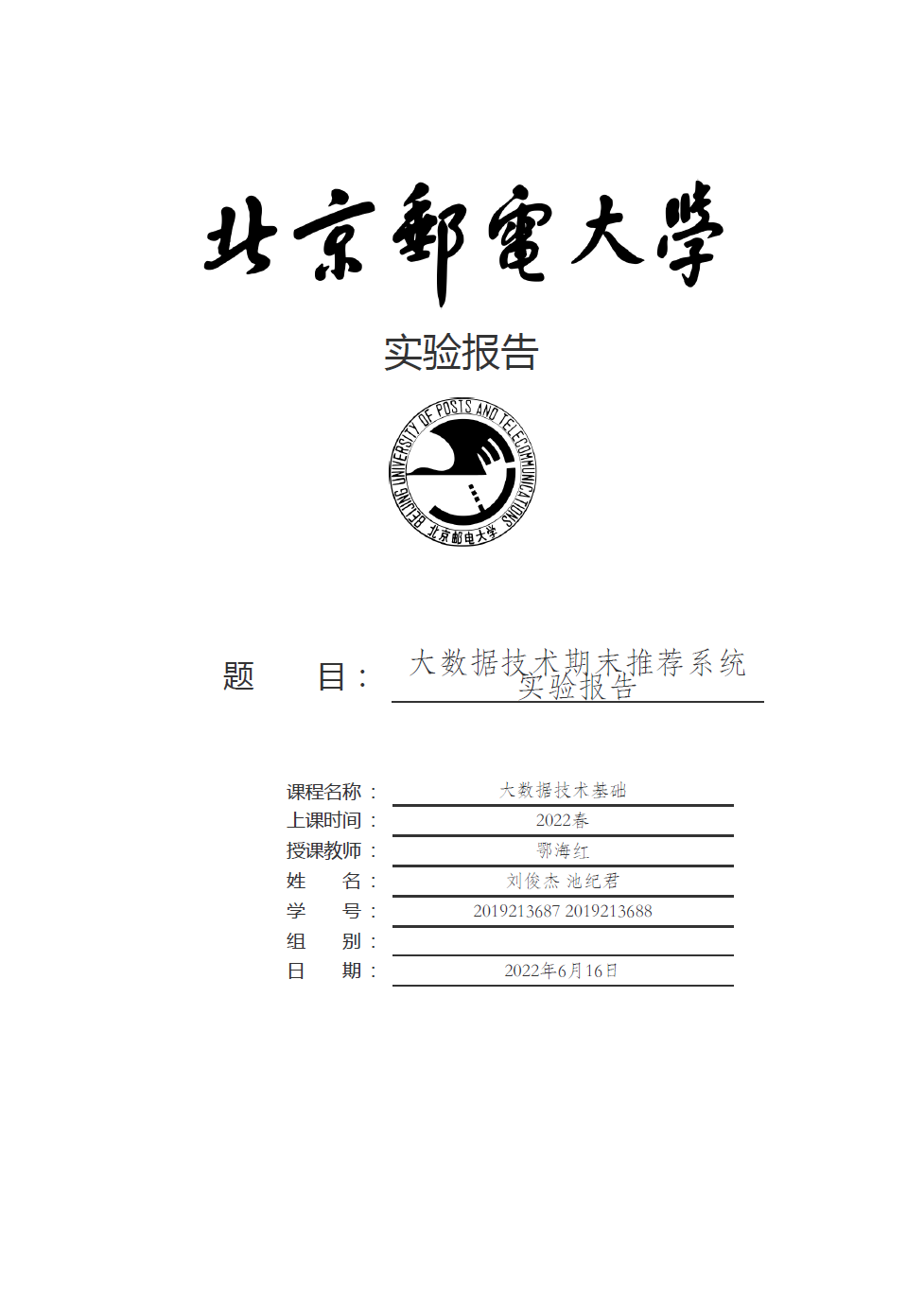
**大数据技术期末推荐系统实验报告**



**1 实验目的**

掌握Lambda架构

**2 实验平台**

操作系统：CentOS 7.6 64bit

**3 实验内容**

**3.1 使用Flume、Kafka实现数据的收集**

本次实验将略去Flume组件。启动一个Kafka服务器，并创建名为“movie\_rating\_records”的Topic。

**3.2 使用HBase实现原始数据的存储**

在HBase中创建一个表并命名为“movie\_records”，数据格式在附加说明（2）部分。编写一个Kafka消费者，定期从“movie\_rating\_records”中获取数据并写入HBase。

**3.3 使用Redis作为缓存数据库实现特征/模型参数的存储**

在本次实验中，将使用Redis作为批式、流式、服务等组件之间的数据交换中间件。例如，将批式计算的得到的特征写入Redis中，等待模型训练时再从Redis读取特征。

**3.4 使用MapReduce或Spark实现定时启动的批式计算任务**

使用MapReduce/Spark启动定时任务，定时时间为5分钟（5*60*1000ms），计算以下两部分内容：①历史特征的计算②CTR预测模型的训练

**3.5 使用Spark Streaming或Flink实现流式计算任务**

不同于批式计算，流式计算直接从Kafka中消费“movie\_rating\_records”Topic，定时时间为30秒（30\*1000ms）并完成以下两部分内容的计算：①实时特征的计算②实时TopK的计算

**3.6 实现服务应答**

服务端根据客户端的请求，从Redis中读取多种召回方式得到的候选列表、批式特征、流式特征、CTR模型参数，根据CTR模型参数与特征，为列表中每个电影打分，并将评分最高的TopK部电影返回给客户端。

**3.7 （选做内容）提升推荐系统的推荐效果**

1. 增加更多的召回：使用多种方式进行召回，如spark.mllib中的协同过滤模型，对每个userId产生单独的召回列表。
2. 使用更多特征：进一步从数据中进行挖掘，例如使用上电影的年份等信息。

**3.8 （提高部分）**

1. 调研现有的大数据软件架构，并撰写调研报告/综述（需要附上引用）
2. 搭建新的架构（包括但不限于上面提到kappa架构和delta架构）来完成本次实验（推荐系统）

**4 实验步骤**

**4.1 安装Kafka**

**4.1.1 解压kafka**

|  |
| --- |
| Plaintext tar -zxvf kafka\_2.11-0.10.2.2.tgz |

**4.1.2 将解压得到的文件夹移到/home/modules目录下**

|  |
| --- |
| Plaintext mv kafka\_2.11-0.10.2.2 /home/modules/ |

**4.1.3 编辑config/server.properties文件**

编辑config/server.properties文件，修改delete.topic.enable和zookeeper.connect

|  |
| --- |
| Plaintext vim /home/modules/kafka\_2.11-0.10.2.2/config/server.properties delete.topic.enable=ture zookeeper.connect=cjj-2019213688-0001:2181,cjj-2019213688-0002:2181,cjj-2019213688-0003:2181,cjj-2019213688-0004:2181 |

**4.1.4 将kafka文件夹通过scp发送到其余结点对应目录下**

|  |
| --- |
| Plaintext scp -r /home/modules/kafka\_2.11-0.10.2.2 root@cjj-2019213688-0002:/home/modules/ scp -r /home/modules/kafka\_2.11-0.10.2.2 root@cjj-2019213688-0003:/home/modules/ scp -r /home/modules/kafka\_2.11-0.10.2.2 root@cjj-2019213688-0004:/home/modules/ |

**4.1.5 编辑config/server.properties文件**

编辑config/server.properties文件，修改broker.id分别为1、2、3、4（各节点不一样）

|  |
| --- |
| Plaintext vim /home/modules/kafka\_2.11-0.10.2.2/config/server.properties broker.id=4 |

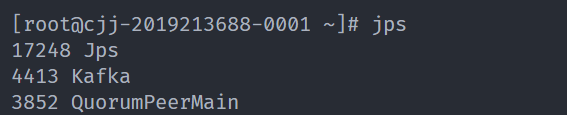
**4.1.6 各节点启动zookeeper**

|  |
| --- |
| Plaintext /usr/local/zookeeper/bin/zkServer.sh start |

**4.1.7 各节点启动kafka**

|  |
| --- |
| Plaintext /home/modules/kafka\_2.11-0.10.2.2/bin/kafka-server-start.sh /home/modules/kafka\_2.11-0.10.2.2/config/server.properties |

**4.1.8 jps确认kafka启动成功（10分）**



**4.2 安装Redis（单机部署）**

**4.2.1 升级gcc**

|  |
| --- |
| Plaintext yum -y install centos-release-scl yum -y install devtoolset-9-gcc devtoolset-9-gcc-c++ devtoolset-9-binutils devtoolset-9-libatomic-devel scl enable devtoolset-9 bash |

**4.2.2 解压redis**

|  |
| --- |
| Plaintext tar -zxvf redis-6.0.6.tar.gz |

**4.2.3 进入解压得到的文件夹，编译，安装**

|  |
| --- |
| Plaintext cd redis-6.0.6 make make install |

**4.2.4 修改redis.conf**

修改redis.conf，设置redis为守护进程，并允许远程连接，关闭保护模式

|  |
| --- |
| Plaintext vim /root/redis-6.0.6/redis.conf daemonize yes #bind 127.0.0.1 protected-mode no requirepass Kd7J16@6d2NWM9znN4$V |

**4.2.5 启动redis**

|  |
| --- |
| Plaintext redis-server redis-6.0.6/redis.conf |

**4.2.6 ps确认redis运行成功（10分）**



**4.3 安装python3**

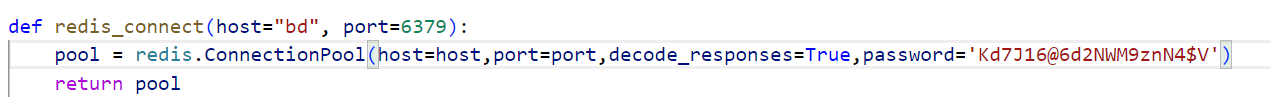
|  |
| --- |
| Plaintext yum –y install python3 |

