**西南石油大学实验报告**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **课 程** | 大数据综合应用实践 | | **实验项目** | | | 任务3 <商品推荐> | | | | | **成绩** | | |  |
| **专业班级** | 2016网络工程 | | **组别** | |  | | **学号** | 201631063211 | | **指导教师** | | 赖俊良 | | |
| **姓 名** | 古皓 | **同组人姓名** | | 无 | | | | | **实验日期** | | | | 2019/7/12 | |

1. **实验方案**

记录拟实现该实验的总体思路和方法，包括：数据预处理方案、采用何种算法、算法测试方案等，也包含一些改进点和创新点。

----内容为宋体小四，1.5倍行距----

正则表达式进行数据清理，自己通过购买次数和数量设计评分方案将数据转换成用户对商品的评分。将数据划分为训练集train（0.6），验证集validation（0.2）和测试集test（0.2），协同过滤算法，设置不同参数的模型，用训练集训练，验证集计算每种模型的均方差误差来判断最优的模型。确定好模型进行测试集测试。推荐模型将所有数据录入，协同过滤选择出与输入用户id相似度最高的评分rating值取得商品product字段返回给用户作为推荐的商品。

1. **实验步骤**

记录任务完成的具体操作过程、参数、中间结果和结果，以文字、代码、流程图和截图方式等展示

----内容为宋体小四，1.5倍行距----

1. 数据录入

val regex = new Regex(" ")

val input = sc.textFile("file:\\E:\\有用的文档\\Hadoop大数据\\实践\\题目三\\OnlineRetail.csv")  
val lines = input.map(line => line.split(",")).filter(s => !s(1).equals("") && !s(6).equals("") && !s(6).equals("CustomerID") && regex.findFirstIn(s(1)).isEmpty)

进行数据清理，过滤掉商品id字段为空和用户ID字段为空的数据，OnlineRetail.csv中商品id字段有的带有字母，我将26个字母映射为1到26数值替换，通过判断用户id是否为CustomerID来过滤掉第一行表头

然后，计算购买次数和购买数量并合并计算评分pingfen(x,y)为计算评分的方法，传入的x为购买次数，y为购买数量，返回double类型的评分值

val buytime = lines.map(s => ((s(6), s(1)), 1)).reduceByKey((x, y) => x + y)  
val buyquantity = lines.map(s=>((s(6),s(1)),s(3).toInt)).reduceByKey((x,y) => x + y)  
val ratings1 = buytime.join(buyquantity).map{case ((a,b),(x,y))=>((a,b),*pingfen*(x,y))}.cache()

1. 划分数据(training,validation,test)

var a=0  
val ratings=ratings1.map(s => ({a=a+1  
a%10}, Rating(s.\_1.\_1.toInt, s.\_1.\_2.toInt, s.\_2.toDouble))).cache()  
 val numPartitions = 4  
val training = ratings.filter(s=>s.\_1<6).values  
.repartition(numPartitions).cache()  
val validation = ratings.filter(s=>s.\_1>=7&&s.\_1<8).values  
 .repartition(numPartitions).cache()  
val test = ratings.filter(c=>c.\_1>=8).values  
 .repartition(numPartitions).cache()

给ratings添加一个自增id（int）将id除以10的余数为0~6分为训练集，余数为6~8分为验证集，余数为8~9分为测试集

1. 交叉验证，计算最佳模型

模型

val model = ALS.train(training, rank, numIter, lambda)

training，为训练集，数据格式：(用户id 商品id 评分)

rank，根据数据的分散情况测试出来的值，特征向量纬度，如果这个值太小拟合的就会不够，误差就很大；如果这个值很大，就会导致模型大泛化能力较差；所以就需要自己把握一个度了，一般情况下10～1000都是可以的；

numIter，循环次数，设置越大越精确，设置越大也越耗时；

lambda，和rank一样的，如果设置很大就可以防止过拟合问题，如果设置很小，其实可以理解为直接设置为0，那么就不会有防止过拟合的功能了；

val ranks = *List*(10,12)  
val lambdas =*List*(0.1,0.2)  
val numIters = *List*(6,8,10)

for(rank <- ranks;lambda <- lambdas;numIter <- numIters){  
 val model = ALS.*train*(training,rank,numIter,lambda)  
 re= *computeRmse*(model,validation)  
 if(re<bestValidationRmse ){  
 bestModel = *Some*(model)  
 bestValidationRmse = re  
 bestRank = rank  
 bestLambda = lambda  
 bestNumber = numIter  
 }  
}

设置几组不同的参数，通过循环比较选择数均方差误差最小的方案为最佳模型computeRmse(model,validation)计算均方差的自定义方法，返回值为均方差误差值，确定好模型后用测试集test测试计算均方差误差

采用交叉验证重新划分训练集，验证集，测试集交叉验证计算最好的模型

1. 推荐商品,加载用户，给每个用户推荐两个评分最高的商品

val model = ALS.*train*(ratings.values,bestRank,bestNumber,bestLambda)  
for (e <- customers){  
 val pre = model.recommendProducts(e,2).toSeq.map(x=>(x.product,x.rating))  
 for (elem <- pre) {  
 message+=(id->("为用户"+e+"推荐商品："+elem.\_1+",推荐评分："+elem.\_2))  
 id=id+1  
 }  
 message+=(id->"----------------------------------------------------------------------------")  
 id=id+1  
}

model.recommendProducts(x,y)方法x为用户id，y为返回推荐商品的数量，通过循环将信息写入message(Map)这里我是给每个用户推荐两个商品。

