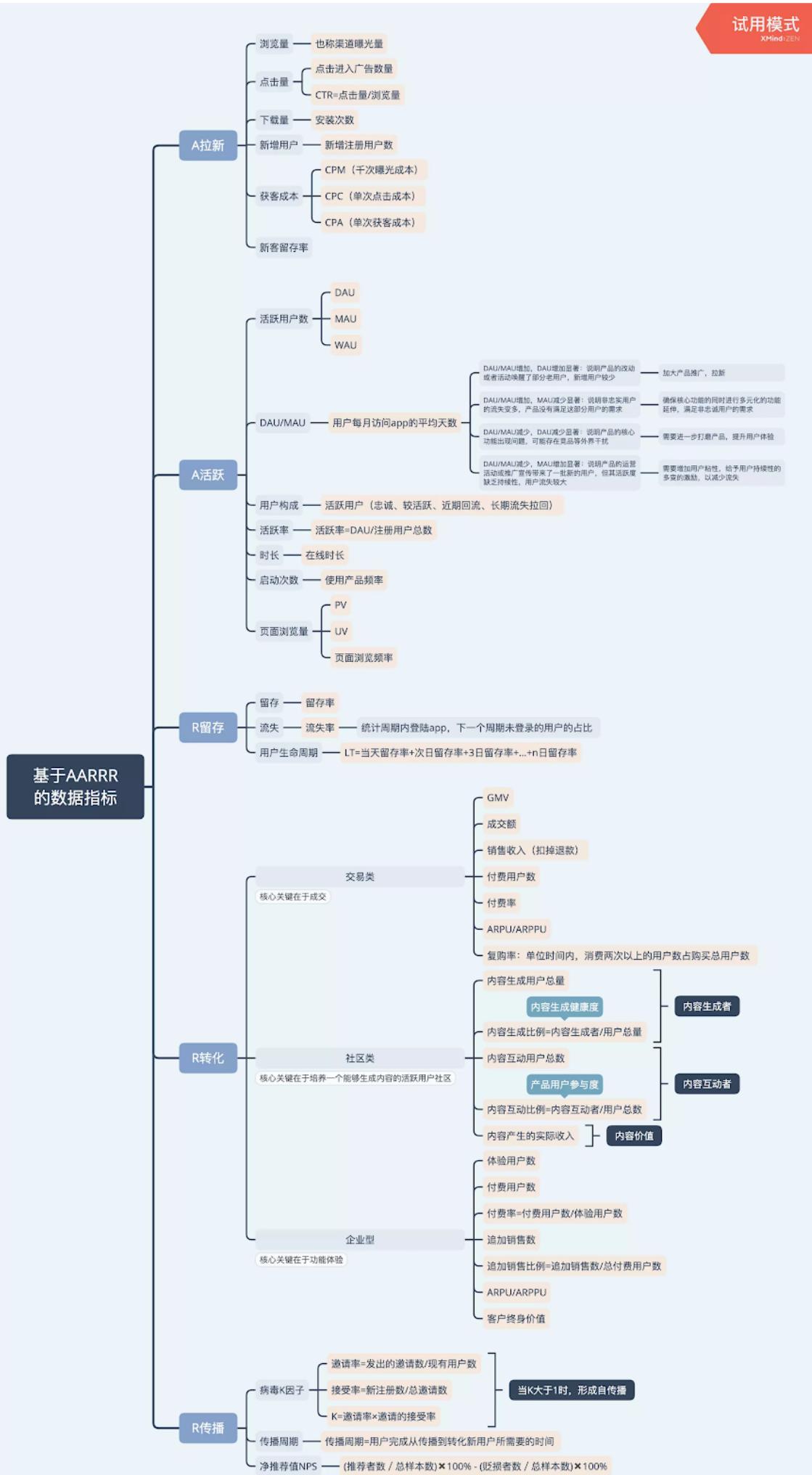
A拉新：通过各种推广渠道，以各种方式获取目标用户，并对各种营销渠道的效果评估，不断优化投入策略，降低获客成本。利用这个模块可以很好帮助市场推广部门比较各个渠道的拉新效果，评估新用户的用户质量。

