**Mahout Item-based推荐的分布式实现**

[](https://www.jianshu.com/u/70b4fd000153)

[Skye\_kh](https://www.jianshu.com/u/70b4fd000153) 关注

2016.07.25 16:06\* 字数 2790 阅读 1232评论 1喜欢 5

title: Mahout Item-based推荐的分布式实现  
date: 2016/7/25 16:02:36  
tags:

* Mahout
* 推荐系统  
  categories: 大数据

Mahout API地址：[http://apache.github.io/mahout/0.10.1/docs/mahout-mr/overview-summary.html](https://link.jianshu.com/?t=http://apache.github.io/mahout/0.10.1/docs/mahout-mr/overview-summary.html)

[Mahout推荐算法API详解](https://link.jianshu.com/?t=http://blog.fens.me/mahout-recommendation-api/)

**Mahout算法框架自带的推荐器有下面这些：**

* GenericUserBasedRecommender：基于用户的推荐器，用户数量少时速度快；
* GenericItemBasedRecommender：基于商品推荐器，商品数量少时速度快，尤其当外部提供了商品相似度数据后效率更好；
* SlopeOneRecommender：基于slope-one算法的推荐器，在线推荐或更新较快，需要事先大量预处理运算，物品数量少时较好；
* SVDRecommender：奇异值分解，推荐效果较好，但之前需要大量预处理运算；
* KnnRecommender：基于k近邻算法(KNN)，适合于物品数量较小时；
* TreeClusteringRecommender：基于聚类的推荐器，在线推荐较快，之前需要大量预处理运算，用户数量较少时效果好；
* Mahout最常用的三个推荐器是上述的前三个，本文主要讨论前两种的使用。

**接口相关介绍**

基于用户或物品的推荐器主要包括以下几个接口：

DataModel 是用户喜好信息的抽象接口，它的具体实现支持从任意类型的数据源抽取用户喜好信息。Taste 默认提供 JDBCDataModel 和 FileDataModel，分别支持从数据库和文件中读取用户的喜好信息。  
UserSimilarity 和 ItemSimilarity。UserSimilarity 用于定义两个用户间的相似度，它是基于协同过滤的推荐引擎的核心部分，可以用来计算用户的“邻居”，这里我们将与当前用户口味相似的用户称为他的邻居。ItemSimilarity 类似的，计算内容之间的相似度。  
UserNeighborhood 用于基于用户相似度的推荐方法中，推荐的内容是基于找到与当前用户喜好相似的邻居用户的方式产生的。UserNeighborhood 定义了确定邻居用户的方法，具体实现一般是基于 UserSimilarity 计算得到的。  
Recommender 是推荐引擎的抽象接口，Taste 中的核心组件。程序中，为它提供一个 DataModel，它可以计算出对不同用户的推荐内容。实际应用中，主要使用它的实现类 GenericUserBasedRecommender 或者 GenericItemBasedRecommender，分别实现基于用户相似度的推荐引擎或者基于内容的推荐引擎。  
RecommenderEvaluator：评分器。  
RecommenderIRStatsEvaluator：搜集推荐性能相关的指标，包括准确率、召回率等等。  
目前，Mahout为DataModel提供了以下几种实现：

* org.apache.mahout.cf.taste.impl.model.GenericDataModel
* org.apache.mahout.cf.taste.impl.model.GenericBooleanPrefDataModel
* org.apache.mahout.cf.taste.impl.model.PlusAnonymousUserDataModel
* org.apache.mahout.cf.taste.impl.model.file.FileDataModel
* org.apache.mahout.cf.taste.impl.model.hbase.HBaseDataModel
* org.apache.mahout.cf.taste.impl.model.cassandra.CassandraDataModel
* org.apache.mahout.cf.taste.impl.model.mongodb.MongoDBDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.SQL92JDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.MySQLJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.PostgreSQLJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.GenericJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.SQL92BooleanPrefJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.MySQLBooleanPrefJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.PostgreBooleanPrefSQLJDBCDataModel
* org.apache.mahout.cf.taste.impl.model.jdbc.ReloadFromJDBCDataModel  
  从类名上就可以大概猜出来每个DataModel的用途，奇怪的是竟然没有HDFS的DataModel，有人实现了一个，请参考[MAHOUT-1579](https://link.jianshu.com/?t=https://issues.apache.org/jira/browse/MAHOUT-1579)。

UserSimilarity 和 ItemSimilarity 相似度实现有以下几种：

* CityBlockSimilarity：基于Manhattan距离相似度
* EuclideanDistanceSimilarity：基于欧几里德距离计算相似度
* LogLikelihoodSimilarity：基于对数似然比的相似度
* PearsonCorrelationSimilarity：基于皮尔逊相关系数计算相似度
* SpearmanCorrelationSimilarity：基于皮尔斯曼相关系数相似度
* TanimotoCoefficientSimilarity：基于谷本系数计算相似度
* UncenteredCosineSimilarity：计算 Cosine 相似度  
  以上相似度的说明，请参考Mahout推荐引擎介绍。

UserNeighborhood 主要实现有两种：

* NearestNUserNeighborhood：对每个用户取固定数量N个最近邻居
* ThresholdUserNeighborhood：对每个用户基于一定的限制，取落在相似度限制以内的所有用户为邻居

Recommender分为以下几种实现：

* GenericUserBasedRecommender：基于用户的推荐引擎
* GenericBooleanPrefUserBasedRecommender：基于用户的无偏好值推荐引擎
* GenericItemBasedRecommender：基于物品的推荐引擎
* GenericBooleanPrefItemBasedRecommender：基于物品的无偏好值推荐引擎

RecommenderEvaluator有以下几种实现：

* AverageAbsoluteDifferenceRecommenderEvaluator：计算平均差值
* RMSRecommenderEvaluator：计算均方根差
* RecommenderIRStatsEvaluator的实现类是GenericRecommenderIRStatsEvaluator。

**单机运行**

首先，需要在maven中加入对mahout的依赖：

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-core</artifactId>

<version>0.9</version>

</dependency>

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-integration</artifactId>

<version>0.9</version>

</dependency>

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-math</artifactId>

<version>0.9</version>

</dependency>

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-examples</artifactId>

<version>0.9</version>

</dependency>

基于用户的推荐，以FileDataModel为例：

File modelFile modelFile = new File("intro.csv");

DataModel model = new FileDataModel(modelFile);

//用户相似度，使用基于皮尔逊相关系数计算相似度

UserSimilarity similarity = new PearsonCorrelationSimilarity(model);

//选择邻居用户，使用NearestNUserNeighborhood实现UserNeighborhood接口，选择邻近的4个用户

UserNeighborhood neighborhood = new NearestNUserNeighborhood(4, similarity, model);

Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);

//给用户1推荐4个物品

List<RecommendedItem> recommendations = recommender.recommend(1, 4);

for (RecommendedItem recommendation : recommendations) {

System.out.println(recommendation);

}

注意： FileDataModel要求输入文件中的字段分隔符为逗号或者制表符，如果你想使用其他分隔符，你可以扩展一个FileDataModel的实现，例如，mahout中已经提供了一个解析MoiveLens的数据集（分隔符为::）的实现GroupLensDataModel。

GenericUserBasedRecommender是基于用户的简单推荐器实现类，推荐主要参照传入的DataModel和UserNeighborhood，总体是三个步骤：

1. 从UserNeighborhood获取当前用户Ui最相似的K个用户集合{U1, U2, …Uk}；
2. 从这K个用户集合排除Ui的偏好商品，剩下的Item集合为{Item0, Item1, …Itemm}；
3. 对Item集合里每个Itemj计算Ui可能偏好程度值pref(Ui, Itemj)，并把Item按此数值从高到低排序，前N个item推荐给用户Ui。

对相同用户重复获得推荐结果，我们可以改用CachingRecommender来包装GenericUserBasedRecommender对象，将推荐结果缓存起来：

Recommender cachingRecommender = new CachingRecommender(recommender);

上面代码可以在main方法中直接运行，然后，我们可以获取推荐模型的评分：

//使用平均绝对差值获得评分

RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

// 用RecommenderBuilder构建推荐引擎

RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

UserSimilarity similarity = new PearsonCorrelationSimilarity(model);

UserNeighborhood neighborhood = new NearestNUserNeighborhood(4, similarity, model);

return new GenericUserBasedRecommender(model, neighborhood, similarity);

}

};

// Use 70% of the data to train; test using the other 30%.

double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);

System.out.println(score);

接下来，可以获取推荐结果的查准率和召回率：

RecommenderIRStatsEvaluator statsEvaluator = new GenericRecommenderIRStatsEvaluator();

// Build the same recommender for testing that we did last time:

RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

UserSimilarity similarity = new PearsonCorrelationSimilarity(model);

UserNeighborhood neighborhood = new NearestNUserNeighborhood(4, similarity, model);

return new GenericUserBasedRecommender(model, neighborhood, similarity);

}

};

// 计算推荐4个结果时的查准率和召回率

IRStatistics stats = statsEvaluator.evaluate(recommenderBuilder,null, model, null, 4,

GenericRecommenderIRStatsEvaluator.CHOOSE\_THRESHOLD,1.0);

System.out.println(stats.getPrecision());

System.out.println(stats.getRecall());

如果是基于物品的推荐，代码大体相似，只是没有了UserNeighborhood，然后将上面代码中的User换成Item即可，完整代码如下：

File modelFile modelFile = new File("intro.csv");

DataModel model = new FileDataModel(new File(file));

// Build the same recommender for testing that we did last time:

RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {

@Override

public Recommender buildRecommender(DataModel model) throws TasteException {

ItemSimilarity similarity = new PearsonCorrelationSimilarity(model);

return new GenericItemBasedRecommender(model, similarity);

}

};

//获取推荐结果

List<RecommendedItem> recommendations = recommenderBuilder.buildRecommender(model).recommend(1, 4);

for (RecommendedItem recommendation : recommendations) {

System.out.println(recommendation);

}

//计算评分

RecommenderEvaluator evaluator =

new AverageAbsoluteDifferenceRecommenderEvaluator();

// Use 70% of the data to train; test using the other 30%.

double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);

System.out.println(score);

//计算查全率和查准率

RecommenderIRStatsEvaluator statsEvaluator = new GenericRecommenderIRStatsEvaluator();

// Evaluate precision and recall "at 2":

IRStatistics stats = statsEvaluator.evaluate(recommenderBuilder,

null, model, null, 4,

GenericRecommenderIRStatsEvaluator.CHOOSE\_THRESHOLD,

1.0);

System.out.println(stats.getPrecision());

System.out.println(stats.getRecall());

**在Spark中运行**

在Spark中运行，需要将Mahout相关的jar添加到Spark的classpath中，修改/etc/spark/conf/spark-env.sh，添加下面两行代码：

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/mahout/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/mahout/\*"

然后，以本地模式在spark-shell中运行下面代码交互测试：

//注意：这里是本地目录

val model = new FileDataModel(new File("intro.csv"))

val evaluator = new RMSRecommenderEvaluator()

val recommenderBuilder = new RecommenderBuilder {

override def buildRecommender(dataModel: DataModel): Recommender = {

val similarity = new LogLikelihoodSimilarity(dataModel)

new GenericItemBasedRecommender(dataModel, similarity)

}

}

val score = evaluator.evaluate(recommenderBuilder, null, model, 0.95, 0.05)

println(s"Score=$score")

val recommender=recommenderBuilder.buildRecommender(model)

val users=trainingRatings.map(\_.user).distinct().take(20)

import scala.collection.JavaConversions.\_

val result=users.par.map{user=>

user+","+recommender.recommend(user,40).map(\_.getItemID).mkString(",")

}

[https://github.com/sujitpal/mia-scala-examples](https://link.jianshu.com/?t=https://github.com/sujitpal/mia-scala-examples)上面有一个评估基于物品或是用户的各种相似度下的评分的类，叫做 RecommenderEvaluator，供大家学习参考。

**分布式运行**

Mahout提供了org.apache.mahout.cf.taste.hadoop.item.RecommenderJob类以MapReduce的方式来实现基于物品的协同过滤，查看该类的使用说明：

ubuntu@Master:~/data$ mahout org.apache.mahout.cf.taste.hadoop.item.RecommenderJob

MAHOUT\_LOCAL is not set; adding HADOOP\_CONF\_DIR to classpath.