1. 方法pingfen(),computeRmse()

def pingfen(a:Int,b:Int): Double ={  
 var bt=a.toDouble;var bq=b.toDouble  
 val minTime = 0  
 val maxTime = 10  
 val minQuan = 0  
 val maxQuan = 100  
 if(bt>10)bt=maxTime  
 else {  
 if(bt==1)bt=5  
 else if(bt>1)bt=5\*math.*log10*(bt\*6)  
 }  
 if(bq>100)bq=maxQuan  
 if(bq<0){bt=5;bq=minQuan}  
 (bt+math.*log10*(bq+1)).formatted("%.2f").toDouble }

评分方案确定

数据记录共267605

经计算 购买次数大于50的记录共5条,大于30的记录共41条，大于20的记录共124条,大于10的记录共1006条 最大购买次数记为10 购买次数为1有198881条

经计算 购买数量大于100的记录共5772条，大于300的记录共1470条，大于500的记录共782, 最大购买数量记为100 最小购买数量记为0

经过统计发现购买次数1占很大一部分，够买次数都包含在10次以内，极少数超过10次于是将超过10次购买的统统计为10次

购买数量超多100的有5772条，大部分都在100以内，将超过100购买数量的统统计为100，购买次数为负数的（可能是存在退货的情况）统统计为0，这样

购买次数范围缩小到1~10次，购买数量范围缩小到0~100

评分初始值都为5，根据这样数据分布于是采用log函数来计算评分，随着自变量增加，因变量增加的越缓慢，斜率k越小，在评分为5 的基础上计算额外的加分，log函数就是要体现出计算细微的变化比巨大的变化更明显，于是经过深思熟虑有了上面的方法。

def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating]): Double ={  
 val userproducts = data.map{case *Rating*(user, product, rate)=>(user,product)}  
 val predict = model.predict(userproducts).map{case *Rating*(user,product,rate)=>((user,product),rate)}  
 val rateandpreds=data.map{case *Rating*(user,product,rate)=>((user,product),rate)}.join(predict)  
  
 rateandpreds.map{case ((user,product),(r1,r2))=>val err=r1-r2  
 err\*err}.mean()  
}

计算均方差是scala Apache网上文档的方法，传入参数model（模型），data（数据集），返回均方差误差值

定义一个map类型的message变量，将所有信息保存到message(id:int,message:string)id自增

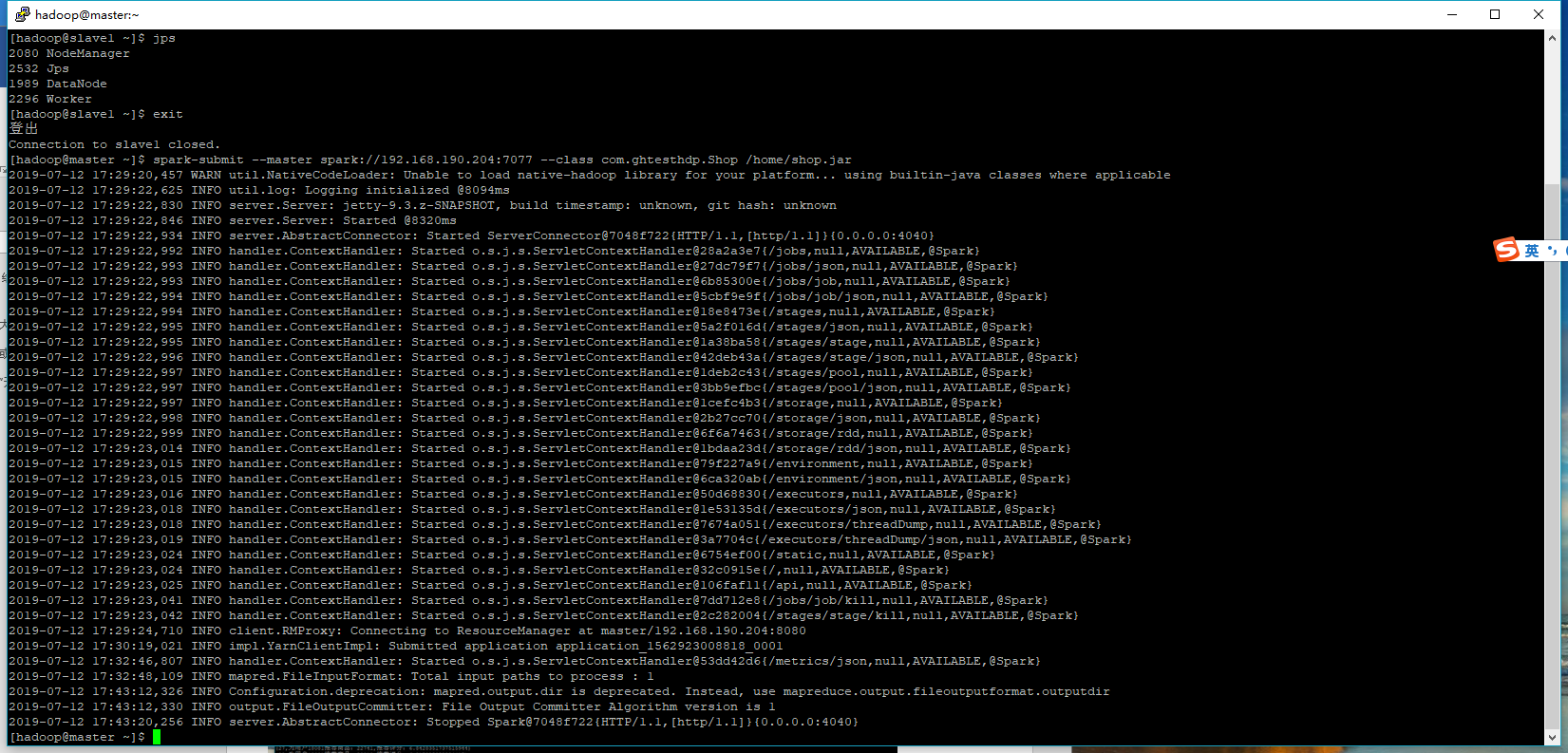
sc.makeRDD(message.toSeq.sortBy(\_.\_1).toArray).saveAsTextFile("hdfs://192.168.190.204:9000/shop/message")