A活跃：活跃用户指真正开始使用了产品提供的价值，我们需要掌握用户的行为数据，监控产品健康程度。这个模块主要反映用户进入产品的行为表现，是产品体验的核心所在。

R留存：衡量用户粘性和质量的指标。

R转化（变现）：主要用来衡量产品商业价值。

R传播：衡量用户自传播程度和口碑情况。

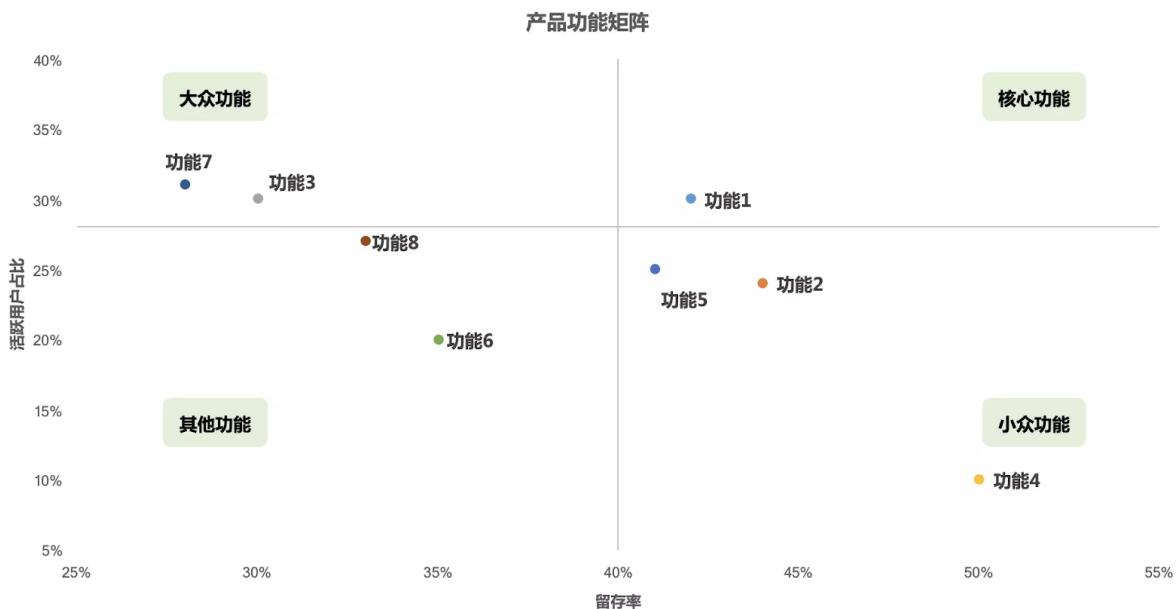


以上图片基于AARRR模型梳理了相关重要指标，当然，对于不同产品，还会有更多基于个性化功能的指标，这里就不做延展。

整合各模块指标可以得出用户**每个步骤的健康程度**，再通过时间维度（环比同比）、空间维度（各个渠道对比、各个地域）交叉对比，可以初步发现数据异常点所在，然后就需要分析师针对异常指标深入分析原因，找出解决办法。这里推荐一篇文章《产品日活DAU下降，我该如何着手分析？》很好地给出了一个DAU异常的分析方法。

Step3 产品的功能模块指标监控体系

对于一个产品，首先需要梳理出各功能模块有哪些，可以建立产品功能矩阵，对各功能的定位和表现有个大概认知。



上图即产品功能矩阵，**横轴是功能留存率**，表示当前功能的用户粘性；纵轴是活跃用户占比，表示某周期使用当前功能的**用户量/该周期的产品活跃用户量**。做出这个矩阵后，我们就可以看到不同的功能在矩阵中的位置分布（注：原点可以选择：**各产品留存率、活跃用户占比均值**）。

1. 大众功能（曝光量大，使用率高）需要了解用户的使用流程，对可优化的点进行优化，不断简化升级。
2. 核心功能（用户使用率低，功能留存率比大众功能要高）我们在核心功能中应该寻找是否存在高价值需求，我们可以推广这个需求的使用率来提升用户留存，探索增长突破口，不断完善扩充。
3. 小众功能（功能留存适中但用户使用率低）可以寻找可能有潜力的模块优化，不断筛选优化。
4. 其他功能（使用率低，留存率也低）对于不常用的功能模块适当摒弃，对于有必要保留的功能模块分析问题所在，或者简化操作流程，不断迭代更新。

