**第二小组：Spark生态圈面试题**

1.Spark中的数据本地性有哪三种：

 PROCESS\_LOCAL是指读取缓存在本地节点的数据

NODE\_LOCAL是指读取本地节点硬盘数据

 ANY是指读取非本地节点数据

### **2.[过滤器有什么用途：](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%231%E8%BF%87%E6%BB%A4%E5%99%A8%E6%9C%89%E4%BB%80%E4%B9%88%E7%94%A8%E9%80%94%EF%BC%9A" \t "https://zhuanlan.zhihu.com/p/_blank)**

增强hbase查询数据的功能

减少服务端返回给客户端的数据量

reduce之后数据的输出量有多大（结合具体场景，比如pi）

Sca阶段的增强日志（1.5T—2T）

过滤性质的mr程序，输出比输入少

解析性质的mr程序，输出比输入多（找共同朋友）

### **[3、RDD缓存](https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md" \l "2rdd%E7%BC%93%E5%AD%98)类型**

rdd.cache

rdd.persist

### **4.[什么是spark](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%233%E4%BB%80%E4%B9%88%E6%98%AFspark" \t "https://zhuanlan.zhihu.com/p/_blank)**

基于内存计算发数据分析引擎，提高在大数据环境下数处理的实时性，spark仅涉及数据计算

### **5.[RDD 是什么](https://link.zhihu.com/?target=https://gitee.com/souyunku/DevBooks/blob/master/docs/%E5%A4%A7%E6%95%B0%E6%8D%AE/%E5%A4%A7%E6%95%B0%E6%8D%AE%E6%9C%80%E6%96%B02021%E5%B9%B4%E9%9D%A2%E8%AF%95%E9%A2%98%E9%99%84%E7%AD%94%E6%A1%88%E8%A7%A3%E6%9E%90%EF%BC%8C%E5%A4%A7%E6%B1%87%E6%80%BB.md%238rdd-%E6%98%AF%E4%BB%80%E4%B9%88" \t "https://zhuanlan.zhihu.com/p/_blank)**

弹性分布式数据集，是spark中最基本的数据抽象，可以存于内存中或者磁盘中，分布式存储可用于分布式计算

6.什么是窄依赖

窄依赖指的是每一个父RDD的partition最多被子RDD的一个artition使用一对一

1. 什么是宽依赖

是多个子RDD的partition会依赖于同一个父RDD的partition多对一

# 简述**RDD的血缘(Lineage)**

RDD血缘（RDD Lineage），也可以叫：RDD依赖关系图。当我们计算一个RDD时，会依赖一个或多个父RDD的数据，而这些父RDD又会依赖它自身的父RDD，这样RDD之间的依赖关系就形成了一个有向无环图（也叫DAG图），这些依赖关系被记录在一个图中，这就是RDD的血缘（也叫RDD Lineage）。

1. RDD之间的依赖关系是如何产生的呢？

这些依赖关系的产生是由于对RDD进行的多个转换操作。由于RDD是懒评价的，一般情况下在没有action操作时，不会执行转换操作。也就是说在执行计算任务之前，执行的操作基本上都是转换操作，而我们知道每个转换操作的输出是一个新的RDD，这样通过多个转换操作就会得到多个新的RDD，而新的RDD之间就形成了RDD 的血缘关系（RDD Lineage）。

10.1.什么时候该使用持久化（缓存）

在[Spark](https://so.csdn.net/so/search?q=Spark&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_43161811/article/details/_blank)应用开发中，有时我们希望能多次使用同一个RDD。如果简单地对RDD进行进行调用，每执行一次action操作，Spark每次都会重算RDD以及它的所有依赖，这在迭代算法中消耗格外的大。

11.简述RDD CheckPoint 检查点

所谓的检查点其实就是通过将 RDD 中间结果写入磁盘由于血缘依赖过长会造成容错成本过高，这样就不如在中间阶段做检查点容错，如果检查点之后有节点出现问题，可以从检查点开始重做血缘，减少了开销。对 RDD 进行 checkpoint 操作并不会马上被执行，必须执行 Action 操作才能触发。

