1. 在hadoop集群上运行任务：

./spark-submit --master yarn

--deploy-mode client

--class testALS scalatest2.jar

/input/ratings.dat 0.8 12 0.068 20 测试类的参数

需要事先设置yarn环境变量：

export YARN\_HOME=$HOME/hadoop/hadoop-2.7.3

export YARN\_CONF\_DIR=$YARN\_HOME/etc/Hadoop

将会自动找到机器上安装的hadoop集群并提交任务

<http://spark.apache.org/docs/latest/running-on-yarn.html>

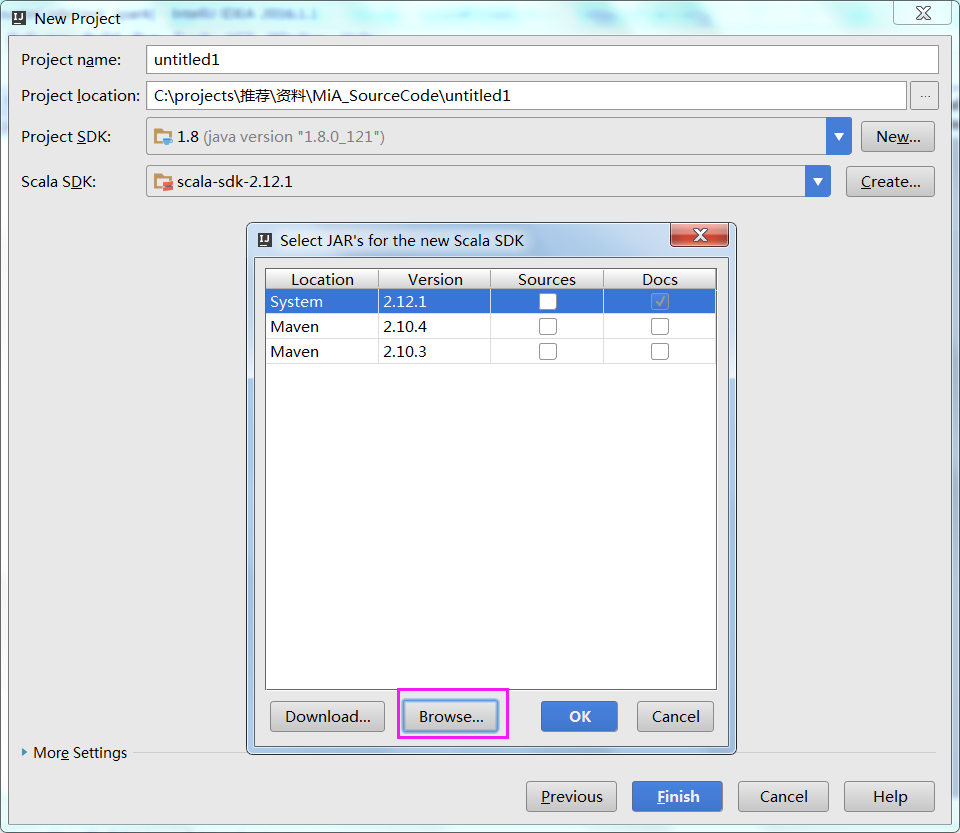
<http://spark.apache.org/docs/latest/submitting-applications.html>

(经实验，上述指令似乎不会将任务提交到集群上运行，原因未知)

提交到集群上：

./spark-submit --master "spark://192.168.146.38:7077" --conf "spark.executor.extraJavaOptions=-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xss8172k" --name "My APP13" --driver-memory 4g --executor-memory 4G --class testALS scalatest2.jar /input/ratings.dat 0.8 12 0.065 20

1. Spark解压到不包含空格的目录下（C:\Program Files (x86)这种是不行的），例如C:\spark-2.1.0-bin-hadoop2.7
2. IntelliJ中新建scala项目，选择spark目录下的jars目录作为scala sdk路径：

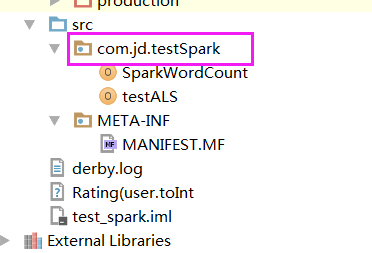


选择C:\spark-2.1.0-bin-hadoop2.7\jars

（spark要求指定的scala版本，不能随意选择scala版本，最好使用spark自带的，否则很可能运行出错）

1. 在项目中添加C:\spark-2.1.0-bin-hadoop2.7\jars到libraries路径中
2. 然后写个简单的word count程序就可以跑啦：