注：后续使用python3解释器需要在命令行使用python3而非python，安装一些库需要用pip3而不是pip

**4.4 打jar包**

pom.xml和3个scala代码文件均已提供

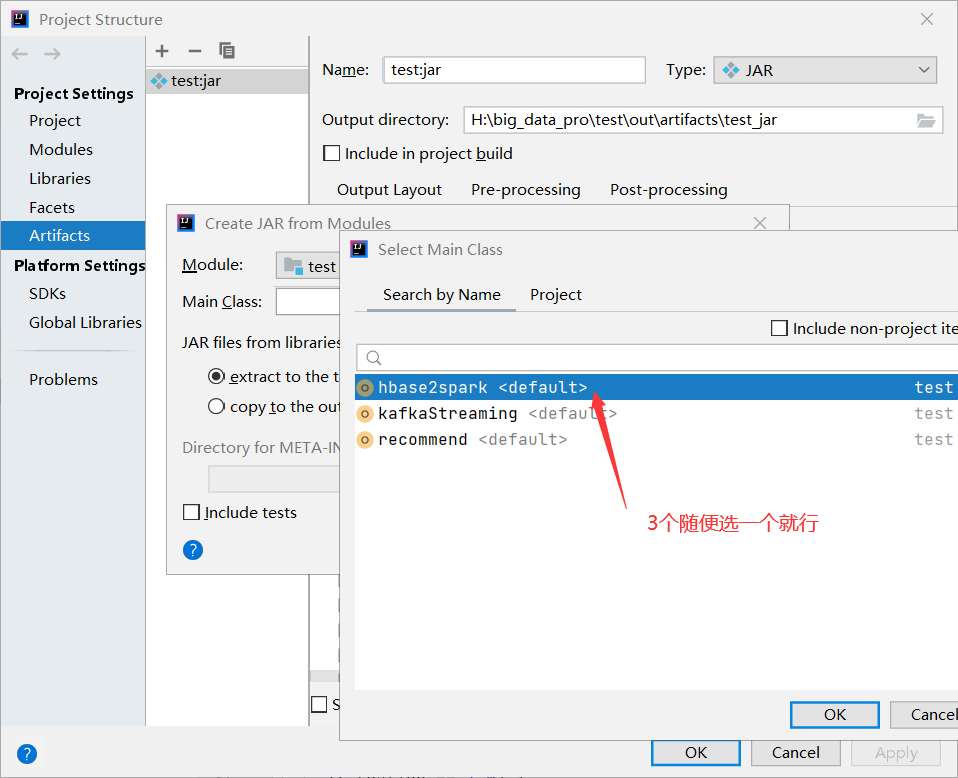
修改redis相关的python文件，增加参数password



修改 redis相关的scala代码，在每次建立redis连接后，进行认证



打jar包的过程和之前的实验类似，需要注意的是，Main Class随便选一个就可以了



**4.5 运行简易推荐系统**

**4.5.1 代码说明**

1. load\_train\_ratings\_hbase.py：将json\_train\_ratings.json作为历史数据直接存储到HBase中
2. load\_movie\_redis.py：将movies.csv中存储的movieId与movie title的映射关系写入Redis中，用于后续推荐时从movieId转换成movie title
3. generatorRecord.py：将json\_test\_ratings.json作为实时数据发送到Kafka中
4. recommend\_server.py与recommend\_client.py：recommend\_server等待recommend\_client连接，连接后recommend\_client可以向recommend\_server发送一个userId，recommend\_server根据推荐算法从Redis中获取召回列表、特征参数与模型参数后，向recommend\_client返回推荐列表。向recommend\_client是一个交互式程序，输入用户id，会输出推荐结果。
5. hbase2spark.scala：批式计算程序，计算后将结果写入 Redis
6. kafkaStreaming.scala：流式计算程序，计算后将结果写入 Redis。
7. recommend.scala：推荐模型程序，计算后将结果写入 Redis

**4.5.2 运行流程**

|  |
| --- |
| \*表示在所有结点都需执行，init表示在初始化系统的时候执行，其余指令只需在主节点上执行。  下面出现的121.36.12.46统一修改为自己主节点的公网ip地址  node001统一修改为自己主节点名称  spark-sparkstreaming-recommend.jar统一修改为自己的jar包名称 |

**4.5.2.1 启动HDFS**

|  |
| --- |
| Plaintext start-all.sh |

**4.5.2.2 启动zookeeper（\*）**

|  |
| --- |
| Plaintext zkServer.sh start |

**4.5.2.3 启动HBase**

|  |
| --- |
| Plaintext start-hbase.sh |

**4.5.2.4 配置HBase Thrift连接，以便python中的happybase库能够连接Hbase**

|  |
| --- |
| Plaintext hbase-daemon.sh start thrift |

**4.5.2.5 在HBase中创建对应的表（init）**

|  |
| --- |
| Plaintext create 'movie\_records','details' |

查看HBase数据：

|  |
| --- |
| Plaintext scan 'movie\_records',{LIMIT=>5} |

查看movie\_records表的前5行

**4.5.2.6 启动load\_train\_ratings\_hbase.py（init）**

|  |
| --- |
| Plaintext python ./load\_train\_ratings\_hbase.py 121.36.12.46 9090 "movie\_records" "../../data/json\_train\_ratings.json" |

**4.5.2.7 启动redis**

|  |
| --- |
| Plaintext redis-server /root/redis-6.0.6/redis.conf |

查看redis数据：

进入redis

|  |
| --- |
| Plaintext redis-cli -h 127.0.0.1 -p 6379 |

查看redis中的key

|  |
| --- |
| Plaintext keys [pattern] |

e.g.: keys \*（查看所有的keys）

获取key对应的value

|  |
| --- |
| Plaintext get [key] |

e.g.: get 'movieId2movieTitle\_1'

查看redis中的表(list)

|  |
| --- |
| Plaintext lrange [表的名字] 0 -1 |

**4.5.2.8 启动load\_movie\_redis.py（init）**

可在本地执行

|  |
| --- |
| Plaintext `python .\load\_movie\_redis.py 121.36.88.159 6379 "../../data/movies.csv"` |

**4.5.2.9 启动Kafka（\*）**

|  |
| --- |
| Plaintext kafka-server-start.sh /home/modules/kafka/config/server.properties |

查看Kafka运行状态：Kafka默认端口为\*\*9092\*\*，可以使用命令：netstat -anlpt | grep 9092 或者 lsof -i:9092 来查看9092端口占用情况

**4.5.2.10 创建 Kafka Topic**

|  |
| --- |
| Plaintext kafka-topics.sh --zookeeper node001:2181 --create --topic movie\_rating\_records --partitions 1 --replication-factor 1 |

**4.5.2.11 启动generatorRecord.py（这个程序会一直运行，不需要等待停止）**

最好在服务器上运行，若要本地Windows/macOS运行，需额外配置kafka外网连接

|  |
| --- |
| Plaintext python3 /root/code/load/generatorRecord.py -h node001:9092 -f "/root/data/json\_test\_ratings.json" |

**4.5.2.12 启动hbase2spark、kafkaStreaming、recommend**

|  |
| --- |
| Plaintext spark-submit --class hbase2spark --master yarn --num-executors 3 --driver-memory 512m --executor-memory 512m --executor-cores 1 /root/spark-sparkstreaming-recommend.jar spark-submit --class kafkaStreaming --master yarn --num-executors 3 --driver-memory 512m --executor-memory 512m --executor-cores 1 /root/spark-sparkstreaming-recommend.jar spark-submit --class recommend --master yarn --num-executors 3 --driver-memory 512m --executor-memory 512m --executor-cores 1 /root/spark-sparkstreaming-recommend.jar |

delta:

|  |
| --- |
| Plaintext spark-submit --class kStream2delta --master yarn --num-executors 3 --driver-memory 512m --executor-memory 512m --executor-cores 1 /root/spark-sparkstreaming-recommend\_d.jar  --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.2.1 --packages io.delta:delta-core\_2.12:1.2.1 |

**4.5.2.13 启动recommend\_server.py**

可在本地执行

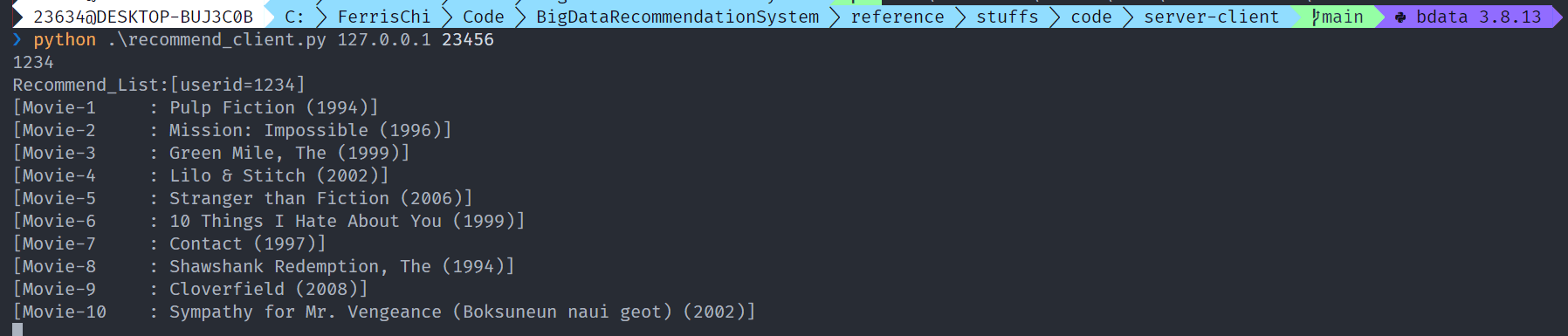
|  |
| --- |
| Plaintext `python code/server-client/recommend\_server.py "121.36.88.159" 6379 23456` |

**4.5.2.14 启动recommend\_client.py**

可在本地执行

|  |
| --- |
| Plaintext python code/server-client/recommend\_client.py 127.0.0.1 23456 |

在客户端输入userid，返回结果截图，返回结果可能与图片不一致（40分）



**4.6 选做内容2 提升推荐系统的推荐效果（20分）**

**4.6.1 增加更多的召回（10分）**

**4.6.1.1 需求说明**

使用多种方式进行召回，如spark.mllib中的协同过滤模型，或对每个userId产生单独的召回列表。

**4.6.1.2 算法说明**

协同过滤算法是最常用的推荐算法，其主要有两种具体形式：

1. 基于用户的推荐算法，通过考察具有相同爱好的用户对相同物品的评分标准进行计算。
2. 基于物品的推荐算法，考察具有相同特质的物品从而推荐给选择了某件物品的用户。

本次使用基于用户的推荐算法。

**4.6.1.3 实现说明**

1. 在hbase2spark.scala中，从hbase中读取所有历史数据data，将数据转换成MLLIb中ALS算法有固定的数据格式case class Rating(user： Int, product： Int,rating： Double)。然后使用ALS算法对数据训练模型model。最后对所有用户调用模型model的推荐函数得到推荐列表。

|  |
| --- |
| Plaintext //统计 协同过滤模型推荐列表 val ratingUserId5MovieId = data.map(x=>{  val rating = Rating(x.\_2.toInt,x.\_3.toInt,x.\_4.toDouble)  rating }) val rank = 2 //设置隐藏因子 val numIterations = 2 //设置迭代次数 val model = ALS.*train*(ratingUserId5MovieId, rank, numIterations, 0.01) //进行模型训练 val userIds = data.map(x => x.\_2).distinct().collect() //根据已有数据集建立协同过滤模型后用recommendProducts为用户推荐10个电影 *println*("Calculating CF for "+userIds.length.toString + " users...") val userIdCFMovieId = userIds.map(x => {  val products = model.recommendProducts(x, 10)  (x,products) }) *println*("Calculating done!") |