12.cache & persist & checkpoint 的特点和区别

特点

cache：

将数据临时存储在内存中进行数据重用

会在血缘关系中添加新的依赖，一旦出现问题，可以重头读取数据

persist：

将数据临时存储在磁盘文件中进行数据重用

因为涉及到磁盘IO，性能较低，但是数据安全

如果作业执行完毕，临时保存的数据文件会丢失

会在血缘关系中添加新的依赖，一旦出现问题，可以重头读取数据

checkpoint：

将数据长久的保存在磁盘文件中进行数据重用

因为涉及到磁盘IO，性能较低，但是数据安全

为了保证数据安全，所以一般情况下，会独立执行作业，

即调用检查点的rdd以前的流程都会重新执行一遍，所以效率比较低

为了能够提高效率，一般情况下，是需要和cache联合使用的

执行过程中，会切断血缘关系，重新建立新的血缘关系，

相当于将一个作业的数据源，由原文件切到检查点落盘的文件

下次重读数据时直接从checkpoint文件中读取

checkpoint等同于改变数据源

区别

cache/persist缓存只是将数据保存起来，不切断血缘依赖，只是加一条缓存的依赖关系。而checkpoint检查点会切断血缘依赖，checkpoint下游的RDD直接从checkpoint落盘的文件中读取数据，checkpoint等同于改变了数据源。（如下图）

cache/persist缓存的数据通常存储在磁盘、内存等地方，可靠性低。Checkpoint的数据通常存储在HDFS等容错性强、高可用的文件系统中，可靠性高。

建议对checkpoint的RDD使用cache缓存，这样checkpoint的job只需从cache缓存中读取数据即可，否则需要从头计算一次RDD。

13.简述Spark的几种持久化级别：

1.MEMORY\_ONLY

使用未序列化的Java对象格式，将数据保存在内存中。如果内存不够存放所有的数据，Spark会自动利用最近最少用(LRU)的缓存策略把最老的分区从内存中移除。下一次要用到已经被移除的分区数据时，这些分区需要从源头处重新计算一遍。这是默认的持久化策略，使用cache()方法时，实际就是使用的这种持久化策略。

2.MEMORY\_AND\_DISK

使用未序列化的Java对象格式，优先尝试将数据保存在内存中。如果内存不够存放所有的数据，Spark会自动利用最近最少用(LRU)的缓存策略把最老的分区从内存中移除，并将数据溢写到磁盘文件中，下次对这个RDD执行算子时，持久化在磁盘文件中的数据会被读取出来使用。

3.MEMORY\_ONLY\_SER

基本含义同MEMORY\_ONLY。唯一的区别是，会将RDD中的数据进行序列化，RDD的每个partition会被序列化成一个字节数组。这种方式会使缓存过程变慢，因为序列化对象也会消耗一些代价，不过这更加节省内存，从而可以避免持久化的数据占用过多内存导致频繁GC，同时也可以显著减少JVM的GC时间。

4.MEMORY\_AND\_DISK\_SER

基本含义同MEMORY\_AND\_DISK。唯一的区别是，会将RDD中的数据进行序列化，RDD的每个partition会被序列化成一个字节数组。这种方式会使缓存过程变慢，因为序列化对象也会消耗一些代价，不过这更加节省内存，从而可以避免持久化的数据占用过多内存导致频繁GC，同时也可以显著减少JVM的GC时间。

5.DISK\_ONLY

使用未序列化的Java对象格式，将数据全部写入磁盘文件中。

1. Spark SQL简介：是Spark用来处理结构化数据的一个模块，它提供了一个编程抽象叫做DataFrame并且作为分布式SQL查询引擎的作用。
2. Spark SQL的特点：1.容易整合（集成）；2.统一的数据访问方式；3.兼容Hive；44.标准的数据连接
3. Spark SQL允许用户提交SQL文本，支持哪些手段编写sql文本：1. spark 代码2. spark-sql的shell3. thriftserver
4. 什么是DataFrame：DataFrame是组织成命名列的数据集。它在概念上等同于关系数据库中的表，但在底层具有更丰富的优化。DataFrames可以从各种来源构建，例如：结构化数据文件(JSON)、外部数据库或现有RDDs。DataFrame API支持的语言有Scala，Java，Python和R。
5. 什么是DataSet：Dataset是数据的分布式集合。Dataset是在Spark 1.6中添加的一个新接口，是DataFrame之上更高一级的抽象。它提供了RDD的优点（强类型化）以及Spark SQL优化后的执行引擎的优点。一个Dataset 可以从JVM对象构造，然后使用函数转换（map， flatMap，filter等）去操作。 Dataset API 支持Scala和Java。 Python不支持Dataset API。