**import** org.apache.spark.SparkContext.\_  
**import** org.apache.spark.{SparkConf, SparkContext}  
  
**object** SparkWordCount{  
 **def** main(args: Array[String]) {  
 *//以本地线程方式运行，可以指定线程个数，  
 //如.setMaster("local[2]")，两个线程执行  
 //下面给出的是单线程执行* **val** conf = **new** SparkConf().setAppName(**"SparkWordCount"**).setMaster(**"local"**)  
 **val** sc = **new** SparkContext(conf)  
  
 *//wordcount操作，计算文件中包含Spark的行数* **val** count=sc.textFile(**"C:\\spark-2.1.0-bin-hadoop2.7\\NOTICE"**).filter(line => line.contains(**"this"**)).count()  
  
 *//打印结果  
 println*(**"count="**+count)  
 sc.stop()  
 }  
}

1. Spark上只有ALS推荐算法，如果要UserCF、ItemCF则需要配合mahout
2. Jar打包要点：
3. scala/java源文件要放到package下
4. 源文件名跟类名保持一致
5. 源文件开头指定package
6. Artifacts中指定main class
7. Spark submit任务时报错：

17/03/22 06:29:41 ERROR cluster.YarnClientSchedulerBackend: Yarn application has already exited with state FINISHED!

17/03/22 06:29:41 ERROR spark.SparkContext: Error initializing SparkContext.

java.lang.IllegalStateException: Spark context stopped while waiting for backend

at org.apache.spark.scheduler.TaskSchedulerImpl.waitBackendReady(TaskSchedulerImpl.scala:614)

at org.apache.spark.scheduler.TaskSchedulerImpl.postStartHook(TaskSchedulerImpl.scala:169)

at org.apache.spark.SparkContext.<init>(SparkContext.scala:567)

at org.apache.spark.SparkContext.<init>(SparkContext.scala:117)

at scalafirst.examples.SHelloWorld$.main(SHelloWorld.scala:9)

at scalafirst.examples.SHelloWorld.main(SHelloWorld.scala)

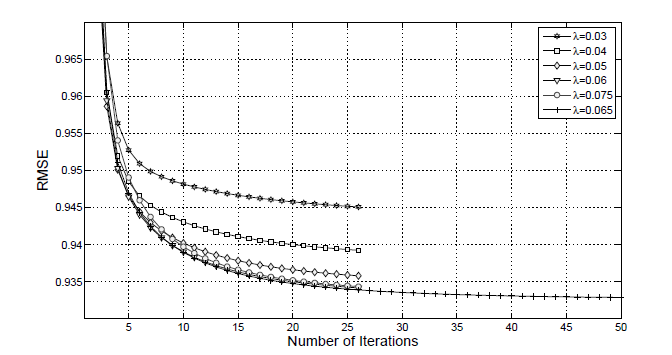
在hadoop的yarn-site.xml中增加：

1. **<property>**
2. **<name>**yarn.nodemanager.pmem-check-enabled**</name>**
3. **<value>**false**</value>**
4. **</property>**
6. **<property>**
7. **<name>**yarn.nodemanager.vmem-check-enabled**</name>**
8. **<value>**false**</value>**
9. **</property>**

<http://blog.csdn.net/linhanyichen/article/details/54909339>

1. ALS算法中的lamdba值取0.065时RMSE最小，原因未知（可能是经验值）。

Yunhong Zhou的经典论文Large-scale Parallel Collaborative Filtering the Netflix Prize中也是lambda=0.065时误差最小：



采用movielens测试数据，同样验证了以上结论：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 训练集比例 | 特征数 | Lambda | 迭代次数 | RMSE |
| 0.8 | 12 | 0.01 | 20 | 0.901451338025801 |
| 0.8 | 12 | 0.06 | 20 | 0.854433794711784 |
| 0.8 | 12 | 0.064 | 20 | 0.854011958626786 |
| 0.8 | 12 | 0.065 | 20 | 0.853513806063940 |
| 0.8 | 12 | 0.066 | 20 | 0.854303118377421 |
| 0.8 | 12 | 0.067 | 20 | 0.853825983940470 |
| 0.8 | 12 | 0.068 | 20 | 0.854193499261030 |
| 0.8 | 12 | 0.099 | 20 | 0.861335352213247 |
| 0.8 | 12 | 0.3 | 20 | 0.958779432247187 |
| 0.8 | 12 | 0.9 | 20 | 1.283270566137736 |

1. ALS中的迭代次数设为20比较合理，通常不超过20次迭代即能收敛到合理范围。Spark官方说法：ALS typically converges to a reasonable solution in 20 iterations or less。
2. Spark-sbumit报错Got exception: org.apache.hadoop.yarn.exceptions.YarnException: Unauthorized request to start container.