1. 将统计结果依次写入到redis

|  |
| --- |
| Plaintext // 依次输出统计结果 val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000) jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V") userIdCFMovieId.foreach(x=> {  jedisIns.del(s"userId\_perfer\_movieId\_**$**{x.\_1}")  for (i <- 0 until 10) {  jedisIns.rpush(s"userId\_perfer\_movieId\_**$**{x.\_1}", x.\_2(i).product.toString)  } }) jedisIns.close() |

1. 在recommend\_server.py中，召回给特定用户推荐的基于协同过滤模型的电影列表，和其他列表一起，经过回归模型的参数计算后推荐给用户。

|  |
| --- |
| Python  # (3)协同过滤模型的list  recall\_list += get\_CF\_recommend\_list(r,userId)  recall\_list = list(set(recall\_list)) # 去重  # print(recall\_list)  # 2. 排序  coefficients, intercept = get\_model\_arams(r)  recommend\_list = sort\_recall\_list\_by\_model(r,userId,recall\_list,coefficients,intercept)   # 发送推荐列表 |

1. 函数get\_CF\_recommend\_list查询redis中userId\_perfer\_movieId\_相关项目并返回。

|  |
| --- |
| Python def get\_CF\_recommend\_list(r,userId):  # 得到协同过滤模型推荐列表的list  CF\_recommend\_list = r.lrange(f"userId\_perfer\_movieId\_{userId}",0,-1)  return [int(x) for x in CF\_recommend\_list] |

**4.6.2 使用更多特征（10分）**

**4.6.2.1 需求说明**

进一步从数据中进行挖掘，例如使用上电影的年份等信息。

**4.6.2.2 算法说明**

我们将电影年份的信息加入Redis缓存数据库中，并引入CTR模型中进行模型参数计算。

**4.6.2.3 实现实现**

1. 更改load\_movie\_redis.py，将电影年份信息统一上传至Redis中的movieId2movieYear中。

|  |
| --- |
| Python # add movie year r.delete(f"movieId2movieYear\_{nowl[0]}") r.set(f"movieId2movieYear\_{nowl[0]}", nowl[3]) |

1. 在Recommend.scala中，将电影年份信息视为线性回归CTR模型特征中的一维，参与回归计算。

|  |
| --- |
| Scala // add movie year to feature val yearkey = s"movieId2movieYear\_${movieId}" features = features :+ {  if (jedis.exists(yearkey)) {  jedis.get(yearkey).toDouble  } else  0  } |

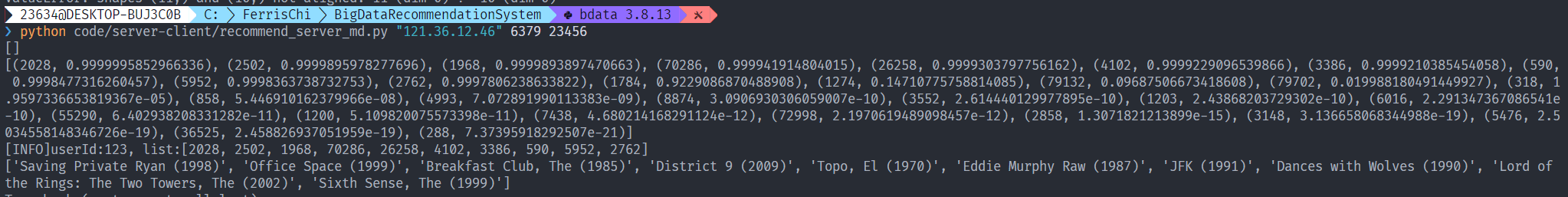
**4.6.3 结果展示**

* 客户端结果



说明：对用户ID为123的用户和ID为219的用户进行电影推荐。客户端将服务端的推荐结果返回。

* 服务端结果



说明：服务端收集客户端的推荐请求，生成找回列表，包括popular\_list、liking\_genre\_list和CF\_recommend\_list。其中命令行的打印值为CF\_recommend\_list，即协同过滤模型的推荐列表。可以看到，用户123并没有进行过评分，没有相关推荐，而用户219由于拥有历史评分数据，协同滤波系统为其推荐了ID为2028, 2502, 1968, 70286, 26258, 4102, 3386, 590, 5952, 2762的电影。在此之后，服务端在Redis数据库中查找相关特征和CTR模型参数，对电影列表进行排序并返回前十个电影。

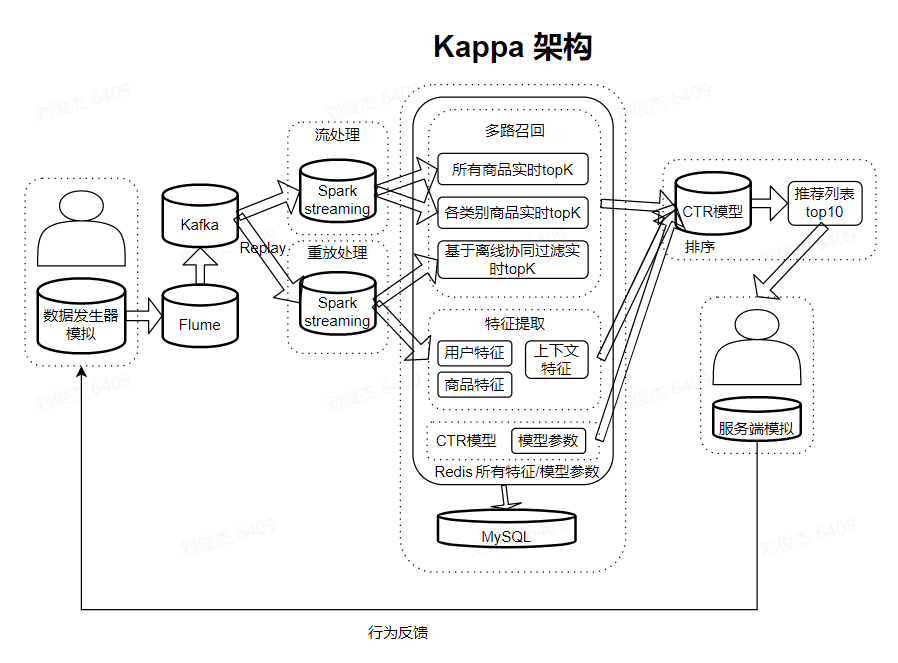
**4.7 提高部分 大数据架构研究与搭建**

**4.7.1 调研现有的大数据软件架构，撰写调研报告/综述（需要附上引用）（5分）**

**[大数据架构调研.pdf]**

**4.7.2 搭建新的kappa架构来完成本次实验（推荐系统）（15分）**

**4.7.2.1 架构图**

**4.7.2.2 Kappa框架介绍**

* Kappa架构是真正意义上的流批一体的处理方式。它是随着流处理引擎的逐步完善后，由LinkedIn公司提出的一种实时数仓架构。Kappa 架构完全废弃了批处理层，即便是批处理也是通过消息队列回溯以流计算的⽅式来完成，这样就实现了批流⼀体，从⽽避免了冗余⼯作，并简化了技术栈。其核⼼思想是通过改进流计算系统来解决数据全量处理的问题，使得实时计算和批处理过程使⽤同⼀套代码。
* 此外Kappa架构认为只有在有必要的时候才会对历史数据进⾏重复计算，⽽如果需要重复计算时，Kappa架构下可以启动很多个实例进⾏重复计算,⽅式是通过上游重放完成(从数据源拉取数据重新计算)。

**4.7.2.3 架构说明**

1. 总体思想：利用Kafka架构的天然流处理特性和与Spark Streaming的结合，对大数据管理系统进行去批处理化设计。
2. 系统说明：
3. 用户数据通过Flume收集（实际省去）
4. 用户数据由Flume传输至Kafka的某一特定Topic中。
5. 流处理部件对Kafka的位于某一Topic中的最新流数据进行实时处理，主要包括：计算所有商品实时TopK、各类别商品试试TopK
6. 重放处理部件对Kafka同一Topic中的流数据进行消息重放，进行历史数据特征计算，主要包括：用户特征、商品特征、上下文特征等
7. 缓存数据库Redis对所有的特征进行集群的缓存，供外部推荐服务器和内部CTR模型等调用。其底层为MySQL等数据库。
8. 服务器收到客户端请求后，请求访问Redis数据库，获取多路召回列表、电影特征和CTR模型参数等信息，对电影排序，并返回Top10的推荐列表。
9. 客户端对推荐列表中电影的点击行为被反馈到数据发生器中，形成系统闭环。

**4.7.2.4 实现要点**

1. 在系统运行前，我们先将历史评分，即json\_train\_ratings.json文件上传至Kafka Topic中，以供重放处理部件使用。
2. 在设置重放部件时，将读取Topic的offset值设置为earliest，以保证可以重放所有历史评分记录。

**4.7.2.5 结果展示**

由于改进框架后，客户端和服务端信息没有变动，故不做展示。重放功能部件的部分输出如下：



**5 实验总结**

**5.1 池纪君总结**

1. 在完成本次实验的过程中，我对在课堂上讲过的kafka、hbase、redis等大数据组件的理论知识有了更切身的实践体会。
2. 本次综合实验的完成过程中，我对前几次实验内容进行了回顾，对大数据架构认识更加深刻。
3. 在完成过程中，由于没有配置Redis安全认证，导致服务器被黑，不仅引发了中间人攻击，还导致服务器IP被封，必须全部重装系统才能恢复。这次教训让我对网络安全的重要性有了清醒的认识。
4. 本学期前没有接触过scala代码，在选做任务中也因为语法问题和对RDD不熟悉（例如.collect()操作）导致耗时较多。
5. 对现代的大数据架构，例如Delta做了一些调研和体验。尽管最后的提高任务并没有选用delta架构，但是在华为云服务器上（被黑之前）进行了kafka stream读写数据湖的实验，由于目前社区资源较少，对delta认识不足，最后放弃。

**5.2 刘俊杰总结**

1. 通过完成本次大作业，更深入学习了kafka、hbase、redis、Spark等大数据技术理论，并将理论知识应用于实践，有了更深入的理解和体会。
2. 由于对配置环境和一些操作仍然不够熟悉，导致完成基础实验的过程中出现了一些错误。
3. 由于对批处理和流处理的一些知识不太熟，在协同过滤中涉及RDD的一些操作出现了一点问题。
4. 对现代的大数据架构，例如Delta架构做了一些调研和尝试，但是由于学习材料较少，没有办法在时间限制内实现。最后找到了替代的Kapper架构，这个架构比较容易从原有架构中修改得到，最后实现了Kapper架构。

**6 参考**

1. [spark MLlib机器学习实战之协同过滤算法](https://zhuanlan.zhihu.com/p/400530786)
2. [Spark Streaming整合Kafka的两种方式\_kingloneye的博客-CSDN博客](https://blog.csdn.net/weixin_38887752/article/details/106907280)
3. Armbrust M, Das T, Sun L, et al. Delta lake: high-performance ACID table storage over cloud object stores[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 3411-3424.
4. [数据仓库架构的发展进程 - 墨天轮](https://www.modb.pro/db/325012)