建立完功能矩阵后，针对**核心功能和大众小众功能**，都需要**分别搭建监控体系**，步骤如下：

1. 确认产品功能核心目标，可以和产品经理确认功能目的。
2. 梳理功能使用路径，可以通过看产品的RPD文档，或者自己体验得到。

3. 监测指标确认，根据使用路径，整理出每一步骤的表现指标，目的是希望用户可以完成我们设想的任务，如果产品功能有做出调整，也可以用来评估前后效果。

「抖音」的数据指标体系搭建

DAU下降归因分析

<http://www.woshipm.com/data-analysis/2467030.html>

产品核心数据异常是在工作中经常会遇到的问题，一款信息流APP平时日活稳定在79w-80w之间，但是在6月13日起突然掉到了78.8w，到6月15日已经掉到78.5w，这时产品负责人着急了，让你尽快排查一下数据下跌的原因。这样的问题对大多数人来说还是比较头疼的，因为对于80w量级的产品，一两万并不是一个非常大的波动，但原因还是要排查。

核心点：先做数据异常原因的假设，后用数据验证假设。

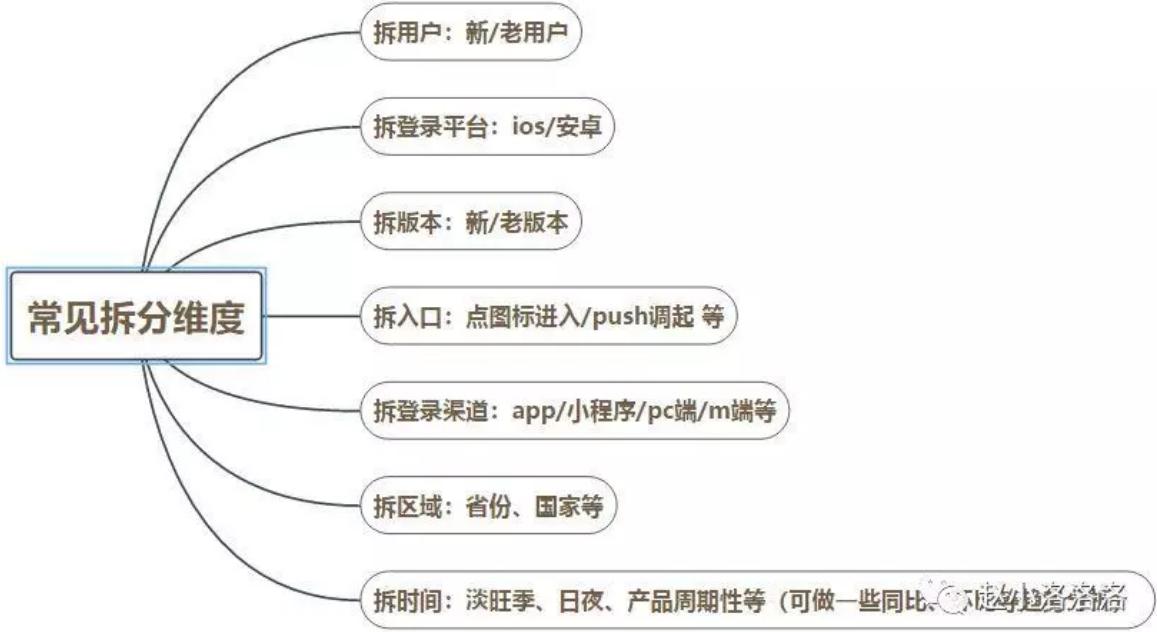
不建议大家第一步先自己对着数据去拆，影响日活数据的因素很多，不可能把所有维度逐一拆解对比，容易浪费时间却没有任何有价值的发现。做数据异常原因分析的核心就是结合以往经验及各种信息，找出最有可能的原因假设，通过数据的拆分进行多维度分析来验证假设，定位问题所在。过程中可能会在原假设基础上建立新的假设或者是调整原来假设，直到定位原因。

第一步：确认数据真实性

在开始着手分析前，建议先确认数据的真实性。我们经常会遇到数据服务、数据上报、数据统计上的BUG，在数据报表上就会出现异常值。所以，找数据流相关的产品和研发确认下数据的真实性吧。