This token is expired. current time is 1490864524006 found 1490864383777

是因为集群中机器的时间相差比较大，同步一下机器的时间即可

1. RDD做repartition是不合理的，会进行shuffle操作影响性能。实际测试下来，使用repartition(20)对RDD重新分区，job总共耗时13分钟；而不重新分区的话，总共只花了6分钟。Rapartition本身耗时不长，但之后进行map时花的时间更长了。

（这个要看具体情况，某些情况下repartition是能提高性能的）

1. Spark报错：

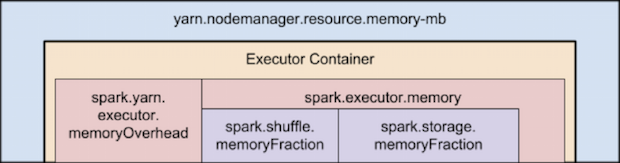
Caused by: [**Java**](http://lib.csdn.net/base/javase).lang.IllegalArgumentException: Compression codec com.[**Hadoop**](http://lib.csdn.net/base/hadoop).compression.lzo.LzoCodec not found.

at org.apache.hadoop.io.compress.CompressionCodecFactory.getCodecClasses(CompressionCodecFactory.java:135)  
        at org.apache.hadoop.io.compress.CompressionCodecFactory.<init>(CompressionCodecFactory.java:175)  
        at org.apache.hadoop.mapred.TextInputFormat.configure(TextInputFormat.java:45)

将编译好的hadoop-lzo\*.jar放到服务器上，然后修改spark-env.sh：

export SPARK\_CLASSPATH=$SPARK\_CLASSPATH:~/lzo/targetall/jarfile/hadoop-lzo-0.4.21-SNAPSHOT.jar

1. Executor内存分布如下：



Executor中的内存主要分为excution(shuffles, joins, sorts and aggregations…)和storage(cache, persist)两部分。

当execution内存不够时可以将storage中的blocks驱逐一部分进而占用他的内存，但不能无限借用，最少要为storage保存spark.memory.storageFraction的内存。

反之，由于实现比较复杂，storage不能借用execution的内存。

[https://spark.apache.org/docs/latest/tuning.html#memory-management-overview](https://spark.apache.org/docs/latest/tuning.html" \l "memory-management-overview)

（**猜测**：当shuffle时内存不够，而明确知道storage较小的情况下，可以调低storageFraction提升性能？）

1. 任务运行异常：

17/05/24 18:27:26 WARN TaskSetManager: Lost task 65821.0 in stage 13.0 (TID 70035, BJYFF3-Druid-19062.hadoop.jd.local, executor 681): FetchFailed(BlockManagerId(329, BJYF-Druid-176106.hadoop.jd.local, 6741, None), shuffleId=0, mapId=114, reduceId=21, message=

org.apache.spark.shuffle.FetchFailedException: java.io.FileNotFoundException: /data0/yarn1/local/usercache/mart\_jypt/appcache/application\_1495535432046\_117823/blockmgr-a3cfe5c5-99f1-4612-a471-e46360439fd6/17/shuffle\_0\_114\_0.index (No such file or directory)

at java.io.FileInputStream.open(Native Method)

at java.io.FileInputStream.<init>(FileInputStream.java:146)

at org.apache.spark.shuffle.IndexShuffleBlockResolver.getBlockData(IndexShuffleBlockResolver.scala:199)

at org.apache.spark.storage.BlockManager.getBlockData(BlockManager.scala:302)

at org.apache.spark.network.netty.NettyBlockRpcServer$$anonfun$2.apply(NettyBlockRpcServer.scala:60)

at org.apache.spark.network.netty.NettyBlockRpcServer$$anonfun$2.apply(NettyBlockRpcServer.scala:60)

at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)

at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)

at scala.collection.IndexedSeqOptimized$class.foreach(IndexedSeqOptimized.scala:33)

at scala.collection.mutable.ArrayOps$ofRef.foreach(ArrayOps.scala:186)

at scala.collection.TraversableLike$class.map(TraversableLike.scala:234)

at scala.collection.mutable.ArrayOps$ofRef.map(ArrayOps.scala:186)

at org.apache.spark.network.netty.NettyBlockRpcServer.receive(NettyBlockRpcServer.scala:60)