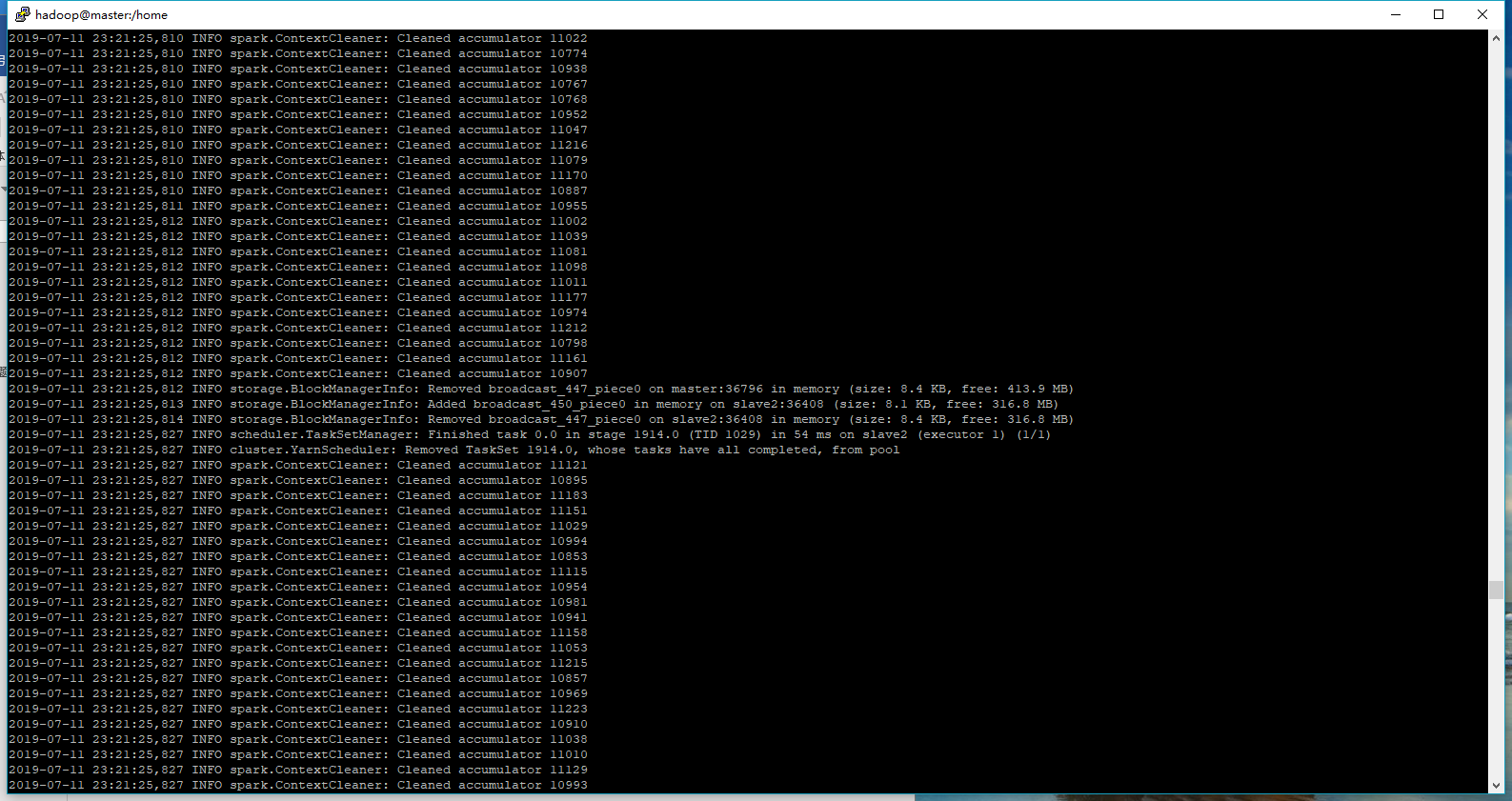
1. **实验结果**