第二步：根据几个常见维度初步拆分数据

或者拆已做好的用户画像。



计算影响系数：每一项数据都要和以往正常值做对比，算出影响系数。

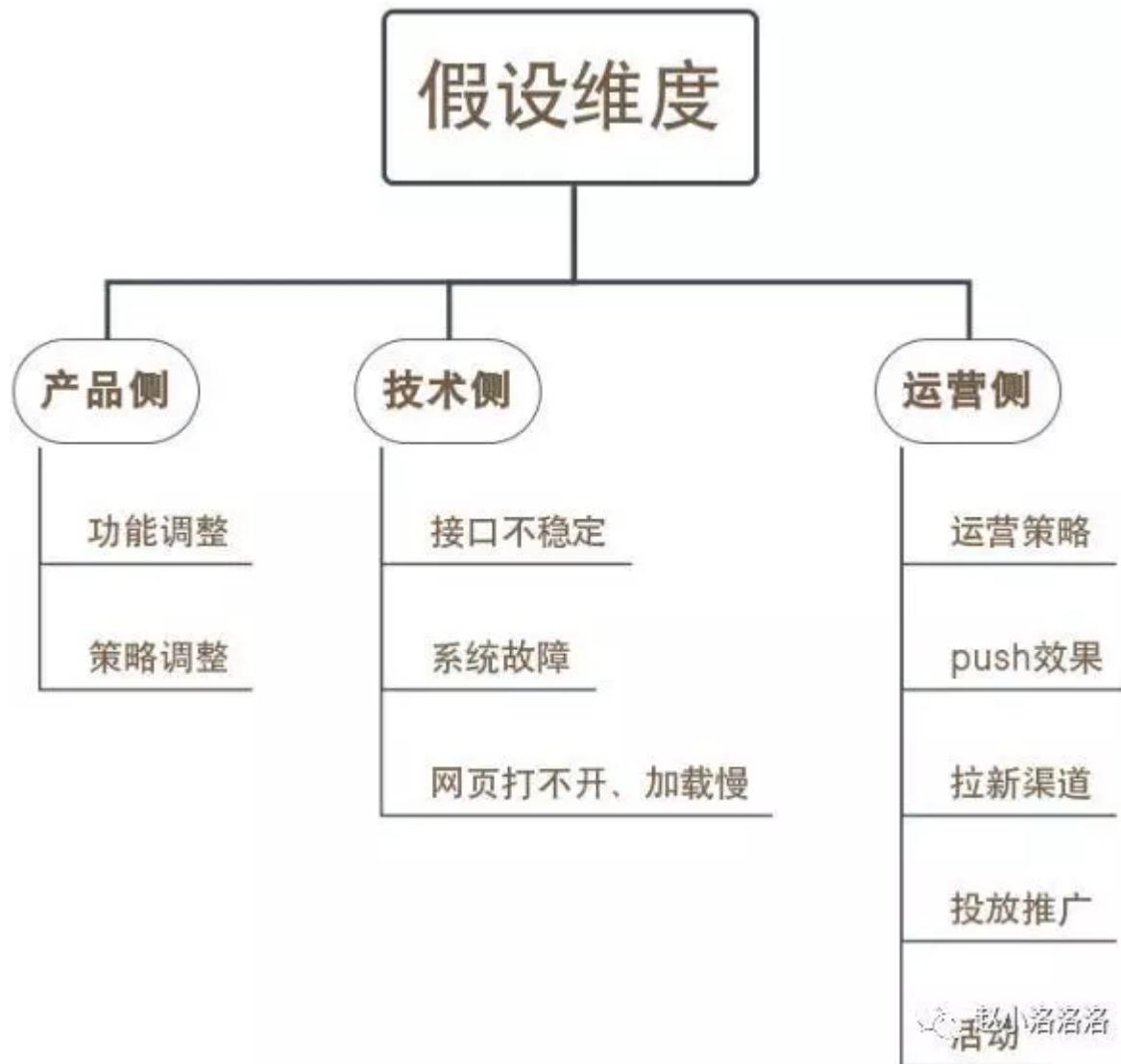
影响系数=(今日量-昨日量)/(今日总量-昨日总量)

