# 一、机器学习原理

### 最大熵模型

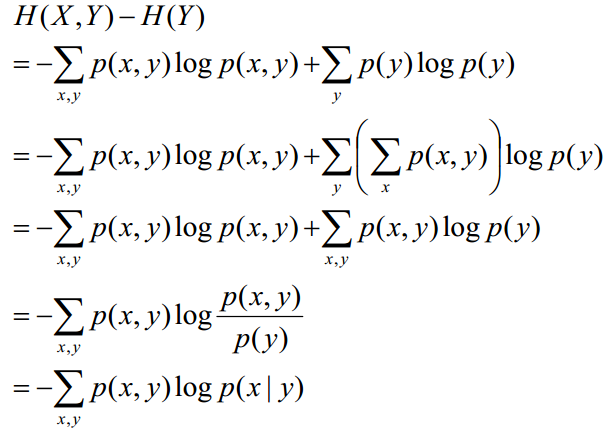
定义：

熵 的理解：熵是随机变量不确定性的度量，不确定性越大，熵值越大；若随机变量退化成定值，熵为0。

#### ****条件熵****

条件熵H(X|Y)= H(X，Y)- H(Y)

推导：



#### ****相对熵****

### 聚类

#### ****k-means聚类算法****

k-means算法目标是：以k为参数，把n个对象分成k个簇，使簇内具有较高的相似度，而簇间的相似度较低。

 k-means算法的处理过程如下：首先，随机地 选择k个对象，每个对象初始地代表了一个簇的平均值或中心;对剩余的每个对象，根据其与各簇中心的距离，将它赋给最近的簇;然后重新计算每个簇的平均值。 这个过程不断重复，直到准则函数收敛。通常，采用平方误差准则，其定义如下：

http://img.blog.csdn.net/20150903211428816?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

       优点：简单直接（体现在逻辑思路以及实现难度上），易于理解，在低维数据集上有不错的效果（简单的算法不见得就不能得到优秀的效果）。

       缺点：对于高维数据（如成百上千维，现实中还不止这么多），其计算速度十分慢，主要是慢在计算距离上（参考欧几里得距离，当然并行化处理是可以的，这是算法实现层面的问题），它的另外一个缺点就是它需要我们设定希望得到的聚类数k，若我们对于数据没有很好的理解，那么设置k值就成了一种估计性的工作。

#### ****层次聚类算法****

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

**分裂型层次聚类：**

    1.将样本集中的所有的样本归为一个类簇；

    repeat：

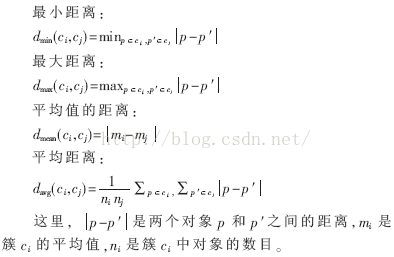
        2.在同一个类簇（计为c）中计算两两样本之间的距离，找出距离最远的两个样本a,b；

        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；

    util： 达到聚类的数目或者达到设定的条件

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



这里给出采用最小距离的凝聚层次聚类算法流程：

(1) 将每个对象看作一类，计算两两之间的最小距离；

(2) 将距离最小的两个类合并成一个新类；

(3) 重新计算新类与所有类之间的距离；

(4) 重复(2)、(3)，直到所有类最后合并成一类。

总结：

       优点：1，距离和规则的相似度容易定义，限制少；2，不需要预先制定聚类数；3，可以发现类的层次关系（在一些特定领域如生物有很大作用）；

       缺点：1，计算复杂度太高（考虑并行化）；2，奇异值也能产生很大影响；3，算法很可能聚类成链状（一层包含着一层）；4，算法不需要预定聚类数，但是我们选择哪个层次的聚类作为我们需要的聚类效果，这需要我们按照实际客观情况以及经验来完成，毕竟就凝聚聚类来说，从最底层的每个个体作为一个个体，到最顶层所有个体合并为一个个体，其中的聚类结果可能有许许多多种。当然针对这个问题也有许多解决方案，其中一个常用的就是凝聚到某个程度其聚类之间的距离都大于某个阈值k，就停止凝聚。

#### ****密度聚类****

密度聚类方法的指导思想是，只要样本点的密度大于某阈值，则将该样本添加到最近的簇中。如：一个中国地图上标着每平方公里的人口密度，于是我们就可以通过这个密度进行聚类，某个地方很密，我就可以认为这个地方是个城市。

优点：这类算法能克服基于距离的算法只能发现“类圆形”(凸)的聚类的缺点，可发现任意形状的聚类，且对噪声数据不敏感。

缺点：计算密度单元的计算复杂度大，因此需要建立空间索引来降低计算量。

DBSCAN算法、密度最大值算法。

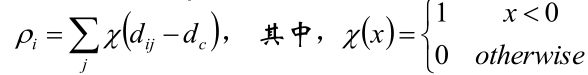
##### 1）DBSCAN算法

它将簇定义为密度相连的点的最大集合，于是能够把具有足够高密度的区域划分为簇，并可在有“噪声”的数据中发现任意形状的聚类。

##### 2）密度最大值算法

密度最大值聚类是一种简洁优美的聚类算法, 可以识别各种形状的类簇, 并且参数很容易确定。

**定义：局部密度ρi**

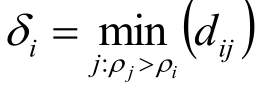


dc是一个截断距离, ρi 即到对象i的距离小于dc的对象的个数，

即：ρi  = 任何一个点以dc为半径的圆内的样本点的数量。

由于该算法只对ρi 的相对值敏感, 所以对dc的选择是稳健的，一种推荐做法是选择dc ，使得平均每个点的邻居数为所有点的1%-2%。

**定义：高局部密度点距离δi(简称“高密距离”(注：该称呼不具代表性))**

****

**解释：**

对第i个样本，我们可以算其局部密度ρi，第j个样本，我们可以算其局部密度ρj，其他样本点同理比如有：ρ1，ρ2，ρ3。为了方便说明，假设：ρi= 8，ρ1=9，ρ2=10，ρ3=4，ρj=20.然后对于第i个样本，将其局部密度和其他所有样本的局部密度作比较，于是乎：因为ρi  < ρ1，所以算算样本i和1之间的距离。同理，算算i和2之间的距离，j和i之间的距离。而因为ρi  > ρ3，所以样本i和3之间的距离就不算了。总之就是对于第i个样本，算算密度比它高的样本与其的距离。最后，在这些距离中取最小的那个，就是**高局部密度点距离**。

**一句话描述就是：**

   比我**高**的**局部密度点**到我的最小**距离**。(把这句话的黑体字连起来看看)。**簇中心和异常点的识别：**

**簇中心：**那些有着比较大的局部密度ρi和很大的高密距离δi 的点被认为是簇的中心。

**异常点：**高密距离δi较大但局部密度ρi较小的点是异常点；

##### 3）谱聚类

　谱聚类算法的主要优点有：

　　1）谱聚类只需要数据之间的相似度矩阵，因此对于处理稀疏数据的聚类很有效。这点传统聚类算法比如K-Means很难做到

　　2）由于使用了降维，因此在处理高维数据聚类时的复杂度比传统聚类算法好。

　谱聚类算法的主要缺点有：

　　1）如果最终聚类的维度非常高，则由于降维的幅度不够，谱聚类的运行速度和最后的聚类效果均不好。

　　2) 聚类效果依赖于相似矩阵，不同的相似矩阵得到的最终聚类效果可能很不同。

### 决策树

#### ID3决策树

##### 理论基础：

熵度量了事物的不确定性，越不确定的事物，它的熵就越大。具体的，随机变量X的熵的表达式如下：



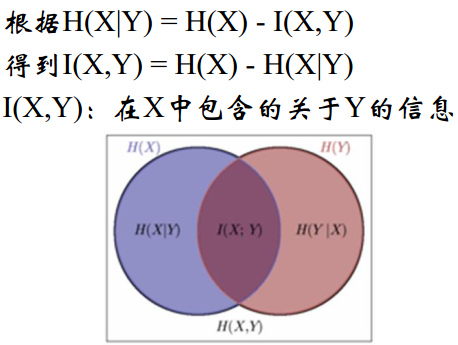
举个例子，比如X有2个可能的取值，而这两个取值各为1/2时X的熵最大，此时X具有最大的不确定性。值为H(X)=−(1/2log1/2+1/2log1/2)=log2。如果一个概率1/3，一个概率2/3，则对应熵为H(X)=−(1/3log1/3+2/3log2/3)=log3−2/3log2 < log2

**联合熵：**



**条件熵：**





称为互信息或者信息增益。

##### 算法思路：

ID3算法就是用**信息增益大小**来判断当前节点应该用什么特征来构建决策树，用计算出的信息增益最大的特征来建立决策树的当前节点。这里我们举一个信息增益计算的具体的例子。比如我们有15个样本D，输出为0或者1。其中有9个输出为1， 6个输出为0。 样本中有个特征A，取值为A1，A2和A3。在取值为A1的样本的输出中，有3个输出为1， 2个输出为0，取值为A2的样本输出中,2个输出为1,3个输出为0， 在取值为A3的样本中，4个输出为1，1个输出为0.

**样本D的熵为：**



**样本D在特征A下的条件熵为：**

 **信息增益为：**



##### 算法缺点：

1. **不能处理连续属性。**ID3没有考虑连续特征，比如长度，密度都是连续值，无法在ID3运用。这大大限制了ID3的用途。
2. **用信息增益选择属性时偏向于选择分枝比较多的属性值。**ID3采用信息增益大的特征优先建立决策树的节点。很快就被人发现，在相同条件下，取值比较多的特征比取值少的特征信息增益大。比如一个变量有2个值，各为1/2，另一个变量为3个值，各为1/3，其实他们都是完全不确定的变量，但是取3个值的比取2个值的信息增益大。如果校正这个问题呢？
3. ID3算法对于缺失值的情况没有做考虑
4. 没有考虑过拟合的问题

#### C4.5决策树

参考：https://blog.csdn.net/gumpeng/article/details/51397737

 C4.5的思路是将连续的特征离散化。比如m个样本的连续特征A有m个，从小到大排列为a1,a2,...,am则C4.5取相邻两样本值的平均数，一共取得m-1个划分点，其中第i个划分点Ti表示为：。对于这m-1个点，分别计算以该点作为二元分类点时的信息增益。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。比如取到的增益最大的点为,则小于的值为类别1，大于的值为类别2，这样我们就做到了连续特征的离散化。要注意的是，与离散属性不同的是，如果当前节点为连续属性，则该属性后面还可以参与子节点的产生选择过程。

**信息增益比：**



其中D为样本特征输出的集合，A为样本特征，对于特征熵



其中n为特征A的类别数，为特征的第i个取值对应的样本个数，D为样本个数。

**算法缺点：**

1. 由于决策树算法非常容易过拟合，因此对于生成的决策树必须要进行剪枝。剪枝的算法有非常多，C4.5的剪枝方法有优化的空间。思路主要是两种，一种是预剪枝，即在生成决策树的时候就决定是否剪枝。另一个是后剪枝，即先生成决策树，再通过交叉验证来剪枝。
2. C4.5生成的是多叉树，即一个父节点可以有多个节点。很多时候，在计算机中二叉树模型会比多叉树运算效率高。如果采用二叉树，可以提高效率。
3. C4.5只能用于分类，如果能将决策树用于回归的话可以扩大它的使用范围。
4. C4.5由于使用了熵模型，里面有大量的耗时的对数运算,如果是连续值还有大量的排序运算。如果能够加以模型简化可以减少运算强度但又不牺牲太多准确性的话，那就更好了。

#### CART决策树

**不纯度计算：**



若是二分类：



对于个给定的样本D,假设有K个类别, 第k个类别的数量为Ck,则样本D的基尼系数表达式为：



特别的，对于样本D,如果根据特征A的某个值a,把D分成D1和D2两部分，则在特征A的条件下，D的基尼系数表达式为：



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

缺点

1. 无论是ID3, C4.5还是CART,在做特征选择的时候都是选择最优的一个特征来做分类决策，但是大多数，分类决策不应该是由某一个特征决定的，而是应该由一组特征决定的。这样绝息到的决策树更加准确。这个决策树叫做多变量决策树(multi-variate decision tree)。在选择最优特征的时候，多变量决策树不是选择某一个最优特征，而是选择最优的一个特征线性组合来做决策。这个算法的代表是OC1
2. 如果样本发生一点点的改动，就会导致树结构的剧烈改变。这个可以通过集成学习里面的随机森林之类的方法解决。