~~加大--executor-memory~~

<http://apache-spark-user-list.1001560.n3.nabble.com/FileNotFoundException-in-appcache-shuffle-files-td17605.html>

应该是加大 spark.yarn.executor.memoryOverhead

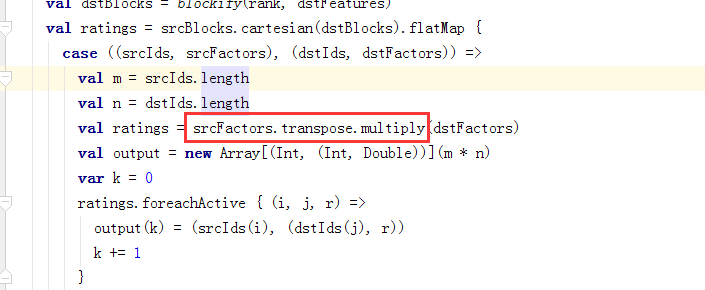
<http://thread.gmane.org/gmane.comp.lang.scala.spark.user/4768>

<http://www.cnblogs.com/LBSer/p/4129481.html>

1. ALS.train速度很快，124306948用户 \* 7061010商品在十分钟以内就能完成；

但是为用户推荐商品巨慢。。。。7个小时都跑不完。。。

原因是进行推荐时直接进行笛卡尔乘积，没有做任何优化：



网上有人提到这个问题：<http://blog.csdn.net/butterluo/article/details/48271361>

似乎他们搞了个扩展版Spark，进行了优化可惜没有公开。思路大致是：





（update:）已经有人给spark提PR来改进推荐过慢的问题了，不过是解决gc time过长的issue，也取得了不错的效果，希望早点merge进master。。。

<https://github.com/apache/spark/pull/17742>

1. 操作数据量较大的sql在reduce阶段报错：

Container killed by the ApplicationMaster.

Container killed on request. Exit code is 143

Container exited with a non-zero exit code 143

Too Many fetch failures.Failing the attempt

Reduce内存不够，加大：

SET mapreduce.reduce.memory.mb=8192;

<http://blog.csdn.net/yijichangkong/article/details/51332432>

1. string特征值映射成数字。某些模型中特征值是字符串，但模型本身只接收数字类型的输入（例如逻辑回归），这时就需要将字符串映射成数字，可以使用StringIndexer配合OneHotEncoder来处理。

[http://spark.apache.org/docs/latest/ml-features.html#stringindexer](http://spark.apache.org/docs/latest/ml-features.html" \l "stringindexer)

<http://dblab.xmu.edu.cn/blog/1297-2/>

<http://blog.csdn.net/zbc1090549839/article/details/50935274>

“StringIndexer可以把一个属性列里的值映射成数值类型。但是逻辑回归分类器默认数据数据是连续的，并且是有序的， 所以StringIndexer生成的数字， 还需要进一步处理。  这里用OneHotEncoder，独热编码即 One-Hot 编码，又称一位有效编码，其方法是使用N位状态寄存器来对N个状态进行编码，每个状态都由他独立的寄存器位，并且在任意时候，其中只有一位有效。

可以这样理解，对于每一个特征，如果它有m个可能值，那么经过独热编码后，就变成了m个二元特征。并且，这些特征互斥，每次只有一个激活。因此，数据会变成稀疏的。

这样做的好处主要有：

解决了分类器不好处理属性数据的问题

在一定程度上也起到了扩充特征的作用”