影响系数越大，说明此处就是主要的下降点

以上是几种常见的初步拆分维度，通过初步拆分，定位原因大致范围。

第三步：异常范围定位后，进一步做假设

针对初步定位的影响范围，进行进一步的排查。分三个维度来做假设，建议针对数据异常问题专门建一个群，拉上相应的产品、技术、运营人员一起，了解数据异常时间点附近做了什么产品、运营、技术侧调整。



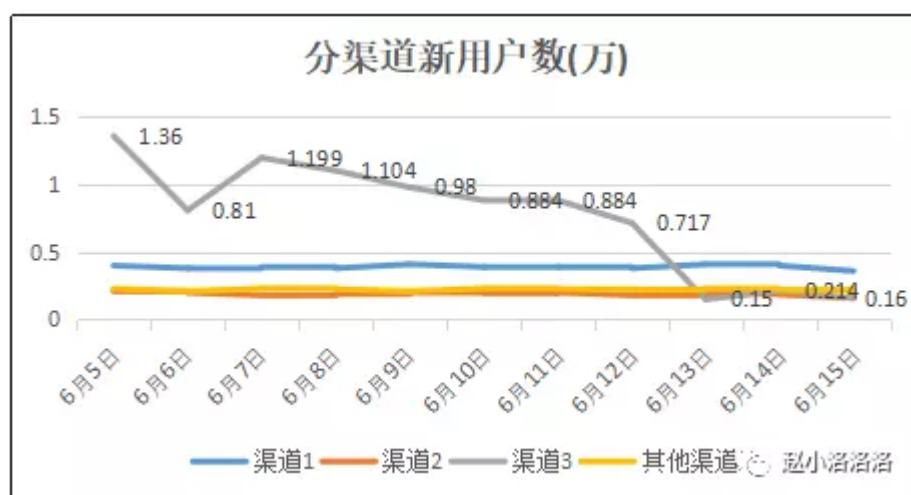
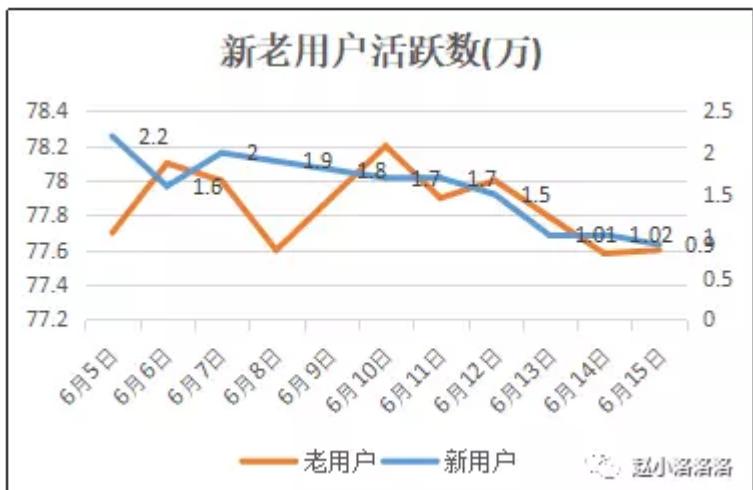
综合考虑以往数据异常原因、产品运营技术侧调整、初步定位的影响范围最可能由什么原因造成，再结合自身业务经验确定几个最可能的原因假设，给这些假设排数据验证的优先级，逐一排查。

最后：细分假设，确立原因

除了上述，可以细分分析的维度实在太多，逻辑上说核心点在于一个假设得到验证后，在这个假设为真的基础上，进行更细维度的数据拆分。我们需要记住这种分析方式，当猜测是某种原因造成数据异常时，只要找到该原因所代表的细分对立面做对比，就可以证明或证伪我们的猜测，直到最后找到真正原因。

案例分析

以上就是核心数据异常的分析套路，是不是刚才拿到问题还不知道从哪开始分析，现在觉得其实有很多点可以去着手？让我们回到刚才的案例吧。



大数据Linux

Linux基础操作

https://blog.csdn.net/zzcv/article/details/2145362?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommentFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommentFromMachineLearnPai2-1.nonecase

Hive

- Hive是什么？

是建立在Hadoop体系架构上的数据仓库工具，用于存（HDFS）和处理（Mapreduce）海量结构化数据。使得数据相关人员使用SQL语言就可以进行海量数据的处理、分析和统计工作。

