问题1，5亿个int找它们的中位数。

这个例子比上面那个更明显。首先我们将int划分为2^16个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

**Mapreduce**分为哪几个阶段，执行流程，多源输入使用什么类？

**Hive** 分组操作有哪几种？ group by distribute by cluster by区别

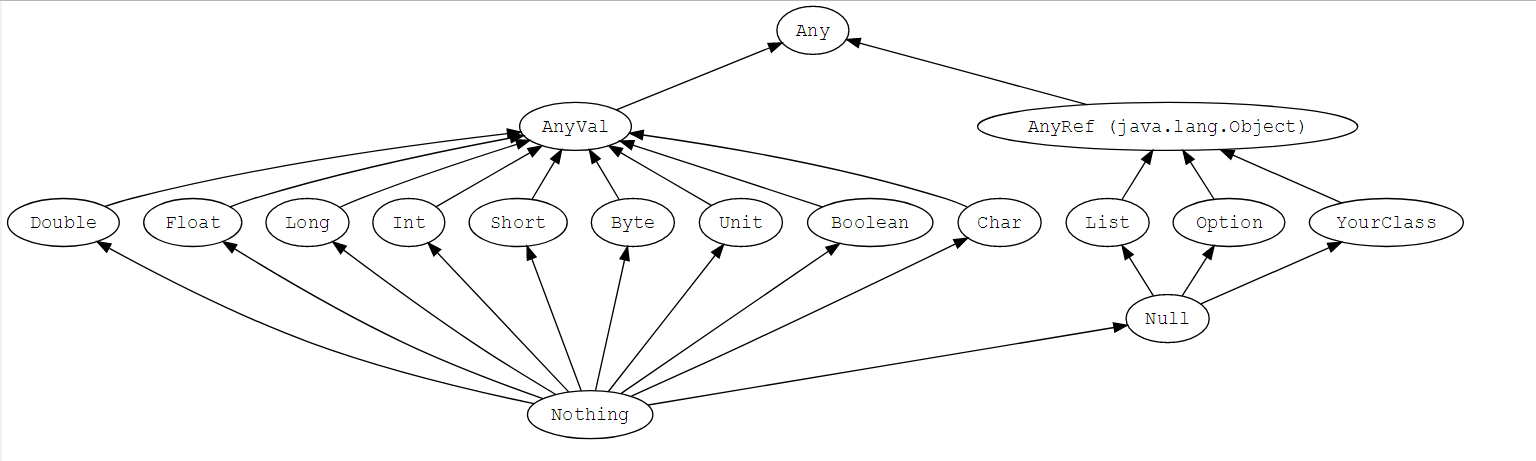
Join的实现原理

如何实现类似mysql的in操作

分区、动态分区

**Flume**版本，spool dir可以递归监控子目录吗？

需要递归监控子目录，应该怎么办

**Scala :**

Nothing也是一个trait（特质），是所有类型Any（包括值类型和引用类型）的子类型，它不在有子类型，它也没有实例

Null是一个trait（特质），是所以引用类型AnyRef的一个子类型，null是Null唯一的实例。

**Option类型的定义和使用场景？**

在Java中，null是一个关键字，不是一个对象，当开发者希望返回一个空对象时，却返回了一个关键字，为了解决这个问题，Scala建议开发者返回值是空值时，使用Option类型，在Scala中null是Null的唯一对象，会引起异常，Option则可以避免。Option有两个子类型，Some和None（空值）

**yield如何工作？**yield用于循环迭代中生成新值

**Option ，Try 和 Either 三者的区别？**

**什么是尾递归？使用 [@tailrec](https://link.jianshu.com/?t=https://www.scala-lang.org/api/current/scala/annotation/tailrec.html" \t "https://www.jianshu.com/p/_blank) 标签可使编译器强制使用尾递归。**正常递归，每一次递归步骤，需要保存信息到堆栈里面，当递归步骤很多时，导致堆栈溢出。  
尾递归就是为了解决上述问题，在尾递归中所有的计算都是在递归之前调用，  
编译器可以利用这个属性避免堆栈错误，尾递归的调用可以使信息不插入堆栈，从而优化尾递归。

**Es问题**

**settings是修改分片和副本数的。**

**mappings是修改字段和类型的。**

ES中以定义的mapping不能修改名字和属性

**分页语句，深度分页有什么问题（页数大于10000时），应如何优化**

GET /\_search

{

"from": 30,

"size": 10

}

**Put和post区别**

ES可以使用PUT或者POST对文档进行更新(全部更新)