Running on hadoop, using /home/ubuntu/hadoop/bin/hadoop and HADOOP\_CONF\_DIR=/home/ubuntu/hadoop/etc/hadoop

MAHOUT-JOB: /home/ubuntu/apache-mahout-distribution-0.10.1/mahout-examples-0.10.1-job.jar

16/07/25 07:41:32 WARN driver.MahoutDriver: No org.apache.mahout.cf.taste.hadoop.item.RecommenderJob.props found on classpath, will use command-line arguments only

16/07/25 07:41:33 ERROR common.AbstractJob: Missing required option --similarityClassname

Missing required option --similarityClassname

Usage:

[--input <input> --output <output> --numRecommendations <numRecommendations>

--usersFile <usersFile> --itemsFile <itemsFile> --filterFile <filterFile>

--userItemFile <userItemFile> --booleanData <booleanData> --maxPrefsPerUser

<maxPrefsPerUser> --minPrefsPerUser <minPrefsPerUser> --maxSimilaritiesPerItem

<maxSimilaritiesPerItem> --maxPrefsInItemSimilarity <maxPrefsInItemSimilarity>

--similarityClassname <similarityClassname> --threshold <threshold>

--outputPathForSimilarityMatrix <outputPathForSimilarityMatrix> --randomSeed

<randomSeed> --sequencefileOutput --help --tempDir <tempDir> --startPhase

<startPhase> --endPhase <endPhase>]

--similarityClassname (-s) similarityClassname Name of distributed

similarity measures class to

instantiate, alternatively

use one of the predefined

similarities

([SIMILARITY\_COOCCURRENCE,

SIMILARITY\_LOGLIKELIHOOD,

SIMILARITY\_TANIMOTO\_COEFFICIEN

T, SIMILARITY\_CITY\_BLOCK,

SIMILARITY\_COSINE,

SIMILARITY\_PEARSON\_CORRELATION

,

SIMILARITY\_EUCLIDEAN\_DISTANCE]

)

也可输入mahout org.apache.mahout.cf.taste.hadoop.item.RecommenderJob --help查看详细说明

可见，该类可以接收的命令行参数如下：

--input(path)(-i): 存储用户偏好数据的目录，该目录下可以包含一个或多个存储用户偏好数据的文本文件；  
--output(path)(-o): 结算结果的输出目录  
--numRecommendations (integer): 为每个用户推荐的item数量，默认为10  
--usersFile (path): 指定一个包含了一个或多个存储userID的文件路径，仅为该路径下所有文件包含的userID做推荐计算 (该选项可选)  
--itemsFile (path): 指定一个包含了一个或多个存储itemID的文件路径，仅为该路径下所有文件包含的itemID做推荐计算 (该选项可选)  
--filterFile (path): 指定一个路径，该路径下的文件包含了[userID,itemID]值对，userID和itemID用逗号分隔。计算结果将不会为user推荐[userID,itemID]值对中包含的item (该选项可选)  
--booleanData (boolean): 如果输入数据不包含偏好数值，则将该参数设置为true，默认为false  
--maxPrefsPerUser (integer): 在最后计算推荐结果的阶段，针对每一个user使用的偏好数据的最大数量，默认为10  
--minPrefsPerUser (integer): 在相似度计算中，忽略所有偏好数据量少于该值的用户，默认为1  
--maxSimilaritiesPerItem (integer): 针对每个item的相似度最大值，默认为100  
--maxPrefsPerUserInItemSimilarity (integer): 在item相似度计算阶段，针对每个用户考虑的偏好数据最大数量，默认为1000  
--similarityClassname (classname): 向量相似度计算类  
outputPathForSimilarityMatrix：SimilarityMatrix输出目录  
--randomSeed：随机种子 –sequencefileOutput：序列文件输出路径  
--tempDir (path): 存储临时文件的目录，默认为当前用户的home目录下的temp目录  
--startPhase  
--endPhase  
--threshold (double): 忽略相似度低于该阀值的item对

一个例子如下，使用SIMILARITY\_LOGLIKELIHOOD相似度推荐物品：

$ hadoop jar /usr/lib/mahout/mahout-examples-0.9-cdh5.4.0-job.jar org.apache.mahout.cf.taste.hadoop.item.RecommenderJob --input /tmp/mahout/part-00000 --output /tmp/mahout-out -s SIMILARITY\_LOGLIKELIHOOD

自己运行的例子如下：

部分实验数据：

1 25 0.0136316222

1 116 0.0090877481

1 5 0.0045438741

1 23 0.1862988368

1 17 0.0272632444

1 3 1.4122360602

1 11 0.0363509925

1 12 0.4543874068

1 120 0.0027263244

1 93 0.0136316222

1 21 0.0036350993

1 6 0.7688234922

1 47 0.0018175496

1 66 0.0454387407

1 27 0.0254456948

1 44 0.0245369200

1 315 0.0045438741

1 28 0.0545264888

1 138 0.0636142369

1 108 0.0045438741

1 1 7.2695320732

1 85 12.5188577359

1 168 0.0545264888

10 4 0.6772009029

10 6 0.6772009029

10 217 0.0112866817

10 2 1.7607223476

10 1 1.8735891648

100 25 0.4788867023

100 5 0.0047793084

100 17 0.2915378128

100 26 0.0047793084

100 3 0.1194827101

100 11 0.6987348890

100 4 0.6652797301

100 12 0.0047793084

100 30 0.0736013495

100 32 0.5257239247

100 31 0.0076468934

100 37 0.0430137757

100 29 0.0592634242

100 44 0.0009558617

100 13 4.4313747540

100 1 9.5461906101

100 10 0.0439696373

1000 8 0.2902055623

1000 14 0.0483675937

1000 5 0.0725513906

1000 9 0.0725513906

1000 26 0.3869407497

1000 3 0.1451027811

1000 436 0.0120918984

1000 2 3.9177750907

1000 15 2.4304715840

ubuntu@Master:~/data$ mahout org.apache.mahout.cf.taste.hadoop.item.RecommenderJob -i /test/item/ckm\_pre\_result1000000.txt -o /test/item/outputPersonCorr --similarityClassname org.apache.mahout.math.hadoop.similarity.cooccurrence.measures.PearsonCorrelationSimilarity

部分结果：

1 [196:10.482155,14:9.271373,145:8.875873,177:7.779633,360:7.537198,114:7.1917353,635:7.085346,75:6.8879533,235:6.796164,210:6.586777]

2 [386:4.339762,631:4.2806735,194:4.274664,153:4.1018524,362:3.7975848,934:3.422003,195:3.0110214,188:2.7676048,30:2.6990044,746:2.693153]

3 [45:4.2422075,212:4.1731844,270:3.9618893,309:3.960001,204:3.933118,275:3.6498196,321:3.6286862,179:3.487534,240:3.3450491,170:3.28568]

4 [293:4.3900704,746:3.9469879,51:3.3795352,52:3.3444872,312:2.818981,24:2.719058,649:2.2690945,28:2.1947412,196:2.170363,145:2.008545]

5 [590:1.1531421,332:1.1508745,336:1.134177,852:1.1335075,561:1.121143,36:1.1099223,535:1.0878772,129:1.0850264,236:1.0413511,83:1.0349866]

默认情况下，mahout使用的reduce数目为1，这样造成大数据处理时效率较低，可以通过参数mahout执行脚本中的MAHOUT\_OPTS中的-Dmapred.reduce.tasks参数指定reduce数目。

上面命令运行完成之后，会在当前用户的hdfs主目录生成temp目录，该目录可由--tempDir (path)参数设置：

$ hadoop fs -ls temp

Found 10 items

-rw-r--r-- 3 root hadoop 7 2015-06-10 14:42 temp/maxValues.bin

-rw-r--r-- 3 root hadoop 5522717 2015-06-10 14:42 temp/norms.bin

drwxr-xr-x - root hadoop 0 2015-06-10 14:41 temp/notUsed

-rw-r--r-- 3 root hadoop 7 2015-06-10 14:42 temp/numNonZeroEntries.bin

-rw-r--r-- 3 root hadoop 3452222 2015-06-10 14:41 temp/observationsPerColumn.bin

drwxr-xr-x - root hadoop 0 2015-06-10 14:47 temp/pairwiseSimilarity

drwxr-xr-x - root hadoop 0 2015-06-10 14:52 temp/partialMultiply

drwxr-xr-x - root hadoop 0 2015-06-10 14:39 temp/preparePreferenceMatrix

drwxr-xr-x - root hadoop 0 2015-06-10 14:50 temp/similarityMatrix

drwxr-xr-x - root hadoop 0 2015-06-10 14:42 temp/weights

观察yarn的管理界面，该命令会生成9个任务，任务名称依次是：

* PreparePreferenceMatrixJob-ItemIDIndexMapper-Reducer
* PreparePreferenceMatrixJob-ToItemPrefsMapper-Reducer
* PreparePreferenceMatrixJob-ToItemVectorsMapper-Reducer
* RowSimilarityJob-CountObservationsMapper-Reducer
* RowSimilarityJob-VectorNormMapper-Reducer
* RowSimilarityJob-CooccurrencesMapper-Reducer
* RowSimilarityJob-UnsymmetrifyMapper-Reducer
* partialMultiply
* RecommenderJob-PartialMultiplyMapper-Reducer

从任务名称，大概可以知道每个任务在做什么，如果你的输入参数不一样，生成的任务数可能不一样，这个需要测试一下才能确认。

在hdfs上查看输出的结果，用户和推荐结果用\t分隔，推荐结果中物品之间用逗号分隔，物品后面通过冒号连接评分：

843 [10709679:4.8334665,8389878:4.833426,9133835:4.7503786,10366169:4.7503185,9007487:4.750272,8149253:4.7501993,10366165:4.750115,9780049:4.750108,8581254:4.750071,10456307:4.7500467]

6253 [10117445:3.0375953,10340299:3.0340924,8321090:3.0340924,10086615:3.032164,10436801:3.0187714,9668385:3.0141575,8502110:3.013954,10476325:3.0074399,10318667:3.0004222,8320987:3.0003839]