**7 源程序**

**7.1 客户端/服务器端**

**recommend\_client.py**

|  |
| --- |
| Python # 作用：模拟客户端，向server\_host:server\_port建立连接并请求推荐列表 # 启动参数：两个 # - server\_host # - server\_port import socket import sys import pickle  def getArgs():  argv = sys.argv[1:]  return argv  def init\_socket(host,port):  client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  client\_socket.connect((host,port))  return client\_socket  def send\_data(cs, datas):  byteStream = pickle.dumps(datas)  length = len(byteStream)  byteStream = bytes(f"{length:<16}", 'utf-8')+byteStream  cs.sendall(byteStream)  def recv\_data(cs):  msg = cs.recv(1024)  length = int(msg[:16])  full\_msg = b''  full\_msg += msg[16:]  nowsize = len(full\_msg)  while nowsize < length:  more = cs.recv(length - nowsize)  full\_msg = full\_msg + more  nowsize += len(more)  return pickle.loads(full\_msg)  if \_\_name\_\_=="\_\_main\_\_":  argv = getArgs()  client\_socket = init\_socket(argv[0],int(argv[1]))  while True:  data = input().strip()  send\_data(client\_socket, data)  response = recv\_data(client\_socket)  print(f"Recommend\_List:[userid={data}]")  for i, moview\_name in enumerate(response):  print(f"[Movie-{i+1:<6}: {moview\_name}]") |

**recommend\_server.py**

|  |
| --- |
| Python # 作用：模拟服务端，连接redis，并监听端口等待客户端的连接 # 1. 请求redis中"popular\_movie\_all" # 2. 请求redis中的特征 # 2. 请求redis中"movieId2movieTitle\_xxx" # 启动参数：三个 # - redis\_host # - redis\_port # - listen\_port:监听端口 import json import socket import redis import sys import pickle import numpy as np from math import e  def redis\_connect(host="bd", port=6379):   pool = redis.ConnectionPool(host=host,port=port,decode\_responses=True, password='Kd7Jdddd16@6djie8gce342NWM9znN4$V')  return pool  def getArgs():  argv = sys.argv[1:]  return argv  def init\_socket(port):  server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  server\_socket.bind(("",port))  server\_socket.listen(128)  return server\_socket  def send\_data(cs, datas):  byteStream = pickle.dumps(datas)  length = len(byteStream)  byteStream = bytes(f"{length:<16}", 'utf-8')+byteStream  cs.sendall(byteStream)  def recv\_data(cs):  msg = cs.recv(1024)  length = int(msg[:16])  full\_msg = b''  full\_msg += msg[16:]  nowsize = len(full\_msg)  while nowsize < length:  more = cs.recv(length - nowsize)  full\_msg = full\_msg + more  nowsize += len(more)  return pickle.loads(full\_msg)  def get\_popular\_list(r):  popular\_list = r.lrange("popular\_movies\_all",0,-1)  return [int(x) for x in popular\_list]  def get\_liking\_genre\_list(r,userId):  # 得到最喜欢  genre\_Id,best\_rating = 0, 0.0  for i in range(19):  value = r.get(f"streaming2feature\_userId\_to\_genresId\_{userId}\_{i}")  if not value:  continue  value = float(value)  if value>best\_rating:  genre\_Id, best\_rating = i, value  #   liking\_genre\_list = r.lrange(f"popular\_movies\_genreId\_{genre\_Id}",0,-1)  # 再求最喜欢的list  return [int(x) for x in liking\_genre\_list]  def get\_CF\_recommend\_list(r,userId):  # 得到协同过滤模型推荐列表的list  CF\_recommend\_list = r.lrange(f"userId\_perfer\_movieId\_{userId}",0,-1)  return [int(x) for x in CF\_recommend\_list]  def get\_model\_arams(redis):  coefficients = redis.lrange("params\_coefficients",0,-1)  coefficients = [float(x) for x in coefficients]  coefficients = np.array(coefficients)  intercept = float(redis.get("params\_intercept"))  return coefficients,intercept  def get\_features\_userId(r, userId):  user\_features = dict()  value = r.get(f"batch2feature\_userId\_rating1\_{userId}")  user\_features["batch2feature\_userId\_rating1"] = float(value) if value else 0  value = r.get(f"batch2feature\_userId\_rating0\_{userId}")  user\_features["batch2feature\_userId\_rating0"] = float(value) if value else 0  value = r.get(f"streaming2feature\_userId\_rating1\_{userId}")  user\_features["streaming2feature\_userId\_rating1"] = float(value) if value else 0  value = r.get(f"streaming2feature\_userId\_rating0\_{userId}")  user\_features["streaming2feature\_userId\_rating0"] = float(value) if value else 0  return user\_features  def get\_features\_movieId(r, userId, movieId):  movie\_features = dict()  value = r.get(f"batch2feature\_movieId\_rating1\_{movieId}")  movie\_features["batch2feature\_movieId\_rating1"] = float(value) if value else 0  value = r.get(f"batch2feature\_movieId\_rating0\_{movieId}")  movie\_features["batch2feature\_movieId\_rating0"] = float(value) if value else 0  value = r.get(f"movieId2movieYear\_{movieId}")  movie\_features['batch2feature\_movieId\_year'] = float(value) if value else 0   value = r.get(f"streaming2feature\_movieId\_rating1\_{movieId}")  movie\_features["streaming2feature\_movieId\_rating1"] = float(value) if value else 0  value = r.get(f"streaming2feature\_movieId\_rating0\_{movieId}")  movie\_features["streaming2feature\_movieId\_rating0"] = float(value) if value else 0  return movie\_features  def get\_userId2movieId(r,userId,movieId):  userId\_movieId\_features = dict()  liking\_genre\_list = r.lrange(f"movie2genres\_movieId\_{movieId}",0,-1)  sum = 0.0  for liking\_genrdId in liking\_genre\_list:  value = r.get(f"batch2feature\_userId\_to\_genresId\_{userId}\_{liking\_genrdId}")  sum += float(value) if value else 0  userId\_movieId\_features[f'batch2feature\_userId\_to\_movieId'] = sum  sum = 0   for liking\_genrdId in liking\_genre\_list:  value = r.get(f"batch2feature\_userId\_to\_genresId\_{userId}\_{liking\_genrdId}")  sum += float(value) if value else 0  userId\_movieId\_features[f'streaming2feature\_userId\_to\_movieId'] = sum  return userId\_movieId\_features  def sort\_recall\_list\_by\_model(r,userId,recalls,coefficients,intercept):  user\_features = get\_features\_userId(r, userId)  rating = []  coefficients = np.array(coefficients)  for movieId in recalls:   movie\_features = get\_features\_movieId(r, userId, movieId)  user\_movie\_features = get\_userId2movieId(r,userId,movieId)  # print(f"movie\_features={movieId}:{movie\_features},{user\_movie\_features}")  # features = [0 for i in range(10)]  features = [0 for i in range(11)]  # 批式：5维  features[0],features[1] = user\_features['batch2feature\_userId\_rating1'],user\_features['batch2feature\_userId\_rating0']  features[2],features[3] = movie\_features['batch2feature\_movieId\_rating1'],movie\_features['batch2feature\_movieId\_rating0']  features[4] = user\_movie\_features[f'batch2feature\_userId\_to\_movieId']  # 实时：5维  features[5],features[6] = user\_features['streaming2feature\_userId\_rating1'],user\_features['streaming2feature\_userId\_rating1']  features[7],features[8] = movie\_features['streaming2feature\_movieId\_rating1'],movie\_features['streaming2feature\_movieId\_rating0']  features[9] = user\_movie\_features[f'streaming2feature\_userId\_to\_movieId']  # 新增：1维（年份）  features[10] = movie\_features['batch2feature\_movieId\_year']  features = np.array(features)  # print(features)  # 模型计算  pred = np.dot(features, coefficients) + intercept  rating.append((movieId,e\*\* (pred) / (1. + e\*\* (pred))))  rating.sort(key=lambda x:x[1],reverse=True)  print(rating)  return [x[0]for x in rating[:10]]  def get\_recommend\_list(redis,userId):  # 1. 召回  recall\_list = []  # （1）载入popular\_list  recall\_list += get\_popular\_list(r)  # （2）载入最喜欢的list  recall\_list += get\_liking\_genre\_list(r,userId)  # (3)协同过滤模型的list  recall\_list += get\_CF\_recommend\_list(r,userId)  recall\_list = list(set(recall\_list)) # 去重  # print(recall\_list)  # 2. 排序  coefficients, intercept = get\_model\_arams(r)  recommend\_list = sort\_recall\_list\_by\_model(r,userId,recall\_list,coefficients,intercept)   # 发送推荐列表  print(f"[INFO]userId:{userId}, list:{recommend\_list}")  print([r.get(f"movieId2movieTitle\_{x}") for x in recommend\_list])  return [r.get(f"movieId2movieTitle\_{x}") for x in recommend\_list]   if \_\_name\_\_=="\_\_main\_\_":  argv = getArgs()  pool = redis\_connect(argv[0],argv[1])  r = redis.Redis(connection\_pool=pool)  # ----Test-------  # userId = 51  # print(get\_recommend\_list(r,userId))  # ---------------  server\_socket = init\_socket(int(argv[2]))  client\_socket, client\_address = server\_socket.accept()  while True:  r = redis.Redis(connection\_pool=pool)  userId = recv\_data(client\_socket)  response = get\_recommend\_list(r,userId)  send\_data(client\_socket,response) |