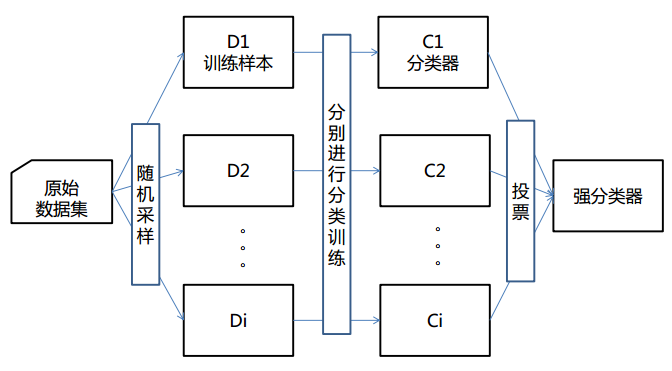
#### 三种决策树对比：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

### Bagging的策略

bootstrap aggregation

从样本集中重采样(有重复的)选出n个样本在所有属性上，对这n个样本建立分类器(ID3、C4.5、CART、SVM、Logistic回归等)，重复以上两步m次，即获得了m个分类器，将数据放在这m个分类器上，最后根据这m个分类器的投票结果，决定数据属于哪一类。



#### 随机森林

随机森林在bagging基础上做了修改

从样本集中用Bootstrap采样选出n个样本；

从所有属性中随机选择k个属性，选择最佳分割属性作为节点建立CART决策树；

重复以上两步m次，即建立了m棵CART决策树；

这m个CART形成随机森林，通过投票表决结果，决定数据属于哪一类。

### Adaboost

#### 算法原理：

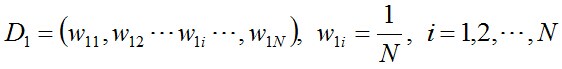
（1）初始化训练数据（每个样本）的权值分布：如果有N个样本，则每一个训练的样本点最开始时都被赋予相同的权重：1/N。

（2）训练弱分类器。具体训练过程中，如果某个样本已经被准确地分类，那么在构造下一个训练集中，它的权重就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。同时，得到弱分类器对应的话语权。然后，更新权值后的样本集被用于训练下一个分类器，整个训练过程如此迭代地进行下去。

（3）将各个训练得到的弱分类器组合成强分类器。各个弱分类器的训练过程结束后，分类误差率小的弱分类器的话语权较大，其在最终的分类函数中起着较大的决定作用，而分类误差率大的弱分类器的话语权较小，其在最终的分类函数中起着较小的决定作用。换言之，误差率低的弱分类器在最终分类器中占的比例较大，反之较小。

#### 算法流程：

**第一步：**

初始化训练数据（每个样本）的权值分布。每一个训练样本，初始化时赋予样的权值w=1/N。N为样本总数。

D1表示，第一次迭代每个样本的权值。w11表示，第1次迭代时的第一个样本的权值。 N为样本总数。

**第二步：进行多次迭代，m=1，2….M。m表示迭代次数。**

1. 使用具有权值分布Dm（m=1,2,3…N)的训练样本集进行学习，得到弱的分类器。

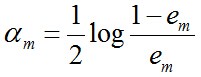
这里写图片描述

该式子表示，第m次迭代时的弱分类器，将样本x要么分类成-1，要么分类成1.**那么根据什么准则得到弱分类器？ （可以理解为分类器的阈值设置成多大）**

准则：该弱分类器的误差函数最小，也就是分错的样本对应的权值之和，最小。



1. 计算弱分类器Gm（x）的话语权，话语权表示Gm（x）在最终分类器中的重要程度。



该式是随em减小而增大。即误差率小的分类器，在最终分类器的 重要程度大。

1. 更新训练样本集的权值分布。用于下一轮迭代。其中，被误分的样本的权值会增大，被正确分的权值减小。





Dm+1是用于下次迭代时样本的权值，Wm+1,i是下一次迭代时，第i个样本的权值。其中，yi代表第i个样本对应的类别（1或-1），Gm（xi）表示弱分类器对样本xi的分类（1或-1）。若果分对，yi\*Gm（xi）的值为1，反之为-1。其中Zm是归一化因子，使得所有样本对应的权值之和为1.



**第三步：组合各个弱分类器**

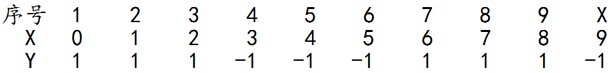


从而得到终分类器



**举例：**

  给定下列训练样本，请用AdaBoost算法学习一个强分类器。



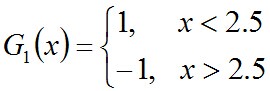
    求解过程：初始化训练数据的权值分布，令每个权值W1i = 1/N = 0.1，其中，N = 10，i = 1,2, ..., 10，然后分别对于m = 1,2,3, ...等值进行迭代。

    拿到这10个数据的训练样本后，根据 X 和 Y 的对应关系，要把这10个数据分为两类，一类是“1”，一类是“-1”，根据数据的特点发现：“0 1 2”这3个数据对应的类是“1”，“3 4 5”这3个数据对应的类是“-1”，“6 7 8”这3个数据对应的类是“1”，9是比较孤独的，对应类“-1”。抛开孤独的9不讲，“0 1 2”、“3 4 5”、“6 7 8”这是3类不同的数据，分别对应的类是1、-1、1，直观上推测可知，可以找到对应的数据分界点，比如2.5、5.5、8.5 将那几类数据分成两类。当然，这只是主观臆测，下面实际计算下这个具体过程。

对于m=1，在权值分布为**D1**（10个数据，每个数据的权值皆初始化为0.1）的训练数据上，经过计算可得：

* 1. **阈值v取2.5**时误差率为0.3（x < 2.5时取1，x > 2.5时取-1，**则6 7 8分错**，误差率为0.3），
  2. 阈值v取5.5时误差率最低为0.4（x < 5.5时取1，x > 5.5时取-1，则3 4 5 6 7 8皆分错，误差率0.6大于0.5，不可取。故令x > 5.5时取1，x < 5.5时取-1，则0 1 2 9分错，误差率为0.4），
  3. 阈值v取8.5时误差率为0.3（x < 8.5时取1，x > 8.5时取-1，则3 4 5分错，误差率为0.3）。

可以看到，无论阈值v取2.5，还是8.5，总得分错3个样本，故可任取其中任意一个如2.5，弄成第一个基本分类器为：

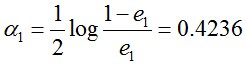


上面说阈值v取2.5时则6 7 8分错，所以误差率为0.3，更加详细的解释是：因为样本集中

* 1. 0 1 2对应的类（Y）是1，因它们本身都小于2.5，所以被G1(x)分在了相应的类“1”中，分对了。
  2. 3 4 5本身对应的类（Y）是-1，因它们本身都大于2.5，所以被G1(x)分在了相应的类“-1”中，分对了。
  3. 但6 7 8本身对应类（Y）是1，却因它们本身大于2.5而被G1(x)分在了类"-1"中，所以这3个样本被分错了。
  4. 9本身对应的类（Y）是-1，因它本身大于2.5，所以被G1(x)分在了相应的类“-1”中，分对了。

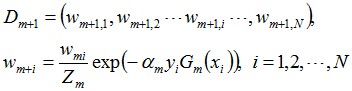
从而得到G1(x)在训练数据集上的误差率（被G1(x)误分类样本“6 7 8”的权值之和）**e1=P(G1(xi)≠yi) = 3\*0.1 = 0.3**。

然后根据误差率e1计算G1的系数：



这个代表G1(x)在最终的分类函数中所占的权重，为0.4236。

接着更新训练数据的权值分布，用于下一轮迭代：



值得一提的是，由权值更新的公式可知，每个样本的新权值是变大还是变小，取决于它是被分错还是被分正确。

即如果某个样本被分错了，则yi \* Gm(xi)为负，负负得正，结果使得整个式子变大（样本权值变大），否则变小。

第一轮迭代后，最后得到各个数据**新**的权值分布**D2**= (0.0715, 0.0715, 0.0715, 0.0715, 0.0715,  0.0715,0.1666, 0.1666, 0.1666, 0.0715)。由此可以看出，因为样本中是数据“6 7 8”被G1(x)分错了，所以它们的权值由之前的0.1增大到0.1666，反之，其它数据皆被分正确，所以它们的权值皆由之前的0.1减小到0.0715。

分类函数f1(x)= a1\*G1(x) = 0.4236G1(x)。

此时，得到的第一个基本分类器sign(f1(x))在训练数据集上有3个误分类点（即6 7 8）。

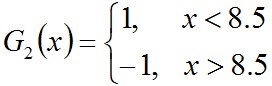
    从上述第一轮的整个迭代过程可以看出：被误分类样本的权值之和影响误差率，误差率影响基本分类器在最终分类器中所占的权重。

**迭代过程2**

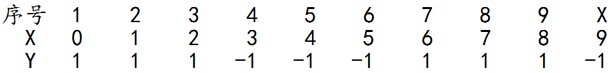
对于m=2，在权值分布为**D2**= (0.0715, 0.0715, 0.0715, 0.0715, 0.0715,  0.0715, 0.1666, 0.1666, 0.1666, 0.0715)的训练数据上，经过计算可得：

* 1. 阈值v取2.5时误差率为0.1666\*3（x < 2.5时取1，x > 2.5时取-1，则6 7 8分错，误差率为0.1666\*3），
  2. 阈值v取5.5时误差率最低为0.0715\*4（x > 5.5时取1，x < 5.5时取-1，则0 1 2 9分错，误差率为0.0715\*3 + 0.0715），
  3. **阈值v取8.5**时误差率为0.0715\*3（x < 8.5时取1，x > 8.5时取-1，**则3 4 5分错**，误差率为0.0715\*3）。

所以，阈值v取8.5时误差率最低，故第二个基本分类器为：

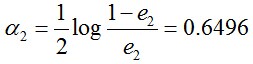


面对的还是下述样本：

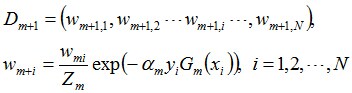


很明显，G2(x)把样本“3 4 5”分错了，根据D2可知它们的权值为0.0715, 0.0715,  0.0715，所以G2(x)在训练数据集上的误差率e2=P(G2(xi)≠yi) = 0.0715 \* 3 = 0.2143。

计算G2的系数：



更新训练数据的权值分布：



D3 = (0.0455, 0.0455, 0.0455, 0.1667, 0.1667,  0.01667, 0.1060, 0.1060, 0.1060, 0.0455)。被分错的样本“3 4 5”的权值变大，其它被分对的样本的权值变小。  
f2(x)=0.4236G1(x) + 0.6496G2(x)

此时，得到的第二个基本分类器sign(f2(x))在训练数据集上有3个误分类点（即3 4 5）。

**迭代过程3**

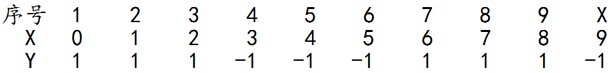
对于m=3，在权值分布为**D3**= (0.0455, 0.0455, 0.0455, 0.1667, 0.1667,  0.01667, 0.1060, 0.1060, 0.1060, 0.0455)的训练数据上，经过计算可得：

* 1. 阈值v取2.5时误差率为0.1060\*3（x < 2.5时取1，x > 2.5时取-1，则6 7 8分错，误差率为0.1060\*3），
  2. **阈值v取5.5**时误差率最低为0.0455\*4（x > 5.5时取1，x < 5.5时取-1，**则0 1 2 9分错**，误差率为0.0455\*3 + 0.0715），
  3. 阈值v取8.5时误差率为0.1667\*3（x < 8.5时取1，x > 8.5时取-1，则3 4 5分错，误差率为0.1667\*3）。

所以阈值v取5.5时误差率最低，故第三个基本分类器为：

http://img.blog.csdn.net/20141110170258640

依然还是原样本：



此时，被误分类的样本是：0 1 2 9，这4个样本所对应的权值皆为0.0455，

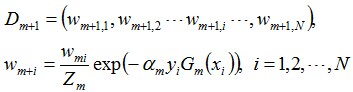
所以G3(x)在训练数据集上的**误差率e3**

= P(G3(xi)≠yi) = **0.0455\*4** = 0.1820。

计算G3的系数：

http://img.blog.csdn.net/20141103005116515

更新训练数据的权值分布：



**D4**= (0.125, 0.125, 0.125, 0.102, 0.102,  0.102, 0.065, 0.065, 0.065, 0.125)。被分错的样本“0 1 2 9”的权值变大，其它被分对的样本的权值变小。

f3(x)=0.4236G1(x) + 0.6496G2(x)+0.7514G3(x)