### Spark-SQL连接Hive 的五种方法：1）内嵌的 HIVE。2）外部的 HIVE。3）运行 Spark beeline。4）运行Spark-SQL CLI。5）代码操作Hive

1. 简述 GraphX应用背景： Spark GraphX是一个分布式图处理框架，它是基于Spark平台提供对图计算和图挖掘简洁易用的而丰富的接口，极大的方便了对分布式图处理的需求。社交网络中人与人之间有很多关系链，例如Twitter、Facebook、微博和微信等，这些都是大数据产生的地方都需要图计算，现在的图处理基本都是分布式的图处理，而并非单机处理。Spark GraphX由于底层是基于Spark来处理的，所以天然就是一个分布式的图处理系统。图的分布式或者并行处理其实是把图拆分成很多的子图，然后分别对这些子图进行计算，计算的时候可以分别迭代进行分阶段的计算，即对图进行并行计算。
2. 什么是边分割（Edge-Cut）：每个顶点都存储一次，但有的边会被打断分到两台机器上。这样做的好处是节省存储空间；坏处是对图进行基于边的计算时，对于一条两个顶点被分到不同机器上的边来说，要跨机器通信传输数据，内网通信流量大。
3. 什么是点分割（Vertex-Cut）：每条边只存储一次，都只会出现在一台机器上。邻居多的点会被复制到多台机器上，增加了存储开销，同时会引发数据同步问题。好处是可以大幅减少内网通信量。
4. GraphX存储模式：Graphx借鉴PowerGraph，使用的是Vertex-Cut(点分割)方式存储图，用三个RDD存储图数据信息：lVertexTable(id, data)：id为Vertex id，data为Edge data
5. 图计算模式： 目前基于图的并行计算框架已经有很多，比如来自Google的Pregel、来自Apache开源的图计算框架Giraph/HAMA以及最为著名的GraphLab，其中Pregel、HAMA和Giraph都是非常类似的，都是基于BSP（Bulk Synchronous Parallell）模式。

25.图数据库neo4j和spark下面的graphx有什么区别？

neo4j是native graph database，也就是有自己的数据库存储。它的长处在于支持交互式查询，属于oltp系统，很多人说不支持分片存储使其无法应付海量数据，本人觉得恰恰相反，可以说neo4j的存储方式是教科书式的以空间换时间，每台服务器配备ssd磁盘阵列虽然贵，但是可以大幅减少分片存储的带宽占用和通信时间开销，保证oltp的效率。 neo4j很容易上手，特有的cypher查询语言以画草图的方式查询和建模数据，很直观。适当构建查询计划的情况下，neo4j的查询效率很高，能够迅速从整网中找出符合特定模式的子网，供随后分析之用。 此外，neo4j实现了tinkerpop接口，tinkerpop是刚刚毕业的一个阿帕奇项目，有望建立图数据库的一套标准用户接口。同样实现tinkerpop的还有titan，orient等主流图数据库。再来看graphX ： graphX是spark的系统组件，存储是基于spark rdd的，有节点和边两种rdd。熟悉spark的朋友对rdd该不会陌生，spark通过缓存rdd的操作节省了大量计算和io开支，因此spark特别适合对海量数据进行运算，此理同样适用于graphX。因此，graphX自设计之初就是奔着图计算的目标去的，属于olap系统，而非oltp系统。 graphX有丰富的函数库，能完成很多经典图算法，如pagerank、三角计数、社群发现、最短路径计算等等。此外，图存储和计算的方式不禁让人想到神经网络算法，如果将隐层用节点rdd表示，隐层之间的边用边rdd表示，运用graphX的计算优势搭建起一套多层神经网络的想法很美妙，这应该就是MLlab相应算法模块的工作原理。 因此跟graphx相关的概念集中在图计算，而非图存储和查询领域。所以经常浏览db-engines的朋友们不难发现，图数据库列表里就没有graphx这一项。在比较图存储和图查询性能时，比较集合多是neo4j、orientdb、titan、arangodb等图数据库系统。而比较图计算时，比较集合多是graphlab、giraph、graphX。 简言之，图数据库系统和图计算系统不是一回事：前者是为了存储完整数据，并根据需求从中查询数据子集供分析展示之用；后者的任务是拿到一个图结构的数据集，从中计算一些有用的东西。