使用Java API方式执行，请参考[Mahout分步式程序开发 基于物品的协同过滤ItemCF](https://link.jianshu.com/?t=http://blog.fens.me/hadoop-mahout-mapreduce-itemcf/)。

在Scala或者Spark中，可以以Java API或者命令方式运行，最后还可以通过Spark来处理推荐的结果，例如：过滤、去重、补足数据，这部分内容不做介绍。

**用Maven构建Mahout项目**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
<http://blog.fens.me/hadoop-mahout-maven-eclipse/>

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-maven-logo.png)

**前言**

基于Hadoop的项目，不管是MapReduce开发，还是Mahout的开发都是在一个复杂的编程环境中开发。Java的环境问题，是困扰着每个程序员的噩梦。Java程序员，不仅要会写Java程序，还要会调linux，会配hadoop，启动hadoop，还要会自己运维。所以，新手想玩起Hadoop真不是件简单的事。

不过，我们可以尽可能的简化环境问题，让程序员只关注于写程序。特别是像算法程序员，把精力投入在算法设计上，要比花时间解决环境问题有价值的多。

**目录**

1. Maven介绍和安装
2. Mahout单机开发环境介绍
3. 用Maven构建Mahout开发环境
4. 用Mahout实现协同过滤userCF
5. 用Mahout实现kmeans
6. 模板项目上传github

**1. Maven介绍和安装**

请参考文章：[用Maven构建Hadoop项目](http://blog.fens.me/hadoop-maven-eclipse/)

**开发环境**

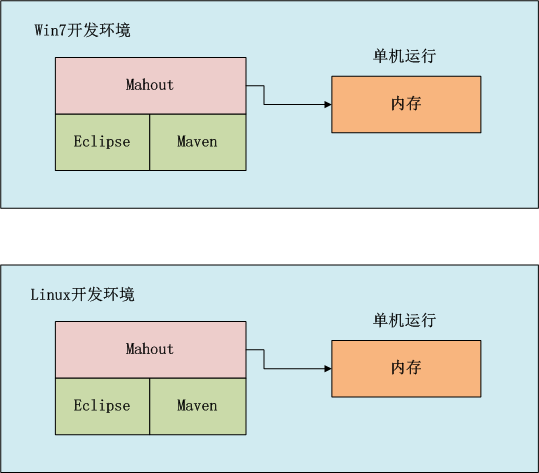
* Win7 64bit
* Java 1.6.0\_45
* Maven 3
* Eclipse Juno Service Release 2
* Mahout 0.6

这里要说明一下mahout的运行版本。

* mahout-0.5, mahout-0.6, mahout-0.7，是基于hadoop-0.20.2x的。
* mahout-0.8, mahout-0.9，是基于hadoop-1.1.x的。
* mahout-0.7，有一次重大升级，去掉了多个算法的单机内存运行，并且了部分API不向前兼容。

注：本文关注于“用Maven构建Mahout的开发环境”，文中的 2个例子都是基于单机的内存实现，因此选择0.6版本。Mahout在Hadoop集群中运行会在下一篇文章介绍。

**2. Mahout单机开发环境介绍**

[](http://blog.fens.me/wp-content/uploads/2013/10/hadoop-mahout-dev.png)

如上图所示，我们可以选择在win中开发，也可以在linux中开发，开发过程我们可以在本地环境进行调试，标配的工具都是Maven和Eclipse。

**3. 用Maven构建Mahout开发环境**

* 1. 用Maven创建一个标准化的Java项目
* 2. 导入项目到eclipse
* 3. 增加mahout依赖，修改pom.xml
* 4. 下载依赖

**1). 用Maven创建一个标准化的Java项目**

~ D:\workspace\java>mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes

-DgroupId=org.conan.mymahout -DartifactId=myMahout -DpackageName=org.conan.mymahout -Dversion=1.0-SNAPSHOT -DinteractiveMode=false

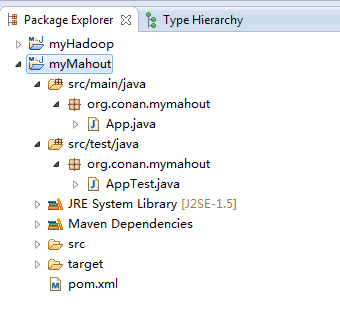
进入项目，执行mvn命令

~ D:\workspace\java>cd myMahout

~ D:\workspace\java\myMahout>mvn clean install

**2). 导入项目到eclipse**

我们创建好了一个基本的maven项目，然后导入到eclipse中。 这里我们最好已安装好了Maven的插件。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-eclipse-folder.png)

**3). 增加mahout依赖，修改pom.xml**

这里我使用hadoop-0.6版本，同时去掉对junit的依赖，修改文件：pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.conan.mymahout</groupId>

<artifactId>myMahout</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>myMahout</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<mahout.version>0.6</mahout.version>

</properties>

<dependencies>

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-core</artifactId>

<version>${mahout.version}</version>

</dependency>

<dependency>

<groupId>org.apache.mahout</groupId>

<artifactId>mahout-integration</artifactId>

<version>${mahout.version}</version>

<exclusions>

<exclusion>

<groupId>org.mortbay.jetty</groupId>

<artifactId>jetty</artifactId>

</exclusion>

<exclusion>

<groupId>org.apache.cassandra</groupId>

<artifactId>cassandra-all</artifactId>

</exclusion>

<exclusion>

<groupId>me.prettyprint</groupId>

<artifactId>hector-core</artifactId>

</exclusion>

</exclusions>

</dependency>

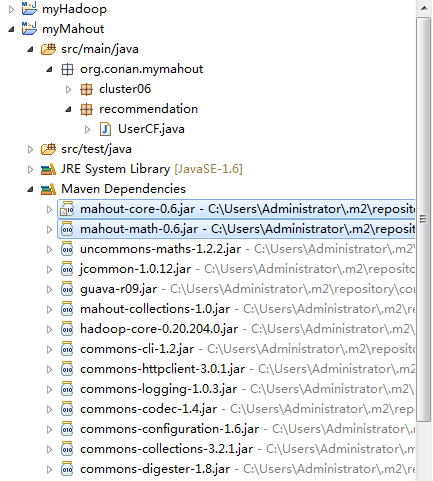
</dependencies>

</project>

**4). 下载依赖**

~ mvn clean install

在eclipse中刷新项目：

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-eclipse-package.png)

项目的依赖程序，被自动加载的库路径下面。

**4. 用Mahout实现协同过滤userCF**

Mahout协同过滤UserCF深度算法剖析，请参考文章：[用R解析Mahout用户推荐协同过滤算法(UserCF)](http://blog.fens.me/r-mahout-usercf/)

实现步骤：

* 1. 准备数据文件: item.csv
* 2. Java程序：UserCF.java
* 3. 运行程序
* 4. 推荐结果解读

**1). 新建数据文件: item.csv**

~ mkdir datafile

~ vi datafile/item.csv

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.5

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

5,103,2.0

5,104,4.0

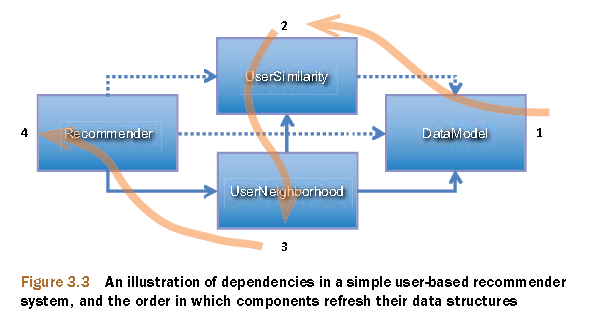
5,105,3.5

5,106,4.0

数据解释：每一行有三列，第一列是用户ID，第二列是物品ID，第三列是用户对物品的打分。

**2). Java程序：UserCF.java**

Mahout协同过滤的数据流，调用过程。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-recommendation-process.png)

上图摘自：Mahout in Action

新建JAVA类：org.conan.mymahout.recommendation.UserCF.java

package org.conan.mymahout.recommendation;

import java.io.File;

import java.io.IOException;

import java.util.List;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;

import org.apache.mahout.cf.taste.impl.neighborhood.NearestNUserNeighborhood;

import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;

import org.apache.mahout.cf.taste.impl.similarity.EuclideanDistanceSimilarity;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;

import org.apache.mahout.cf.taste.recommender.Recommender;

import org.apache.mahout.cf.taste.similarity.UserSimilarity;

public class UserCF {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws IOException, TasteException {

String file = "datafile/item.csv";

DataModel model = new FileDataModel(new File(file));

UserSimilarity user = new EuclideanDistanceSimilarity(model);

NearestNUserNeighborhood neighbor = new NearestNUserNeighborhood(NEIGHBORHOOD\_NUM, user, model);

Recommender r = new GenericUserBasedRecommender(model, neighbor, user);

LongPrimitiveIterator iter = model.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = r.recommend(uid, RECOMMENDER\_NUM);

System.out.printf("uid:%s", uid);

for (RecommendedItem ritem : list) {

System.out.printf("(%s,%f)", ritem.getItemID(), ritem.getValue());

}

System.out.println();

}

}

}

**3). 运行程序**  
控制台输出:

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

uid:1(104,4.274336)(106,4.000000)

uid:2(105,4.055916)

uid:3(103,3.360987)(102,2.773169)

uid:4(102,3.000000)

uid:5

**4). 推荐结果解读**

* 向用户ID1，推荐前二个最相关的物品, 104和106
* 向用户ID2，推荐前二个最相关的物品, 但只有一个105
* 向用户ID3，推荐前二个最相关的物品, 103和102
* 向用户ID4，推荐前二个最相关的物品, 但只有一个102
* 向用户ID5，推荐前二个最相关的物品, 没有符合的

**5. 用Mahout实现kmeans**

* 1. 准备数据文件: randomData.csv
* 2. Java程序：Kmeans.java
* 3. 运行Java程序
* 4. mahout结果解读
* 5. 用R语言实现Kmeans算法
* 6. 比较Mahout和R的结果

**1). 准备数据文件: randomData.csv**

~ vi datafile/randomData.csv

-0.883033363823402,-3.31967192630249

-2.39312626419456,3.34726861118871

2.66976353341256,1.85144276077058

-1.09922906899594,-6.06261735207489

-4.36361936997216,1.90509905380532

-0.00351835125495037,-0.610105996559153

-2.9962958796338,-3.60959839525735

-3.27529418132066,0.0230099799641799

2.17665594420569,6.77290756817957

-2.47862038335637,2.53431833167278

5.53654901906814,2.65089785582474

5.66257474538338,6.86783609641077

-0.558946883114376,1.22332819416237

5.11728525486132,3.74663871584768

1.91240516693351,2.95874731384062

-2.49747101306535,2.05006504756875

3.98781883213459,1.00780938946366

这里只截取了一部分，更多的数据请查看源代码。

注：我是通过R语言生成的randomData.csv

x1<-cbind(x=rnorm(400,1,3),y=rnorm(400,1,3))

x2<-cbind(x=rnorm(300,1,0.5),y=rnorm(300,0,0.5))