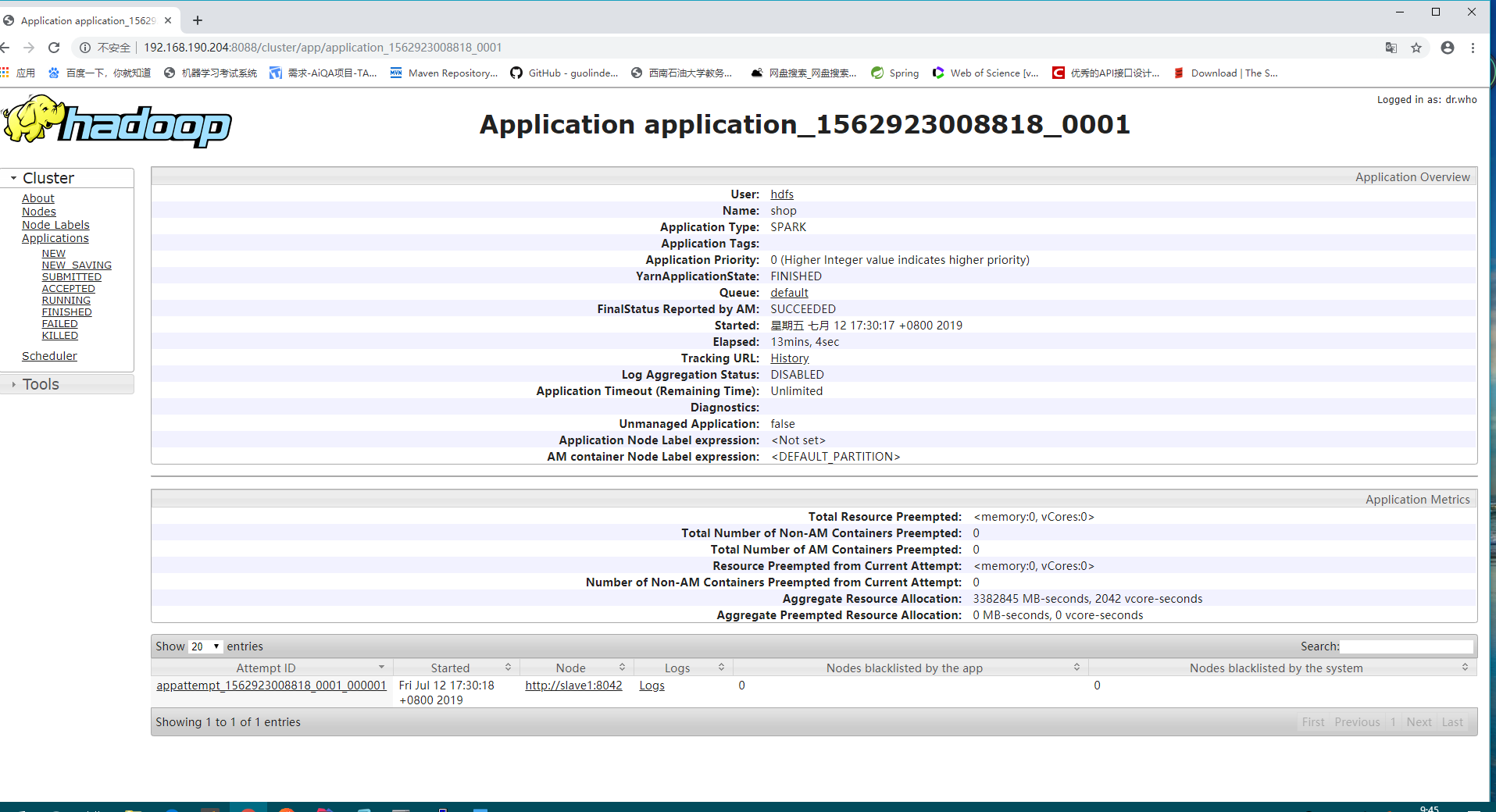
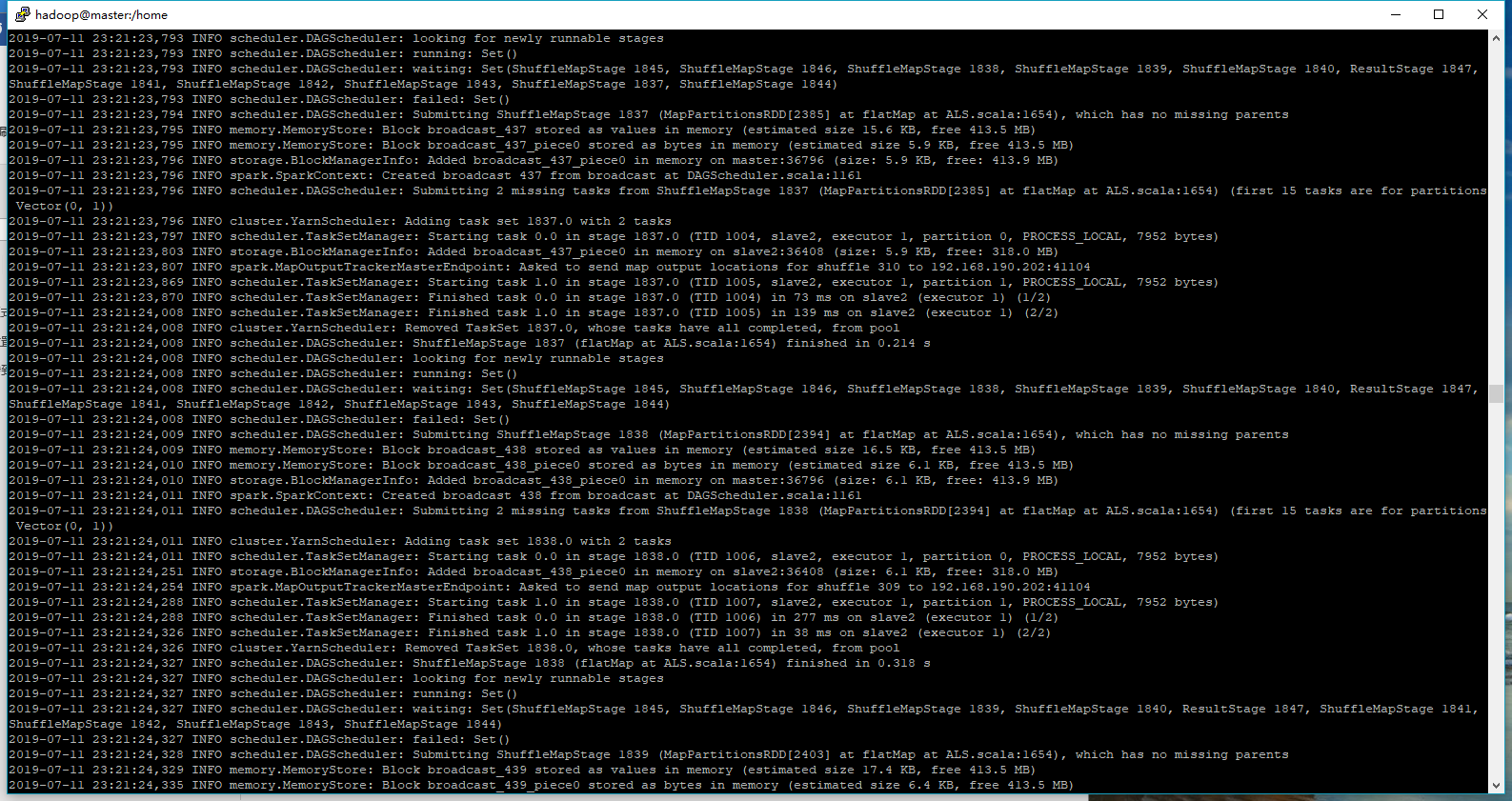