1. 图（GraphX）的基本概念：图是由顶点集合(vertex)及顶点间的关系集合（边edge）组成的一种数据结构。这里的图并非指代数中的图。图可以对事物以及事物之间的关系建模，图可以用来表示自然发生的连接数据，如：社交网络、互联网web页面。常用的应用有：在地图应用中找到最短路径、基于与他人的相似度图，推荐产品、服务、人际关系或媒体。
2. 有向图和无向图在有向图中，一条边的两个顶点一般扮演者不同的角色，比如父子关系、页面A连接向页面B；在一个无向图中，边没有方向，即关系都是对等的，比如qq中的好友。GraphX中有一个重要概念，所有的边都有一个方向，那么图就是有向图，如果忽略边的方向，就是无向图。
3. 有环图和无环图：有环图是包含循环的，一系列顶点连接成一个环。无环图没有环。在有环图中，如果不关心终止条件，算法可能永远在环上执行，无法退出。
4. 超步：图进行迭代计算时，每一轮的迭代叫做一个超步
5. 图数据分析：Google Pregel、Spark GraphX、GraphLab等图计算软件。传统的数据分析方法侧重于事物本身，即实体，例如银行交易、资产注册等等。而图数据不仅关注事物，还关注事物之间的联系。例如，如果在通话记录中发现张三曾打电话给李四，就可以将张三和李四关联起来，这种关联关系提供了与两者相关的有价值的信息，这样的信息是不可能仅从两者单纯的个体数据中获取的。
6. 图数据可视化：OLTP风格的图数据库或者OLAP风格的图数据分析系统（或称为图计算软件），都可以应用图数据库可视化技术。需要注意的是，图可视化与关系数据可视化之间有很大的差异，关系数据可视化的目标是对数据取得直观的了解，而图数据可视化的目标在于对数据或算法进行调试。

31.GraphX实现架构大致分为哪3个层：

• 实现层：该层定义了 GraphX由最基本的数据结构，即顶点、边和边点三元组；实现了GraphX不同的顶点切分策略以及数据存储方式以及图计算过程使用的数据存储结构，如路由表、重复顶点视图等。

• 操作层：主要包括抽象类GraphX及其实现类GraphImpl,在这两个类中定义了构建图操作、转换操作、结构操作、聚合操作和缓存操作等。另外，在GraphOps类中也实现了图基本属性操作和连接操作等。

• 算法层：GraphX根据实现层和操作层实现了常用的算法，如PageRank、三角关系统计、最短路径等。

32.Spark有哪些适用场景：

Spark是基于内存的迭代计算框架，适用于需要多次操作特定数据集的应用场合。需要反复操作的次数越多，所需读取的数据量越大，受益越大，数据量小但是计算密集度较大的场合，受益就相对较小。

33.Spark有哪些不适用场景：

由于RDD的特性，Spark不适用那种异步细粒度更新状态的应用，例如web服务的存储或者是增量的web爬虫和索引。就是对于那种增量修改的应用模型不适合。数据量不是特别大，但是要求近实时统计分析需求· 内存hold不住的场景，在内存不足的情况下，Spark会下放到磁盘，会降低应有的性能 ；高实时性要求的流式计算业务，例如实时性要求毫秒级；由于RDD设计上的只读特点，所以Spark对于待分析数据频繁变动的情景很难做（并不是不可以），比如题主例子里的搜索，假设你的数据集在频繁变化（不停增删改），而且又需要结果具有很强的一致性（不一致时间窗口很小），那么就不合适了。

### ****34.描述spark计算模型：****

RDD可以看做是对各种数据计算模型的统一抽象，Spark的计算过程主要是RDD的迭代计算过程。RDD的迭代计算过程非常类似于管道。分区数量取决于partition数量的设定，每个分区的数据只会在一个Task中计算。所有分区可以在多个机器节点的Executor上并行执行。

35.叙述spark 运行流程

· 构建Spark Application的运行环境，启动SparkContext

· SparkContext向资源管理器（可以是Standalone，Mesos，Yarn）申请运行Executor资源，并启动StandaloneExecutorbackend，

· Executor向SparkContext申请Task

· SparkContext将应用程序分发给Executor

· SparkContext构建成DAG图，将DAG图分解成Stage、将Taskset发送给Task Scheduler，最后由Task Scheduler将Task发送给Executor运行

· Task在Executor上运行，运行完释放所有资源

36.叙述spark 运行特点：

每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行Task。这种Application隔离机制是有优势的，无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统

Spark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了

提交SparkContext的Client应该靠近Worker节点（运行Executor的节点），最好是在同一个Rack里，因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换Task采用了数据本地性和推测执行的优化机制