- 为什么要使用Hive？

如今公司面对的数据量是亿级别的，需要面临大规模数据处理的问题，如果使用传统的比如MySQL数据库系统，做个简单的查询都要一到两天时间。Hive使得拥有SQL技能的人员非常容易地切换到Hadoop领域，充分利用Hadoop的大数据处理能力。

- 数据库和数据仓库的区别

- 1) 简单理解下数据仓库是多个数据库以一种方式组织起来
- 2) 数据库强调范式，尽可能减少冗余
- 3) 数据仓库强调整查询分析的速度，优化读取操作，主要目的是快速做大量数据的查询
- 4) 数据仓库定期写入新数据，但不覆盖原有数据，而是给数据加上时间戳标签
- 5) 数据库采用行存储，数据仓库一般采用列存储
- 6) 数据仓库的特征是面向主题、集成、相对稳定、反映历史变化，存储数历史数据；数据库是面向事务的，存储在线交易数据
- 7) 数据仓库的两个基本元素是维表和事实表，维是看待问题的角度，比如时间、部门等，事实表放着要查询的数据

- Hive的数据类型

基础数据类型：数值型，日期时间型，字符串型等

复杂数据类型：Map（字典），Array（同类型元素数组）

- 内部表和外部表的区别

创建表时：创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。

删除表时：在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。这样外部表相对来说更加安全些，数据组织也更加灵活，方便共享源数据。

- 数据的导入导出

从本地导入：load data local inpath '/home/1.txt' (overwrite) into table student;

从Hdfs导入：load data inpath '/user/hive/warehouse/1.txt' (overwrite) into table student;

查询导入：create table student1 as select * from student;(也可以具体查询某项数据)

查询结果导入：insert (overwrite) into table staff select * from track_log;

Hive面试题整理

https://blog.csdn.net/qq_36174081/article/details/89945050?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommentFromBaidu-5.nonecase&depth_1-utm_source=distribute_pc_relevant.none-task-blog-BlogCommentFromBaidu-5.nonecase

https://blog.csdn.net/wxfghy/article/details/81361400?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-1.nonecase

Hive SQL

https://blog.csdn.net/Thomson617/article/details/83212338?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-10.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-10.nonecase

Hive优化

https://blog.csdn.net/l1028386804/article/details/80629279?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-2.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommentFromMachineLearnPai2-2.nonecase

Spark

相对于Hadoop的MapReduce会在运行完工作后将中介数据存放到磁盘中，Spark使用了内存内运算技术，能在数据尚未写入硬盘时即在内存内分析运算。Spark在内存内运行程序的运算速度能做到比Hadoop MapReduce的运算速度快上100倍，即便是运行程序于硬盘时，Spark也能快上10倍速度。Spark允许用户将数据加载至集群内存，并多次对其进行查询，非常适合用于机器学习算法。

这里出现了与它的一个同类大胸弟——Hadoop的对比，同样是大数据处理的常用工具（具体有关Hadoop的简明教程我会另开一篇文章来写），说明中告诉我们，Spark是一直在内存里面算，Hadoop是运行过程中会将中间的数据不断写入再读出磁盘，而我们也知道，系统I/O操作是一个比较耗时的操作，Hadoop由于其实现机制问题，不得不进行大量I/O操作，而Spark一直在内存中干活，就省去了这些麻烦，只在数据完全处理完毕后一次性输出。

另外，从两者任务的执行机制来看，Spark是将任务做成了DAG有向无环图的模式，而Hadoop是用MapReduce(MR)模式，而DAG模式本身是比MR模式在处理效率上更高，因为DAG可以有效减少Map和Reduce任务之间传递数据的过程。（这里只是大概的一个表述，言语不恰当处请见谅）

其它的具体对比可以参照知乎上的回答。

另外，Spark本身在需要大量迭代的运算上非常有优势，而机器学习算法恰恰是会涉及到许多迭代操作的，因此Spark跑机器学习算法是非常完美的配合。

Spark的运行架构

Spark的一个最大特点就是它的集群式/分布式计算，就是它可以将一个大任务分解成很小任务，交给很多台机器去分别完成，最后汇总。就是“人多力量大”的道理。

而它要完成这种分布式的计算，就需要有一套标准的逻辑去负责分解、分配任务，以及搜集最后的处理结果，这些就是通过以下的架构来实现的。

<https://blog.csdn.net/u011204847/article/details/51010205>