此时，得到的第三个基本分类器sign(f3(x))在训练数据集上有0个误分类点。至此，整个训练过程结束。

    现在，咱们来总结下3轮迭代下来，各个样本权值和误差率的变化，如下所示（其中，样本权值D中加了下划线的表示在上一轮中被分错的样本的新权值）：

1. 训练之前，各个样本的权值被初始化为D1 = (0.1, 0.1,0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)；
2. **第一轮迭代**中，样本“**6 7 8”**被分错，对应的误差率为**e1**=P(G1(xi)≠yi) = 3\*0.1 = 0.3，此第一个基本分类器在最终的分类器中所占的权重为**a1** = 0.4236。第一轮迭代过后，样本新的权值为**D2**= (0.0715, 0.0715, 0.0715, 0.0715, 0.0715,  0.0715, 0.1666, 0.1666, 0.1666, 0.0715)；
3. **第二轮迭代**中，样本**“3 4 5”**被分错，对应的误差率为**e2**=P(G2(xi)≠yi) = 0.0715 \* 3 = 0.2143，此第二个基本分类器在最终的分类器中所占的权重为**a2** = 0.6496。第二轮迭代过后，样本新的权值为**D3**= (0.0455, 0.0455, 0.0455, 0.1667, 0.1667,  0.01667, 0.1060, 0.1060, 0.1060, 0.0455)；
4. **第三轮迭代**中，样本**“0 1 2 9”**被分错，对应的误差率为**e3**
5. = P(G3(xi)≠yi) = 0.0455\*4 = 0.1820，此第三个基本分类器在最终的分类器中所占的权重为**a3** = 0.7514。第三轮迭代过后，样本新的权值为**D4**= (0.125, 0.125, 0.125, 0.102, 0.102,  0.102, 0.065, 0.065, 0.065, 0.125)。

    从上述过程中可以发现，如果某些个样本被分错，它们在下一轮迭代中的权值将被增大，反之，其它被分对的样本在下一轮迭代中的权值将被减小。就这样，分错样本权值增大，分对样本权值变小，而在下一轮迭代中，总是选取让误差率最低的阈值来设计基本分类器，所以误差率e（所有被Gm(x)误分类样本的权值之和）不断降低。

    综上，将上面计算得到的a1、a2、a3各值代入G(x)中，G(x) = sign[f3(x)] = sign[ a1 \* G1(x) + a2 \* G2(x) + a3 \* G3(x) ]，得到**最终的分类器**为：

G(x) = sign[f3(x)] = sign[ 0.4236G1(x) + 0.6496G2(x)+0.7514G3(x) ]。

### GBDT

https://www.cnblogs.com/ModifyRong/p/7744987.html

### 贝叶斯公式

朴素贝叶斯算法的算法思想是：对于给定的输入样本，在已知先验概率，和样本类别的概率分布模型（已知属于某一类别的前提下样本发生的条件概率分布）的前提下，通过贝叶斯公式求得样本类别的后验概率，将后验概率最大的样本所属的类作为分类器的输出。 先验概率+数据=后验概率

朴素贝叶斯的优点

1. 简单易操作，分类效果稳定，良好；
2. 对缺失数据不敏感（适合增量式训练）；
3. 常用于文本分类；缺点：
   1. 模型应用建立在假设基础上，假设特征相互独立，因此在实际应用中特征相关性大时效果不好。（半朴素贝叶斯是在考虑部分关联性后适当改进）；
   2. 需要知道先验概率，先验概率取决于假设，假设服从的分布模型有很多。（但一般先验概率都是用频率替代，大数定理）

#### 1.条件概率



第一角度：在B发生的基础上，A发生的概率。

另一角度：韦恩图看，A在B发生的基础上发生的概率是A和B交集的阴影部分面积占用B的比例。

#### 2. 乘法公式

1）由条件概率公式：



AB同时发生的概率是在A基础上发生B的概率乘以A本身在外部发生的概率，也是B基础上发生A的概率乘以B本身在外部发生的概率.

2）乘法公式推广：对任何正整数，当时，有：



#### 3. 全概率事件

1）如果事件组B1，B2，...满足

（1）B1，B2，...两两互斥，即，且

（2），则称是样本空间的一个划分

 设 B1,B2,...是样本空间Ω的一个划分，A为任一事件，则：



上式即为全概率公式。

（3）全概率公式的意义在于，当直接计算P(A)较为困难,而P(Bi),P(A|Bi)  (i=1,2,...)的计算较为简单时，可以利用全概率公式计算P(A)。思想就是，将事件A分解成几个小事件，通过求小事件的概率，然后相加从而求得事件A的概率，而将事件A进行分割的时候，不是直接对A进行分割，而是先找到样本空间Ω的一个个划分B1,B2,...Bn,这样事件A就被事件AB1,AB2,...ABn分解成了n部分，即A=AB1+AB2+...+ABn, 每一Bi发生都可能导致A发生相应的概率是P(A|Bi)，由加法公式得

         P(A)=P(AB1)+P(AB2)+....+P(ABn)

               =P(A|B1)P(B1)+P(A|B2)P(B2)+...+P(A|Bn)P(PBn)

#### 4. 贝叶斯公式

与全概率公式解决的问题相反，贝叶斯公式是建立在条件概率的基础上寻找事件发生的原因（即大事件A已经发生的条件下，分割中的小事件Bi的概率），设B1,B2,...是样本空间Ω的一个划分，则对任一事件A（P(A)>0),有



  上式即为贝叶斯公式（Bayes formula)，Bi常被视为导致试验结果A发生的”原因“，P(Bi)(i=1,2,...)表示各种原因发生的可能性大小，故称先验概率；P(Bi|A)(i=1,2...)则反映当试验产生了结果A之后，再对各种原因概率的新认识，故称后验概率。

### 多元线性回归

对于n维特征样本数据，多元线性回归公式：



简写：



矩阵形式：



设一共有m个样本，每个样本n个特征，则X是的矩阵。为的向量。

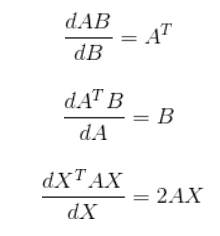
均方误差作为损失函数：



用矩阵表达：



矩阵求导法则：



**梯度下降法求解：**



令 ，求得：

### 范数

0范数，向量中非零元素的个数。

1范数，为绝对值之和。

2范数，就是通常意义上的模。



### 岭回归(Ridge Regression)

#### 一般线性回归遇到的问题

多元线性回归预测精度：这里要处理好这样一对为题，即样本的数量m和特征的数量n

时，最小二乘回归会有较小的方差

时，容易产生过拟合

时，最小二乘回归得不到有意义的结果

多元线性回归中，若是奇异矩阵，则无法求得逆矩阵。实际中会面临两个问题，一是X是否是奇异矩阵，二是X中变量是不是都做出贡献。

奇异性和贡献的意思，1，奇异矩阵的充要条件就是X矩阵的行列式为0（|X|=0），我们知道如果一个矩阵中存在某几个个向量共线（就是两个向量成比例），那这个矩阵的行列式就一定是0，即该矩阵叫做奇异矩阵。2，变量有没有贡献就是指某个X指标对结果Y有没有影响，比如Y是某学生的考试平均分，X中有x1（语文成绩），x2（数学成绩），x3（吃饭速度）……这里x3这个分量就是打酱油的数据，对Y的最终贡献率为0，那我们就要把x3这个分量剔除。

#### 岭回归的概念

在多元平方误差的基础上加入**L2正则项**：

 （1）

对求偏导：



令 ，求得：

可以证明（1）式的等价：



当自变量间存在共线性时，｜X′X｜≈0，我们设想给X′X加上一个正常数矩阵kI，（k＞0)，那么X′X+kI接近奇异的程度就会比X′X接近奇异的程度小得多。

弱化共线性对大小的影响。

**岭回归在不抛弃任何一个变量的情况下，缩小了回归系数，是的模型相对而言比较稳定，但是会使得模型的变量特别多，模型解释性差。**

### Lasso回归

Lasso回归有时也叫做线性回归的L1正则化，和Ridge回归的主要区别就是在正则化项，Ridge回归用的是L2正则化，而Lasso回归用的是L1正则化。Lasso回归的损失函数表达式如下：





最小化非零元素的个数，减小维度。

### Logistic Regression

**LR的优点在于实现简单**，并且计算量非常小，速度很快，存储资源低，

**缺点就是因为模型简单**，对于复杂的情况下会出现欠拟合，并且只能处理2分类问题

1. 线性回归公式：



其中表示第i个输入变量的权重或是回归系数，第一个输入变量默认为1。

上述公式表示为：，

其中和均表示n行1列的向量。

1. Logistic函数可以写成如下形式：



其中，为**对数几率**，**几率**反映了输入向量x划分为1类的可能性。

1. 如何求解模型中的权重

输出类别为1和0的概率为：





1. 极大似然估计

极大似然估计的思想为：对于所有的抽样样本，使它们联合概率达到最大的系数便是统计模型最优的系数。因此对于第i个输入数据，它被划分为的概率可由如下公式表示：

上述公式的巧妙之处在于，当时，；而当时, 。

在极大似然法中，我们假设样本之间都是独立同分布的，因此它们的联合概率就是它们各自概率的乘积。因此关于回归系数ω的极大似然函数可由下面公式表示：



两边同时取对数：



根据对数性质：



1. 求解回归系数

有两种方法。a.梯度上升法。b.牛顿法

1）梯度上升法

函数在某一点的梯度总是指向该函数增长最快的方向。因此沿着该函数的梯度方向探寻就能找到该函数的最大值。梯度上升法的迭代公式如下：



公式中，表示每次迭代的步数，表示梯度算子。

根据以上定义，将方程两边同时取微分：



故可简化如下表达：



1. 牛顿法：利用泰勒公式不断迭代，从而逐次逼近零点或极值点。

一阶展开求零点：在处的一阶展开如下：



### SVM 支持向量机

超平面：*wTx*+*b*=0

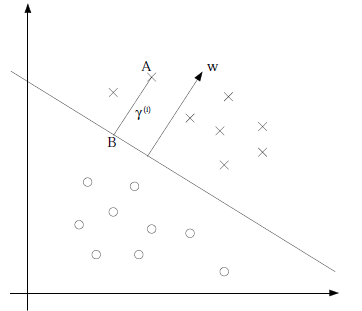
一个超平面，在二维空间中的例子就是一条直线。我们希望的是，通过这个超平面可以把两类数据分隔开来，比如，在超平面一边的数据点所对应的 *y* 全是 -1 ，而在另一边全是 1 。具体来说，我们令 *f*(*x*)=*wTx*+*b* ，显然，如果 *f*(*x*)=0 ，那么 *x* 是位于超平面上的点。我们不妨要求对于所有满足 *f*(*x*)<0 的点，其对应的 *y* 等于 -1 ，而 *f*(*x*)>0 则对应 *y*=1的数据点。

#### 函数间隔

定义函间隔：

前面乘上类别 *y* 之后可以保证这个 margin 的非负性（因为 *f*(*x*)<0 对应于 *y*=−1 的那些点）

#### 几何间隔



点到直线的距离定义成几何距离：



#### 最优间隔分类器

函数间隔和几何间隔相差一个的缩放因子。按照我们前面的分析，对一个数据点进行分类，当它的间隔越大的时候，分类的置信度越大。对于一个包含 *n* 个点的数据集，我们可以很自然地定义它的间隔为所有这 *n* 个点的间隔值中最小的那个。于是，为了使得分类的置信度高，我们希望所选择的超平面能够最大化这个 间隔值。 不过这里我们有两个间隔可以选，不过函数间隔明显是不太适合用来最大化的一个量，因为在超平面固定以后，我们可以等比例地缩放*w*的长度和*b*的值，这样可以使得*f*(*x*)=*wTx*+*b*的值任意大，亦即 函数间隔*γ*ˆ可以在超平面保持不变的情况下被取得任意大，而几何则没有这个问题，因为除上了 这个分母，所以缩放 *w* 和 *b* 的时候 *γ*˜ 的值是不会改变的，它只随着超平面的变动而变动，因此，这是更加合适的一个间隔 。这样一来，我们的最优间隔分类器的目标函数即定义为



当然，还需要满足一些条件，根据 间隔 的定义，我们有



其中 *γ*ˆ=*γ*˜ ，根据我们刚才的讨论，即使在超平面固定的情况下，*γ*ˆ 的值也可以随着 的变化而变化。由于我们的目标就是要确定超平面，因此可以把这个无关的变量固定下来，固定的方式有两种：一是固定  ，当我们找到最优的 *γ*˜ 时 *γ*ˆ 也就可以随之而固定；二是反过来固定 *γ*ˆ ，此时  也可以根据最优的 *γ*˜ 得到。处于方便推导和优化的目的，我们选择第二种，令 *γ*ˆ=1 ，则我们的目标函数化为：