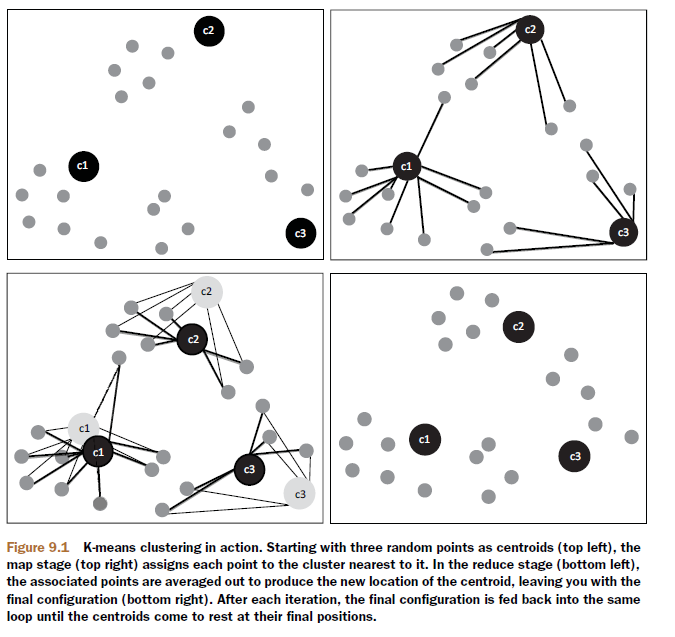
x3<-cbind(x=rnorm(300,0,0.1),y=rnorm(300,2,0.2))

x<-rbind(x1,x2,x3)

write.table(x,file="randomData.csv",sep=",",row.names=FALSE,col.names=FALSE)

**2). Java程序：Kmeans.java**

Mahout中kmeans方法的算法实现过程。

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-kmeans-process.png)

上图摘自：Mahout in Action

新建JAVA类：org.conan.mymahout.cluster06.Kmeans.java

package org.conan.mymahout.cluster06;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import org.apache.mahout.clustering.kmeans.Cluster;

import org.apache.mahout.clustering.kmeans.KMeansClusterer;

import org.apache.mahout.common.distance.EuclideanDistanceMeasure;

import org.apache.mahout.math.Vector;

public class Kmeans {

public static void main(String[] args) throws IOException {

List sampleData = MathUtil.readFileToVector("datafile/randomData.csv");

int k = 3;

double threshold = 0.01;

List randomPoints = MathUtil.chooseRandomPoints(sampleData, k);

for (Vector vector : randomPoints) {

System.out.println("Init Point center: " + vector);

}

List clusters = new ArrayList();

for (int i = 0; i < k; i++) {

clusters.add(new Cluster(randomPoints.get(i), i, new EuclideanDistanceMeasure()));

}

List<List> finalClusters = KMeansClusterer.clusterPoints(sampleData, clusters, new EuclideanDistanceMeasure(), k, threshold);

for (Cluster cluster : finalClusters.get(finalClusters.size() - 1)) {

System.out.println("Cluster id: " + cluster.getId() + " center: " + cluster.getCenter().asFormatString());

}

}

}

**3). 运行Java程序**  
控制台输出:

Init Point center: {0:-0.162693685149196,1:2.19951550286862}

Init Point center: {0:-0.0409782183083317,1:2.09376666042057}

Init Point center: {0:0.158401778474687,1:2.37208412905273}

SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

Cluster id: 0 center: {0:-2.686856800552941,1:1.8939462954763795}

Cluster id: 1 center: {0:0.6334255423230666,1:0.49472852972602105}

Cluster id: 2 center: {0:3.334520309711998,1:3.2758355898247653}

**4). mahout结果解读**

* 1. Init Point center表示，kmeans算法初始时的设置的3个中心点
* 2. Cluster center表示，聚类后找到3个中心点

**5). 用R语言实现Kmeans算法**  
接下来为了让结果更直观，我们再用R语言，进行kmeans实验，操作相同的数据。

R语言代码：

> y<-read.csv(file="randomData.csv",sep=",",header=FALSE)

> cl<-kmeans(y,3,iter.max = 10, nstart = 25)

> cl$centers

V1 V2

1 -0.4323971 2.2852949

2 0.9023786 -0.7011153

3 4.3725463 2.4622609

# 生成聚类中心的图形

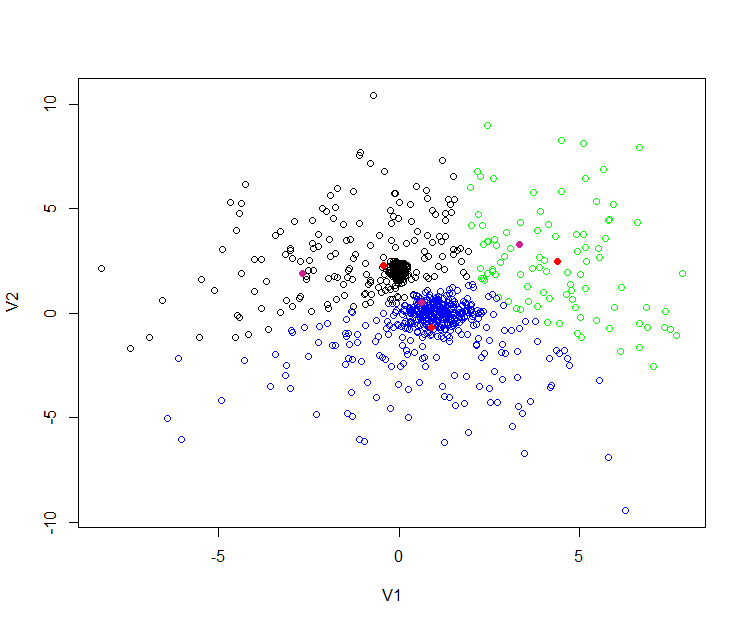
> plot(y, col=c("black","blue","green")[cl$cluster])

> points(cl$centers, col="red", pch = 19)

# 画出Mahout聚类的中心

> mahout<-matrix(c(-2.686856800552941,1.8939462954763795,0.6334255423230666,0.49472852972602105,3.334520309711998,3.2758355898247653),ncol=2,byrow=TRUE)

> points(mahout, col="violetred", pch = 19)

聚类的效果图：  
[](http://blog.fens.me/wp-content/uploads/2013/10/kmeans-center.png)

**6). 比较Mahout和R的结果**  
从上图中，我们看到有 黑，蓝，绿，三种颜色的空心点，这些点就是原始的数据。

3个红色实点，是R语言kmeans后生成的3个中心。  
3个紫色实点，是Mahout的kmeans后生成的3个中心。

R语言和Mahout生成的点，并不是重合的，原因有几点：

* 1. 距离算法不一样：  
  Mahout中，我们用的 “欧氏距离(EuclideanDistanceMeasure)”  
  R语言中，默认是”Hartigan and Wong”
* 2. 初始化的中心是不一样的。
* 3. 最大迭代次数是不一样的。
* 4. 点合并时，判断的”阈值(threshold)”是不一样的。

**6. 模板项目上传github**

<https://github.com/bsspirit/maven_mahout_template/tree/mahout-0.6>

大家可以下载这个项目，做为开发的起点。

~ git clone https://github.com/bsspirit/maven\_mahout\_template

~ git checkout mahout-0.6

**Mahout分步式程序开发 基于物品的协同过滤ItemCF**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
<http://blog.fens.me/hadoop-mahout-mapreduce-itemcf/>

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-hadoop-itemcf.png)

**前言**

Mahout是Hadoop家族一员，从血缘就继承了Hadoop程序的特点，支持HDFS访问和MapReduce分步式算法。随着Mahout的发展，从0.7版本开始，Mahout做了重大的升级。移除了部分算法的单机内存计算，只支持基于Hadoop的MapReduce平行计算。

从这点上，我们能看出Mahout走向大数据，坚持并行化的决心！相信在Hadoop的大框架下，Mahout最终能成为一个大数据的明星产品！

**目录**

1. Mahout开发环境介绍
2. Mahout基于Hadoop的分步环境介绍
3. 用Mahout实现协同过滤ItemCF
4. 模板项目上传github

**1. Mahout开发环境介绍**

在 [用Maven构建Mahout项目](http://blog.fens.me/hadoop-mahout-maven-eclipse/) 文章中，我们已经配置好了基于Maven的Mahout的开发环境，我们将继续完成Mahout的分步式的程序开发。

本文的mahout版本为0.8。

**开发环境：**

* Win7 64bit
* Java 1.6.0\_45
* Maven 3
* Eclipse Juno Service Release 2
* Mahout 0.8
* Hadoop 1.1.2

找到pom.xml，修改mahout版本为0.8

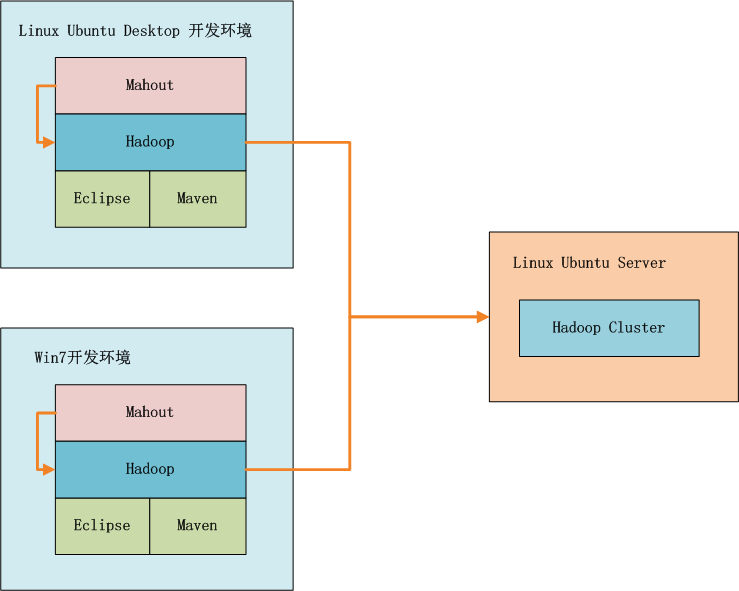
<mahout.version>0.8</mahout.version>

然后，下载依赖库。

~ mvn clean install

由于 org.conan.mymahout.cluster06.Kmeans.java 类代码，是基于mahout-0.6的，所以会报错。我们可以先注释这个文件。

**2. Mahout基于Hadoop的分步环境介绍**

[](http://blog.fens.me/wp-content/uploads/2013/10/hadoop-mahout-cluster-dev.png)

如上图所示，我们可以选择在win7中开发，也可以在linux中开发，开发过程我们可以在本地环境进行调试，标配的工具都是Maven和Eclipse。

Mahout在运行过程中，会把MapReduce的算法程序包，自动发布到Hadoop的集群环境中，这种开发和运行模式，就和真正的生产环境差不多了。

**3. 用Mahout实现协同过滤ItemCF**

实现步骤:

* 1. 准备数据文件: item.csv
* 2. Java程序：HdfsDAO.java
* 3. Java程序：ItemCFHadoop.java
* 4. 运行程序
* 5. 推荐结果解读