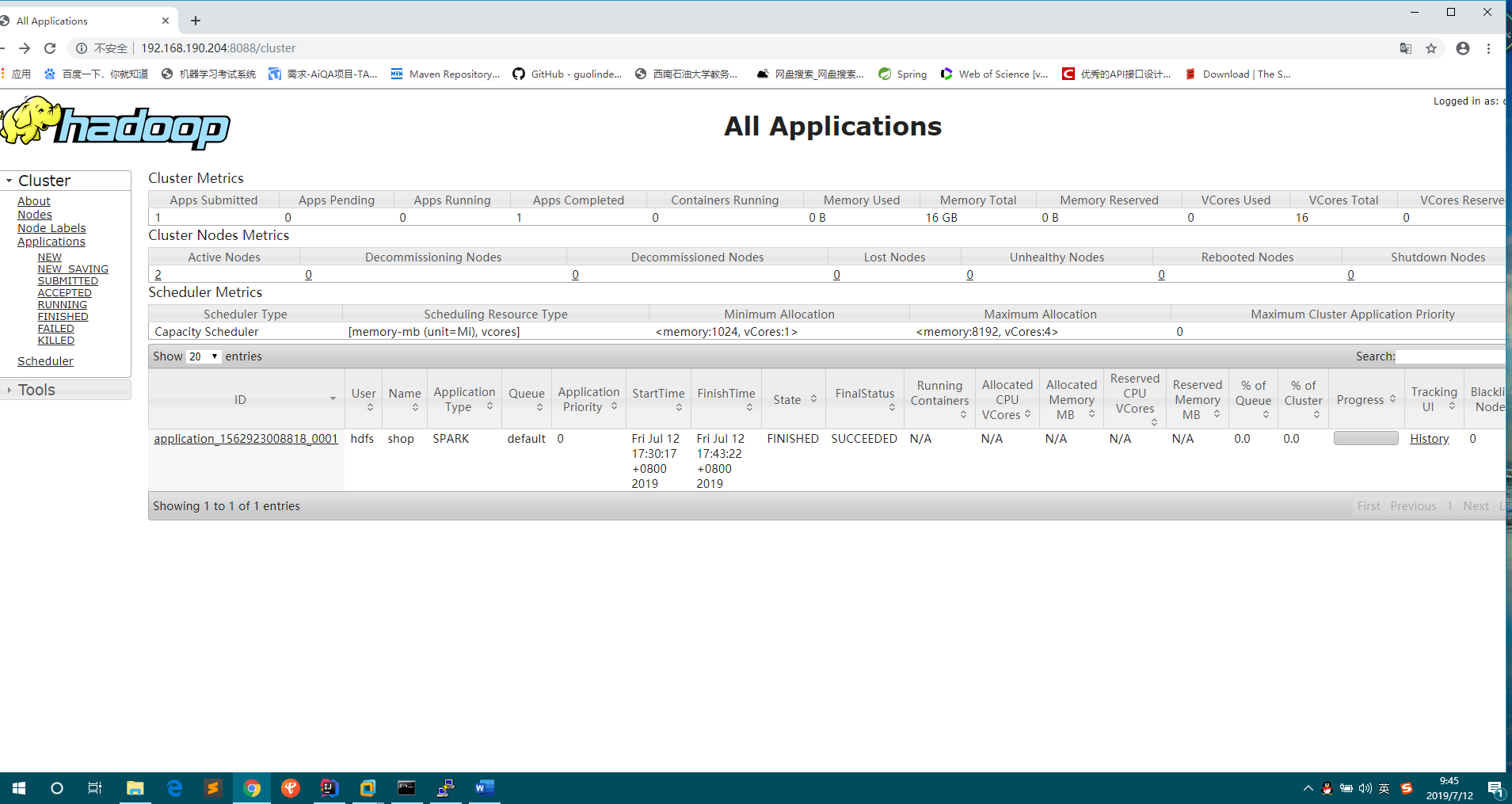
最终的处理结果或测试结果展示，用表格或截图等方式展示