局部更新，可以添加新字段或者更新已有字段（必须使用POST）

PUT会将新的json值完全替换掉旧的；而POST方式只会更新相同字段的值，其他数据不会改变，新提交的字段若不存在则增加

分词之后查询，并且根据lucene的评分机制(TF/IDF)来进行评分。

**详细描述一下Elasticsearch索引文档的过程。**

协调节点默认使用文档ID参与计算（也支持通过routing），以便为路由提供合适的分片。

shard = hash(document\_id) % (num\_of\_primary\_shards)

当分片所在的节点接收到来自协调节点的请求后，会将请求写入到Memory Buffer(客户端不可读)，然后定时（默认是每隔1秒）写入到Filesystem Cache(客户端可读)，这个从Momery Buffer到Filesystem Cache的过程就叫做refresh；

当然在某些情况下，存在Momery Buffer和Filesystem Cache的数据可能会丢失，ES是通过translog的机制来保证数据的可靠性的。其实现机制是接收到请求后，同时也会写入到translog中，当Filesystem cache中的数据写入到磁盘中时，才会清除掉，这个过程叫做flush；

在flush过程中，内存中的缓冲将被清除，内容被写入一个新段，段的fsync将创建一个新的提交点，并将内容刷新到磁盘，旧的translog将被删除并开始一个新的translog。

flush触发的时机是定时触发（默认30分钟）或者translog变得太大（默认为512M）时；

索引历史数据时，可以关闭refresh，之后再refresh

Segment合并

**在并发情况下，Elasticsearch如果保证读写一致？**

**可以通过版本号使用乐观并发控制**，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；

另外对于写操作，一致性级别支持quorum/one/all，默认为quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。

对于读操作，可以设置replication为sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置replication为async时，也可以通过设置搜索请求参数\_preference为primary来查询主分片，确保文档是最新版本。

**热索引分片不均**   
默认情况下，ES 集群的数据均衡策略是以各节点的分片总数(indices\_all\_active)作为基准的。这对于搜索服务来说无疑是均衡搜索压力提高性能的好办法。但是对于 Elastic Stack 场景，一般压力集中在新索引的数据写入方面。正常运行的时候，也没有问题。**但是当集群扩容时，新加入集群的节点，分片总数远远低于其他节点**。这时候如果有新索引创建，ES 的默认策略会导致新索引的所有主分片几乎全分配在这台新节点上。整个集群的写入压力，压在一个节点上，结果很可能是这个节点直接被压死，集群出现异常。   
所以，对于 Elastic Stack 场景，强烈建议大家预先计算好索引的分片数后，配置好单节点分片的限额。比如，一个 5 节点的集群，索引主分片 10 个，副本 1 份。则平均下来每个节点应该有 4 个分片，那么就配置：

# curl -s -XPUT http://127.0.0.1:9200/logstash-2015.05.08/\_settings -d '{

"index": { "routing.allocation.total\_shards\_per\_node" : "5" }

}'

注意，这里配置的是 5 而不是 4。因为我们需要预防有机器故障，分片发生迁移的情况。如果写的是 4，那么分片迁移会失败。

**在必要的时候，还可以通过 ES 的 reroute 接口，手动完成对分片的分配选择的控制。**

**节点下线**：修改setting配置exclude ip，会将节点索引转移至其他节点，然后下线

**Elasticsearch选主算法**

1、对所有可以成为master的节点根据nodeId排序，每次选举每个节点都把自己所知道节点排一次序，然后选出第一个（第0位）节点，暂且认为它是master节点。

2、如果对某个节点的投票数达到一定的值（可以成为master节点数n/2+1）并且该节点自己也选举自己，那这个节点就是master。否则重新选举。

3、对于brain split问题，需要把候选master节点最小值设置为可以成为master节点数n/2+1（quorum ）

**Sqoo问题：**

1. Sqoop导数据，关系型数据库和hive字段数据类型不一致，使用什么命令参数？

有没有碰见过字段串行错位的，Sqoop中分隔符去除参数

sqoop import \

--connect jdbc:mysql://db.dajiangtai.net:3306/djtdb\_hadoop \

--username sqoop \

--password sqoop \

--table user \

--target-dir /junior/sqoop/ \ //可选，不指定目录，数据默认导入到/user下

--where "sex='female'" \ //可选

--as-sequencefile \ //可选，不指定格式，数据格式默认为 Text 文本格式

--num-mappers 10 \ //可选，这个数值不宜太大

--null-string '\\N' \ //可选

--null-non-string '\\N' \ //可选

-schema tracking

--map-column-java wsresp=String

--map-column-hive wsresp=STRING

--hive-drop-import-delims Drops \n, \r, and \01 from string fields when importing to Hive.