**7.2 选做**

**hbase2spark.scala**

|  |
| --- |
| Python import breeze.linalg.Broadcaster import org.apache.hadoop.hbase.mapreduce.TableInputFormat import org.apache.hadoop.hbase.util.Bytes import org.apache.hadoop.conf.Configuration import org.apache.hadoop.hbase.{HBaseConfiguration, HTableDescriptor, TableName} import org.apache.hadoop.hbase.client.{HBaseAdmin, HTable} import org.apache.log4j.{Level, Logger} import org.apache.spark import org.apache.spark.\_ import org.apache.spark.broadcast.Broadcast import org.apache.spark.{SparkConf, SparkContext} import org.apache.spark.mllib.recommendation.{ALS, Rating} import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig}  import java.lang.Thread.sleep import java.text.SimpleDateFormat import java.util.Date  class hbase2spark{}  object hbase2spark {  def getHBaseConfiguration(quorum:String, port:String) = {  val conf = HBaseConfiguration.create()  conf.set("hbase.zookeeper.quorum",quorum)  conf.set("hbase.zookeeper.property.clientPort",port)  conf  }  def isEqual(x:(Int,Float), y:Float):List[(Int, Int)] = {  if (x.\_2==y)  List((x.\_1,1))  else  List()  }  def batch2feature(sc:SparkContext) {  val hbaseconf = getHBaseConfiguration("ljj-2019213687-0001","2181")  hbaseconf.set(TableInputFormat.INPUT\_TABLE,"movie\_records")  // HBase数据转成RDD  val hBaseRDD = sc.newAPIHadoopRDD(hbaseconf,classOf[TableInputFormat],  classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],  classOf[org.apache.hadoop.hbase.client.Result]).cache()   // RDD数据操作  val data = hBaseRDD.map(x => {  val result = x.\_2  val key = Bytes.toString(result.getRow)  val rating = Bytes.toString(result.getValue("details".getBytes,"rating".getBytes)).toFloat  val userId = Bytes.toString(result.getValue("details".getBytes,"userId".getBytes)).toInt  val movieId = Bytes.toString(result.getValue("details".getBytes,"movieId".getBytes)).toInt  val timestamp = Bytes.toString(result.getValue("details".getBytes,"timestamp".getBytes))  (key,userId,movieId,rating,timestamp)  })   //统计 a)用户历史正反馈次数  val counterUserIdPos = data.flatMap(x => isEqual((x.\_2,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 b)用户历史负反馈次数  val counterUserIdNeg = data.flatMap(x => isEqual((x.\_2,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 c)电影历史正反馈次数  val counterMovieIdPos = data.flatMap(x => isEqual((x.\_3,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 d)电影历史负反馈次数  val counterMovieIdNeg = data.flatMap(x => isEqual((x.\_3,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)   //统计 e)用户历史点击该分类比例  val counterUserId2MovieId = data.filter(x=>x.\_4==1.0)  .map(x=>(x.\_2,x.\_3))  .groupByKey()  .flatMapValues(x=>{  var sum = 0  val one\_hot: Array[Int] = new Array[Int](19)  // val jedisIns = new JedisIns("bd",6379,100000)  val jedisIns:Jedis = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  for (record<-x) {  sum=sum+1  val genres\_list = jedisIns.lrange("movie2genres\_movieId\_" + record.toString,0,-1)  val it = genres\_list.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  one\_hot(genresId) = one\_hot(genresId)+1  }  }  jedisIns.close()  var counter:List[(Int,Float)] = List()  for (i<-one\_hot.indices) {  if (one\_hot(i)>0) counter = counter :+ (i,one\_hot(i).toFloat/sum)  }  counter  })   //统计 协同过滤模型推荐列表  val ratingUserId5MovieId = data.map(x=>{  val rating = Rating(x.\_2.toInt,x.\_3.toInt,x.\_4.toDouble)  rating  })  val rank = 2 //设置隐藏因子  val numIterations = 2 //设置迭代次数  val model = ALS.train(ratingUserId5MovieId, rank, numIterations, 0.01) //进行模型训练  val userIds = data.map(x => x.\_2).distinct().collect()  //根据已有数据集建立协同过滤模型后用recommendProducts为用户推荐10个电影  println("Calculating CF for "+userIds.length.toString + " users...")  val userIdCFMovieId = userIds.map(x => {  val products = model.recommendProducts(x, 10)  (x,products)  })  println("Calculating done!")   // 依次输出统计结果  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  userIdCFMovieId.foreach(x=> {  jedisIns.del(s"userId\_perfer\_movieId\_${x.\_1}")  for (i <- 0 until 10) {  jedisIns.rpush(s"userId\_perfer\_movieId\_${x.\_1}", x.\_2(i).product.toString)  }  })  jedisIns.close()   counterUserIdPos.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_userId\_rating1\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterUserIdNeg.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_userId\_rating0\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterMovieIdPos.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_movieId\_rating1\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterMovieIdNeg.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_movieId\_rating0\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterUserId2MovieId.foreach(x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set(s"batch2feature\_userId\_to\_genresId\_${x.\_1.toString}\_${x.\_2.\_1}", x.\_2.\_2.toString)  jedisIns.close()  })  }  def NowDate(): String = {  val now: Date = new Date()  val dateFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")  val date = dateFormat.format(now)  date  }  def main(args: Array[String]): Unit = {  // Spark  // val jedisIns = new JedisIns("bd",6379,100000)  // jedisIns.testJedis()  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)  while (true) {  println(s"${NowDate()} [INFO] Begin to calculate batch features")  val sparkConf = new SparkConf().setAppName("HBaseReadTest").setMaster("local[2]")  val sc = new SparkContext(sparkConf)  // val broadcast: Broadcast[JedisPoolUtil] = sc.broadcast(jedis\_pool)  batch2feature(sc)  sc.stop()  println(s"${NowDate()} [INFO] Success!")  sleep(1000\*60\*5)  }  } } |

**kafkaStreaming.scala**

|  |
| --- |
| Python import com.alibaba.fastjson.serializer.SerializerFeature import com.alibaba.fastjson.{JSON, JSONObject} import org.apache.hadoop.hbase.{HBaseConfiguration, TableName} import org.apache.hadoop.hbase.client.{ConnectionFactory, Put} import org.apache.kafka.clients.consumer.ConsumerRecord import org.apache.kafka.common.serialization.StringDeserializer import org.apache.log4j.{Level, Logger} import org.apache.spark.SparkConf import org.apache.spark.streaming.{Durations, StreamingContext} import org.apache.spark.streaming.api.java.{JavaDStream, JavaInputDStream, JavaStreamingContext} import org.apache.spark.streaming.dstream.DStream import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe import org.apache.spark.streaming.kafka010.\_ import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig} import org.apache.hadoop.hbase.util.Bytes import org.apache.spark.broadcast.Broadcast  import java.util import collection.JavaConverters.\_  class kafkaStreaming {}  object kafkaStreaming {  def getSc() = {  val sparkConf = new SparkConf()  sparkConf.setMaster("local[2]").setAppName("kafkaConsumer")  val streamingContext = new StreamingContext(sparkConf, Durations.seconds(30))  streamingContext  }  def isEqual(x:(Int,Float), y:Float):List[(Int, Int)] = {  if (x.\_2==y)  List((x.\_1,1))  else  List()  }  def getStream(sc:StreamingContext) = {  val kafkaParams = Map[String, Object] (  "bootstrap.servers" -> "ljj-2019213687-0001:9092",  "key.deserializer" -> classOf[StringDeserializer],  "value.deserializer" -> classOf[StringDeserializer],  "group.id" -> "kafkaStreaming",  "auto.offset.reset" -> "latest",  "enable.auto.commit" -> (false: java.lang.Boolean),  "partition.assignment.strategy"->"org.apache.kafka.clients.consumer.RangeAssignor"  )  val topics = Array("movie\_rating\_records")  val stream = KafkaUtils.createDirectStream[String,String](  sc,  PreferConsistent,  Subscribe[String, String](topics, kafkaParams)  )  stream  }  def streamingCore(sc: StreamingContext): Unit = {  val stream = kafkaStreaming.getStream(sc).map(x=>{  val json: JSONObject = JSON.parseObject(x.value())  (x.key(),json.get("userId").toString.toInt,json.get("movieId").toString.toInt,json.get("rating").toString.toFloat,json.get("timestamp").toString)  })   //缓存最近的记录,写入redis && hbase  stream.foreachRDD( rdd =>  rdd.foreachPartition { part => {  //redis connection  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  //hbase connection  val config = HBaseConfiguration.create  config.set("hbase.zookeeper.property.clientPort", "2181")  config.set("hbase.zookeeper.quorum", "ljj-2019213687-0001")  val connection = ConnectionFactory.createConnection(config)  val table = connection.getTable(TableName.valueOf("movie\_records"))  var records: List[Put] = List()  part.foreach { x => {  //写入redis  while (jedisIns.llen("streaming\_records") >= 200) jedisIns.lpop("streaming\_records")  val record: Map[String, String] = Map("userId" -> x.\_2.toString,  "movieId" -> x.\_3.toString,  "rating" -> x.\_4.toString,  "timestamp" ->x.\_5.toString)  // println(record)  // println(JSON.toJSONString(record.asJava,SerializerFeature.WriteMapNullValue))  jedisIns.rpush("streaming\_records", JSON.toJSONString(record.asJava, SerializerFeature.WriteMapNullValue))  //写入hbase  val put = new Put(Bytes.toBytes(x.\_1))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("userId"),Bytes.toBytes(record("userId")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("movieId"),Bytes.toBytes(record("movieId")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("rating"),Bytes.toBytes(record("rating")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("timestamp"),Bytes.toBytes(record("timestamp")))  records = records :+ put  }  import scala.collection.JavaConverters.\_  table.put(records.asJava)  table.close()  jedisIns.close()  }  }  }  )  //全局topK统计  stream.map(x=>(x.\_3,1))  .reduceByKey((x,y)=>(x+y))  .foreachRDD(line=> {  line.sortBy(x => x.\_2, ascending = false).take(10).foreach(  x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  while (jedisIns.llen("popular\_movies\_all")>=30) jedisIns.lpop("popular\_movies\_all")  jedisIns.rpush("popular\_movies\_all",x.\_1.toString)  jedisIns.close()  }  )  }  )  //热点topK统计  stream.flatMap(x=> {  var seqList: Seq[(Int, (Int, Int))] = Seq()  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  val genresList = jedisIns.lrange(s"movie2genres\_movieId\_${x.\_3}",0,-1)  val it = genresList.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  seqList = seqList :+ (genresId,(x.\_3,1))  }  jedisIns.close()  seqList  }).groupByKey()  //写入redis  .mapValues(records=>{  val answers: Array[(Int, Int)] = new Array[(Int, Int)](11)  var len = 0  for(record <- records) {  answers(len) = record  var i = len  while (i>0 && answers(i-1).\_2<answers(i).\_2) {  val tmp = answers(i-1)  answers(i-1) = answers(i)  answers(i) = tmp  i-=1  }  len+=1  if (len==11){  len = 10  }  }  (len,answers)  })  .foreachRDD(  rdd => rdd.foreach{  x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.del(s"popular\_movies\_genreId\_${x.\_1}")  for (i <- 0 until x.\_2.\_1) {  jedisIns.rpush(s"popular\_movies\_genreId\_${x.\_1}", x.\_2.\_2(i).\_1.toString)  }  jedisIns.close()  }  }  )  //  //特征抽取  //统计 a)用户历史正反馈次数  val counterUserIdPos = stream.flatMap(x => isEqual((x.\_2,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 b)用户历史负反馈次数  val counterUserIdNeg = stream.flatMap(x => isEqual((x.\_2,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 c)电影历史正反馈次数  val counterMovieIdPos = stream.flatMap(x => isEqual((x.\_3,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 d)电影历史负反馈次数  val counterMovieIdNeg = stream.flatMap(x => isEqual((x.\_3,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 e)用户历史点击该分类比例  val counterUserId2MovieId = stream.filter(x=>x.\_4==1.0)  .map(x=>(x.\_2,x.\_3))  .groupByKey()  .flatMapValues(x=>{  var sum = 0  val one\_hot: Array[Int] = new Array[Int](19)  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  for (record<-x) {  sum=sum+1  val genres\_list = jedisIns.lrange("movie2genres\_movieId\_" + record.toString,0,-1)  val it = genres\_list.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  one\_hot(genresId) = one\_hot(genresId)+1  }  }  var counter:List[(Int,Float)] = List()  for (i<-one\_hot.indices) {  if (one\_hot(i)>0) counter = counter :+ (i,one\_hot(i).toFloat/sum)  }  jedisIns.close()  counter  })  counterUserIdPos.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_userId\_rating1\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterUserIdNeg.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_userId\_rating0\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterMovieIdPos.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_movieId\_rating1\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterMovieIdNeg.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_movieId\_rating0\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterUserId2MovieId.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set(s"streaming2feature\_userId\_to\_genresId\_${x.\_1.toString}\_${x.\_2.\_1}", x.\_2.\_2.toString)  jedisIns.close()  }}  )  }  val redis\_host:String = "ljj-2019213687-0001"  val redis\_port:Int = 6379  val redis\_timeout:Int = 10000  def main(args:Array[String]) = {  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)  val sc:StreamingContext = kafkaStreaming.getSc()  streamingCore(sc)  sc.start()  sc.awaitTermination()  } } |