----内容为宋体小四，1.5倍行距----

Spark-submit,屏蔽掉日志info

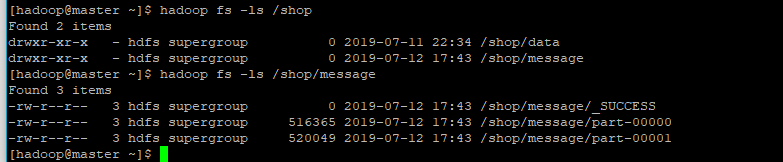
Logger.*getLogger*("org.apache.spark").setLevel(Level.*ERROR*)  
Logger.*getLogger*("org.eclipse.jetty.server").setLevel(Level.*OFF*)

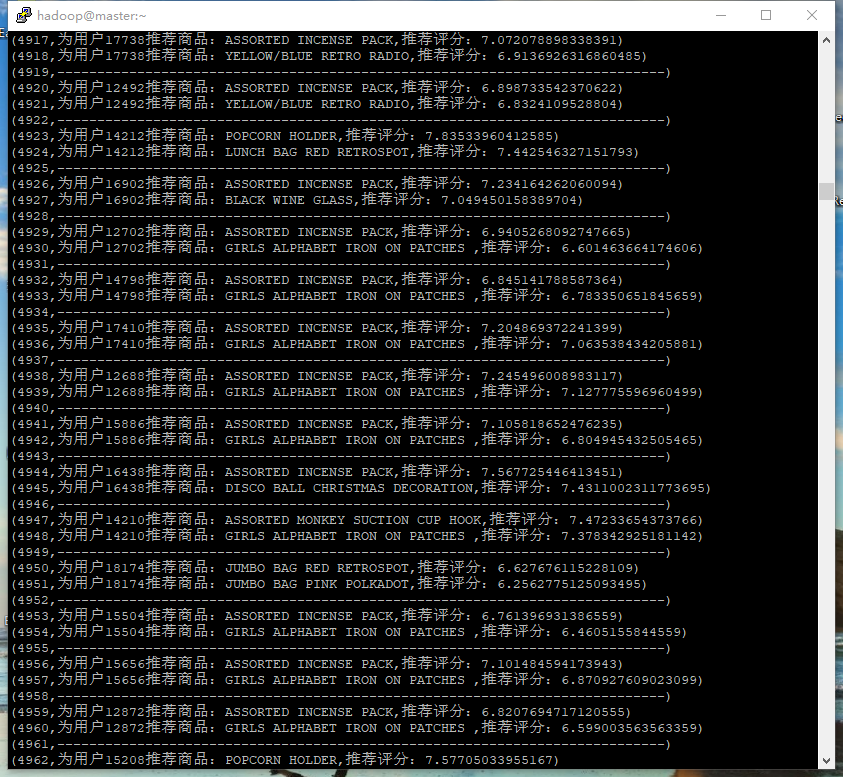
没有屏蔽info的运行部分截图

通过网页查看状态

将需要打印出来的信息保存到message并保存到文件里面

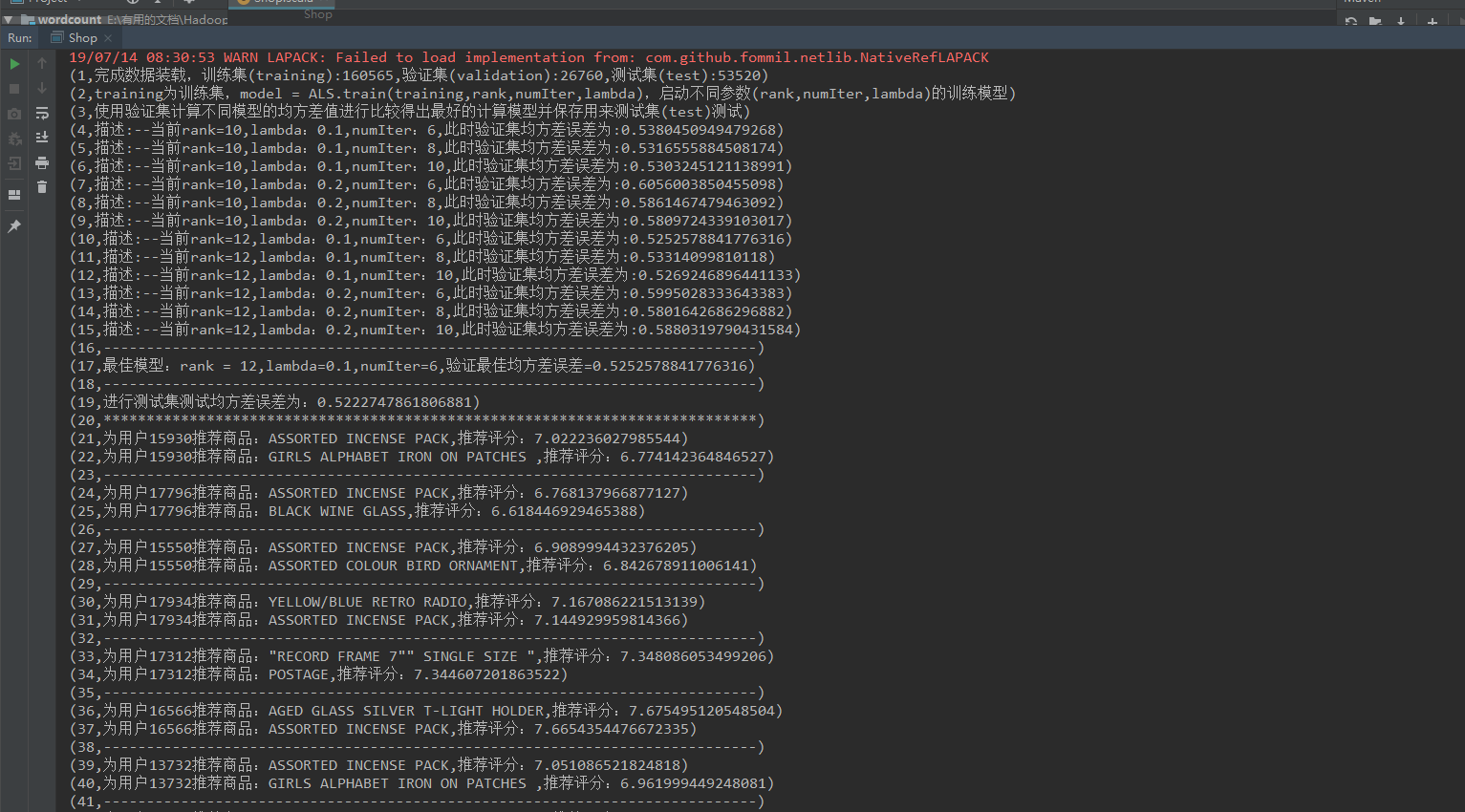
sc.makeRDD(message.toSeq.sortBy(\_.\_1).toArray).saveAsTextFile("hdfs://192.168.190.204:9000/shop/message")

目录下出现需要的文件信息

Hadoop fs -cat打开查看

Hdfs上查看不全传到linux上



Windows idea测试结果Windows上测试将，spark-mllib\_2.11依赖的<scope>值设为compile才能测试

1. **体会和展望**

实验过程中的一些困难和心得体会，算法的不足和今后的改进思路

----内容为宋体小四，1.5倍行距----

两周学习scala，spark做一个商品推荐的实践，做出来还是很有成就感的。一开始摸不着头脑只有看网上的样本实例一步一步去理解去弄明白，花了很长时间也做了很多无用功。但结果还是好的进一步加深了对spark，scala的理解和认知，是一门功能强大的语言，写的代码不是很多却实现了很多功能。算法的不足，每次跑代码要执行很长时间，没有去了解如何提升运行效率，代码中的map循环很多可能是效率不高的原因，每执行一次都要打印很长一段info，于是索性将info日志屏蔽掉了。后续改进思路就是进一步优化，增加执行效率。

代码:

import org.apache.log4j.{Level, Logger}

import org.apache.spark.mllib.recommendation.{ALS, MatrixFactorizationModel, Rating}

import org.apache.spark.rdd.RDD

import org.apache.spark.{SparkConf, SparkContext}

import scala.util.matching.Regex

object Shop {

def main(args: Array[String]): Unit = {

Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)

Logger.getLogger("org.eclipse.jetty.server").setLevel(Level.OFF)

var id=1

var message:Map[Int,String] = Map()

var re = 10.0

val regex = new Regex(" ")

val conf = new SparkConf().setMaster("yarn-client").setAppName("shop")

val sc = new SparkContext(conf)

val input = sc.textFile("hdfs://192.168.190.204:9000/shop/data/OnlineRetail.csv")