**1). 准备数据文件: item.csv**  
上传测试数据到HDFS，单机内存实验请参考文章：[用Maven构建Mahout项目](http://blog.fens.me/hadoop-mahout-maven-eclipse/)

~ hadoop fs -mkdir /user/hdfs/userCF

~ hadoop fs -copyFromLocal /home/conan/datafiles/item.csv /user/hdfs/userCF

~ hadoop fs -cat /user/hdfs/userCF/item.csv

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.5

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

5,103,2.0

5,104,4.0

5,105,3.5

5,106,4.0

**2). Java程序：HdfsDAO.java**  
HdfsDAO.java，是一个HDFS操作的工具，用API实现Hadoop的各种HDFS命令，请参考文章：[Hadoop编程调用HDFS](http://blog.fens.me/hadoop-hdfs-api/)

我们这里会用到HdfsDAO.java类中的一些方法：

HdfsDAO hdfs = new HdfsDAO(HDFS, conf);

hdfs.rmr(inPath);

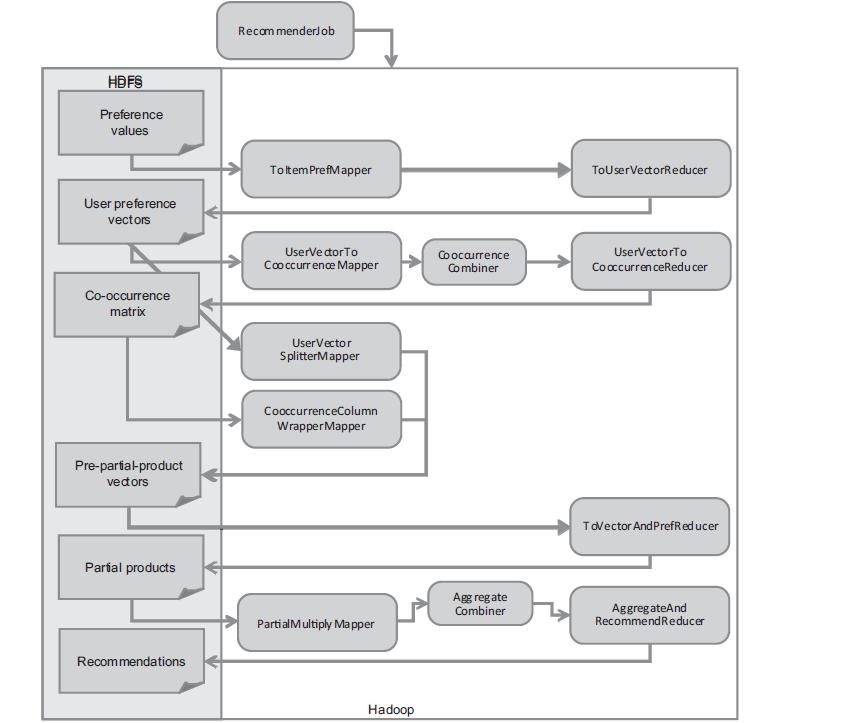
hdfs.mkdirs(inPath);

hdfs.copyFile(localFile, inPath);

hdfs.ls(inPath);

hdfs.cat(inFile);

**3). Java程序：ItemCFHadoop.java**  
用Mahout实现分步式算法，我们看到Mahout in Action中的解释。

[](http://blog.fens.me/wp-content/uploads/2013/10/aglorithm_2.jpg)

实现程序：

package org.conan.mymahout.recommendation;

import org.apache.hadoop.mapred.JobConf;

import org.apache.mahout.cf.taste.hadoop.item.RecommenderJob;

import org.conan.mymahout.hdfs.HdfsDAO;

public class ItemCFHadoop {

private static final String HDFS = "hdfs://192.168.1.210:9000";

public static void main(String[] args) throws Exception {

String localFile = "datafile/item.csv";

String inPath = HDFS + "/user/hdfs/userCF";

String inFile = inPath + "/item.csv";

String outPath = HDFS + "/user/hdfs/userCF/result/";

String outFile = outPath + "/part-r-00000";

String tmpPath = HDFS + "/tmp/" + System.currentTimeMillis();

JobConf conf = config();

HdfsDAO hdfs = new HdfsDAO(HDFS, conf);

hdfs.rmr(inPath);

hdfs.mkdirs(inPath);

hdfs.copyFile(localFile, inPath);

hdfs.ls(inPath);

hdfs.cat(inFile);

StringBuilder sb = new StringBuilder();

sb.append("--input ").append(inPath);

sb.append(" --output ").append(outPath);

sb.append(" --booleanData true");

sb.append(" --similarityClassname org.apache.mahout.math.hadoop.similarity.cooccurrence.measures.EuclideanDistanceSimilarity");

sb.append(" --tempDir ").append(tmpPath);

args = sb.toString().split(" ");

RecommenderJob job = new RecommenderJob();

job.setConf(conf);

job.run(args);

hdfs.cat(outFile);

}

public static JobConf config() {

JobConf conf = new JobConf(ItemCFHadoop.class);

conf.setJobName("ItemCFHadoop");

conf.addResource("classpath:/hadoop/core-site.xml");

conf.addResource("classpath:/hadoop/hdfs-site.xml");

conf.addResource("classpath:/hadoop/mapred-site.xml");

return conf;

}

}

RecommenderJob.java，实际上就是封装了，上面整个图的分步式并行算法的执行过程！如果没有这层封装，我们需要自己去实现图中8个步骤MapReduce算法。

关于上面算法的深度剖析，请参考文章：[R实现MapReduce的协同过滤算法](http://blog.fens.me/rhadoop-mapreduce-rmr/)

**4). 运行程序**  
控制台输出：

Delete: hdfs://192.168.1.210:9000/user/hdfs/userCF

Create: hdfs://192.168.1.210:9000/user/hdfs/userCF

copy from: datafile/item.csv to hdfs://192.168.1.210:9000/user/hdfs/userCF

ls: hdfs://192.168.1.210:9000/user/hdfs/userCF

==========================================================

name: hdfs://192.168.1.210:9000/user/hdfs/userCF/item.csv, folder: false, size: 229

==========================================================

cat: hdfs://192.168.1.210:9000/user/hdfs/userCF/item.csv

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.5

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

5,103,2.0

5,104,4.0

5,105,3.5

5,106,4.0SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".

SLF4J: Defaulting to no-operation (NOP) logger implementation

SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.

2013-10-14 10:26:35 org.apache.hadoop.util.NativeCodeLoader

警告: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

2013-10-14 10:26:35 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:35 org.apache.hadoop.io.compress.snappy.LoadSnappy

警告: Snappy native library not loaded

2013-10-14 10:26:36 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0001

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:36 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:36 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:36 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:36 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:36 org.apache.hadoop.io.compress.CodecPool getCompressor

信息: Got brand-new compressor

2013-10-14 10:26:36 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0001\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:36 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0001\_m\_000000\_0' done.

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:36 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:36 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:36 org.apache.hadoop.io.compress.CodecPool getDecompressor

信息: Got brand-new decompressor

2013-10-14 10:26:36 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 42 bytes

2013-10-14 10:26:36 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0001\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:36 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0001\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:36 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0001\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/preparePreferenceMatrix/itemIDIndex

2013-10-14 10:26:36 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:36 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0001\_r\_000000\_0' done.

2013-10-14 10:26:37 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:37 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0001

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Counters: 19

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=187

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=3287330

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=916

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=3443292

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=645

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=229

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=46

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Map input records=21

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=14

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=84

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=376569856

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=116

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Combine input records=21

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=7

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=7

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Combine output records=7

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=7

2013-10-14 10:26:37 org.apache.hadoop.mapred.Counters log

信息: Map output records=21

2013-10-14 10:26:37 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:37 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0002

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:37 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:37 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:37 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:37 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:37 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0002\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:37 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0002\_m\_000000\_0' done.

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:37 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:37 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:37 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 68 bytes

2013-10-14 10:26:37 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0002\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:37 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0002\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:37 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0002\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/preparePreferenceMatrix/userVectors

2013-10-14 10:26:37 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:37 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0002\_r\_000000\_0' done.

2013-10-14 10:26:38 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:38 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0002

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Counters: 20

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: org.apache.mahout.cf.taste.hadoop.item.ToUserVectorsReducer$Counters

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: USERS=5

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=288

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=6574274

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=1374

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=6887592

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=1120

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=229

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=72

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Map input records=21

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=42

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=63

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=575930368

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=116

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Combine input records=0

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=21

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=5

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Combine output records=0

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=5

2013-10-14 10:26:38 org.apache.hadoop.mapred.Counters log

信息: Map output records=21

2013-10-14 10:26:38 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:38 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0003

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:38 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:38 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:38 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:38 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:38 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0003\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:38 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0003\_m\_000000\_0' done.

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:38 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:38 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:38 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 89 bytes

2013-10-14 10:26:38 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0003\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:38 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0003\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:38 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0003\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/preparePreferenceMatrix/ratingMatrix

2013-10-14 10:26:38 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:38 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0003\_r\_000000\_0' done.

2013-10-14 10:26:39 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:39 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0003

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Counters: 21

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=335

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: org.apache.mahout.cf.taste.hadoop.preparation.ToItemVectorsMapper$Elements

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: USER\_RATINGS\_NEGLECTED=0

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: USER\_RATINGS\_USED=21

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=9861349

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=1950

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=10331958

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=1751

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=288

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=93

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Map input records=5

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=14

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=336

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=775290880

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=157

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Combine input records=21

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=7

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=7

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Combine output records=7

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=7

2013-10-14 10:26:39 org.apache.hadoop.mapred.Counters log

信息: Map output records=21

2013-10-14 10:26:39 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:39 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0004

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:39 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:39 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:39 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:39 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:39 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0004\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:39 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0004\_m\_000000\_0' done.

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:39 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:39 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:39 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 118 bytes

2013-10-14 10:26:39 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0004\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:39 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0004\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:39 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0004\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/weights

2013-10-14 10:26:39 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:39 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0004\_r\_000000\_0' done.

2013-10-14 10:26:40 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:40 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0004

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Counters: 20

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=381

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=13148476

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=2628

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=13780408

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=2551

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=335

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: org.apache.mahout.math.hadoop.similarity.cooccurrence.RowSimilarityJob$Counters

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: ROWS=7

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=122

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Map input records=7

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=16

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=516

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=974651392

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=158

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Combine input records=24

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=8

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=8

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Combine output records=8

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=5

2013-10-14 10:26:40 org.apache.hadoop.mapred.Counters log

信息: Map output records=24

2013-10-14 10:26:40 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:40 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0005

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:40 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:40 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:40 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:40 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:40 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0005\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:40 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0005\_m\_000000\_0' done.

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:40 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:40 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:40 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 121 bytes

2013-10-14 10:26:40 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0005\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:40 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0005\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:40 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0005\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/pairwiseSimilarity

2013-10-14 10:26:40 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:40 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0005\_r\_000000\_0' done.

2013-10-14 10:26:41 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:41 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0005

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Counters: 21

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=392

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=16435577

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=3488

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=17230010

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=3408

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=381

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: org.apache.mahout.math.hadoop.similarity.cooccurrence.RowSimilarityJob$Counters

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: PRUNED\_COOCCURRENCES=0

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: COOCCURRENCES=57

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=125

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Map input records=5

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=14

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=744

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=1174011904

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=129

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Combine input records=21

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=7

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=7

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Combine output records=7

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=7

2013-10-14 10:26:41 org.apache.hadoop.mapred.Counters log

信息: Map output records=21

2013-10-14 10:26:41 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:41 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0006

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:41 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:41 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:41 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:41 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:41 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0006\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:41 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0006\_m\_000000\_0' done.