**recommend.scala**

|  |
| --- |
| Python  import com.alibaba.fastjson import com.alibaba.fastjson.JSON.parseObject import com.alibaba.fastjson.{JSON, JSONObject} import org.apache.kafka.clients.consumer.KafkaConsumer import org.apache.kafka.common.TopicPartition import org.apache.log4j.{Level, Logger} import org.apache.spark import org.apache.spark.{SparkConf, SparkContext} import org.apache.spark.api.java.{JavaRDD, JavaSparkContext} import org.apache.spark.ml.classification.LogisticRegression import org.apache.spark.ml.linalg.{Vector, Vectors} import org.apache.spark.sql.{Row,DataFrame, SparkSession} import org.apache.spark.streaming.{Seconds, StreamingContext} import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig}  import java.io.{BufferedWriter, FileWriter} import java.lang.Thread.sleep import java.text.SimpleDateFormat import java.time.Duration import java.util.{Collections, Date, Properties} import scala.collection.mutable import scala.concurrent.duration.Duration  class recommend {}  object recommend {  def add\_record(map:mutable.Map[String,Int],jedis: Jedis,json:JSONObject) = {  // println(json)  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toFloat.toInt  if (rating == 1.0) {  //统计 a)用户历史正反馈次数  var key = s"counteruserId1\_${userId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 c)电影历史正反馈次数  key = s"countermovieId1\_${movieId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 e)用户历史点击该分类比例  key = s"counteruserId2movie\_${userId}\_${movieId}"  val genres = jedis.lrange(s"movie2genres\_movieId\_${movieId}",0,-1)  val it = genres.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  val key2 = s"counteruserId2genre\_${userId}\_${genresId}"  map += key2 -> (map.getOrElseUpdate(key2, 0)+1)  }  }  else {  //统计 b)用户历史负反馈次数  var key = "counteruserId0\_" + userId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 d)电影历史负反馈次数  key = "countermovieId0\_" + movieId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  }  map  }  def delete\_record(map:mutable.Map[String,Int],jedis: Jedis,json:JSONObject) = {  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toFloat.toInt  if (rating == 1.0) {  //统计 a)用户历史正反馈次数  var key = s"counteruserId1\_${userId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 c)电影历史正反馈次数  key = s"countermovieId1\_${movieId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 e)用户历史点击该分类比例  key = s"counteruserId2movie\_${userId}\_${movieId}"  val genres = jedis.lrange(s"movie2genres\_movieId\_${movieId}",0,-1)  val it = genres.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  val key2 = s"counteruserId2genre\_${userId}\_${genresId}"  map += key2 -> (map.getOrElseUpdate(key2, 0)-1)  }  }  else {  //统计 b)用户历史负反馈次数  var key = "counteruserId0\_" + userId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 d)电影历史负反馈次数  key = "countermovieId0\_" + movieId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  }  map  }  // def getRedisList(jedis: Jedis,key:String) = {  // val value = jedis.lrange(key,0,-1)  // jedis.close()  // value  // }  def collect\_train\_data() = {  val jedis = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedis.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  val records\_length = jedis.llen("streaming\_records")  var map: mutable.Map[String, Int] = mutable.Map()  for(i <- 0 to records\_length.toInt/2) {  val json = JSON.parseObject(jedis.lindex("streaming\_records",i))  map = add\_record(map,jedis,json)  }  var data:Seq[(Double,Vector)] = Seq()  for(i <- records\_length.toInt / 2 + 1 until records\_length.toInt) {  var json = JSON.parseObject(jedis.lindex("streaming\_records",i-records\_length.toInt/2))  map = delete\_record(map,jedis,json)  json = JSON.parseObject(jedis.lindex("streaming\_records",i))  map = add\_record(map,jedis,json)  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toDouble  var features:Seq[Double] = Seq()  for (key<-Array(  s"batch2feature\_userId\_rating1\_${userId}",  s"batch2feature\_userId\_rating0\_${userId}",  s"batch2feature\_movieId\_rating1\_${movieId}",  s"batch2feature\_movieId\_rating0\_${movieId}")) {  features = features :+ {  if (jedis.exists(key)) {  jedis.get(key).toDouble  } else  0  }  }  var sum:Double = 0.0  val key = s"batch2feature\_userId\_to\_genresId\_${userId}"  for(i <- 0 to 19) {  sum = sum + {  if (jedis.exists(s"${key}\_${i}"))  jedis.get(s"${key}\_${i}").toDouble  else  0  }  }  features = features :+ sum  for (key<-Array(  s"counteruserId1\_${userId}",  s"counteruserId0\_${userId}",  s"countermovieId1\_${movieId}",  s"countermovieId0\_${movieId}")) {  features = features :+ map.getOrElseUpdate(key,0).toDouble  }  val counteruserIdsum = map.getOrElseUpdate(s"counteruserId1\_${userId}",0) + map.getOrElseUpdate(s"counteruserId0\_${userId}",0)  //简单完成一个加和即可  sum = 0.0  for(i <- 0 to 19) {  sum = sum + {  val value = map.getOrElseUpdate(s"counteruserId2genre\_${userId}\_${i}",0).toDouble  if (value==0)  0.toDouble  else  value/counteruserIdsum.toDouble  }  }  features = features :+ sum   val yearkey = s"movieId2movieYear\_${movieId}"  features = features :+ {  if (jedis.exists(yearkey)) {  jedis.get(yearkey).toDouble  } else  0  }   data = data :+ (rating, Vectors.dense(features.toArray))  }  jedis.close()  data  }   def trainRecommendModel(spark:SparkSession) = {  val train\_data = collect\_train\_data()  val train\_dataframe = spark.createDataFrame(train\_data).toDF("label","features")  train\_dataframe.show(20,truncate = false)  import spark.implicits.\_  // val training = spark.read.format("libsvm").load("./sample\_libsvm\_data.txt")  val lr = new LogisticRegression()  .setMaxIter(20)   // Fit the model  val lrModel = lr.fit(train\_dataframe)  // val trainingSummary = lrModel.binarySummary  // val objectiveHistory = trainingSummary.objectiveHistory  // println("objectiveHistory:")  // objectiveHistory.foreach(loss => println(loss))  // println(s"areaUnderROC: ${trainingSummary.areaUnderROC}")   val params\_coefficients = lrModel.coefficients.toDense.toArray  println(params\_coefficients.mkString("Array(", ", ", ")"))  val jedis = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedis.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedis.del("params\_coefficients")  for (i<- params\_coefficients.indices)  jedis.rpush("params\_coefficients",params\_coefficients(i).toString)  jedis.set("params\_intercept",lrModel.intercept.toString)   }  def NowDate(): String = {  val now: Date = new Date()  val dateFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")  val date = dateFormat.format(now)  date  }  val redis\_host:String = "ljj-2019213687-0001"  val redis\_port:Int = 6379  val redis\_timeout:Int = 10000  def main(args:Array[String]) = {  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)  while (true) {  print(s"${NowDate()} [INFO] Begin to train lr model")  val spark = SparkSession.builder().master("local[2]").appName("ALSExample").getOrCreate()  trainRecommendModel(spark)  spark.close()  print(s"${NowDate()} [INFO] Success!")  sleep(1000\*60\*5)  }  } } |