//val input = sc.textFile("file:\\E:\\有用的文档\\Hadoop大数据\\实践\\题目三\\OnlineRetail.csv")

val lines = input.map(line => line.split(",")).filter(s => !s(1).equals("") && !s(6).equals("") && !s(6).equals("CustomerID") && regex.findFirstIn(s(1)).isEmpty)

val products = lines.map(s=>(s(1),s(2))).distinct.cache()

val customers = lines.map(s=>s(6).toInt).distinct.collect()

val buytime = lines.map(s => ((s(6), s(1)), 1)).reduceByKey((x, y) => x + y)

val buyquantity = lines.map(s=>((s(6),s(1)),s(3).toInt)).reduceByKey((x,y) => x + y)

val ratings1 = buytime.join(buyquantity).map{case ((a,b),(x,y))=>((a,b),pingfen(x,y))}.cache()

var a=0

val ratings=ratings1.map(s => ({a=a+1

a%10}, Rating(s.\_1.\_1.toInt, s.\_1.\_2.toInt, s.\_2.toDouble))).cache()

val numPartitions = 4

val training = ratings.filter(s=>s.\_1<6)

.values

.repartition(numPartitions)

.cache()

val validation = ratings.filter(s=>s.\_1>=7&&s.\_1<8)

.values

.repartition(numPartitions)

.cache()

val test = ratings.filter(c=>c.\_1>=8)

.values

.repartition(numPartitions)

.cache()

// val model = ALS.train(training, 10, 10, 0.5)

message+=(id -> ("完成数据装载，训练集(training):"+training.count() + ",验证集(validation):" + validation.count() + ",测试集(test):" + test.count()))

id=id+1

val ranks = List(10,12)

val lambdas =List(0.1,0.2)

val numIters = List(6,8,10)

var bestModel:Option[MatrixFactorizationModel] = None

var bestValidationRmse = Double.MaxValue

var bestRank = 0

var bestLambda = 0.0

var bestNumber = 0

message+=(id->"training为训练集，model = ALS.train(training,rank,numIter,lambda)，启动不同参数(rank,numIter,lambda)的训练模型")

id=id+1

message+=(id->"使用验证集计算不同模型的均方差值进行比较得出最好的计算模型并保存用来测试集(test)测试")

id=id+1

for(rank <- ranks;lambda <- lambdas;numIter <- numIters){

val model = ALS.train(training,rank,numIter,lambda)

re= computeRmse(model,validation)

message+=(id->("描述:--当前rank="+rank +",lambda："+lambda+",numIter："+numIter+",此时验证集均方差误差为:"+re))

id=id+1

if(re<bestValidationRmse ){

bestModel = Some(model)

bestValidationRmse = re

bestRank = rank

bestLambda = lambda

bestNumber = numIter

}

}

message+=(id->"----------------------------------------------------------------------------")

id=id+1

message+=(id->("最佳模型：rank = "+bestRank+",lambda="+bestLambda+",numIter="+bestNumber+",验证最佳均方差误差="+bestValidationRmse))

id=id+1

val re1= computeRmse(bestModel.get,test)

message+=(id->"----------------------------------------------------------------------------")

id=id+1

message+=(id->("进行测试集测试均方差误差为："+re1))

id=id+1

message+=(id->"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

id=id+1

val model = ALS.train(ratings.values,bestRank,bestNumber,bestLambda)

for (e <- customers){

val pre = model.recommendProducts(e,2).toSeq.map(x=>(x.product,x.rating))

for (elem <- pre) {

val pro=products.filter(s=>s.\_1.toInt ==elem.\_1).first()

message+=(id->("为用户"+e+"推荐商品："+pro.\_2+",推荐评分："+elem.\_2))

id=id+1

}

message+=(id->"----------------------------------------------------------------------------")

id=id+1

}

sc.makeRDD(message.toSeq.sortBy(\_.\_1).toArray).saveAsTextFile("hdfs://192.168.190.204:9000/shop/message")

//message.toSeq.sortBy(\_.\_1).toArray.foreach(println)

}

//评分

//数据记录共267605

//经计算 购买次数大于50的记录共5条,大于30的记录共41条，大于20的记录共124条,大于10的记录共1006条 最大购买次数记为10 购买次数为1有198881条

//经计算 购买数量大于100的记录共5772条，大于300的记录共1470条，大于500的记录共782条，大于100的记录共5722条 最大购买数量记为100 最小购买数量记为0

def pingfen(a:Int,b:Int): Double ={

var bt=a.toDouble;var bq=b.toDouble

val minTime = 0

val maxTime = 10

val minQuan = 0

val maxQuan = 100

if(bt>10)bt=maxTime

else {

if(bt==1)bt=5

else if(bt>1)bt=5\*math.log10(bt\*6)

}

if(bq>100)bq=maxQuan

if(bq<0){bt=5;bq=minQuan}

(bt+math.log10(bq+1)).formatted("%.2f").toDouble

}

//均方差

def computeRmse(model: MatrixFactorizationModel, data: RDD[Rating]): Double ={

val userproducts = data.map{case Rating(user, product, rate)=>(user,product)}

val predict = model.predict(userproducts).map{case Rating(user,product,rate)=>((user,product),rate)}

val rateandpreds=data.map{case Rating(user,product,rate)=>((user,product),rate)}.join(predict)

rateandpreds.map{case ((user,product),(r1,r2))=>val err=r1-r2

err\*err}.mean()

}

}