2、在实际应用中还存在这样一个问题，比如导入数据的时候，Map Task 执行失败， 那么该 Map 任务会转移到另外一个节点执行重新运行，这时候之前导入的数据又要重新导入一份，造成数据重复导入。 因为 Map Task 没有回滚策略，一旦运行失败，已经导入数据库中的数据就无法恢复。Sqoop export 提供了一种机制能保证原子性， 使用--staging-table 选项指定临时导入的表。Sqoop export 导出数据的时候会分为两步：第一步，将数据导入数据库中的临时表，如果导入期间 Map Task 失败，会删除临时表数据重新导入；第二步，确认所有 Map Task 任务成功后，会将临时表名称为指定的表名称。

方法一：sqoop export \

--connect jdbc:mysql://db.dajiangtai.net:3306/djtdb\_hadoop \

--username sqoop \

--password sqoop \

--table user \

--staging-table staging\_user

 方法二:使用 --update-key id和--update-mode allowinsert 两个选项的情况下，如果数据已经存在，则更新数据，如果数据不存在，则插入新数据记录。

sqoop export \

--connect jdbc:mysql://db.dajiangtai.net:3306/djtdb\_hadoop \

--username sqoop \

--password sqoop \

--table user \

--update-key id \

--update-mode allowinsert

**java问题**

1. 讲讲Gc机制，垃圾收集方法，新生代、老年代。
2. Hashmap数据结构，是否线程安全，如何线程安全的适用

|  |  |
| --- | --- |
|  | //Hashtable  Map<String, String> hashtable = new Hashtable<>();  //synchronizedMap  Map<String, String> synchronizedHashMap = Collections.synchronizedMap(new HashMap<String, String>());  //ConcurrentHashMap性能高于另外两个  Map<String, String> concurrentHashMap = new ConcurrentHashMap<>(); |

Map是否是有序的，有没有有序的实现类

Treemap key的顺序，也可以自定义排序方式Comparator comparable

LinkedHashMap元素进入顺序

3、

**int** ret = 0;

**try**{

**throw** **new** Exception();

}

**catch**(Exception e){

ret = 1;

**return** ret;

}

**finally**{

ret = 2;

}

主要的考点就是catch中的return在finally之后执行 但是会将return的值放到一个地方存起来，所以finally中的ret=2会执行，但返回值是1。

4、Spring中用到的设计模式

工厂模式:IOC容器  
代理模式:AOP  
策略模式:在spring采取动态代理时，根据代理的类有无实现接口有JDK和CGLIB两种代理方式，就是采用策略模式实现的  
单例模式:默认情况下spring中的bean只存在一个实例

5、mysql索引内部数据结构，oracle索引内部数据结构

B+树，三层，真实的数据存储在叶子节点

1. redis数据结构有几种？

Redis集群中，如何将一个对象映射到对应的缓存服务器？

接上个问题，缓存集群中如果新增一台服务器，怎么才能不影响大部分缓存数据的命中？

一共有5种，字符串，散列，列表，集合，有序集合。

一般就是hash%N,就是用对象的hash值对缓存服务器的个数取余

7、如何判断一个链表有没有环，环的入口位置如何确定

对于如何判断链表有环，可以从起点发出两个指针，一个一次一步，另一个一次两步，如果两个指针相遇，那么这个单链表就有环。

第一问得出相遇点后，再发出一个指针，统计这个指针再次回到这个点的距离，也就是环的距离。然后从起点再发出两个指针，一个指针在另一个前面，两个指针的距离就是环的距离，当两个指针再次相遇的时候就是环的入口

8、二叉树层序遍历

层序遍历与先序、中序、后序遍历不同。层序遍历用到了队列，而先、中、后序需要用到栈。

因此，先、中、后序遍历 可以 采用递归方式来实现，而层序遍历则没有递归方式。

private void levelTraverse(BinaryNode<T> root){

5 if(root == null)

6 return;

7

8 Queue<BinaryNode<T>> queue = new LinkedList<>();//层序遍历时保存结点的队列

9 queue.offer(root);//初始化

10 while(!queue.isEmpty()){

11 BinaryNode<T> node = queue.poll();

12 System.out.print(node.element + " ");//访问节点

13 if(node.left != null)

14 queue.offer(node.left);