**7.2 Kappa**

**kafkareplay2spark.scala**

|  |
| --- |
| Python import breeze.linalg.Broadcaster import com.alibaba.fastjson.{JSON, JSONObject} import kafkaStreaming.streamingCore import org.apache.hadoop.hbase.mapreduce.TableInputFormat import org.apache.hadoop.hbase.util.Bytes import org.apache.hadoop.conf.Configuration import org.apache.hadoop.hbase.{HBaseConfiguration, HTableDescriptor, TableName} import org.apache.hadoop.hbase.client.{HBaseAdmin, HTable} import org.apache.kafka.common.serialization.StringDeserializer import org.apache.log4j.{Level, Logger} import org.apache.spark import org.apache.spark.\_ import org.apache.spark.broadcast.Broadcast import org.apache.spark.{SparkConf, SparkContext} import org.apache.spark.mllib.recommendation.{ALS, Rating} import org.apache.spark.rdd.RDD import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe import org.apache.spark.streaming.kafka010.KafkaUtils import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent import org.apache.spark.streaming.{Durations, StreamingContext} import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig}  import java.lang.Thread.sleep import java.text.SimpleDateFormat import java.util.Date  class hbase2spark{}  object hbase2spark {  def getHBaseConfiguration(quorum:String, port:String) = {  val conf = HBaseConfiguration.create()  conf.set("hbase.zookeeper.quorum",quorum)  conf.set("hbase.zookeeper.property.clientPort",port)  conf  }  def isEqual(x:(Int,Float), y:Float):List[(Int, Int)] = {  if (x.\_2==y)  List((x.\_1,1))  else  List()  } // def getSc() = { // val sparkConf = new SparkConf() // sparkConf.setMaster("local[2]").setAppName("kafkaConsumer") // val streamingContext = new StreamingContext(sparkConf, Durations.seconds(30)) // streamingContext // }  def getStream(sc:StreamingContext) = {  val kafkaParams = Map[String, Object] (  "bootstrap.servers" -> "ljj-2019213687-0001:9092",  "key.deserializer" -> classOf[StringDeserializer],  "value.deserializer" -> classOf[StringDeserializer],  "group.id" -> "Hbase2spark",  "auto.offset.reset" -> "earliest",  "enable.auto.commit" -> (false: java.lang.Boolean),  "partition.assignment.strategy"->"org.apache.kafka.clients.consumer.RangeAssignor"  )  val topics = Array("movie\_rating\_records")  val stream = KafkaUtils.createDirectStream[String,String](  sc,  PreferConsistent,  Subscribe[String, String](topics, kafkaParams)  )  stream  } // sc:SparkContext,  def batch2feature(sc:SparkContext, sc1:StreamingContext) { // val hbaseconf = getHBaseConfiguration("ljj-2019213687-0001","2181") // hbaseconf.set(TableInputFormat.INPUT\_TABLE,"movie\_records") // // HBase数据转成RDD // val hBaseRDD = sc.newAPIHadoopRDD(hbaseconf,classOf[TableInputFormat], // classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable], // classOf[org.apache.hadoop.hbase.client.Result]).cache()   // RDD数据操作 // var data = hBaseRDD.map(x => { // val result = x.\_2 // val key = Bytes.toString(result.getRow) // val rating = Bytes.toString(result.getValue("details".getBytes,"rating".getBytes)).toFloat // val userId = Bytes.toString(result.getValue("details".getBytes,"userId".getBytes)).toInt // val movieId = Bytes.toString(result.getValue("details".getBytes,"movieId".getBytes)).toInt // val timestamp = Bytes.toString(result.getValue("details".getBytes,"timestamp".getBytes)) // (key,userId,movieId,rating,timestamp) // })   println("getting stream data")  val stream = hbase2spark.getStream(sc1).map(x=>{  val json: JSONObject = JSON.parseObject(x.value())  (x.key(),json.get("userId").toString.toInt,json.get("movieId").toString.toInt,json.get("rating").toString.toFloat,json.get("timestamp").toString)  })  // var data = sc.makeRDD(Array(("", 0, 0, (0.0).toFloat, "")))   println("Before union") // println(data)  stream.foreachRDD(data => {  println(data.count()) // data = data.union(rdd)   //统计 a)用户历史正反馈次数  val counterUserIdPos = data.flatMap(x => isEqual((x.\_2,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 b)用户历史负反馈次数  val counterUserIdNeg = data.flatMap(x => isEqual((x.\_2,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 c)电影历史正反馈次数  val counterMovieIdPos = data.flatMap(x => isEqual((x.\_3,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 d)电影历史负反馈次数  val counterMovieIdNeg = data.flatMap(x => isEqual((x.\_3,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)   //统计 e)用户历史点击该分类比例  val counterUserId2MovieId = data.filter(x=>x.\_4==1.0)  .map(x=>(x.\_2,x.\_3))  .groupByKey()  .flatMapValues(x=>{  var sum = 0  val one\_hot: Array[Int] = new Array[Int](19)  // val jedisIns = new JedisIns("bd",6379,100000)  val jedisIns:Jedis = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  for (record<-x) {  sum=sum+1  val genres\_list = jedisIns.lrange("movie2genres\_movieId\_" + record.toString,0,-1)  val it = genres\_list.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  one\_hot(genresId) = one\_hot(genresId)+1  }  }  jedisIns.close()  var counter:List[(Int,Float)] = List()  for (i<-one\_hot.indices) {  if (one\_hot(i)>0) counter = counter :+ (i,one\_hot(i).toFloat/sum)  }  counter  })   //统计 协同过滤模型推荐列表  val ratingUserId5MovieId = data.map(x=>{  val rating = Rating(x.\_2.toInt,x.\_3.toInt,x.\_4.toDouble)  rating  })  val rank = 2 //设置隐藏因子  val numIterations = 2 //设置迭代次数  val model = ALS.train(ratingUserId5MovieId, rank, numIterations, 0.01) //进行模型训练  val userIds = data.map(x => x.\_2).distinct().collect()  //根据已有数据集建立协同过滤模型后用recommendProducts为用户推荐10个电影  println("Calculating CF for "+userIds.length.toString + " users...")  val userIdCFMovieId = userIds.map(x => {  val products = model.recommendProducts(x, 10)  (x,products)  })  println("Calculating done!")   // 依次输出统计结果  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  userIdCFMovieId.foreach(x=> {  jedisIns.del(s"userId\_perfer\_movieId\_${x.\_1}")  for (i <- 0 until 10) {  jedisIns.rpush(s"userId\_perfer\_movieId\_${x.\_1}", x.\_2(i).product.toString)  }  })  jedisIns.close()   counterUserIdPos.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_userId\_rating1\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterUserIdNeg.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_userId\_rating0\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterMovieIdPos.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_movieId\_rating1\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterMovieIdNeg.foreach( x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("batch2feature\_movieId\_rating0\_"+x.\_1.toString, x.\_2.toString)  jedisIns.close()  })  counterUserId2MovieId.foreach(x=> {  val jedisIns = new Jedis("ljj-2019213687-0001",6379,100000)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set(s"batch2feature\_userId\_to\_genresId\_${x.\_1.toString}\_${x.\_2.\_1}", x.\_2.\_2.toString)  jedisIns.close()  })  }) // println("After union") // println(data)   println("done")   }  def NowDate(): String = {  val now: Date = new Date()  val dateFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")  val date = dateFormat.format(now)  date  }    def main(args: Array[String]): Unit = {  // Spark  Logger.*getLogger*("org.apache.spark").setLevel(Level.*WARN*)   while (true) {  *println*(s"**$**{*NowDate*()} [INFO] Begin to calculate batch features")  val sparkConf = new SparkConf().setAppName("HBaseReadTest").setMaster("local[2]")  val sc = new SparkContext(sparkConf)  val sc1:StreamingContext = new StreamingContext(sc, Durations.*seconds*(30))  *batch2feature*(sc, sc1)  sc1.start()  sc1.awaitTermination()  sc1.stop()  *println*(s"**$**{*NowDate*()} [INFO] Success!")  }  } } |

**kafkaStreaming.scala**

|  |
| --- |
| Python import com.alibaba.fastjson.serializer.SerializerFeature import com.alibaba.fastjson.{JSON, JSONObject} import org.apache.hadoop.hbase.{HBaseConfiguration, TableName} import org.apache.hadoop.hbase.client.{ConnectionFactory, Put} import org.apache.kafka.clients.consumer.ConsumerRecord import org.apache.kafka.common.serialization.StringDeserializer import org.apache.log4j.{Level, Logger} import org.apache.spark.SparkConf import org.apache.spark.streaming.{Durations, StreamingContext} import org.apache.spark.streaming.api.java.{JavaDStream, JavaInputDStream, JavaStreamingContext} import org.apache.spark.streaming.dstream.DStream import org.apache.spark.streaming.kafka010.ConsumerStrategies.Subscribe import org.apache.spark.streaming.kafka010.\_ import org.apache.spark.streaming.kafka010.LocationStrategies.PreferConsistent import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig} import org.apache.hadoop.hbase.util.Bytes import org.apache.spark.broadcast.Broadcast  import java.util import collection.JavaConverters.\_  class kafkaStreaming {}  object kafkaStreaming {  def getSc() = {  val sparkConf = new SparkConf()  sparkConf.setMaster("local[2]").setAppName("kafkaConsumer")  val streamingContext = new StreamingContext(sparkConf, Durations.seconds(30))  streamingContext  }  def isEqual(x:(Int,Float), y:Float):List[(Int, Int)] = {  if (x.\_2==y)  List((x.\_1,1))  else  List()  }  def getStream(sc:StreamingContext) = {  val kafkaParams = Map[String, Object] (  "bootstrap.servers" -> "ljj-2019213687-0001:9092",  "key.deserializer" -> classOf[StringDeserializer],  "value.deserializer" -> classOf[StringDeserializer],  "group.id" -> "kafkaStreaming",  "auto.offset.reset" -> "latest",  "enable.auto.commit" -> (false: java.lang.Boolean),  "partition.assignment.strategy"->"org.apache.kafka.clients.consumer.RangeAssignor"  )  val topics = Array("movie\_rating\_records")  val stream = KafkaUtils.createDirectStream[String,String](  sc,  PreferConsistent,  Subscribe[String, String](topics, kafkaParams)  )  stream  }  def streamingCore(sc: StreamingContext): Unit = {  val stream = kafkaStreaming.getStream(sc).map(x=>{  val json: JSONObject = JSON.parseObject(x.value())  (x.key(),json.get("userId").toString.toInt,json.get("movieId").toString.toInt,json.get("rating").toString.toFloat,json.get("timestamp").toString)  })   //缓存最近的记录,写入redis && hbase  stream.foreachRDD( rdd =>  rdd.foreachPartition { part => {  //redis connection  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  //hbase connection  val config = HBaseConfiguration.create  config.set("hbase.zookeeper.property.clientPort", "2181")  config.set("hbase.zookeeper.quorum", "ljj-2019213687-0001")  val connection = ConnectionFactory.createConnection(config)  val table = connection.getTable(TableName.valueOf("movie\_records"))  var records: List[Put] = List()  part.foreach { x => {  //写入redis  while (jedisIns.llen("streaming\_records") >= 200) jedisIns.lpop("streaming\_records")  val record: Map[String, String] = Map("userId" -> x.\_2.toString,  "movieId" -> x.\_3.toString,  "rating" -> x.\_4.toString,  "timestamp" ->x.\_5.toString)  // println(record)  // println(JSON.toJSONString(record.asJava,SerializerFeature.WriteMapNullValue))  jedisIns.rpush("streaming\_records", JSON.toJSONString(record.asJava, SerializerFeature.WriteMapNullValue))  //写入hbase  val put = new Put(Bytes.toBytes(x.\_1))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("userId"),Bytes.toBytes(record("userId")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("movieId"),Bytes.toBytes(record("movieId")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("rating"),Bytes.toBytes(record("rating")))  put.addColumn(Bytes.toBytes("details"),Bytes.toBytes("timestamp"),Bytes.toBytes(record("timestamp")))  records = records :+ put  }  import scala.collection.JavaConverters.\_ // table.put(records.asJava)  table.close()  jedisIns.close()  }  }  }  )  //全局topK统计  stream.map(x=>(x.\_3,1))  .reduceByKey((x,y)=>(x+y))  .foreachRDD(line=> {  line.sortBy(x => x.\_2, ascending = false).take(10).foreach(  x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  while (jedisIns.llen("popular\_movies\_all")>=30) jedisIns.lpop("popular\_movies\_all")  jedisIns.rpush("popular\_movies\_all",x.\_1.toString)  jedisIns.close()  }  )  }  )  //热点topK统计  stream.flatMap(x=> {  var seqList: Seq[(Int, (Int, Int))] = Seq()  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  val genresList = jedisIns.lrange(s"movie2genres\_movieId\_${x.\_3}",0,-1)  val it = genresList.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  seqList = seqList :+ (genresId,(x.\_3,1))  }  jedisIns.close()  seqList  }).groupByKey()  //写入redis  .mapValues(records=>{  val answers: Array[(Int, Int)] = new Array[(Int, Int)](11)  var len = 0  for(record <- records) {  answers(len) = record  var i = len  while (i>0 && answers(i-1).\_2<answers(i).\_2) {  val tmp = answers(i-1)  answers(i-1) = answers(i)  answers(i) = tmp  i-=1  }  len+=1  if (len==11){  len = 10  }  }  (len,answers)  })  .foreachRDD(  rdd => rdd.foreach{  x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.del(s"popular\_movies\_genreId\_${x.\_1}")  for (i <- 0 until x.\_2.\_1) {  jedisIns.rpush(s"popular\_movies\_genreId\_${x.\_1}", x.\_2.\_2(i).\_1.toString)  }  jedisIns.close()  }  }  )  //  //特征抽取  //统计 a)用户历史正反馈次数  val counterUserIdPos = stream.flatMap(x => isEqual((x.\_2,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 b)用户历史负反馈次数  val counterUserIdNeg = stream.flatMap(x => isEqual((x.\_2,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 c)电影历史正反馈次数  val counterMovieIdPos = stream.flatMap(x => isEqual((x.\_3,x.\_4),1.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 d)电影历史负反馈次数  val counterMovieIdNeg = stream.flatMap(x => isEqual((x.\_3,x.\_4),0.0.toFloat))  .reduceByKey((x,y)=> x+y)  //统计 e)用户历史点击该分类比例  val counterUserId2MovieId = stream.filter(x=>x.\_4==1.0)  .map(x=>(x.\_2,x.\_3))  .groupByKey()  .flatMapValues(x=>{  var sum = 0  val one\_hot: Array[Int] = new Array[Int](19)  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  for (record<-x) {  sum=sum+1  val genres\_list = jedisIns.lrange("movie2genres\_movieId\_" + record.toString,0,-1)  val it = genres\_list.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  one\_hot(genresId) = one\_hot(genresId)+1  }  }  var counter:List[(Int,Float)] = List()  for (i<-one\_hot.indices) {  if (one\_hot(i)>0) counter = counter :+ (i,one\_hot(i).toFloat/sum)  }  jedisIns.close()  counter  })  counterUserIdPos.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_userId\_rating1\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterUserIdNeg.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_userId\_rating0\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterMovieIdPos.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_movieId\_rating1\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterMovieIdNeg.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set("streaming2feature\_movieId\_rating0\_" + x.\_1.toString, x.\_2.toString)  jedisIns.close()  }}  )  counterUserId2MovieId.foreachRDD(  rdd => rdd.foreach { x => {  val jedisIns = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedisIns.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedisIns.set(s"streaming2feature\_userId\_to\_genresId\_${x.\_1.toString}\_${x.\_2.\_1}", x.\_2.\_2.toString)  jedisIns.close()  }}  )  }  val redis\_host:String = "ljj-2019213687-0001"  val redis\_port:Int = 6379  val redis\_timeout:Int = 10000  def main(args:Array[String]) = {  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)  val sc:StreamingContext = kafkaStreaming.getSc()  streamingCore(sc)  sc.start()  sc.awaitTermination()  } } |