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:41 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:41 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:41 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 158 bytes

2013-10-14 10:26:41 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0006\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:41 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0006\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:41 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0006\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/similarityMatrix

2013-10-14 10:26:41 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:41 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0006\_r\_000000\_0' done.

2013-10-14 10:26:42 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:42 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0006

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Counters: 19

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=554

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=19722740

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=4342

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=20674772

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=4354

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=392

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=162

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Map input records=7

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=14

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=599

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=1373372416

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=140

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Combine input records=25

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=7

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=7

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Combine output records=7

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=7

2013-10-14 10:26:42 org.apache.hadoop.mapred.Counters log

信息: Map output records=25

2013-10-14 10:26:42 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:42 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:42 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0007

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0007\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0007\_m\_000000\_0' done.

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:42 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0007\_m\_000001\_0 is done. And is in the process of commiting

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0007\_m\_000001\_0' done.

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:42 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 2 sorted segments

2013-10-14 10:26:42 org.apache.hadoop.io.compress.CodecPool getDecompressor

信息: Got brand-new decompressor

2013-10-14 10:26:42 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 2 segments left of total size: 233 bytes

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0007\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0007\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:42 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0007\_r\_000000\_0' to hdfs://192.168.1.210:9000/tmp/1381717594500/partialMultiply

2013-10-14 10:26:42 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:42 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0007\_r\_000000\_0' done.

2013-10-14 10:26:43 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:43 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0007

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Counters: 19

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=572

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=34517913

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=8751

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=36182630

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=7934

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=0

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=241

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Map input records=12

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=56

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=453

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=2558459904

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=665

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Combine input records=0

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=28

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=7

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Combine output records=0

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=7

2013-10-14 10:26:43 org.apache.hadoop.mapred.Counters log

信息: Map output records=28

2013-10-14 10:26:43 org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus

信息: Total input paths to process : 1

2013-10-14 10:26:43 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Running job: job\_local\_0008

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:43 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: io.sort.mb = 100

2013-10-14 10:26:43 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: data buffer = 79691776/99614720

2013-10-14 10:26:43 org.apache.hadoop.mapred.MapTask$MapOutputBuffer

信息: record buffer = 262144/327680

2013-10-14 10:26:43 org.apache.hadoop.mapred.MapTask$MapOutputBuffer flush

信息: Starting flush of map output

2013-10-14 10:26:43 org.apache.hadoop.mapred.MapTask$MapOutputBuffer sortAndSpill

信息: Finished spill 0

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0008\_m\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:43 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0008\_m\_000000\_0' done.

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task initialize

信息: Using ResourceCalculatorPlugin : null

2013-10-14 10:26:43 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:43 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Merging 1 sorted segments

2013-10-14 10:26:43 org.apache.hadoop.mapred.Merger$MergeQueue merge

信息: Down to the last merge-pass, with 1 segments left of total size: 206 bytes

2013-10-14 10:26:43 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task done

信息: Task:attempt\_local\_0008\_r\_000000\_0 is done. And is in the process of commiting

2013-10-14 10:26:43 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息:

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task commit

信息: Task attempt\_local\_0008\_r\_000000\_0 is allowed to commit now

2013-10-14 10:26:43 org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter commitTask

信息: Saved output of task 'attempt\_local\_0008\_r\_000000\_0' to hdfs://192.168.1.210:9000/user/hdfs/userCF/result

2013-10-14 10:26:43 org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate

信息: reduce > reduce

2013-10-14 10:26:43 org.apache.hadoop.mapred.Task sendDone

信息: Task 'attempt\_local\_0008\_r\_000000\_0' done.

2013-10-14 10:26:44 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: map 100% reduce 100%

2013-10-14 10:26:44 org.apache.hadoop.mapred.JobClient monitorAndPrintJob

信息: Job complete: job\_local\_0008

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Counters: 19

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: File Output Format Counters

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Bytes Written=217

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: FileSystemCounters

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_READ=26299802

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_READ=7357

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: FILE\_BYTES\_WRITTEN=27566408

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: HDFS\_BYTES\_WRITTEN=6269

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: File Input Format Counters

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Bytes Read=572

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Map-Reduce Framework

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Map output materialized bytes=210

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Map input records=7

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Reduce shuffle bytes=0

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Spilled Records=42

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Map output bytes=927

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Total committed heap usage (bytes)=1971453952

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: SPLIT\_RAW\_BYTES=137

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Combine input records=0

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Reduce input records=21

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Reduce input groups=5

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Combine output records=0

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Reduce output records=5

2013-10-14 10:26:44 org.apache.hadoop.mapred.Counters log

信息: Map output records=21

cat: hdfs://192.168.1.210:9000/user/hdfs/userCF/result//part-r-00000

1 [104:1.280239,106:1.1462644,105:1.0653841,107:0.33333334]

2 [106:1.560478,105:1.4795978,107:0.69935876]

3 [103:1.2475469,106:1.1944525,102:1.1462644]

4 [102:1.6462644,105:1.5277859,107:0.69935876]

5 [107:1.1993587]

**5). 推荐结果解读**  
我们可以把上面的日志分解析成3个部分解读

* a. 初始化环境
* b. 算法执行
* c. 打印推荐结果

**a. 初始化环境**  
出初HDFS的数据目录和工作目录，并上传数据文件。

Delete: hdfs://192.168.1.210:9000/user/hdfs/userCF

Create: hdfs://192.168.1.210:9000/user/hdfs/userCF

copy from: datafile/item.csv to hdfs://192.168.1.210:9000/user/hdfs/userCF

ls: hdfs://192.168.1.210:9000/user/hdfs/userCF

==========================================================

name: hdfs://192.168.1.210:9000/user/hdfs/userCF/item.csv, folder: false, size: 229

==========================================================

cat: hdfs://192.168.1.210:9000/user/hdfs/userCF/item.csv

**b. 算法执行**  
分别执行，上图中对应的8种MapReduce算法。  
Job complete: job\_local\_0001  
Job complete: job\_local\_0002  
Job complete: job\_local\_0003  
Job complete: job\_local\_0004  
Job complete: job\_local\_0005  
Job complete: job\_local\_0006  
Job complete: job\_local\_0007  
Job complete: job\_local\_0008

**c. 打印推荐结果**

方便我们看到计算后的推荐结果

cat: hdfs://192.168.1.210:9000/user/hdfs/userCF/result//part-r-00000

1 [104:1.280239,106:1.1462644,105:1.0653841,107:0.33333334]

2 [106:1.560478,105:1.4795978,107:0.69935876]

3 [103:1.2475469,106:1.1944525,102:1.1462644]

4 [102:1.6462644,105:1.5277859,107:0.69935876]

5 [107:1.1993587]

**4. 模板项目上传github**

<https://github.com/bsspirit/maven_mahout_template/tree/mahout-0.8>

大家可以下载这个项目，做为开发的起点。

~ git clone https://github.com/bsspirit/maven\_mahout\_template

~ git checkout mahout-0.8

我们完成了基于物品的协同过滤分步式算法实现，下面将继续介绍Mahout的Kmeans的分步式算法实现，请参考文章：[Mahout分步式程序开发 聚类Kmeans](http://blog.fens.me/hadoop-mahout-kmeans/)

**Mahout推荐算法API详解**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
[http://blog.fens.me/mahout-recommendation-api](http://blog.fens.me/mahout-recommendation-api/mahout-recommendation-api)

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-Recommendation.png)

**前言**

用Mahout来构建推荐系统，是一件既简单又困难的事情。简单是因为Mahout完整地封装了“协同过滤”算法，并实现了并行化，提供非常简单的API接口；困难是因为我们不了解算法细节，很难去根据业务的场景进行算法配置和调优。

本文将深入算法API去解释Mahout推荐算法底层的一些事。

**目录**

1. Mahout推荐算法介绍
2. 算法评判标准：召回率与准确率
3. Recommender.java的API接口
4. 测试程序：RecommenderTest.java
5. 基于用户的协同过滤算法UserCF
6. 基于物品的协同过滤算法ItemCF
7. SlopeOne算法
8. KNN Linear interpolation item–based推荐算法
9. SVD推荐算法
10. Tree Cluster-based 推荐算法
11. Mahout推荐算法总结

**1. Mahout推荐算法介绍**

Mahoutt推荐算法，从数据处理能力上，可以划分为2类：

* 单机内存算法实现
* 基于Hadoop的分步式算法实现

**1). 单机内存算法实现**

单机内存算法实现：就是在单机下运行的算法，是由cf.taste项目实现的，像我的们熟悉的UserCF,ItemCF都支持单机内存运行，并且参数可以灵活配置。单机算法的基本实例，请参考文章：[用Maven构建Mahout项目](http://blog.fens.me/hadoop-mahout-maven-eclipse/)

单机内存算法的问题在于，受限于单机的资源。对于中等规模的数据，像1G,10G的数据量，有能力进行计算，但是超过100G的数据量，对于单机来说是不可能完成的任务。

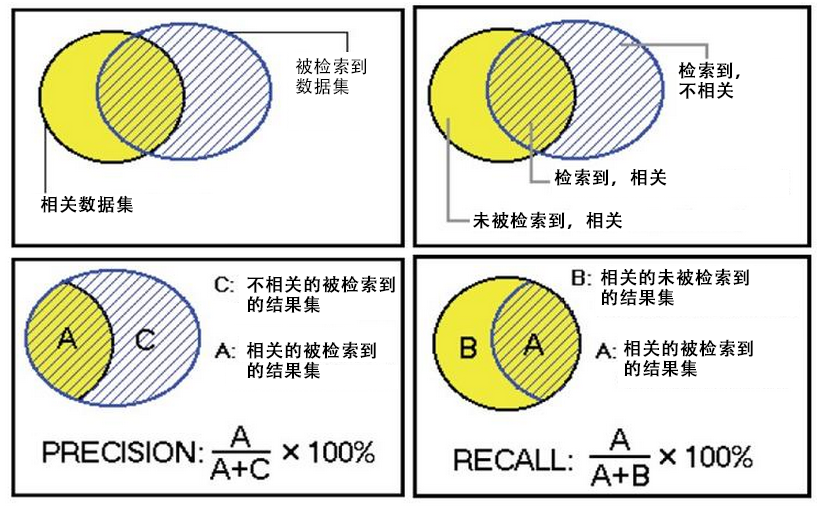
**2). 基于Hadoop的分步式算法实现**

基于Hadoop的分步式算法实现：就是把单机内存算法并行化，把任务分散到多台计算机一起运行。Mahout提供了ItemCF基于Hadoop并行化算法实现。基于Hadoop的分步式算法实现，请参考文章：  
[Mahout分步式程序开发 基于物品的协同过滤ItemCF](http://blog.fens.me/hadoop-mahout-mapreduce-itemcf/)

分步式并行算法的问题在于，如何让单机算法并行化。在单机算法中，我们只需要考虑算法，数据结构，内存，CPU就够了，但是分步式算法还要额外考虑很多的情况，比如多节点的数据合并，数据排序，网路通信的效率，节点宕机重算，数据分步式存储等等的很多问题。