15 if(node.right != null)

16 queue.offer(node.right);

17 }

18 }

4 //前序遍历递归的方式

5 public void preOrder(BinaryTreeNode root){

6 if(null!=root){

7 System.out.print(root.getData()+"\t"); --中--左---右

8 preOrder(root.getLeft()); --在此打印则 --左---中---右

9 preOrder(root.getRight()); --在此打印则 --左--右--中

10 }

11 }

12 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

13 //前序遍历非递归的方式 适用栈，先进后出

14 public void preOrderNonRecursive(BinaryTreeNode root){

15 Stack<BinaryTreeNode> stack=new Stack<BinaryTreeNode>();

16 while(true){

17 while(root!=null){

18 System.out.print(root.getData()+"\t");

19 stack.push(root);

20 root=root.getLeft();

21 }

22 if(stack.isEmpty()) break;

23 root=stack.pop(); --在此打印则中序遍历

24 root=root.getRight(); --

25 }

26 }

集群负载均衡技术，LVS和HAProxy相比，它的缺点是什么?

Lvs配置复杂，性能高

HAProxy配置简单，性能低

**算法：**

集成学习有两个流派，一个是boosting派系，它的特点是各个弱学习器之间有依赖关系。另一种是bagging流派，它的特点是各个弱学习器之间没有依赖关系，可以并行拟合。本文就对集成学习中Bagging与随机森林算法做一个总结。  
随机森林是集成学习中可以和梯度提升树GBDT分庭抗礼的算法，尤其是它可以很方便的并行训练，在如今大数据大样本的的时代很有诱惑力。

作为一个可以高度并行化的算法，RF在大数据时候大有可为。这里也对常规的随机森林算法的优缺点做一个总结。  
RF（随机森林）的主要优点有：  
1）训练可以高度并行化，对于大数据时代的大样本训练速度有优势。个人觉得这是的最主要的优点。  
2）由于可以随机选择决策树节点划分特征，这样在样本特征维度很高的时候，仍然能高效的训练模型。  
3）在训练后，可以给出各个特征对于输出的重要性  
4）由于采用了随机采样，训练出的模型的方差小，泛化能力强。  
5）相对于Boosting系列的Adaboost和GBDT， RF实现比较简单。  
6）对部分特征缺失不敏感。  
RF的主要缺点有：  
1)在某些噪音比较大的样本集上，RF模型容易陷入过拟合。  
2)取值划分比较多的特征容易对RF的决策产生更大的影响，从而影响拟合的模型的效果。

偏差：描述的是预测值（估计值）的期望E’与真实值Y之间的差距。偏差越大，越偏离真实数据。

方差：描述的是预测值P的变化范围，离散程度，是预测值的方差，也就是离其期望值E的距离。方差越大，数据的分布越分散。

模型的真实误差是两者之和

如果是小训练集，高偏差/低方差的分类器（例如，朴素贝叶斯NB）要比低偏差/高方差大分类的优势大（例如，KNN），因为后者会过拟合。但是，随着你训练集的增长，模型对于原数据的预测能力就越好，偏差就会降低，此时低偏差/高方差分类器就会渐渐的表现其优势（因为它们有较低的渐近误差），此时高偏差分类器此时已经不足以提供准确的模型了。

****1.朴素贝叶斯****

朴素贝叶斯属于生成式模型（关于生成模型和判别式模型，主要还是在于是否是要求联合分布），非常简单，你只是做了一堆计数。如果注有条件独立性假设（一个比较严格的条件），朴素贝叶斯分类器的收敛速度将快于判别模型，如逻辑回归，所以你只需要较少的训练数据即可。即使NB条件独立假设不成立，NB分类器在实践中仍然表现的很出色。它的主要缺点是它不能学习特征间的相互作用，用mRMR中R来讲，就是特征冗余。引用一个比较经典的例子，比如，虽然你喜欢Brad Pitt和Tom Cruise的电影，但是它不能学习出你不喜欢他们在一起演的电影。

优点：

朴素贝叶斯模型发源于古典数学理论，有着坚实的数学基础，以及稳定的分类效率。

对小规模的数据表现很好，能个处理多分类任务，适合增量式训练；

对缺失数据不太敏感，算法也比较简单，常用于文本分类。

缺点：

需要计算先验概率；

分类决策存在错误率；

对输入数据的表达形式很敏感。

****Logistic Regression（逻辑回归）****

属于判别式模型，有很多正则化模型的方法（L0， L1，L2，etc），而且你不必像在用朴素贝叶斯那样担心你的特征是否相关。与决策树与SVM机相比，你还会得到一个不错的概率解释，你甚至可以轻松地利用新数据来更新模型（使用在线梯度下降算法，online gradient descent）。如果你需要一个概率架构（比如，简单地调节分类阈值，指明不确定性，或者是要获得置信区间），或者你希望以后将更多的训练数据快速整合到模型中去，那么使用它吧。