这个问题等价于（为了方便求解，我在这里加上了平方，还有一个系数，显然这两个问题是等价的，因为我们关心的并不是最优情况下目标函数的具体数值）：

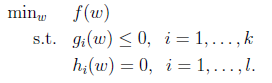


到这个形式以后，就可以很明显地看出来，它是一个凸优化问题，或者更具体地说，它是一个二次优化问题——目标函数是二次的，约束条件是线性的。

#### 拉格朗日乘子法

1）拉格朗日对偶

有不等式约束的极值问题求法，问题如下：



定义一般化的拉格朗日公式：

clip_image008[6]

可以把f(w)写为：，为什么呢？因为h(x)=0, g(x)<=0，现在L(a,b,x)的最大值，a\*g(x)是<=0，所以L(a,b,x)只有在a\*g(x) = 0时才能满足L(a,b,x)取最大值。

所以:



原规划问题变为：



2）目标函数



用拉格朗日乘子算法写成如下形式：



原问题是极小极大问题：



原始问题的对偶问题是极大极小问题：



将拉格朗日函数分别对w，b求偏导，并等于0。即求分别关于w，b的极小化





代入：



3） 整理目标函数



即：



添加负号：



#### 线性可分支持向量机学习算法

构造并求解约束最优化问题



求解最优解：

可以求得：



求得分离超平面：



分离决策函数：



#### 核函数

分类函数：



映射后的分类函数：



将核函数形式化定义，如果原始特征内积是[clip_image014[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034291172.png)，映射后为[clip_image016[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034309711.png)，那么定义核函数（Kernel）为

[clip_image018[8]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034302186.png)

**多项式核：**



更一般地，核函数[clip_image037[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034372921.png)对应的映射后特征维度为[clip_image039[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034378984.png)

由于计算的是内积，我们可以想到IR中的余弦相似度，如果x和z向量夹角越小，那么核函数值越大，反之，越小。因此，核函数值是[clip_image020[11]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034377838.png)和[clip_image041[4]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/20110318203438804.png)的相似度。

**高斯核函数：**

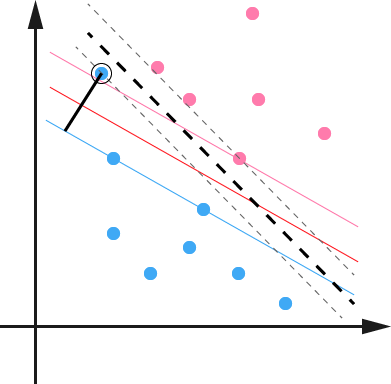


如果x和z很相近（[clip_image044[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034396245.png)），那么核函数值为1，如果x和z相差很大（[clip_image046[6]](http://images.cnblogs.com/cnblogs_com/jerrylead/201103/201103182034398720.png)），那么核函数值约等于0。由于这个函数类似于高斯分布，因此称为高斯核函数，也叫做径向基函数(Radial Basis Function 简称RBF)。它能够把原始特征映射到无穷维。

所以，分类函数可写成：



#### 离群点



用黑圈圈起来的那个蓝点是一个离群点 ，它偏离了自己原本所应该在的那个半空间，如果直接忽略掉它的话，原来的分隔超平面还是挺好的，但是由于这个离群点的出现，导致分隔超平面不得不被挤歪了，变成途中黑色虚线所示（这只是一个示意图，并没有严格计算精确坐标），同时间隔也相应变小了。当然，更严重的情况是，如果这个离群点再往右上移动一些距离的话，我们将无法构造出能将数据分开的超平面来。

为了处理这种情况，SVM 允许数据点在一定程度上偏离一下超平面。例如上图中，黑色实线所对应的距离，就是该离群点偏离的距离，如果把它移动回来，就刚好落在原来的超平面上，而不会使得超平面发生变形了。具体来说，原来的约束条件

*yi*(*wTxi*+*b*) ≥ 1 , *i*=1,…,*n*

现在变成

*yi*(*wTxi*+*b*) ≥ 1− *ξi* , *i*=1,…,*n*

其中 *ξi*≥0 称为松弛变量 (slack variable) ，对应数据点 *xi* 允许偏离的 functional margin 的量。当然，如果我们运行 *ξi* 任意大的话，那任意的超平面都是符合条件的了。所以，我们在原来的目标函数后面加上一项，使得这些 *ξi* 的总和也要最小：



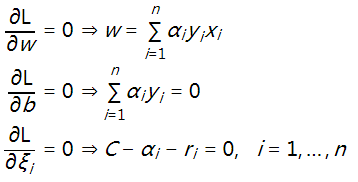
其中 *C* 是一个参数，用于控制目标函数中两项（“寻找 margin 最大的超平面”和“保证数据点偏差量最小”）之间的权重。注意，其中 *ξ* 是需要优化的变量（之一），而 *C* 是一个事先确定好的常量。完整地写出来是这个样子：



拉格朗日方程：



分析方法和前面一样，转换为另一个问题之后，我们先让 L 针对 *w*、*b* 和 *ξ* 最小化：



W代入L式



因为，，因此



和之前的结果对比一下，可以看到唯一的区别就是现在*α* 多了一个上限 *C* 。而 Kernel 化的非线性形式也是一样的，只要把 （*xi*,*xj*） 换成 *κ*(*xi*,*xj*) 即可。

AR模型是一种线性预测，即已知N个数据，可由模型推出第N点前面或后面的数据（设推出P点），所以其本质类似于插值。

MA模型(moving average model)滑动平均模型，模型参量法谱分析方法之一。

ARMA模型(auto regressive moving average model)自回归滑动平均模型，模型参量法高分辨率谱分析方法之一。这种方法是研究平稳随机过程有理谱的典型方法。它比AR模型法与MA模型法有较精确的谱估计及较优良的谱分辨率性能，但其参数估算比较繁琐。

GARCH模型称为广义ARCH模型，是ARCH模型的拓展， GARCH对误差的 方差进行了进一步的建模，特别适用于波动性的分析和 预测。

HK算法

HMM算法

常采用特征选择方法。常见的六种特征选择方法：

1）DF(Document Frequency) 文档频率

DF:统计特征词出现的文档数量，用来衡量某个特征词的重要性

2）MI(Mutual Information) 互信息法

互信息法用于衡量特征词与文档类别直接的信息量。

如果某个特征词的频率很低，那么互信息得分就会很大，因此互信息法倾向"低频"的特征词。

相对的词频很高的词，得分就会变低，如果这词携带了很高的信息量，互信息法就会变得低效。

3）(Information Gain) 信息增益法

通过某个特征词的缺失与存在的两种情况下，语料中前后信息的增加，衡量某个特征词的重要性。

4）CHI(Chi-square) 卡方检验法

利用了统计学中的"假设检验"的基本思想：首先假设特征词与类别直接是不相关的

如果利用CHI分布计算出的检验值偏离阈值越大，那么更有信心否定原假设，接受原假设的备则假设：特征词与类别有着很高的关联度。

5）WLLR(Weighted Log Likelihood Ration)加权对数似然

6）WFO（Weighted Frequency and Odds）加权频率和可能性

推荐算法 （Collaborative Filtering, SVD++）；

### 生成模型使用联合概率建模，判别模型直接使用条件概率建模

常见的判别模型有：

支持向量机

传统的神经网络

线性判别分析

线性回归

CRF 条件随机厂 无向图

产生式模型常见的主要有：

高斯

朴素贝叶斯

混合多项式

混合高斯模型

专家的混合物

隐马尔可夫模型

马尔可夫的随机场

### 准确率、召回率



**1、准确率（Accuracy）**

准确率(accuracy)计算公式为：   


**2. precision是精确性**

表示被分为正例的示例中实际为正例的比例

precision=TP/（TP+FP）

**2、召回率（recall）**

召回率是覆盖面的度量，度量有多个正例被分为正例，recall=TP/(TP+FN)=TP/P=sensitive，可以看到召回率与灵敏度是一样的。

**3、综合评价指标（F-Measure）**

P和R指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）。

F-Measure是Precision和Recall加权调和平均：



当参数α=1时，就是最常见的F1，也即 ：



可知F1综合了P和R的结果，当F1较高时则能说明试验方法比较有效

### XGboost

* 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）。
* 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。
* 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。
* 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）
* 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。
* 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。
* 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。
* 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。

### HMM

 HMM的三个问题 第一个问题被称为：概率计算问题。 解决办法：前向-后向算法(一种动态规划算法)。

第二个问题被称为：学习问题。 解决办法：如果状态序列已知，那用最大似然估计就好了，但HMM的状态序列未知，即含有隐变量，所以要使用Baum-welch算法(其实其本质就是EM算法)。

第三个问题被称为：预测问题/解码问题。 解决办法：Viterbi算法(一种动态规划算法)。

### 序列标准模型

现有的序列标注模型主要有 HMM（ 隐马尔科夫模型） ， MEMM（最大熵隐马尔科夫模型） 以及 CRF （条件随机场模型）

### PCA

PCA通过线性变换将原始数据变换为一组各维度线性无关的表示，可用于提取数据的主要特征分量，常用于高维数据的降维。

降维的标准为：样本点到这个超平面的距离足够近,或者说样本点在这个超平面上的投影能尽可能的分开。

### 常见的损失函数

#### 1. 0-1损失函数和绝对值损失函数

0-1损失是指，预测值和目标值不相等为1，否则为0：

L(Y,f(X))={1,Y≠f(X)0,Y=f(X)

**感知机**就是用的这种损失函数。但是由于相等这个条件太过严格，因此我们可以放宽条件，即满足 |Y−f(X)|<T|Y−f(X)|<T 时认为相等。

L(Y,f(X))={1,|Y−f(X)|≥T0,|Y=f(X)|<TL(Y,f(X))={1,|Y−f(X)|≥T0,|Y=f(X)|<T

绝对值损失函数为：

L(Y,f(X)=|Y−f(X)|

#### 2.log对数损失函数

**逻辑斯特回归**的损失函数就是对数损失函数，在逻辑斯特回归的推导中，它假设样本服从伯努利分布（0-1）分布，然后求得满足该分布的似然函数，接着用对数求极值。逻辑斯特回归并没有求对数似然函数的最大值，而是把极大化当做一个思想，进而推导它的风险函数为最小化的负的似然函数。从损失函数的角度上，它就成为了log损失函数。

交叉熵

#### 3.  平方损失函数

**最小二乘法**是线性回归的一种方法，它将回归的问题转化为了凸优化的问题。最小二乘法的基本原则是：**最优拟合曲线应该使得所有点到回归直线的距离和最小**。通常用欧几里得距离进行距离的度量。平方损失的损失函数为：

L(Y|f(X))=∑N(Y−f(X))2

#### 4. 指数损失函数

**AdaBoost**就是一指数损失函数为损失函数的。   
指数损失函数的标准形式：

L(Y|f(X))=exp[−yf(x)]

#### 5. Hinge损失函数

Hinge损失函数和SVM是息息相关的。在线性支持向量机中，最优化问题可以等价于

minw,b∑iN(1−yi(wxi+b))+λ||w2|

# 二、java篇

### 一、java线程的状态

java中，线程常有5中状态，**创建、就绪、运行、阻塞、死亡**

**1.创建**：生成线程对象，并没有调用该对象的start方法，这时线程处于创建状态

**2.就绪：**调用线程对象的start方法后，该线程进入了就绪状态，但是此线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行后，从等待或者睡眠中回来之后，也会处于就绪状态。

**3.运行：**线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行run函数当中的代码。

**4.阻塞：**线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend，wait等方法都可以导致线程阻塞。

**5.死亡：**如果一个线程的run方法执行结束或者调用stop方法后，该线程就会死亡。对于已经死亡的线程，无法再使用start方法令其进入就绪。

### 二、线程中阻塞和死锁的区别

**死锁**：当两个线程相互等待对方释放同步监视器的时候就会发生死锁， JVM 没有监测，也没有措施来处理死锁情况，多线程情况下应该避免死锁。死锁一旦发生，整个程序既不会发生异常，也不会给出任何提示，只是所有线程处于阻塞状态，无法继续，死锁容易发生，尤其是在系统中出现多个同步监视器的时候。

导致死锁的根源在于不适当地运用“synchronized”关键词来管理线程对特定对象的访问。

**阻塞**：线程正在运行的时候，被暂停，通常是为了等待某个时间的发生（比如说某项资源就绪）之后再继续运行。 sleep,suspend 等方法都可以导致线程阻塞。

### 三、参数传递

实参向形参传递有两种，按值传递和按引用传递。按值传递是针对基本变量，传递的是基本变量的副本。按引用传递是针对类变量，是传递实例化对象的地址。

### 四、null值

任何对象都可以赋值为 null，包括类的引用对象， null 可以被强制转换为任何类，转换后也为 null，因此无法调用对象的方法，但可以调用类的方法（静态方法）。