**注意**：分类变量会为特征的每一个值生成新的特征列，由此会导致特征数急剧膨胀。

例如，某模型原来的有171个特征:

|jd\_5903a35e9acd0|46699989|1 |(171,[0,1,2,3,4,5,6,7,8,9,12,13,14,15,16,17,18,19,20,21,22,23,24,28,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170],[999.0,50.0,0.05005005005005005,2000.0,122.0,0.061,18.0,0.15,102.0,0.85,16.0,49.0,110.0,1.0,6.0,13.0,3.0,-1.0,-1.0,55.0,179.0,208.6,208.6,4444.0,170.0,336103.0,1977.0764705882352,51221.0,301.3,0.19765992505894567,0.17735472265523441,5319.0,31.288235294117648,0.015825505871711946,0.04671231092436975,549.0,8.073529411764707,0.10335090361445783,0.4334889577523593,361.0,24.066666666666666,0.06795933734939759,0.8438924809930834,4402.0,49.46067415730337,0.8286897590361446,0.8861164455948687,69.0,182.0,378.0,4.0,21.0,57.0])

当我们将字符串类型的收货地址字段“u\_cpp\_addr\_county”使用分类变量进行处理后，特征数变为了2995：

|jd\_563bb132f613f|44953945|1 |(2995,[0,1,2,3,4,5,6,7,8,9,12,13,14,15,16,17,18,19,20,21,22,23,24,28,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,363],[50.0,2.0,0.04,1000000.0,5340.0,0.00534,2526.0,0.47312230754823,2813.0,0.52687769245177,73.0,295.0,737.0,33.0,157.0,374.0,3.0,-1.0,-1.0,87.0,224.9,520.9,459.5,4444.0,30.0,9709979.0,323665.9666666667,217.0,7.233333333333333,0.0830781010719755,0.0968499212598276,33741.0,1124.7,0.0034748787819211557,0.045108972746412575,10970.0,522.3809523809524,0.325191201754906,0.19047683231706222,9050.0,1810.0,0.2682753305270647,0.5751051075799795,13714.0,527.4615384615385,0.4065334677180293,0.8509408070554538,122.0,495.0,1199.0,42.0,194.0,460.0,1.0])

因此，在进行分类变量处理时要特别注意，取值很多的字段最好不要进行分类变量处理，否则特征数过多spark运行时非常耗资源非常慢，甚至根本无法运行报奇奇怪怪的错误，例如：

17/09/22 11:31:23 WARN TaskSetManager: Lost task 73.0 in stage 44.0 (TID 20475, BJLFRZ-10k-39-99.hadoop.jd.local, executor 16): ExecutorLostFailure (executor 16 exited caused by one of the running tasks) Reason: Container marked as failed: container\_e02\_1505878256677\_275564\_01\_000020 on host: BJLFRZ-10k-39-99.hadoop.jd.local. Exit status: 134. Diagnostics: Exception from container-launch.

Container id: container\_e02\_1505878256677\_275564\_01\_000020

Exit code: 134

Stack trace: org.apache.hadoop.yarn.server.nodemanager.containermanager.runtime.ContainerExecutionException: Launch container failed

at org.apache.hadoop.yarn.server.nodemanager.containermanager.linux.runtime.DefaultLinuxContainerRuntime.launchContainer(DefaultLinuxContainerRuntime.java:111)

at org.apache.hadoop.yarn.server.nodemanager.containermanager.linux.runtime.DelegatingLinuxContainerRuntime.launchContainer(DelegatingLinuxContainerRuntime.java:102)

at org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor.launchContainer(LinuxContainerExecutor.java:368)

at org.apache.hadoop.yarn.server.nodemanager.containermanager.launcher.ContainerLaunch.call(ContainerLaunch.java:307)

at org.apache.hadoop.yarn.server.nodemanager.containermanager.launcher.ContainerLaunch.call(ContainerLaunch.java:81)

at java.util.concurrent.FutureTask.run(FutureTask.java:266)

at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)

at java.lang.Thread.run(Thread.java:745)

Shell output: main : command provided 1

main : run as user is yarn

main : requested yarn user is mart\_srd

Getting exit code file...

Creating script paths...

Writing pid file...

Writing to tmp file /data1/yarn1/local/nmPrivate/application\_1505878256677\_275564/container\_e02\_1505878256677\_275564\_01\_000020/container\_e02\_1505878256677\_275564\_01\_000020.pid.tmp

Writing to cgroup task files...

Creating local dirs...

Launching container...

Getting exit code file...

Creating script paths...