**2. 算法评判标准：召回率(recall)与查准率(precision)**

Mahout提供了2个评估推荐器的指标，查准率和召回率（查全率），这两个指标是搜索引擎中经典的度量方法。

[](http://blog.fens.me/wp-content/uploads/2013/10/precision_recall.png)

相关 不相关

检索到 A C

未检索到 B D

* A：检索到的，相关的 （搜到的也想要的）
* B：未检索到的，但是相关的 （没搜到，然而实际上想要的）
* C：检索到的，但是不相关的 （搜到的但没用的）
* D：未检索到的，也不相关的 （没搜到也没用的）

被检索到的越多越好，这是追求“查全率”，即A/(A+B)，越大越好。  
被检索到的，越相关的越多越好，不相关的越少越好，这是追求“查准率”，即A/(A+C)，越大越好。

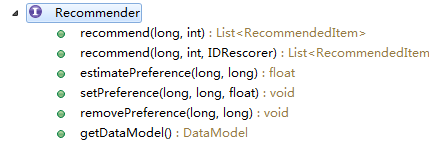
在大规模数据集合中，这两个指标是相互制约的。当希望索引出更多的数据的时候，查准率就会下降，当希望索引更准确的时候，会索引更少的数据。

**3. Recommender的API接口**

**1). 系统环境:**

* Win7 64bit
* Java 1.6.0\_45
* Maven 3
* Eclipse Juno Service Release 2
* Mahout 0.8
* Hadoop 1.1.2

**2). Recommender接口文件：**  
org.apache.mahout.cf.taste.recommender.Recommender.java

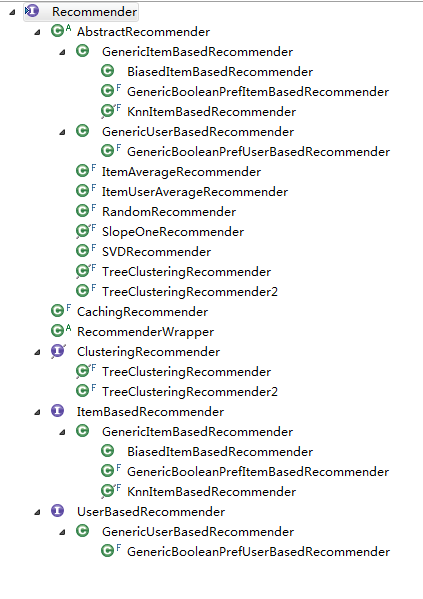
[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-Recommender-class.png)

接口中方法的解释：

* recommend(long userID, int howMany): 获得推荐结果，给userID推荐howMany个Item
* recommend(long userID, int howMany, IDRescorer rescorer): 获得推荐结果，给userID推荐howMany个Item，可以根据rescorer对结构重新排序。
* estimatePreference(long userID, long itemID): 当打分为空，估计用户对物品的打分
* setPreference(long userID, long itemID, float value): 赋值用户，物品，打分
* removePreference(long userID, long itemID): 删除用户对物品的打分
* getDataModel(): 提取推荐数据

通过Recommender接口，我可以猜出核心算法，应该会在子类的estimatePreference()方法中进行实现。

**3). 通过继承关系到Recommender接口的子类：**

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-Recommender-hierarchy.png)

推荐算法实现类：

* GenericUserBasedRecommender: 基于用户的推荐算法
* GenericItemBasedRecommender: 基于物品的推荐算法
* KnnItemBasedRecommender: 基于物品的KNN推荐算法
* SlopeOneRecommender: Slope推荐算法
* SVDRecommender: SVD推荐算法
* TreeClusteringRecommender：TreeCluster推荐算法

下面将分别介绍每种算法的实现。

**4. 测试程序：RecommenderTest.java**

测试数据集：item.csv

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.5

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

5,103,2.0

5,104,4.0

5,105,3.5

5,106,4.0

测试程序：org.conan.mymahout.recommendation.job.RecommenderTest.java

package org.conan.mymahout.recommendation.job;

import java.io.IOException;

import java.util.List;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.eval.RecommenderBuilder;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;

import org.apache.mahout.common.RandomUtils;

public class RecommenderTest {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

RandomUtils.useTestSeed();

String file = "datafile/item.csv";

DataModel dataModel = RecommendFactory.buildDataModel(file);

slopeOne(dataModel);

}

public static void userCF(DataModel dataModel) throws TasteException{}

public static void itemCF(DataModel dataModel) throws TasteException{}

public static void slopeOne(DataModel dataModel) throws TasteException{}

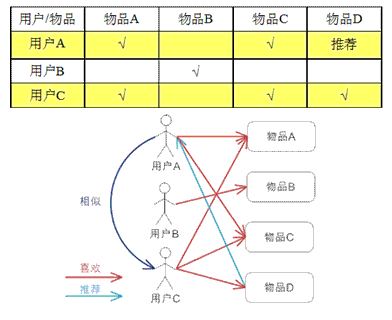
...

每种算法都一个单独的方法进行算法测试，如userCF(),itemCF(),slopeOne()….

**5. 基于用户的协同过滤算法UserCF**

基于用户的协同过滤，通过不同用户对物品的评分来评测用户之间的相似性，基于用户之间的相似性做出推荐。简单来讲就是：给用户推荐和他兴趣相似的其他用户喜欢的物品。

举例说明：

[](http://blog.fens.me/wp-content/uploads/2013/10/image015.gif)

*基于用户的 CF 的基本思想相当简单，基于用户对物品的偏好找到相邻邻居用户，然后将邻居用户喜欢的推荐给当前用户。计算上，就是将一个用户对所有物品的偏好作为一个向量来计算用户之间的相似度，找到 K 邻居后，根据邻居的相似度权重以及他们对物品的偏好，预测当前用户没有偏好的未涉及物品，计算得到一个排序的物品列表作为推荐。图 2 给出了一个例子，对于用户 A，根据用户的历史偏好，这里只计算得到一个邻居 – 用户 C，然后将用户 C 喜欢的物品 D 推荐给用户 A。*

上文中图片和解释文字，摘自： <https://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy2/>

**算法API: org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender**

@Override

public float estimatePreference(long userID, long itemID) throws TasteException {

DataModel model = getDataModel();

Float actualPref = model.getPreferenceValue(userID, itemID);

if (actualPref != null) {

return actualPref;

}

long[] theNeighborhood = neighborhood.getUserNeighborhood(userID);

return doEstimatePreference(userID, theNeighborhood, itemID);

}

protected float doEstimatePreference(long theUserID, long[] theNeighborhood, long itemID) throws TasteException {

if (theNeighborhood.length == 0) {

return Float.NaN;

}

DataModel dataModel = getDataModel();

double preference = 0.0;

double totalSimilarity = 0.0;

int count = 0;

for (long userID : theNeighborhood) {

if (userID != theUserID) {

// See GenericItemBasedRecommender.doEstimatePreference() too

Float pref = dataModel.getPreferenceValue(userID, itemID);

if (pref != null) {

double theSimilarity = similarity.userSimilarity(theUserID, userID);

if (!Double.isNaN(theSimilarity)) {

preference += theSimilarity \* pref;

totalSimilarity += theSimilarity;

count++;

}

}

}

}

// Throw out the estimate if it was based on no data points, of course, but also if based on

// just one. This is a bit of a band-aid on the 'stock' item-based algorithm for the moment.

// The reason is that in this case the estimate is, simply, the user's rating for one item

// that happened to have a defined similarity. The similarity score doesn't matter, and that

// seems like a bad situation.

if (count <= 1) {

return Float.NaN;

}

float estimate = (float) (preference / totalSimilarity);

if (capper != null) {

estimate = capper.capEstimate(estimate);

}

return estimate;

}

测试程序:

public static void userCF(DataModel dataModel) throws TasteException {

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:1.0

Recommender IR Evaluator: [Precision:0.5,Recall:0.5]

uid:1,(104,4.333333)(106,4.000000)

uid:2,(105,4.049678)

uid:3,(103,3.512787)(102,2.747869)

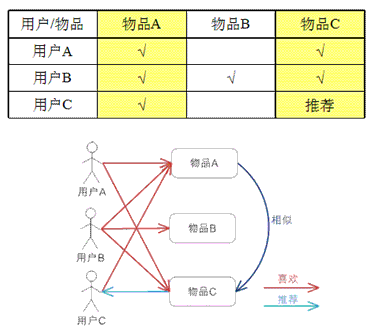
uid:4,(102,3.000000)

用R语言重写UserCF的实现，请参考文章：[用R解析Mahout用户推荐协同过滤算法(UserCF)](http://blog.fens.me/r-mahout-usercf/)

**6. 基于物品的协同过滤算法ItemCF**

基于item的协同过滤，通过用户对不同item的评分来评测item之间的相似性，基于item之间的相似性做出推荐。简单来讲就是：给用户推荐和他之前喜欢的物品相似的物品。

举例说明：

[](http://blog.fens.me/wp-content/uploads/2013/10/image017.gif)

*基于物品的 CF 的原理和基于用户的 CF 类似，只是在计算邻居时采用物品本身，而不是从用户的角度，即基于用户对物品的偏好找到相似的物品，然后根据用户的历史偏好，推荐相似的物品给他。从计算的角度看，就是将所有用户对某个物品的偏好作为一个向量来计算物品之间的相似度，得到物品的相似物品后，根据用户历史的偏好预测当前用户还没有表示偏好的物品，计算得到一个排序的物品列表作为推荐。图 3 给出了一个例子，对于物品 A，根据所有用户的历史偏好，喜欢物品 A 的用户都喜欢物品 C，得出物品 A 和物品 C 比较相似，而用户 C 喜欢物品 A，那么可以推断出用户 C 可能也喜欢物品 C。*

上文中图片和解释文字，摘自： <https://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy2/>

**算法API: org.apache.mahout.cf.taste.impl.recommender.GenericItemBasedRecommender**

@Override

public float estimatePreference(long userID, long itemID) throws TasteException {

PreferenceArray preferencesFromUser = getDataModel().getPreferencesFromUser(userID);

Float actualPref = getPreferenceForItem(preferencesFromUser, itemID);

if (actualPref != null) {

return actualPref;

}

return doEstimatePreference(userID, preferencesFromUser, itemID);

}

protected float doEstimatePreference(long userID, PreferenceArray preferencesFromUser, long itemID)

throws TasteException {

double preference = 0.0;

double totalSimilarity = 0.0;

int count = 0;

double[] similarities = similarity.itemSimilarities(itemID, preferencesFromUser.getIDs());

for (int i = 0; i < similarities.length; i++) {

double theSimilarity = similarities[i];

if (!Double.isNaN(theSimilarity)) {

// Weights can be negative!

preference += theSimilarity \* preferencesFromUser.getValue(i);

totalSimilarity += theSimilarity;

count++;

}

}

// Throw out the estimate if it was based on no data points, of course, but also if based on

// just one. This is a bit of a band-aid on the 'stock' item-based algorithm for the moment.