### 五、String、StringBuffer、StringBuilder的区别

1.类型区别

String：字符串常量

StringBuilder：字符串变量

StringBuffer：字符串变量

2.执行速度

StringBuilder > StringBuffer > String

原因： String 是字符串常量，用 String 改变字符串的值，不是在原来的字符串上操作，是创建新的字符串，而原来的对象值变为垃圾会被 GC 回收掉，所以执行效率低。StringBuilder 和 StringBuffer 是在原来的字符串改变。

3.线程安全

StringBuilder：线程非安全， 单线程下优先使用；

StringBuffer：线程安全， 多线程下使用；

4.总结  
1） 少量字符串处理用： String；  
2） 单线程在字符串缓冲区操作大量字符串： StringBuilder；  
3） 多线程在字符串缓冲区操作大量字符串： StringBuffer；

### 六、Double保留n位小数位的方法

1. String.format 转化  
String.format("%.2f", double) 返回 double 数四舍五入后的 String 类型；  
2. 使用 DecimalFormat 类  
DecimalFormat dFormat = new DecimalFormat("#.00"); 构建输出格式；  
dFormat.format(double) 返回double数四舍五入后的String类型

### 七、ArrayList与LinkedList的区别

1. ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。  
2. 对于随机访问 get 和 set， ArrayList 觉得优于 LinkedList，因为 LinkedList 要移动指针。  
3. 对于新增和删除操作 add 和 remove， LinkedList 比较占优势，因为 ArrayList 要移动数据。（ArrayList 默认构造的容量是 10）

### 八、HashMap的工作原理

HashMap 是基于 hashing 的原理，使用 put(key， value)存储对象到 HashMap 中，使用get（key）从 HashMap 中获取对象。当给 put()方法传递键和值时，首先对键调用 hasCode()方法，返回的 hasCode 用于找到 bucket 位置来存储 Entry 位置，作为 Map.Entry。

### 九、 内存泄漏

程序运行就会有分配内存和释放内存。但如果程序分配的对象或变量因为某种原因在不被使用的时候一直未被释放，始终占据内存，任务不断重复，最终会耗尽内存并异常终止，或无法继续运行。

### 十、垃圾回收器 GC

GC 是指 JVM 自动运行管理内存，释放回收无引用对象所占的内存。对于 GC 来说，当程序创建对象时，GC 就开始监控这个对象的地址、大小和使用情况。Java 中的内存分为两种：堆内存 Heap 和栈内存 Stack。1.堆是开发人员用的，把局部变量都放进去，存放由 new创建的对象和数组，生命周期和进程有关。通常 GC 采用有向图的方式记录和管理堆（heap）中所有的对象。通过这种方式确定哪些对象是“可达的”，哪些是 “不可达”。当 GC 确认一些对象是不可达时，就收回这些内存空间。程序员可以手动执行 System.gc()，通知 GC 运行，但 java 语言规范并不保证 GC 一定会执行。2.栈是留给 jvm 自己用的，用来存放类的信息，它和堆不同，运行期内 GC 不会释放空间，当超过变量的作用域后，java 会自动释放掉该变量所分配的内存空间。

a.如果程序声明了 static 的变量，就直接在栈中运行，进程销毁

了，不一定销毁 static 变量。b.在函数中，基本类型变量和对象的引用变量都在函数的栈内存中分配。

### 十一、进程和线程的区别

进程：是执行中一段程序，即一旦程序被载入到内存中并准备执行，它就是一个进程。进程是表示资源分配和调度运行的一个独立单位。

线程：是一个实体，单个进程中执行每个任务就是一个线程。线程是进程中执行运算的最小单位。

区别：

* + 1. 线程和进程是不同操作系统资源管理方式；
    2. 进程有独立的地址空间，一旦进程崩溃，在保护模式下不会对其他进程产生影响，线程只是一个进程中的不同执行路径；
    3. 线程有自己的堆栈和局部变量，但线程之间没有独立的地址空间，一个线程死掉等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，单进程之间切换耗资源效率差；
    4. 一个程序至少有一个进程，一个进程至少有一个线程；
    5. 进程之间互相独立，通信比较困难，线程之间共享一块内存，通信方便；
    6. 进程在执行过程中包含比较固定的入口、执行顺序和出口，而线程的这些过程被应用程序控制；

### 十三、重写与重载

1、就是在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。

多个同名函数同时存在，具有不同的参数个数/类型。重载Overloading是一个类中多态性的一种表现。

2、参数列表必须完全与被重写的方法相同，否则不能称其为重写而是重载。

返回的类型必须一直与被重写的方法的返回类型相同，否则不能称其为重写而是重载。

### 十二、Java 开启进程的方式

1.利用 cmd 方式

2.

### 十三、静态变量和实例变量的区别

类变量叫静态变量，变量前加 static；实例变量也叫对象变量，不加 static。

区别：类变量是所有对象，其中一个 new 对象调用改变其值，其他变量调用也是改变后的结果；而实例变量属于对象私有，某个 new 对象调用改变其值，不影响其他 new 变量调用。

### 十四、JVM 内存划分

#### 1.栈区

JVM 虚拟就留给自己用的内存区域。

1. 每个线程包含一个栈区，栈中只保存基础数据类型的对象和自定义对象的引用

(不是对象)，对象都存放在堆区中；

1. 每个栈中的数据(原始类型和对象引用)都是私有的，其他栈不能访问；
2. 栈分为 3 个部分：基本类型变量区、执行环境上下文、操作指令区(存放操作指令)。

#### 2、堆区

程序员自己管理的内存区域；

1. 存储的全部是对象，每个对象都包含一个与之对应的 class 的信息。(class 的目的是得到操作指令)；
2. 一个堆区(heap)被所有线程共享，堆中不存放基本类型和对象引用，只存放对象本身；

#### 3、方法区

在方法区中，存储了每个类的信息（包括类的名称、方法信息、字段信息）、静态变量、常量以及编译器编译后的代码等。

### 十五、设计模式

#### 1、单例模式

单例模式的作用就是保证在整个应用程序的生命周期中，任何一个时刻，单例类的实例只有一个；

#### 2、工厂模式

工厂模式专门负责实例化有大量公共接口的类，工厂模式可以动态的决定将哪一个类实例化，而不必事先知道每次要实例化哪一个类。

#### 3、适配器模式

适配器模式也成为变压器模式，它把一个类的接口转换成客户端所期望的另一种接口，从而使原本因接口不匹配而无法一起工作的两个类能够一起工作；

#### 4、观察者模式

观察者模式提供了避免组件之间紧密耦合的另一种方法，它将观察者和被观察者的对象分离开来。

### 十六、JVM 加载类文件的原理机制

Java 语言是一种具有动态性的解释语言，类（class）只有被加载到 JVM 后才能运行，

当运行指定程序时，JVM 会将编译生产的.class 文件按照需求和一定的规则加载到内存中，并组织成一个完整的 Java 应用程序。这个加载过程是由类加载器来完成的，具体来说是由

ClassLoader 和它的子类来实现的。类加载器本身也是一个类，其实质就是把类文件从硬盘读取到内存中。

类的加载方式分为隐式加载和显示加载。隐式加载指的是程序在使用 new 等方式创建对象时，会隐式的调用类的加载器把对应的类加载到 JVM 中，显式加载时直接通过调用

class.forName()方法来把它所需的类加载到JVM 中。

当程序启动时，只把需要的类加载到 JVM 中，其他类只有被使用到的时候才会被加载，这样可以加快加载速度，也可以节约程序运行时对内存的开销。

加载器类有三种：系统类加载器，扩展类加载器和自定义类加载器；

Bootstrap Loader -负责加载系统类（jre/lib/rt.jar 的类）

|

------Extension Loader -负责加载扩展类（jar/lib/ext \*.jar 的类）

|

-----Application Loader -负责加载应用类（classpath 指定的目录或 jar 中的类）

volatile 关键字的作用 答完之后追问 i++是原子性的么？内存模型

为什么要设置工作内存和主内存工作 内存类比为处理器的高速缓存

GC 的过程吧

GC 时会对程序有什么影响设计模式

java 中有什么类可以实现拦截器的作用？ 过滤器 和拦截器 和 controller 的区别？

知道哪些设计模式？答单例，装饰者，工厂，动态代理 （以为问动态代理， 就可以回答 Spring, jdk 代理和 cglib 区别 balabla）

java 的 I/O 类中有哪些用到了装饰者模式？会 JVM 调优吗？

# 三、数据结构

### 一、栈

n元素固定顺序入栈，对应的出栈次数

### 二、二叉树

#### 1、二叉树节点个数性质

1. 非空二叉树的第 n 层上至多有 2^(n-1)个元素。
2. 深度为 h 的二叉树至多有 2^h-1 个结点。
3. 在满二叉树中若其深度为h，则其所包含的结点数必为 2^h-1。

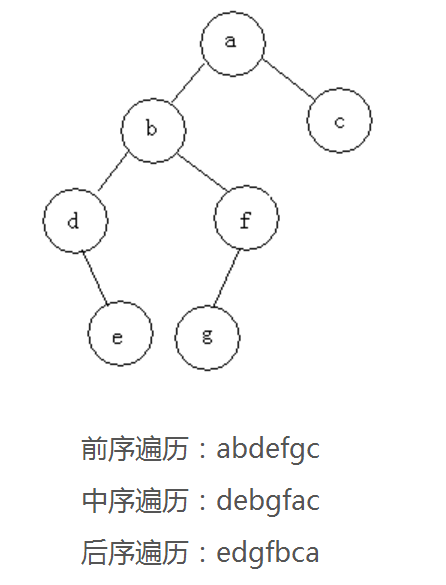
#### 2、二叉树遍历

前序遍历：根节点->左子树->右子树

中序遍历：左子树->根节点->右子树

后序遍历：左子树->右子树->根节点

二叉树遍历的前中后都是针对根节点而言，根节点在坐前面是前序，在中间是中序，在最后是后序，而三种遍历方式左子树都在右子树前面。



### 三、排序算法



#### 1、冒泡排序 平均复杂度 *n*2

**思想**：比较两个相邻元素的大小，如果前者大于后者则交换，不断重复把最大的值浮动动到最末端，进行 n-1 趟排序，每趟排序进行 n-i 次；

#### 2、快排排序 平均复杂度

*n* log2

**思想**：采用分治法，先比大小，再分区，把大于等于的放到右边，小于的放到左边，然后对分区递归快排；

#### 3、选择排序 平均复杂度 *n*2

**思想**：将已知的无序数据按照升序排列。首先比较 a[1]和 a[2]~a[n]一次比较，如果有比 a[1]小的就交换两个的值，一趟比较 a[1]肯定是最小的。然后比较 a[2]和 a[3]~a[n]，重复比较，直至所有的元素比较完；其实比较次数与冒泡一样，但是数据移动次数比冒泡少；

#### 4、桶排序

**思想**：就是把可能出现的数据情况穷尽，然后建立一个空桶，空桶的下标和数据有换算关系，空桶内所有的元素初始化为 0，然后依次统计所有的数据出现的次数更改空桶中的初始值，最后按照空桶顺序对数组元素大于 0 的元素进行输出，输出的个数是数组元素的值，输出值是通过下标关系换算的到的。所以桶排序是有限制条件的。