Container exited with a non-zero exit code 134

（我们可以通过StringIndexerModel.labels.length来查看分类变量处理之后的特征数，然后酌情选取需要处理的字段）

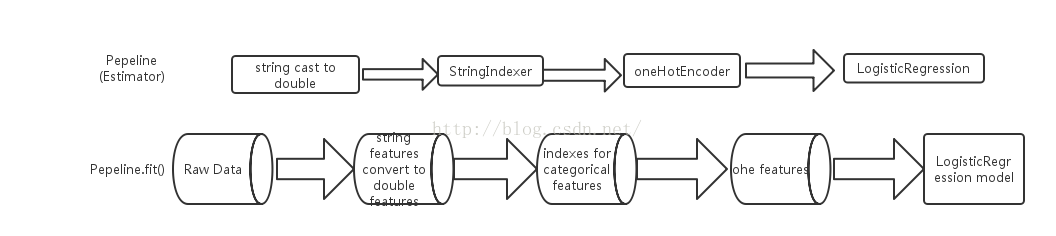
还有，如果使用StringIndexer处理多个字段，应该保证训练和预测时处理顺序是一致的，否则数据会有问题而不报错，最终预测的概率全为1

1. spark ml应尽量使用pipeline。

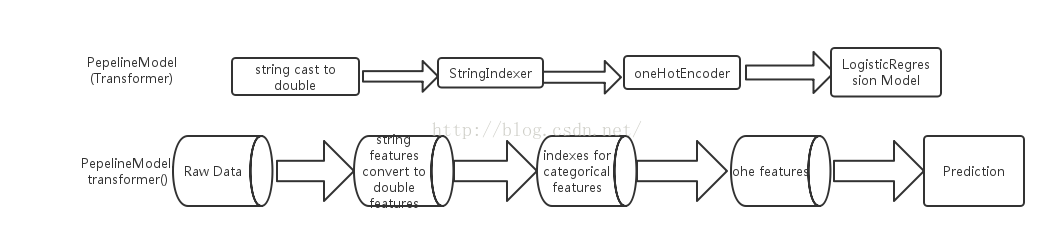
<http://spark.apache.org/docs/2.1.0/ml-pipeline.html>

<https://vimsky.com/article/2049.html> (翻译)

训练阶段：



预测阶段：



1. 启动时报错：

org.apache.spark.rpc.RpcTimeoutException: Futures timed out after [120 seconds]. This timeout is controlled by spark.rpc.askTimeout

分配的core不够, 多分配几核的CPU

20. spark任务运行慢，查看监控发现stage的timeline显示有很多executor不断的退出，然后增加新的executor，所以导致很慢。



查看退出的executor的日志发现都正常，只有一行日志比较奇怪：

18/05/17 13:34:22 ERROR CoarseGrainedExecutorBackend: RECEIVED SIGNAL TERM

根据网上的反馈，有人说是executor内存不够，于是减少每个executor的core，增加executor的内存

[https://mail-archives.apache.org/mod\_mbox/spark-user/201507.mbox/%3C55A372C5.9050801@googlemail.com%3E](https://mail-archives.apache.org/mod_mbox/spark-user/201507.mbox/<55A372C5.9050801@googlemail.com>)

有可能还需要增加 spark.yarn.executor.memoryOverhead

**PS:** 后来证明任务跑的慢另有原因，是在map时使用了一个巨大的dict，每个task都要从driver上把这个dict变量给拉取到executor上，大量时间都浪费在这上面了。上图中每个stage之间都有一段长时间的空白，就是在传输这个变量。知道原因就好办，将该dict广播到各executor就可以了，这样每个task直接使用executor中的变量，而不需要从driver拉取，果然就快了很多。

22. join产生重复列。

下面的写法会产生两个名为sentence的列:

df1 = df1.join(df\_embeddings, df1.sentence == df\_embeddings.sentence, **"left"**)

使用如下方式避免：

df1 = df1.join(df\_embeddings, "sentence", "left")

https://docs.databricks.com/spark/latest/faq/join-two-dataframes-duplicated-column.html

23.应尽量使用reduceByKey取代groupBy+map,因为reduceByKey会先在本地计算然后在交换数据，能大大减少需要交换的数据量。

数据量大的情况下，下面第一种写法非常耗资源，甚至不可能完成任务：

rdd = rdd.groupBy(lambda x: x.md5\_s1)\

.map(lambda x: (x[0], sorted(x[1], key=lambda x: x.scene\_similarity, reverse=True)[:2]))\

.flatMap(lambda x: x[1])

rdd = rdd.reduceByKey(add)\

.map(lambda x:(x[0],sorted(x[1], key=lambda x:(x.scene\_similarity, x.sentence\_similarity, x.join\_similarity), reverse=True)[:2]))\

.flatMap(lambda x: x[1])