37.MLlib提供了哪些工具：

1.机器学习算法：常规机器学习算法包括分类、回归、聚类和协同过滤。

2.特征工程：特征提取、特征转换、特征选择以及降维。

3.管道：构造、评估和调整的管道的工具。

4.存储：保存和加载算法、模型及管道

5.实用工具：线性代数，统计，数据处理等。

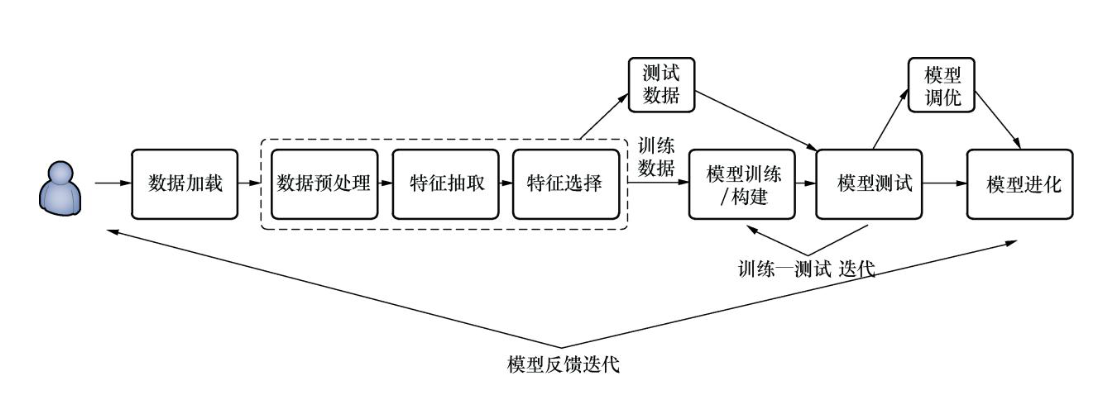
38.为什么MLlib转向数据框接口？

1.数据框可以提供比RDD更容易掌握使用的接口。数据框的主要优点包括Spark数据源来源、结构化查询语言的数据框查询、各编程语言之间统一的接口。

2.基于数据框的MLlib接口为多种机器学习算法与编程语言提供统一的接口。

3.数据框有助于实现机器学习管道，特别是特征转换。管道指南中可查看详细信息。

## 39.请画出**典型的机器学习工作流**



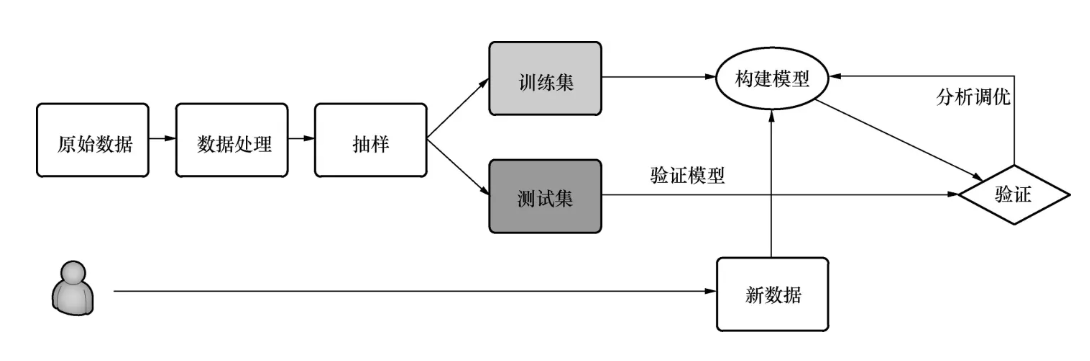
1. 根据学习系统学习反馈的本质，机器学习任务通常被分为哪几类

3类，即监督学习、无监督学习以及增强学习

## 41.**什么是监督学习**

监督学习的目标是学习将输入映射到与现实世界相一致的输出的一般规则。例如，垃圾邮件过滤数据集通常包含垃圾邮件以及非垃圾邮件。因此，能够知道训练集中的数据是垃圾邮件还是正常邮件。我们有机会利用这些信息来训练模型，以便对新来的邮件进行分类。如图 6-3所示，该图为监督学习的示意图。算法找到所需的模式后，可以使用这些模式对未标记的测试数据进行预测。这是最常见的机器学习任务类型，MLlib 也不例外，其中大部分算法都是监督学习，如朴素贝叶斯、逻辑回归、随机森林等。