#### 5、直接插入排序 平均复杂度 *n*2

**思想**：就是对将带排序的记录，按其值的大小插入到一个已经升序的数组的合适位置。不用创建新的空数组，只需要把原来数组的第一个看成是已经升序的数组，其他数组元素看成

是带排序数据。这种思想是直接插入排序，还有其他插入排序，思路都是一样的，只是找合适的插入位置的方式不同，有二分插入排序，希尔插入排序。

# 四、Hadoop平台

## 1、Hadoop 和 Spark

### 1）Hadoop 和 Spark 的区别

1. 计算模式：hadoop 是基于磁盘计算，侧重于离线大批量计算，而 spark 基于内存计算，侧重于实时计算；
2. 操作算子：hadoop 提供两部分内容，hdfs 和 mr，hdfs 存储数据，mr 分布式计算框架，hadoop 中的 job 只要 map 和 reduce 操作，表达能力欠缺，而且在 mr 过程中重复的读写 hdfs 造成大量的 io 操作，多个 job 需要自己管理关系；spark core完全代替 mr，mr 提供的map 和 reduce 操作 spark 都有，并且还提供了很多其他方法，比如 filter，sort by，group by 等，比 mr 方便多，RDD 是个弹性分布式数据集，rdd 上的分区可以理解为与 hdfs 上的数据块一一对应的，在执行的时候每个分区上运行一个 task。
3. 执行过程：两者都是用 mr 模型进行并行计算，hadoop 的一个作业称为 job，job里面分为 map task 和 reduce task，每个 task 都在自己的进程中进行运行，当 task结束时，进程也会结束；spark 用户提交的任务是 application，一个 application 对应一个 sparkcontext，appilication 中存在多个 job，每个触发一次 action 操作就会产生一个job，这些 job 可以并行计算，也可以串行计算，每个 job 中有多个stage，

stage 是 shuffle 过程中 DAGSchaduler 通过 RDD 之间的依赖关系划分 job，每个

stage 里面有多个 task，做成 taskset 由 TaskSchaduler 分发各个 excutor 中执行，

executor 的生命周期和 application 一样，即使没有 job 运行也是存在的，所以 task

可以快速启动读取内存进行计算；

### 2）Hadoop 和 Spark 的 Shuffle 过程

Hadoop：map 端保存分片数据，通过网络收集到 reduce 段；

Spark：spark 的 shuffle 是在 DAGSchedular 划分 Stage 的时候产生，TaskSchedule 要分发 Stage 到 worker 的 executor；

## 2、Hadoop 的项目结构

1. **Common**，原名 Core，为 Hadoop 其他子项目提供支持的常用工具，主要包括文件系统，RPC 和串行库，为搭建云计算环境提供基本的服务；
2. HBase，分布式的列式数据库，具有强大的非结构化数据存储能力，采用 HDFS

作为元数据的存储，具有高可靠性，高性能，可伸缩，实时读写。

1. ZooKeeper，是高效和可靠的协同工作系统，提供分布式锁之类的服务（统一命名，状态同步服务，集群管理等），使用 Java 编程实现，容易编程接入；
2. Hive，基于 Hadoop 的数据仓库，提供了类似于关系数据库 SQL 语言的查询语言， Hive QL，Hive 自身可以将 Hive QL 语句转行成 MapReduce 任务。Hive 是逻辑上的数据仓库，实际操作都是在 HDFS 上的文件，Hive 表存的是表，是和HDFS的映射关系；

Hive 和关系型数据库的关系？没有关系，Hive 是逻辑上的数据仓库，不能和数据库一样进行实时的CURD（create，update，retrieve，delete）操作，是一次写入多次读取的操作，可以看成ETL 工具（Extract-Transform-Load）；

1. Pig，一种语言，数据流语言和运行环境，适用于 Hadoop 和 MapReduce 平台来查询大型半结构化数据。
2. Sqoop，用来在 Hadoop 和关系数据库之间交换数据，Sqoop 主要通过JDBC 和关系数据库进行交互。可以将关系型数据库中的数据导入非结构化的 HDFS，Hive和 HBase，也可以将 HDFS 的数据导入到关系型数据或者文本文件中，使用的是

MR 程序来执行任务，使用JDVC 进行交互，分别为 import 过程和 export 过程：

**import 原理**：通过指定分割符进行数据切分，将切分传入各个 map 中，在 map

任务中每行数据进行写入处理，没有 reduce；

**export 原理**：根据要操作的表明生成一个 java 类，并读取其元数据信息和分隔符对非结构化数据进行匹配，多个 map 作业同时执行写入关系型数据库；

## 3、MapReduce 执行过程

1. MapReduce 框架使用 InputFormat 模块做 Map 前处理，比如使用 InputSplit 对文本进行逻辑切分，只是记录了要处理的数据的位置和长度；
2. 用 RecordReader 对 InputSplit 中的信息来处理 InputSplit 的具体记录，加载数据并转化为合适 Map 任务读取的键值对，输入给 Map 任务；
3. Map 任务根据用户自定义的映射规则，输出一些列的<key,value>作为中间结果。
4. 对 Map 结果进行整理，提供 Reduce 能处理的<key,value-list>形式的中间结果，处理的<key,value-list>的过程称为 Shuffle。对 Map 的输出进行一定的分区、排序

（sort）、合并（Combine）、归并（Merge）等操作，得到<key,value-list>形式的中间结果，再交给对应的Reduce 进行处理。

1. Reduce 将以一系列<key,value-list>Shuffle 过程中间结果作为输入，执行用户定义的逻辑，输出到 OutputFormat 模块。
2. OutputFormat 模块对输出结果进行验证，验证输出目录是否存在和输出结果是否符合配置文件中的配置类型，如果满足，则输出 Reduce 结果到分布式文件系统。

## 4、TextInputFormat 的作用和自定义实现

#### 1）作用

InputFormat 会在 map 操作之前对数据进行两方面的预处理

1． getSplits 返回的是InputSplit 数组，对数据进行 split 分片，每片交给 map 操作一次 ； 2． getRecordReader 返回的是 RecordReader 对象，对每个 split 分片进行转换为 key-value 键值对格式传递给 map；

#### 2）自定义实现

常用的 InputFormat 是 TextInputFormat，使用的是 LineRecordReader 对每个分片进行键值对的转换，以行偏移量作为键，行内容作为值；自定义类继承 InputFormat 接口，重写createRecordReader 和 isSplitable 方法 ，在 createRecordReader 中可以自定义分隔符；

# 五、Spark 平台篇

## 1、基础概念

#### 1）平台搭建步骤

创建相关用户，获取文件权限，建立 SSH 免密码通信，下载各种软件包，解压缩后配置各种环境变量，配置Hadoop 配置文件（hdfs-site.xml，yarn 资源管理器相关内容），Spark配置文件（集群的 masterIP，各内存值）

#### 2）实时计算和流式计算

实时计算强调 response tim（e 实时），就是给定一个计算任务，它希望 immediate response，

服务器必须在用能忍耐的时间范围内得到结果；

流式计算，比实时计算要稍微迟钝一些，但是要比离线计算又实时的多，主要强调计算方法，流的本质是 one pass 和 sequence（序列），它希望计算是 compute on the fly，然后

incremental 地更新，cache（缓存）需要自己管理；比如，服务器端，有一个值，是记录小明订单数量。当小明每买一件东西后，服务端立即发出一个交易成功的事件，该值接收到这个事件后就立即加 1。如果用离线计算的方式来做，估计是在查询时，才慢腾腾的从低速存储中，把小明的所有订单取出来，统计数量。流式计算有点像数据库领域的触发器，又有些像事件总线、中间件之类的计算模式。

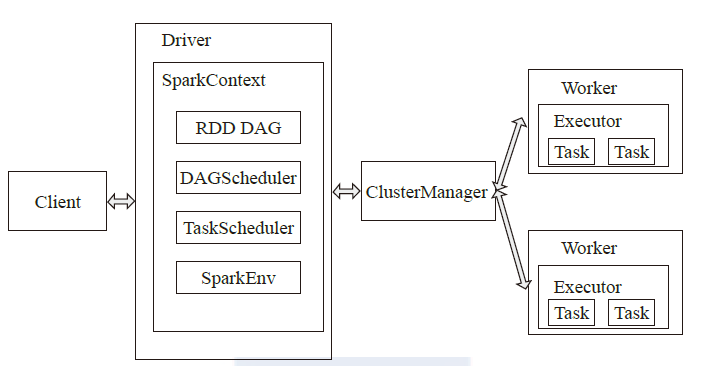
#### 3）Spark Core 的基本介绍

Spark Core 提供 Spark 最基础与最核心的功能，主要包括以下功能：

* + - 1. SparkContext：通常而言，Driver Application 的执行与输出都是通过 SparkContext来完成的。在正式提交Application 之前，首先需要初始化 SparkContext。SparkContext 隐藏了网络通信、分布式部署、消息通信、存储能力、计算能力、缓存、测量系统、文件服务、

Web 服务等内容，应用程序开发者只需要使用 SparkContext 提供的 API 完成功能开发。

SparkContext 内置的 DAGScheduler 负责创建 Job，将 DAG 中的 RDD 划分到不同的 Stage，提交 Stage 等功能。内置的 TaskScheduler 负责资源的申请，任务的提交及请求集群对任务的调度等工作。



(2)存储体系：Spark 优先考虑使用各节点的内存作为存储，当内存不足时才会考虑使用磁盘，这极大地减少了磁盘 IO，提升了任务执行的效率，使得 Spark 适用于实时计算、流式计算等场景。此外，Spark 还提供了以内存为中心的高容错的分布式文件系统 Tachyon供用户进行选择。Tachyon 能够为 Spark 提供可靠的内存级的文件共享服务。

(3)计算引擎：计算引擎由 SparkContext 中的 DAGScheduler、RDD 以及具体节点上的

Executor 负责执行的Map 和Reduce 任务组成。DAGScheduler 和RDD 虽然位于SparkContext内部，但是在任务正式提交与执行之前会将 Job 中的 RDD 组织成有向无环图（DAG），并对 Stage 进行划分，决定了任务执行阶段任务的数量、迭代计算、shuffle 等过程。

(4)部署模式：由于单节点不足以提供足够的存储和计算能力，所以作为大数据处理的Spark 在SparkContext 的TaskScheduler 组件中提供了对Standalone 部署模式的实现和Yarn、

Mesos 等分布式资源管理系统的支持。通过使用 Standalone、Yarn、Mesos 等部署模式为Task

分配计算资源，提高任务的并发执行效率。

#### 4）Spark 子框架

1. Spark SQL：首先使用 SQL 语句解析器（SqlParser）将 SQL 转换为语法树（Tree），并且使用规则执行器（RuleExecutor）将一系列规则（Rule）应用到语法树，最终生成物理执行计划并执行。其中，规则执行器包括语法分析器（Analyzer）和优化器（Optimizer）。
2. Spark Streaming：用于流式计算。Spark Streaming 支持 Kafka、Flume、Twitter、MQTT、

ZeroMQ、Kinesis 和简单的 TCP 套接字等多种数据输入源。输入流接收器（Receiver）负责接入数据，是接入数据流的接口规范。Dstream 是 Spark Streaming 中所有数据流的抽象，

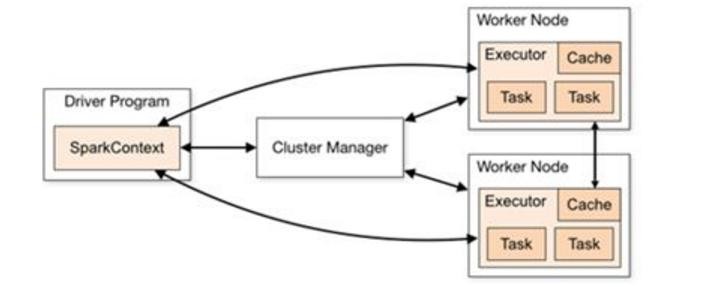
Dstream 可以被组织为Dstream Graph。Dstream 本质上由一系列连续的RDD 组成。

1. GraphX：Spark 提供的分布式图计算框架。GraphX 主要遵循整体同步并行（bulk Synchronous parallel，BSP）计算模式下的Pregel 模型实现。GraphX 提供了对图的抽象Graph，

Graph 由顶点（Vertex），边（Edge）及继承了 Edge 的 EdgeTriplet 三种结构组成。GraphX目前已经封装了最短路径，网页排名，连接组件，三角关系统计等算法的实现，用户可以选择使用。

1. MLlib：Spark 提供的机器学习框架。机器学习是一门设计概率论、统计学、逼近论、凸分析、算法复杂度理论等多领域的交叉学科。MLlib 目前已经提供了基础统计、分析、回归、决策树、随机森林、朴素贝叶斯、保序回归、协同过滤、聚类、维数缩减、特征提取与转型、频繁模式挖掘、预言模型标记语言、管道等多种数理统计、概率论、数据挖掘方面的数学算法。

#### 5）Spark 管理模式



首先有集群资源管理服务(Cluster Manager)和运行作业任务的结点(Worker Node)，然后就是每个应用的任务控制结点 Driver 和每个机器节点上有具体任务的执行进程(Executor)；

Executor 有二个优点：一个是多线程来执行具体的任务，而不是像 MR 那样采用进程模型，减少了任务的启动开稍。二个是 Executor 上会有一个 BlockManager 存储模块，类似于 KV系统(内存和磁盘共同作为存储设备)，当需要迭代多轮时，可以将中间过程的数据先放到这个存储系统上，下次需要时直接读该存储上数据，而不需要读写到 hdfs 等相关的文件系统里，或者在交互式查询场景下，事先将表 Cache 到该存储系统上，提高读写 IO 性能。另外