优点：

实现简单，广泛的应用于工业问题上；

分类时计算量非常小，速度很快，存储资源低；

便利的观测样本概率分数；

对逻辑回归而言，多重共线性并不是问题，它可以结合L2正则化来解决该问题；

缺点：

当特征空间很大时，逻辑回归的性能不是很好；

容易欠拟合，一般准确度不太高

不能很好地处理大量多类特征或变量；

只能处理两分类问题（在此基础上衍生出来的softmax可以用于多分类），且必须线性可分；

对于非线性特征，需要进行转换；

****4.最近领算法——KNN****

KNN即最近邻算法，其主要过程为：

1. 计算训练样本和测试样本中每个样本点的距离（常见的距离度量有欧式距离，马氏距离等）；2. 对上面所有的距离值进行排序；3. 选前k个最小距离的样本；4. 根据这k个样本的标签进行投票，得到最后的分类类别；

如何选择一个最佳的K值，这取决于数据。一般情况下，在分类时较大的K值能够减小噪声的影响。但会使类别之间的界限变得模糊。一个较好的K值可通过各种启发式技术来获取，比如，交叉验证。另外噪声和非相关性特征向量的存在会使K近邻算法的准确性减小。

近邻算法具有较强的一致性结果。随着数据趋于无限，算法保证错误率不会超过贝叶斯算法错误率的两倍。对于一些好的K值，K近邻保证错误率不会超过贝叶斯理论误差率。

****KNN算法的优点****

理论成熟，思想简单，既可以用来做分类也可以用来做回归；

可用于非线性分类；

训练时间复杂度为O(n)；

对数据没有假设，准确度高，对outlier不敏感；

****缺点****

计算量大；

样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；

需要大量的内存；

****决策树****

易于解释。它可以毫无压力地处理特征间的交互关系并且是非参数化的，因此你不必担心异常值或者数据是否线性可分（举个例子，决策树能轻松处理好类别A在某个特征维度x的末端，类别B在中间，然后类别A又出现在特征维度x前端的情况）。它的缺点之一就是不支持在线学习，于是在新样本到来后，决策树需要全部重建。另一个缺点就是容易出现过拟合，但这也就是诸如随机森林RF（或提升树boosted tree）之类的集成方法的切入点。另外，随机森林经常是很多分类问题的赢家（通常比支持向量机好上那么一丁点），它训练快速并且可调，同时你无须担心要像支持向量机那样调一大堆参数，所以在以前都一直很受欢迎。

决策树中很重要的一点就是选择一个属性进行分枝，因此要注意一下信息增益的计算公式，并深入理解它。

信息熵的计算公式如下:

IMG_256

其中的n代表有n个分类类别（比如假设是2类问题，那么n=2）。分别计算这2类样本在总样本中出现的概率p1和p2，这样就可以计算出未选中属性分枝前的信息熵。

现在选中一个属性xixi用来进行分枝，此时分枝规则是：如果xi=vxi=v的话，将样本分到树的一个分支；如果不相等则进入另一个分支。很显然，分支中的样本很有可能包括2个类别，分别计算这2个分支的熵H1和H2,计算出分枝后的总信息熵H’ =p1H1+p2 H2,则此时的信息增益ΔH = H - H’。以信息增益为原则，把所有的属性都测试一边，选择一个使增益最大的属性作为本次分枝属性。

****决策树自身的优点：****

计算简单，易于理解，可解释性强；

比较适合处理有缺失属性的样本；

能够处理不相关的特征；

在相对短的时间内能够对大型数据源做出可行且效果良好的结果。

****缺点****

容易发生过拟合（随机森林可以很大程度上减少过拟合）；

忽略了数据之间的相关性；

对于那些各类别样本数量不一致的数据，在决策树当中,信息增益的结果偏向于那些具有更多数值的特征（只要是使用了信息增益，都有这个缺点，如RF）

****Adaboosting****

Adaboost是一种加和模型，每个模型都是基于上一次模型的错误率来建立的，过分关注分错的样本，而对正确分类的样本减少关注度，逐次迭代之后，可以得到一个相对较好的模型。是一种典型的boosting算法。下面是总结下它的优缺点。