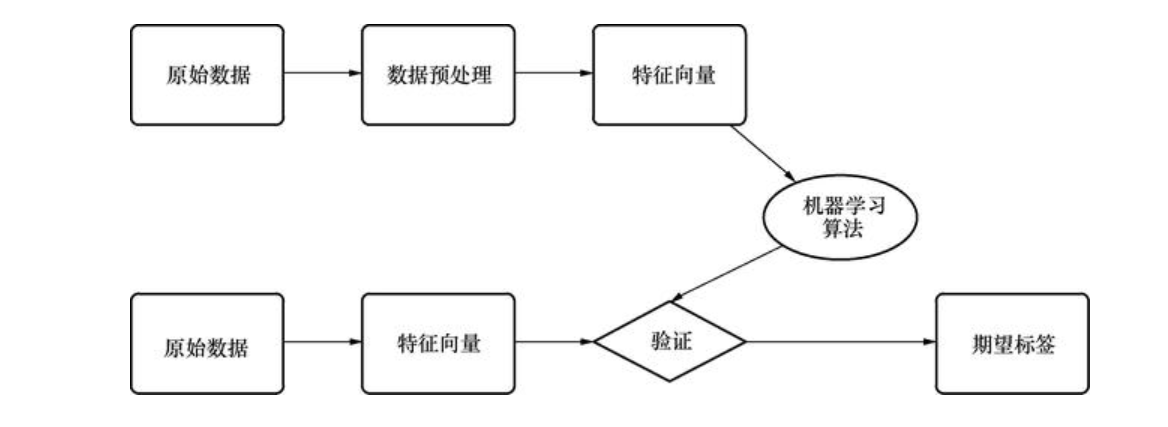
1. 画出监督学习的数据处理流程图



## 什么是**无监督学习**

在无监督学习中，数据没有相关的标签，也就是说无法区分训练集与测试集。因此，我们需要用算法上加上标签。因此，标签必须从数据集中推断出来，这意味着无监督学习算法的目标是通过描述结构，以某种结构化的方式对数据进行预处理。

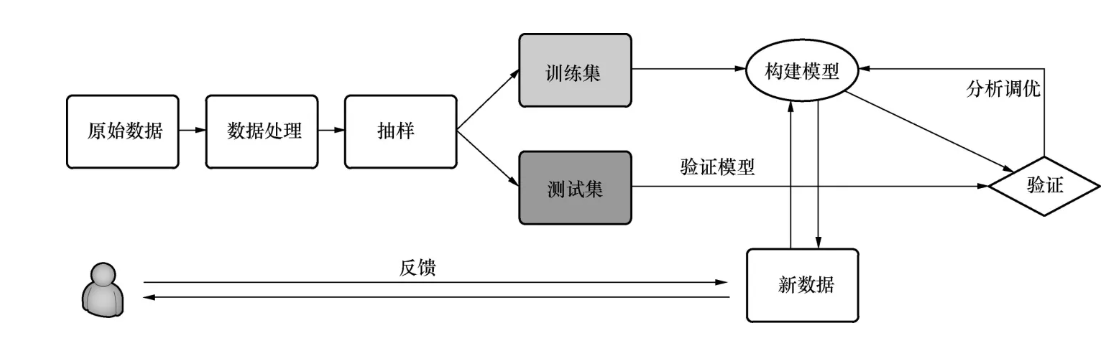
1. .画出无监督学习的数据处理流程图



## 44.什么是**增强学习**

增强学习是一种模型本身从一系列行为中学习的技术。数据集或样本的复杂性对于需要算法成功学习目标函数的增强学习非常重要。此外，为了达到最终目标，每条数据都需要做到一点，即在保证与外部环境相互作用的同时，应确保奖励函数的最大化。

1. 请画出**增强学习流程图**



## **45.试比较park MLlib与Spark ML异同**

在Spark MLlib模块中，可以看到它的源码主要分为两个包：spark.ml与spark.mllib，我们将前者称为Spark ML API，后者称为Spark MLlib API，有些算法在两个包中都可以找到，如协同过滤，有些算法只有MLlib有，如SVD。除此以外，它们还有一些区别。

一言以蔽之，MLlib与ML之间最大的区别在于，ML基于DataFrame，而MLlib API基于RDD，这与GraphX和GraphFrame之间的关系类似。在Spark 2.0后，基于RDD的API，也就是MLlib API，就已经进入了维护状态，而Spark MLlib首要的API为ML API。虽然如此，Spark MLlib仍然会以修复Bug的方式支持MLlib API，但不会增加新特性了。在Spark 2.x的版本中，ML API会逐渐变得与MLlib API一样，在完成这个过程后，MLlib API会被弃用。在Spark 3.0中，MLlib API会被彻底舍弃。