Spark 在做Shuffle 时，在Groupby、Join 等场景下去掉了不必要的Sort操作，相比于MapReduce

只有 Map 和 Reduce 二种模式，Spark 还提供了更加丰富全面的运算操作如 filter、groupby、

join 等。

#### 6）Spark 执行组件

1. master：管理集群和节点，不参与计算。
2. worker：计算节点，进程本身不参与计算，和master 汇报。
3. Driver：运行程序的main 方法，创建 spark context 对象。
4. spark context：控制整个 application 的生命周期，包括 dagsheduler 和 task scheduler

等组件。

1. client：用户提交程序的入口。

#### 7）Spark 工作机制

用户在 client 端提交作业后，会由Driver 运行 main 方法并创建 Spark context 上下文。执行 rdd 算子，形成 DAG 图输入 DAGScheduler，按照 rdd 之间的依赖关系划分 stage输入 task scheduler。

task scheduler 会将 stage 划分为 task set 分发到各个节点的 executor 中执行。

## 2、RDD

#### 1）RDD 综述

RDD 是 spark 的基本计算单元，是一个抽象类，可以通过一系列的算子进行操作，包括

Transformation 和 Action 两种算子操作。这个抽象类的内部实现包括五部分：

1. getPartitions 方法：分区列表（数据块列表）；
2. compute 方法：计算每个分片的函数；
3. getDependencies 方法：对父 RDD 的依赖列表

上面这三个用于描述 RDD 的 Lineage（血统关系，a）RDD 间的数据流图，各分区

RDD 从哪来；b）基于数据流上的操作算子流图，数据转换过程中经过的算子；）

1. partitioner：Key-Value（键-值）RDD 的分区器
2. getPreferredLoaction：每个数据分片的预定义地址列表（如 HDFS 上的数据块的地址）

#### 2）RDD 算子——map 和 flagMap

map 函数会对数据的每一条输入进行指定的操作，然后为每一条输入返回一个对象；

flagMap 函数则是两两个操作的集合——“先映射后扁平化”

1. 同 map 函数一样，对数据的每一条输入进行指定操作，然后每一条输入返回一个对象；
2. 最后将所有对象合并为一个对象；【扁平化处理】

#### 3）RDD 算子——分区算子

1） rdd.toDebugsString：查看 rdd 的 Lineage 关系；

2） rdd.dependencies：查看其父类RDD 依赖关系； 3） sc.defaultParallelism：获取所有 rdd 的默认分区数； 4） rdd.partitions.size：获取该rdd 的分区数；

#### 4）RDD 的持久化

persist 和 cache 方法将RDD 缓存内存或磁盘、Tachyon 文件系统中。

cache 是 lazy 特性，也是 persist 方法使用 MEMORY\_ONLY 储存级别的快捷方式；

如果要看 cache 或 persist 的效果，必须对 rdd 再进行 Action 操作，例如 rdd.count；

unpersist 方法释放缓存，是 eager 特性；

持久化的Web Interface 界面监控在（http://driverhost:4040）

#### Java 序列化与反序列化：

序列化：是指把 java 对象转换成字节序列的过程；反序列化：是把字节序列恢复为 java 对象的过程；**储存级别：**

|  |  |
| --- | --- |
| 存储级别（Storage Level） | 含义 |
| MEMORY\_ONLY | 将 RDD 以反序列化的（deserialized）的 java 对象存储到  JVM。如果 RDD 不能别内存装下，一些分区不会被缓存，并且在需要的时候被重新计算。这是默认级别； |
| MEMORY\_AND\_DISK | 将 RDD 以反序列化 java 对象存储到 JVM。如果 RDD 不  能别内存装下，超出的部分将被保存到硬盘上，并且在需要的时候被读取。这是默认级别； |
| …… | …… |

程序示例：

**import** org.apache.spark.storage.StorageLevel.\_ data.persist(*MEMORY\_AND\_DISK*)

#### RDD 数据原理

Spark 通过驱动器SparkContext 读取本地文本数据，将读入的数据集转化为 RDD 类型，而 RDD 的每行数据的类型都是一个字符串：class java.lang.String，因此可以元素之间的分割符分割字符串后，对其中数据做各种操作，可以获取里面任何元素。

**val** data01 = sc.textFile("D:\\DataSet\\mobile.txt") **val** data91 = data01.map(x => x.split(",")(4))

用 first()或 take()函数获取data01 中的记录，得到的是这行字符串的地址。

println(data01.first())输出：

[Ljava.lang.String;@5a545b0f 用 split()函数转化后，对每行字符串中的元素操作后这行字符创将根据操作变为元组或

者向量形式。

**val** data84 = sc.textFile("D:DataSet\\test.txt") **val** data85 = data84.map(\_.split(",")).map(x =>

(x(0).toDouble,x(1).toDouble))

**val** data86 = data84.map(s => Vectors.*dense*(s.split(",").map(\_.toDouble)))

println("data85 的整体类型" + data85.getClass)

println("data85 的元素类型" + data85.first().getClass)

println("data86 的整体类型" + data86.getClass)

println("data86 的元素类型" + data86.first().getClass)输出：

data85 的整体类型 class org.apache.spark.rdd.MapPartitionsRDD

data85 的元素类型 class scala.Tuple2$mcDD$sp

data86 的整体类型 class org.apache.spark.rdd.MapPartitionsRDD

data86 的元素类型 class org.apache.spark.mllib.linalg.DenseVector

数据字段模型

**val** fields =

*List*("NodeID","CI","IMEI","APP","Time","UplinkBytes","DownlinkBytes") **val** bcfields = sc.broadcast(fields)

用 bcfields.value.indexOf("IMEI")与数据进行对应，相当于带字段名的文本数据；这样就可以利用字段名对原数据字段进行组合操作，行程新的字段，如：

x => (x(bcfields.value.indexOf("IMEI")) + ":" + x(bcfields.value.indexOf("Time"))

## 3、DataFrame

#### DataFrame 综述

DataFrame 是 Spark1.3 引入，用来命名 SchemaRDD 类型，是一个以命名列方式组织的分布式数据集。DataFrame 数据可以从 Hive 表、Parquet 文件、JSON 文件、现有的RDD 转换等方式得到。

RDD 到DataFrame 的两种方式，1.用反射模式；2.编程模型；

#### DataFrame 编程模型具有的功能特点有：

1. 从 KB 到PB 级的数据量支持；
2. 多种数据格式和多种存储系统支持；
3. 通过 Spark SQL 的 Catalyst 优化器进行先进的优化，生成代码；
4. 为 Scala、Java、Scala 和 R（SparkR）提供 API。

# 六、Linux 篇

## 1、常用命令

1. 查看当前进程：ps；
2. 创建文件：touch fileName(创建空文件，无法编辑) vim/vi fileName 创建并编辑文件；
3. 修改文件权限：chmod 751 fileName 等价于 chmod u=rwx,g=rw,o=x filename；
4. 查看文件前几行：head –n filename （n 可以省略）
5. 查看文件尾几行：tail -n filename
6. Linux 下的通配符：？ 单个字符 \* 代替任意多个字符 [charset] 集合中的任意单个字符；
7. 对文件的内容进行统计：wc –clw filename -c 统计字节数 -l 统计行数 -w

统计字数；

1. 强大的文本搜索工具 grep [string] filename 或 grep [^string] filename；
2. 使命令在后台运行：&
3. 显示所有进程： ps -ef；
4. 终止进程命令：kill -9 pid；
5. 查看系统支持的所有信号： kill -l；
6. 搜索文件：find -name “string\*”；
7. 查看磁盘使用情况：df -hl;
8. 查看目录：du
9. 强大的文本分析工具：awk '{pattern + action}' {filenames} 其中 pattern 表示

AWK 在数据中查找的内容，而 action 是在找到匹配内容时所执行的一系列命令。花括号（{}）不需要在程序中始终出现，但它们用于根据特定的模式对一系列指令进行分组。 pattern 就是要表示的正则表达式，用斜杠括起来。

## 2、Stdout 和 stderr

stdout（标准输出），输出方式是行缓冲。输出的字符会先存放在缓冲区，等按下回车键时才进行实际的 I/O 操作。

stderr（标准出错），是不带缓冲的，这使得出错信息可以直接尽快地显示出来。

## 3、文件查找关键词

**locate** 并不真正对硬盘上的文件系统进行查找，而是对文件名数据库进行检索，而且可以使用通配符？和\*

**find** 命令从指定的起始目录开始，递归地搜索其各个子目录，查找满足寻找条件的文件并对之采取相关的操作

**whereis** 命令只能用于程序名的搜索，而且只搜索二进制文件（参数-b）、man 说明文件

（参数-m）和源代码文件（参数-s）

**type** 命令用来显示指定命令的类型，判断给出的指令是内部指令还是外部指令

# 七、数据库篇

## 1、基础语法

#### 大小写问题

1.关键词不区分大小写。2.标识符区分大小写（自定义的变量名），标识符如果不加双引号，默认是按照大写执行；标识符加上双引号，则按照定义时候的原始大小写执行。

#### 创建表基础语法

CREATE TABLE tableName (

attributeName Type(size) constraintCondition, attributeName Type(size) constraintCondition

)

例子：

CREATE TABLE employees( emp\_no int(11) PRIMARY KEY, birth\_day date NOT NULL,

first\_name varchar(14) NOT NULL, last\_name varchar(16) NOT NULL, gender char(1) NOT NULL,

hire\_date date NOT NULL depart\_no varchar(14) UNQUIE, FOREIGN KEY(depart\_no)

)

#### 查询表基础法

SELECT attributeName FROM tableName WHERE conditionExpression

GROUP BY attributeName HAVING conditionExpression ORDER BY attributeName ASC|DESC

例子：

SELECT \* FROM employees

ORDER BY hire\_date DESC Limit 0,1

#### 说明：

1. LIMIT m,n : 表示从第 m+1 条开始，取 n 条数据；

LIMIT n ： 表示从第 0 条开始，取 n 条数据，是 limit(0,n)的缩写。

#### between

x between a and b 等价于 a≤x≤b ;

1、select \* from emp

where sal>=1500 and sal=<3000

等价于

select \* from emp

where sal between 1500 and 3000；

2、select \* from emp

where sal<1500 and sal>3000

等价于

elect \* from emp

where sal not between 1500 and 3000；

#### in【属于若干个孤立值】

1、select \* from emp

where sal in (1500,3000)

等价于

select \* from emp

where sal = 1500 or sal = 3000

2、select \* from emp

where sal not in (1500,3000)

等价于

select \* from emp

where sal!= 1500 and sal!=3000

等价于

select \* from emp

where sal<>1500 and sal<>3000

注意:

数据库中的不等于有两种表达式 != 和 <>；

#### distinct

select distinct comm from emp;

----distinct 输出字段值不允许重复；对 null 同样有效

distinct comm 过滤掉 comm 中相同的值；

select distinct comm,deptno from emp;

----distinct 对 comm 和 deptno 的组合同时过滤；

#### 内连接和外连接区别

内连接和外连接都是以某个共同字段相等为条件进行表关联，但是内连接是按照条件字段取交集，不会有含有 NULL 的记录出现。外连接是取并集，左外连接是以左表为主表，其余表为附表去匹配左表，左表中的所有记录都会保留，匹配不到的会有 NULL 值。右外连接是以右表为主表进行信息匹配。

#### 乐观锁和悲观锁

数据库管理系统（DBMS）中的并发控制的任务是确保在多个事务同时存取数据库中同一数据时不破坏事务的隔离性和统一性以及数据库的统一性。

乐观并发控制(乐观锁)和悲观并发控制（悲观锁）是并发控制主要采用的技术手段。

#### 乐观锁

乐观锁认为别人在每次拿数据的时候不会修改，数据访问过程中不会上锁，但在更新的时候回判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。

#### 悲观锁

悲观锁认为别人每次去拿数据都会修改，所以数据对外界修改持保守态度，整个数据处理过程中，将数据处于锁定状态，悲观锁的实现往往依靠数据库的锁机制。

#### 总结

乐观锁使用于多读的应用类型，这样可以提高吞吐量，数据库中提供的类似于

write\_condition 机制其实都是提供的乐观锁。相反，如果经常发生冲突，上层应用不断进行

retry，这样反而降低性能，适合使用悲观锁。

#### union 和 union all 区别

union 在进行表求并集合后会去掉重复的元素，所以会对产生的结果集进行排序运算，删除重复的记录再返回结果；

union all 只是简单将两个结果集合并后返回结果，如果有重复元素不会去重；

数据查询语言DQL

## 2、数据定义语言 DDL

* 1. **数据操作语言 DML**

数据操纵语言DML 主要有三种形式：插入 INSERT，更新 UPDATE，删除 DELETE

#### 删除 DELETE

drop ：删除表的结构和内容，表将不复存在；

delete：删除整个表的数据或一条或多条数据，可以修复，表空间还在；

truncate ：删除表的数据连通其占用的表空间一起删除，无法恢复，一次性删除表的数据；

#### 区别：

1． truncate 和 delete 只删除数据不删除表的结构，drop 语句删除表的结构被依赖的约束，触发器，索引依赖于该表的存储过程/函数将保留,但是变为 invalid 状态。