**recommend.scala**

|  |
| --- |
| Python  import com.alibaba.fastjson import com.alibaba.fastjson.JSON.parseObject import com.alibaba.fastjson.{JSON, JSONObject} import org.apache.kafka.clients.consumer.KafkaConsumer import org.apache.kafka.common.TopicPartition import org.apache.log4j.{Level, Logger} import org.apache.spark import org.apache.spark.{SparkConf, SparkContext} import org.apache.spark.api.java.{JavaRDD, JavaSparkContext} import org.apache.spark.ml.classification.LogisticRegression import org.apache.spark.ml.linalg.{Vector, Vectors} import org.apache.spark.sql.{Row,DataFrame, SparkSession} import org.apache.spark.streaming.{Seconds, StreamingContext} import redis.clients.jedis.{Jedis, JedisPool, JedisPoolConfig}  import java.io.{BufferedWriter, FileWriter} import java.lang.Thread.sleep import java.text.SimpleDateFormat import java.time.Duration import java.util.{Collections, Date, Properties} import scala.collection.mutable import scala.concurrent.duration.Duration  class recommend {}  object recommend {  def add\_record(map:mutable.Map[String,Int],jedis: Jedis,json:JSONObject) = {  // println(json)  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toFloat.toInt  if (rating == 1.0) {  //统计 a)用户历史正反馈次数  var key = s"counteruserId1\_${userId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 c)电影历史正反馈次数  key = s"countermovieId1\_${movieId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 e)用户历史点击该分类比例  key = s"counteruserId2movie\_${userId}\_${movieId}"  val genres = jedis.lrange(s"movie2genres\_movieId\_${movieId}",0,-1)  val it = genres.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  val key2 = s"counteruserId2genre\_${userId}\_${genresId}"  map += key2 -> (map.getOrElseUpdate(key2, 0)+1)  }  }  else {  //统计 b)用户历史负反馈次数  var key = "counteruserId0\_" + userId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  //统计 d)电影历史负反馈次数  key = "countermovieId0\_" + movieId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt+1)  }  map  }  def delete\_record(map:mutable.Map[String,Int],jedis: Jedis,json:JSONObject) = {  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toFloat.toInt  if (rating == 1.0) {  //统计 a)用户历史正反馈次数  var key = s"counteruserId1\_${userId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 c)电影历史正反馈次数  key = s"countermovieId1\_${movieId.toString}"  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 e)用户历史点击该分类比例  key = s"counteruserId2movie\_${userId}\_${movieId}"  val genres = jedis.lrange(s"movie2genres\_movieId\_${movieId}",0,-1)  val it = genres.iterator()  while (it.hasNext) {  val genresId = it.next().toInt  val key2 = s"counteruserId2genre\_${userId}\_${genresId}"  map += key2 -> (map.getOrElseUpdate(key2, 0)-1)  }  }  else {  //统计 b)用户历史负反馈次数  var key = "counteruserId0\_" + userId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  //统计 d)电影历史负反馈次数  key = "countermovieId0\_" + movieId.toString  map += key-> (map.getOrElseUpdate(key, 0).toString.toInt-1)  }  map  }  // def getRedisList(jedis: Jedis,key:String) = {  // val value = jedis.lrange(key,0,-1)  // jedis.close()  // value  // }  def collect\_train\_data() = {  val jedis = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedis.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  val records\_length = jedis.llen("streaming\_records")  var map: mutable.Map[String, Int] = mutable.Map()  for(i <- 0 to records\_length.toInt/2) {  val json = JSON.parseObject(jedis.lindex("streaming\_records",i))  map = add\_record(map,jedis,json)  }  var data:Seq[(Double,Vector)] = Seq()  for(i <- records\_length.toInt / 2 + 1 until records\_length.toInt) {  var json = JSON.parseObject(jedis.lindex("streaming\_records",i-records\_length.toInt/2))  map = delete\_record(map,jedis,json)  json = JSON.parseObject(jedis.lindex("streaming\_records",i))  map = add\_record(map,jedis,json)  val userId = json.getOrDefault("userId",null).toString.toInt  val movieId = json.getOrDefault("movieId",null).toString.toInt  val rating = json.getOrDefault("rating",null).toString.toDouble  var features:Seq[Double] = Seq()  for (key<-Array(  s"batch2feature\_userId\_rating1\_${userId}",  s"batch2feature\_userId\_rating0\_${userId}",  s"batch2feature\_movieId\_rating1\_${movieId}",  s"batch2feature\_movieId\_rating0\_${movieId}")) {  features = features :+ {  if (jedis.exists(key)) {  jedis.get(key).toDouble  } else  0  }  }  var sum:Double = 0.0  val key = s"batch2feature\_userId\_to\_genresId\_${userId}"  for(i <- 0 to 19) {  sum = sum + {  if (jedis.exists(s"${key}\_${i}"))  jedis.get(s"${key}\_${i}").toDouble  else  0  }  }  features = features :+ sum  for (key<-Array(  s"counteruserId1\_${userId}",  s"counteruserId0\_${userId}",  s"countermovieId1\_${movieId}",  s"countermovieId0\_${movieId}")) {  features = features :+ map.getOrElseUpdate(key,0).toDouble  }  val counteruserIdsum = map.getOrElseUpdate(s"counteruserId1\_${userId}",0) + map.getOrElseUpdate(s"counteruserId0\_${userId}",0)  //简单完成一个加和即可  sum = 0.0  for(i <- 0 to 19) {  sum = sum + {  val value = map.getOrElseUpdate(s"counteruserId2genre\_${userId}\_${i}",0).toDouble  if (value==0)  0.toDouble  else  value/counteruserIdsum.toDouble  }  }  features = features :+ sum   val yearkey = s"movieId2movieYear\_${movieId}"  features = features :+ {  if (jedis.exists(yearkey)) {  jedis.get(yearkey).toDouble  } else  0  }   data = data :+ (rating, Vectors.dense(features.toArray))  }  jedis.close()  data  }   def trainRecommendModel(spark:SparkSession) = {  val train\_data = collect\_train\_data()  val train\_dataframe = spark.createDataFrame(train\_data).toDF("label","features")  train\_dataframe.show(20,truncate = false)  import spark.implicits.\_  // val training = spark.read.format("libsvm").load("./sample\_libsvm\_data.txt")  val lr = new LogisticRegression()  .setMaxIter(20)   // Fit the model  val lrModel = lr.fit(train\_dataframe)  // val trainingSummary = lrModel.binarySummary  // val objectiveHistory = trainingSummary.objectiveHistory  // println("objectiveHistory:")  // objectiveHistory.foreach(loss => println(loss))  // println(s"areaUnderROC: ${trainingSummary.areaUnderROC}")   val params\_coefficients = lrModel.coefficients.toDense.toArray  println(params\_coefficients.mkString("Array(", ", ", ")"))  val jedis = new Jedis(redis\_host,redis\_port,redis\_timeout)  jedis.auth("Kd7Jdddd16@6djie8gce342NWM9znN4$V")  jedis.del("params\_coefficients")  for (i<- params\_coefficients.indices)  jedis.rpush("params\_coefficients",params\_coefficients(i).toString)  jedis.set("params\_intercept",lrModel.intercept.toString)   }  def NowDate(): String = {  val now: Date = new Date()  val dateFormat: SimpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss")  val date = dateFormat.format(now)  date  }  val redis\_host:String = "ljj-2019213687-0001"  val redis\_port:Int = 6379  val redis\_timeout:Int = 10000  def main(args:Array[String]) = {  Logger.getLogger("org.apache.spark").setLevel(Level.WARN)  while (true) {  print(s"${NowDate()} [INFO] Begin to train lr model")  val spark = SparkSession.builder().master("local[2]").appName("ALSExample").getOrCreate()  trainRecommendModel(spark)  spark.close()  print(s"${NowDate()} [INFO] Success!")  sleep(1000\*60\*5)  }  } } |