## **46.什么是数据预处理，为什么要进行数据预处理**

在机器学习实践中，数据科学家拿到的数据通常是不尽如人意的，例如存在大量的缺失值、特征的值是不同的量纲、有一些无关的特征、特征的值需要再次处理等情况，这样的数据无法直接训练，因此我们需要对这些数据进行预处理。预处理在机器学习中是非常重要的步骤，如果没有按照正确的方法对数据进行预处理，往往会得到错误的训练结果。

1. 请介绍集中介绍几种常见的预处理方法

## **数据标准化 缺失值处理 特征抽取 特征选择**

## **介绍机器学习中常见的分类算法应用**

决策树：决策树是一种机器学习的方法，本质上是通过一种树形结构对样本进行分类，每个非叶子结点是一次判断，每个叶子结点代表了分类结果。决策树是一种典型的监督学习，需要一定量的样本来学习的一个树形结构，常见的决策树构造树算法有C4.5与ID3。

随机森林：在[机器学习](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)中，随机森林是一个包含多个[决策树](https://baike.baidu.com/item/%E5%86%B3%E7%AD%96%E6%A0%91/10377049?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)的[分类器](https://baike.baidu.com/item/%E5%88%86%E7%B1%BB%E5%99%A8?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)， 并且其输出的类别是由个别树输出的类别的[众数](https://baike.baidu.com/item/%E4%BC%97%E6%95%B0/44796?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%9A%8F%E6%9C%BA%E6%A3%AE%E6%9E%97/_blank)而定。 Leo Breiman和Adele Cutler发展出推论出随机森林的算法。

49.简述Spark MLlib架构

从架构图可以看出MLlib主要包含三个部分：底层基础：包括Spark的运行库、矩阵库和向量库； 算法库：包含广义线性模型、推荐系统、聚类、决策树和评估的算法；实用程序：包括测试数据的生成、外部数据的读入等功能。

50.简述回归算法：属于监督式学习，每个个体都有一个与之相关联的实数标签，并且我们希望在给出用于表示这些实体的数值特征后，所预测出的标签值可以尽可能接近实际值。MLlib 目前支持回归算法有：线性回归、岭回归、Lasso和决策树。

51.简述聚类算法：聚类算法属于非监督式学习，通常被用于探索性的分析，是根据“物以类聚”的原理，将本身没有类别的样本聚集成不同的组，这样的一组数据对象的集合叫做簇，并且对每一个这样的簇进行描述的过程。它的目的是使得属于同一簇的样本之间应该彼此相似，而不同簇的样本应该足够不相似，常见的典型应用场景有客户细分、客户研究、市场细分、价值评估。MLlib 目前支持广泛使用的KMmeans聚类算法。

52.什么是协同过滤

协同过滤常被应用于推荐系统，这些技术旨在补充用户-商品关联矩阵中所缺失的部分。MLlib当前支持基于模型的协同过滤，其中用户和商品通过一小组隐语义因子进行表达，并且这些因子也用于预测缺失的元素。

## **什么是Spark Streaming？**

数据流是连续到达的无穷序列。流处理将不断流动的输入数据分成独立的单元进行处理。流处理是对流数据的低延迟处理和分析。Spark Streaming是Spark API核心的扩展，可实现实时数据的快速扩展，高吞吐量，高容错处理。Spark Streaming适用于大量数据的快速处理。

54.**Spark Streaming**实时处理用例包括哪些场景：

* 网站监控，网络监控
* 欺诈识别
* 网页点击
* 广告
* 物联网传感器

## 55.**Spark Straming如何工作**

Spark Streaming将数据流每X秒分作一个集合，称为Dstreams，它在内部是一系列RDD。您的Spark应用程序使用Spark API处理RDD，并且批量返回RDD操作的结果。