2． delete 语句是 dml，这个操作会放到 rollback segement 中，事务提交之后才生效;如果有相应的 trigger，执行的时候将被触发。

3． truncate,drop 是 ddl，操作立即生效，原数据不放到 rollback segment 中，不能回滚。操作不触发 trigger。

4． 速度，一般来说: drop> truncate > delete；

5． delete 是DML 语句,不会自动提交。drop/truncate 都是DDL 语句,执行后会自动提交。

数据控制语言DCL

## 3、索引

索引是一种数据结构， 是存储表中一个特定列值的数据结构，是对数据库表中一列或多列的值进行排序的一种结构。

索引就是一种特殊的查询表，数据库的搜索引擎可以利用它加速对数据的检索。它很类似与现实生活中书的目录，不需要查询整本书内容就可以找到想要的数据。

索引两种存储结构：BTREE 索引和HASH 索引，不同的存储引擎支持不同的索引存储结构，InnoDB 和 MyISAM 存储引擎支持BTREE 索引，MEMORY存储引擎支持BTREE 索引和 HASH 索引。

每个索引中保存了特定列中的所有值，并且每个值又指向了与它相关记录的地址，即行地址。相当于存储了一个键值对，key=值，value=行索引。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值又指向与它相关的记录。这种索引的数据结构是经过排序的，因而可以对其执行二分查找。

索引的创建用 INDEX 和 KEY 指定；

#### BTREE 索引

它就是一种平衡多路查找树

BTREE 索引是有序的，

差异较小，且是一个范围，btree效率高

#### HASH 索引

哈希索引只能做等于查找，但是无论多大的Hash表，查找复杂度都是O(1)。

如果值的差异性大，并且以等值查找（=、 <、>、in）为主，Hash索引是更高效的选择，它有O(1)的查找复杂度。

HASH 索引是无序的，并且适合查询键值对-也就是说查询相等的查询

* + 1. **索引优缺点**

第一，提高检索速度。

第二，耗费时间，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第三，耗物理空间，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还

要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第四，降低维护速度，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

## 4、数据库优化思想

1. 选择最适合的字段属性，字段宽度尽可能小，尽量把字段设置为 NOT NULL，这样不用去比较 NULL 值，某些文本类字段可以定义为 ENUM 类型，因为枚举类型在 SQL 中被当成数值型处理，数值型数据被处理的速度比文本类型快；
2. 使用连接（JOIN）来代替子查询(Sub-Queries)；
3. 使用联合(UNION)来代替手动创建的临时表；
4. 使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句

话说，就是可以保持数据库中数据的一致性和完整性。事物以 BEGIN 关键字开始，COMMIT 关键字结束。在这之间的一条SQL 操作失败，那么，ROLLBACK命令就可以把数据库恢复到 BEGIN 开始之前的状态。

1. 锁定表；
2. 使用外键；
3. 使用索引；
4. 优化查询语句；

## 5、触发器的作用？

触发器是一种特殊的存储过程，主要是通过事件来触发而被执行的。它可以强化约束，来维护数据的完整性和一致性，可以跟踪数据库内的操作从而不允许未经许可的更新和变化。

可以联级运算。如，某表上的触发器上包含对另一个表的数据操作，而该操作又会导致该表触发器被触发。

## 6、存储过程？有哪些优缺点？

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。

（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。

* 存储过程是一个预编译的代码块，执行效率比较高；
* 一个存储过程替代大量T\_SQL 语句 ，可以降低网络通信量，提高通信速率；
* 可以一定程度上确保数据安全；

## 6、事务

事务（Transaction）是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。事务是数据库维护数据一致性的单位，在每个事务结束时，都能保持数据一致性。

事务具有四个特征：原子性（ Atomicity ）、一致性（ Consistency ）、隔离性（ Isolation ）和持续性（ Durability ）。这四个特性简称为 ACID 特性。

1 、原子性    
事务是数据库的逻辑工作单位，事务中包含的各操作要么都做，要么都不做    
2 、一致性    
事 务执行的结果必须是使数据库从一个一致性状态变到另一个一致性状态。因此当数据库只包含成功事务提交的结果时，就说数据库处于一致性状态。如果数据库系统 运行中发生故障，有些事务尚未完成就被迫中断，这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这时数据库就处于一种不正确的状态，或者说是 不一致的状态。    
3 、隔离性    
一个事务的执行不能其它事务干扰。即一个事务内部的操作及使用的数据对其它并发事务是隔离的，并发执行的各个事务之间不能互相干扰。    
4 、持续性    
也称永久性，指一个事务一旦提交，它对数据库中的数据的改变就应该是永久性的。接下来的其它操作或故障不应该对其执行结果有任何影响。

## 7、视图

视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，视图通常是有一个表或者多个表的行或列的子集。

对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。

只暴露部分字段给访问者，所以就建一个虚表，就是视图。

查询的数据来源于不同的表，而查询者希望以统一的方式查询，这样也可以建立一个视图，把多个表查询结果联合起来，查询者只需要直接从视图中获取数据，不必考虑数据来源于不同表所带来的差异

优点： 1）对数据库的访问，因为视图可以有选择性的选取数据库里的一部分。 2 ）用户通过简单的查询可以从复杂查询中得到结果。 3 ）维护数据的独立性，试图可从多个表检索数据。 4 ）对于相同的数据可产生不同的视图。 缺点： 性能：查询视图时，必须把视图的查询转化成对基本表的查询，如果这个视图是由一个复杂的多表查询所定义，那么，那么就无法更改数据

## 8、三个范式。

#### 第一范式（1NF）

列不可再分

数据库表中的字段都是单一属性的，不可再分。这个单一属性由基本类型构成，包括整型、实数、字符型、逻辑型、日期型等。

#### 第二范式（2NF）

行可以唯一区分，主键约束

数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖（部分函数依赖指的是存在组合关键字中的某些字段决定非关键字段的情况），也即所有非关键字段都完全依赖于任意一组候选关键字。

#### 第三范式（3NF）

在第二范式的基础上，数据表中如果不存在非关键字段对任一候选关键字段的传递函数依赖则符合第三范式。所谓传递函数依赖，指的是如 果存在"A → B → C"的决定关系，则 C 传递函数依赖于A。因此，满足第三范式的数据库表应该不存在如下依赖关系： 关键字段 → 非关键字段 x → 非关键字段 y

# 八、综合篇

## 用户画像

用户画像就是通过用户人口学特征，网络浏览内容，网络社交活动和消费记录等信息抽象出一个标签化的用户模型

#### 作用

大体分为如下几个方面：

1. 精准营销：精准直邮，短信，app 消息推送，个性化广告等；
2. 用户研究：知道产品优化，甚至做到产品功能的私人订制；
3. 个性服务：个性化推送，个性化搜索；
4. 业务决策：统计排名，地域分析，行业趋势，竞品分析；

#### 内容

对互联网企业，用户画像主要包含人口属性和行为特征；

**人口属性**：用户的年龄，性别，所在省份和城市，教育程度，婚姻情况，生育情况，工作所在的行业和职业等。

**行为特征**：活跃度和忠诚度等；

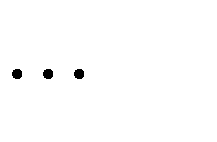
#### 企业的业务不同，提起的用户画像各有侧重：

* + 以内容为主的媒体或阅读网站，或搜索引擎或通用导航网站：提取用户浏览内容特诊，比如体育，娱乐，美食，理财，旅游，房产，汽车等；
  + 以社交网站：提取用户的社交网络，从中发现关系紧密的用户群和社群中起领袖作用的明星节点；
  + 电商购物网站：提取用户的网购兴趣和消费能力等，网购兴趣主要是指购买偏好，如服装类，箱包类，母婴类，饮食类等；

另外还可以考虑环境属性，如当前时间，访问地点 LBS 特征，当地天气，节假日情况等；

#### 应用示例——个性化推荐

很多电商网站的某个页面的个性化推荐，考虑到特征的可解释性，易扩展性和模型的计算性能，许多线上的电商推荐系统采用 LR 模型训练，很多推荐场景都会基于商品的协同过滤，基于商品的协同过滤的核心是一个商品相关性矩阵*W* ，假设有 *n* 个商品，那么*W* 就是一个*n*  *n* 的矩阵，*wij* 代表商品*li* 和*lj* 之间的相关系数。根据用户访问和购买商品的行为特征，

可以把用户表示成一个 *n* 维特征向量*U* [*u*1,*u*2 , *un* ] ， 于是用户对商品的感兴趣程度

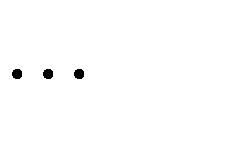
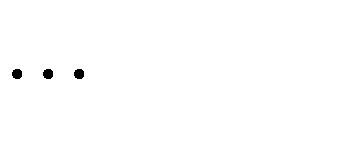
*V*  *U* *W* [*v*1,*v*2 ,



*vn* ]

， 这 里 即 是 用 户 对 商 品 *l*1

的 感 兴 趣 程 度 ，

*v*1  *i*1  *w*11  *i*2  *w*12  *in*  *w*1*n* ，如果把相关系数 *w*11, *w*12 , , *w*1*n* 看成要求的变量，那么就可以

用 LR 模型，把训练集用户的行为向量*U* ，进行求解，这样初步的 LR 模型就训练出来了，效果类似于商品的协同过滤。

## 区块链

狭义来讲，区块链是一种按照时间顺序将数据区块以顺序相连的方式组合成的一种链式

数据结构， 并以密码学方式保证的不可篡改和不可伪造的分布式账本。广义来讲，区块链技术是利用块链式数据结构来验证与存储数据、利用分布式节点共识算法来生成和更新数据、利用密码学的方式保证数据传输和访问的安全、利用由自动化脚本代码组成的智能合约来编

程和操作数据的一种全新的分布式基础架构与计算范式。

## 腾讯消息推送

信鸽（腾讯移动推送）推送的特点、关键字，海量，实时，精准推送。

精准推送的使用提示消息的打开率为衡量标准**海量**：→终端长连接→推送量

**实时**：1KW/s

**精准**：画像/机器学习人群分类

有推送诉求的行业，资讯类应用，游戏，王者荣耀。信鸽服务于数万 APP，日推送消息数 60 亿。

单机性能优化：硬件、操作系统（linux 优化）、协议栈、架构。

硬件优化，CPU 操作指令的差异，Intel CPU 指令优于一般常用的CRC32。倒排索引应用于实时推送。

#### 推送数据分析

系统的实现实时和离线。

离线主要是用来进行人群的挖掘，分类目标人群。应用的运营的目标：提升用户活跃度、潜在流失用户挽回，提升应用收入等。不同目标人群挖掘方法不同，流程都是数据挖掘的通用流程。

***潜在流失用户人群挖掘例子***

原数据保存在 HDFS 或 TDW（腾讯腾讯分布式数据仓库（Tencent distributed Data

Warehouse, 简称 TDW）基于开源软件 Hadoop 和 Hive 进行构建，并且根据公司数据量大、计算复杂等特定情况进行了大量优化和改造，目前单集群最大规模达到 5600 台，每日作业

数达到 100 多万，已经成为公司最大的离线数据处理平台，在往实时化改进。）中，加工数据构造特征，首次注册的时间、每日启动的次数、每日活动状态和最近登陆时间。用到的算法 C4.5 决策树模型（判定规则识别进行预测）、贝叶斯（通过特征类别的条件概率分布进行预测），达到的**召回率**为 0.76 和 0.89，满足场景需求。

效果评估，通常采用CTR（点击通过率 click-through-rate）评估送推送效果，但不能达到运行目标。改进，把目标页面的浏览 PV（浏览量）和 UV（访客）转化到系统配置文件，当进入推送，把配置文件下发到统计系统中。这些**数据的用途**，第一部分是根据推送文件指

定的方法进行数据的抽取、过滤、转换、加载并更新相关的倒排索引数据，当真正需要指标输出的时候，我们会有一个管理的节点，进行计算规则的下发，比如说点击的用户里面有多少是男的，检索以后进行一次 bitmap 的&操作，就能统计出有多少用户了。数据会实时刷新到Redis 存储中，应用开发者在前台查询结果的时候，我们就会把这个结果反馈给用户。