****优点****

adaboost是一种有很高精度的分类器。

可以使用各种方法构建子分类器，Adaboost算法提供的是框架。

当使用简单分类器时，计算出的结果是可以理解的，并且弱分类器的构造极其简单。

简单，不用做特征筛选。

不容易发生overfitting。

关于随机森林和GBDT等组合算法，参考这篇文章：机器学习-组合算法总结

****缺点****：对outlier比较敏感

****SVM支持向量机****

高准确率，为避免过拟合提供了很好的理论保证，而且就算数据在原特征空间线性不可分，只要给个合适的核函数，它就能运行得很好。在动辄超高维的文本分类问题中特别受欢迎。可惜内存消耗大，难以解释，运行和调参也有些烦人，而随机森林却刚好避开了这些缺点，比较实用。

****优点****

可以解决高维问题，即大型特征空间；

能够处理非线性特征的相互作用；

无需依赖整个数据；

可以提高泛化能力；

****缺点****

当观测样本很多时，效率并不是很高；

对非线性问题没有通用解决方案，有时候很难找到一个合适的核函数；

对缺失数据敏感；

对于核的选择也是有技巧的（libsvm中自带了四种核函数：线性核、多项式核、RBF以及sigmoid核）：

第一，如果样本数量小于特征数，那么就没必要选择非线性核，简单的使用线性核就可以了；

第二，如果样本数量大于特征数目，这时可以使用非线性核，将样本映射到更高维度，一般可以得到更好的结果；

第三，如果样本数目和特征数目相等，该情况可以使用非线性核，原理和第二种一样。

对于第一种情况，也可以先对数据进行降维，然后使用非线性核，这也是一种方法。

****7. 人工神经网络的优缺点****

****人工神经网络的优点****：

分类的准确度高；

并行分布处理能力强,分布存储及学习能力强，

对噪声神经有较强的鲁棒性和容错能力，能充分逼近复杂的非线性关系；

具备联想记忆的功能。

****人工神经网络的缺点****：

神经网络需要大量的参数，如网络拓扑结构、权值和阈值的初始值；

不能观察之间的学习过程，输出结果难以解释，会影响到结果的可信度和可接受程度；

学习时间过长,甚至可能达不到学习的目的。

****8、K-Means聚类****

之前写过一篇关于K-Means聚类的文章，博文链接：机器学习算法-K-means聚类。关于K-Means的推导，里面有着很强大的EM思想。

****优点****

算法简单，容易实现 ；

对处理大数据集，该算法是相对可伸缩的和高效率的，因为它的复杂度大约是O(nkt)，其中n是所有对象的数目，k是簇的数目,t是迭代的次数。通常k<<n。这个算法通常局部收敛。

算法尝试找出使平方误差函数值最小的k个划分。当簇是密集的、球状或团状的，且簇与簇之间区别明显时，聚类效果较好。

****缺点****

对数据类型要求较高，适合数值型数据；

可能收敛到局部最小值，在大规模数据上收敛较慢

K值比较难以选取；

对初值的簇心值敏感，对于不同的初始值，可能会导致不同的聚类结果；

不适合于发现非凸面形状的簇，或者大小差别很大的簇。

对于”噪声”和孤立点数据敏感，少量的该类数据能够对平均值产生极大影响。

****算法选择参考****

之前翻译过一些国外的文章，有一篇文章中给出了一个简单的算法选择技巧：

1. 首当其冲应该选择的就是逻辑回归，如果它的效果不怎么样，那么可以将它的结果作为基准来参考，在基础上与其他算法进行比较；

2. 然后试试决策树（随机森林）看看是否可以大幅度提升你的模型性能。即便最后你并没有把它当做为最终模型，你也可以使用随机森林来移除噪声变量，做特征选择；

3. 如果特征的数量和观测样本特别多，那么当资源和时间充足时（这个前提很重要），使用SVM不失为一种选择。

通常情况下：【GBDT>=SVM>=RF>=Adaboost>=Other…】，现在深度学习很热门，很多领域都用到，它是以神经网络为基础的，目前我自己也在学习，只是理论知识不是很厚实，理解的不够深，这里就不做介绍了。

算法固然重要，但好的数据却要优于好的算法，设计优良特征是大有裨益的。假如你有一个超大数据集，那么无论你使用哪种算法可能对分类性能都没太大影响（此时就可以根据速度和易用性来进行抉择）

为什么离职？

爱逛的网站