1. 简谈Spark Streaming的实时计算整合

基于Spark通用计算平台，可以很好地扩展各种计算类型的应用，尤其是Spark提供了内建的计算库支持，像Spark Streaming、Spark SQL、MLlib、GraphX，这些内建库都提供了高级抽象，可以用非常简洁的代码实现复杂的计算逻辑、这也得益于Scala编程语言的简洁性。这里，我们基于1.3.0版本的Spark搭建了计算平台，实现基于Spark Streaming的实时计算。我们的应用场景是分析用户使用手机App的行为。手机客户端会收集用户的行为事件（我们以点击事件为例），将数据发送到[数据服务](https://cloud.tencent.com/solution/data-collect-and-label-service?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)器，我们假设这里直接进入到Kafka[消息队列](https://cloud.tencent.com/product/cmq?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)。后端的实时服务会从Kafka消费数据，将数据读出来并进行实时分析，这里选择Spark Streaming，因为Spark Streaming提供了与Kafka整合的内置支持，经过Spark Streaming实时计算程序分析，将结果写入[Redis](https://cloud.tencent.com/product/crs?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)，可以实时获取用户的行为数据，并可以导出进行离线综合统计分析。Spark Streaming提供了一个叫做DStream（Discretized Stream）的高级抽象，DStream表示一个持续不断输入的数据流，可以基于Kafka、TCP Socket、Flume等输入数据流创建。在内部，一个DStream实际上是由一个RDD序列组成的。Sparking Streaming是基于Spark平台的，也就继承了Spark平台的各种特性，如容错（Fault-tolerant）、可扩展（Scalable）、高吞吐（High-throughput）等。在Spark Streaming中，每个DStream包含了一个时间间隔之内的数据项的集合，我们可以理解为指定时间间隔之内的一个batch，每一个batch就构成一个RDD数据集，所以DStream就是一个个batch的有序序列，时间是连续的，按照时间间隔将数据流分割成一个个离散的RDD数据集。

### **Spark Straming与Storm的区别**

Storm是纯实时的流式处理框架，SparkStreaming是准实时的处理框架（微批处理）。因为微批处理，SparkStreaming的吞吐量比Storm要高。Storm 的事务机制要比SparkStreaming的要完善。Storm支持动态资源调度。(spark1.2开始和之后也支持)

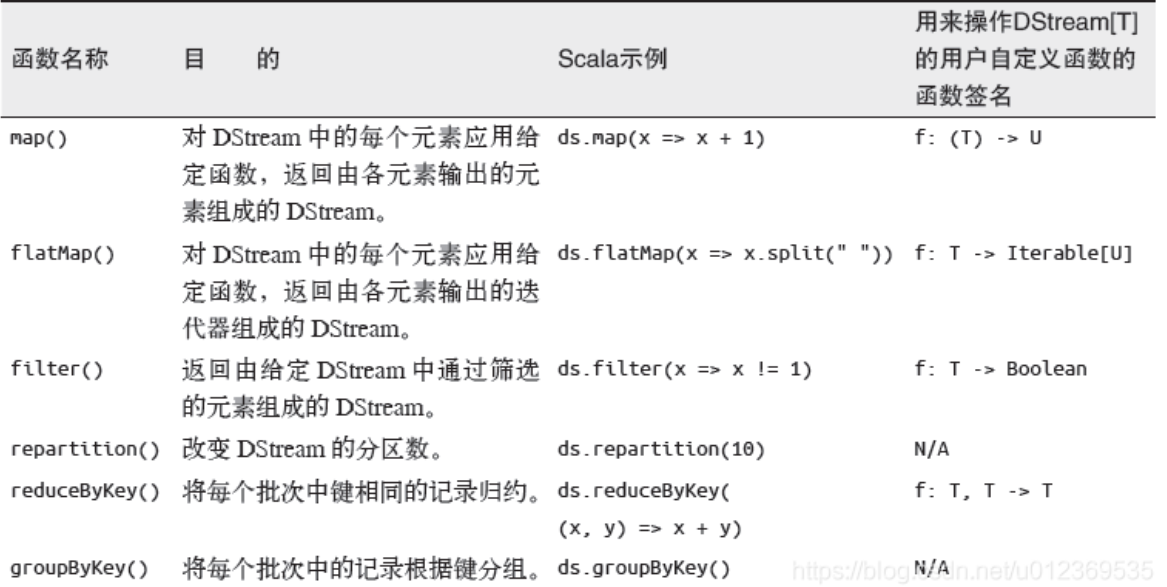
SparkStreaming擅长**复杂的业务处理**，Storm不擅长复杂的业务处理，擅长简单的汇总型计算。

### 列举**SparkStreaming算子**

foreachRDD属于output operation算子； transform属于transformation算子； updateStateByKey

59.描述Spark Streamin运行架构

总体来说，Spark Streaming是将流式计算分解成一系列短小的批处理作业。这里的批处理引擎是Spark Core，也就是Spark Streaming将输入数据按照batch interval（如5秒）分成一段一段的数据（Discretized Stream），每一段数据都转换成Spark中的RDD（Resilient Distributed Dataset），然后将Spark Streaming中对DStream的Transformation操作变为针对DSteam内各个RDD的Transformation操作，将RDD经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加或者存储到外部设备.

60.举例说明DStream的转化操作

·