// The reason is that in this case the estimate is, simply, the user's rating for one item

// that happened to have a defined similarity. The similarity score doesn't matter, and that

// seems like a bad situation.

if (count <= 1) {

return Float.NaN;

}

float estimate = (float) (preference / totalSimilarity);

if (capper != null) {

estimate = capper.capEstimate(estimate);

}

return estimate;

}

测试程序:

public static void itemCF(DataModel dataModel) throws TasteException {

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.8676552772521973

Recommender IR Evaluator: [Precision:0.5,Recall:1.0]

uid:1,(105,3.823529)(104,3.722222)(106,3.478261)

uid:2,(106,2.984848)(105,2.537037)(107,2.000000)

uid:3,(106,3.648649)(102,3.380000)(103,3.312500)

uid:4,(107,4.722222)(105,4.313953)(102,4.025000)

uid:5,(107,3.736842)

**7. SlopeOne算法**

这个算法在mahout-0.8版本中，已经被@Deprecated。

SlopeOne是一种简单高效的协同过滤算法。通过均差计算进行评分。SlopeOne论文下载([PDF](http://www.daniel-lemire.com/fr/documents/publications/lemiremaclachlan_sdm05.pdf))

**1). 举例说明：**  
用户X，Y，Z，对于物品A,B进行打分，如下表，求Z对B的打分是多少？

[](http://blog.fens.me/wp-content/uploads/2013/10/slopeone.png)

Slope one算法认为：平均值可以代替某两个未知个体之间的打分差异，事物A对事物B的平均差是：((5 - 4) + (4 - 2)) / 2 = 1.5，就得到Z对B的打分是，3-1.5 = 1.5。

Slope one算法将用户的评分之间的关系看作简单的线性关系：

Y = mX + b

**2). 平均加权计算：**  
用户X，Y，Z，对于物品A,B,C进行打分，如下表，求Z对A的打分是多少？

[](http://blog.fens.me/wp-content/uploads/2013/10/slopeone2.png)

* 1. 计算A和B的平均差, ((5-3)+(3-4))/2=0.5
* 2. 计算A和C的平均差, (5-2)/1=3
* 3. Z对A的评分，通过AB得到, 2+0.5=2.5
* 4. Z对A的评分，通过AC得到，5+3=8
* 5. 通过加权平均计算Z对A的评分：A和B都有评价的用户数为2,A和C都有评价的用户数为1，权重为别是2和1， (2\*2.5+1\*8)/(2+1)=13/3=4.33

通过这种简单的方式，我们可以快速计算出一个评分项，完成推荐过程！

**算法API: org.apache.mahout.cf.taste.impl.recommender.slopeone.SlopeOneRecommender**

@Override

public float estimatePreference(long userID, long itemID) throws TasteException {

DataModel model = getDataModel();

Float actualPref = model.getPreferenceValue(userID, itemID);

if (actualPref != null) {

return actualPref;

}

return doEstimatePreference(userID, itemID);

}

private float doEstimatePreference(long userID, long itemID) throws TasteException {

double count = 0.0;

double totalPreference = 0.0;

PreferenceArray prefs = getDataModel().getPreferencesFromUser(userID);

RunningAverage[] averages = diffStorage.getDiffs(userID, itemID, prefs);

int size = prefs.length();

for (int i = 0; i < size; i++) {

RunningAverage averageDiff = averages[i];

if (averageDiff != null) {

double averageDiffValue = averageDiff.getAverage();

if (weighted) {

double weight = averageDiff.getCount();

if (stdDevWeighted) {

double stdev = ((RunningAverageAndStdDev) averageDiff).getStandardDeviation();

if (!Double.isNaN(stdev)) {

weight /= 1.0 + stdev;

}

// If stdev is NaN, then it is because count is 1. Because we're weighting by count,

// the weight is already relatively low. We effectively assume stdev is 0.0 here and

// that is reasonable enough. Otherwise, dividing by NaN would yield a weight of NaN

// and disqualify this pref entirely

// (Thanks Daemmon)

}

totalPreference += weight \* (prefs.getValue(i) + averageDiffValue);

count += weight;

} else {

totalPreference += prefs.getValue(i) + averageDiffValue;

count += 1.0;

}

}

}

if (count <= 0.0) {

RunningAverage itemAverage = diffStorage.getAverageItemPref(itemID);

return itemAverage == null ? Float.NaN : (float) itemAverage.getAverage();

} else {

return (float) (totalPreference / count);

}

}

测试程序:

public static void slopeOne(DataModel dataModel) throws TasteException {

RecommenderBuilder recommenderBuilder = RecommendFactory.slopeOneRecommender();

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:1.3333333333333333

Recommender IR Evaluator: [Precision:0.25,Recall:0.5]

uid:1,(105,5.750000)(104,5.250000)(106,4.500000)

uid:2,(105,2.286115)(106,1.500000)

uid:3,(106,2.000000)(102,1.666667)(103,1.625000)

uid:4,(105,4.976859)(102,3.509071)

**8. KNN Linear interpolation item–based推荐算法**

这个算法在mahout-0.8版本中，已经被@Deprecated。

算法来自论文：  
This algorithm is based in the paper of Robert M. Bell and Yehuda Koren in ICDM '07.

(TODO未完)

**算法API: org.apache.mahout.cf.taste.impl.recommender.knn.KnnItemBasedRecommender**

@Override

protected float doEstimatePreference(long theUserID, PreferenceArray preferencesFromUser, long itemID)

throws TasteException {

DataModel dataModel = getDataModel();

int size = preferencesFromUser.length();

FastIDSet possibleItemIDs = new FastIDSet(size);

for (int i = 0; i < size; i++) {

possibleItemIDs.add(preferencesFromUser.getItemID(i));

}

possibleItemIDs.remove(itemID);

List mostSimilar = mostSimilarItems(itemID, possibleItemIDs.iterator(),

neighborhoodSize, null);

long[] theNeighborhood = new long[mostSimilar.size() + 1];

theNeighborhood[0] = -1;

List usersRatedNeighborhood = Lists.newArrayList();

int nOffset = 0;

for (RecommendedItem rec : mostSimilar) {

theNeighborhood[nOffset++] = rec.getItemID();

}

if (!mostSimilar.isEmpty()) {

theNeighborhood[mostSimilar.size()] = itemID;

for (int i = 0; i < theNeighborhood.length; i++) {

PreferenceArray usersNeighborhood = dataModel.getPreferencesForItem(theNeighborhood[i]);

int size1 = usersRatedNeighborhood.isEmpty() ? usersNeighborhood.length() : usersRatedNeighborhood.size();

for (int j = 0; j < size1; j++) {

if (i == 0) {

usersRatedNeighborhood.add(usersNeighborhood.getUserID(j));

} else {

if (j >= usersRatedNeighborhood.size()) {

break;

}

long index = usersRatedNeighborhood.get(j);

if (!usersNeighborhood.hasPrefWithUserID(index) || index == theUserID) {

usersRatedNeighborhood.remove(index);

j--;

}

}

}

}

}

double[] weights = null;

if (!mostSimilar.isEmpty()) {

weights = getInterpolations(itemID, theNeighborhood, usersRatedNeighborhood);

}

int i = 0;

double preference = 0.0;

double totalSimilarity = 0.0;

for (long jitem : theNeighborhood) {

Float pref = dataModel.getPreferenceValue(theUserID, jitem);

if (pref != null) {

double weight = weights[i];

preference += pref \* weight;

totalSimilarity += weight;

}

i++;

}

return totalSimilarity == 0.0 ? Float.NaN : (float) (preference / totalSimilarity);

}

}

测试程序:

public static void itemKNN(DataModel dataModel) throws TasteException {

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemKNNRecommender(itemSimilarity, new NonNegativeQuadraticOptimizer(), 10);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:1.5

Recommender IR Evaluator: [Precision:0.5,Recall:1.0]

uid:1,(107,5.000000)(104,3.501168)(106,3.498198)

uid:2,(105,2.878995)(106,2.878086)(107,2.000000)

uid:3,(103,3.667444)(102,3.667161)(106,3.667019)

uid:4,(107,4.750247)(102,4.122755)(105,4.122709)

uid:5,(107,3.833621)

**9. SVD推荐算法**

(TODO未完)

**算法API: org.apache.mahout.cf.taste.impl.recommender.svd.SVDRecommender**

@Override

public float estimatePreference(long userID, long itemID) throws TasteException {

double[] userFeatures = factorization.getUserFeatures(userID);

double[] itemFeatures = factorization.getItemFeatures(itemID);

double estimate = 0;

for (int feature = 0; feature < userFeatures.length; feature++) {

estimate += userFeatures[feature] \* itemFeatures[feature];

}

return (float) estimate;

}

测试程序:

public static void svd(DataModel dataModel) throws TasteException {

RecommenderBuilder recommenderBuilder = RecommendFactory.svdRecommender(new ALSWRFactorizer(dataModel, 10, 0.05, 10));

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.09990564982096355

Recommender IR Evaluator: [Precision:0.5,Recall:1.0]

uid:1,(104,4.032909)(105,3.390885)(107,1.858541)

uid:2,(105,3.761718)(106,2.951908)(107,1.561116)

uid:3,(103,5.593422)(102,2.458930)(106,-0.091259)

uid:4,(105,4.068329)(102,3.534025)(107,0.206257)

uid:5,(107,0.105169)

**10. Tree Cluster-based 推荐算法**

这个算法在mahout-0.8版本中，已经被@Deprecated。

(TODO未完)

**算法API: org.apache.mahout.cf.taste.impl.recommender.TreeClusteringRecommender**

@Override

public float estimatePreference(long userID, long itemID) throws TasteException {

DataModel model = getDataModel();

Float actualPref = model.getPreferenceValue(userID, itemID);

if (actualPref != null) {

return actualPref;

}

buildClusters();

List topRecsForUser = topRecsByUserID.get(userID);

if (topRecsForUser != null) {

for (RecommendedItem item : topRecsForUser) {

if (itemID == item.getItemID()) {

return item.getValue();

}

}

}

// Hmm, we have no idea. The item is not in the user's cluster

return Float.NaN;

}

测试程序:

public static void treeCluster(DataModel dataModel) throws TasteException {

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

ClusterSimilarity clusterSimilarity = RecommendFactory.clusterSimilarity(RecommendFactory.SIMILARITY.FARTHEST\_NEIGHBOR\_CLUSTER, userSimilarity);

RecommenderBuilder recommenderBuilder = RecommendFactory.treeClusterRecommender(clusterSimilarity, 10);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, true);

}

}

程序输出：

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:NaN

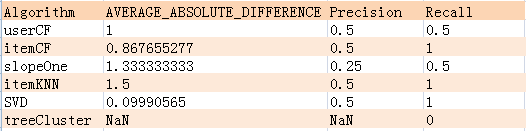
Recommender IR Evaluator: [Precision:NaN,Recall:0.0]

**11. Mahout推荐算法总结**

算法及适用场景：

[](http://blog.fens.me/wp-content/uploads/2013/10/recommender-intro.png)

算法评分的结果：

[](http://blog.fens.me/wp-content/uploads/2013/10/recommender-score.png)

通过对上面几种算法的一平分比较：itemCF,itemKNN,SVD的Rrecision,Recall的评分值是最好的，并且itemCF和SVD的AVERAGE\_ABSOLUTE\_DIFFERENCE是最低的，所以，从算法的角度知道了，哪个算法是更准确的或者会索引到更多的数据集。

另外的一些因素：

* 1. 这3个指标，并不能直接决定计算结果一定itemCF,SVD好
* 2. 各种算法的参数我们并没有调优
* 3. 数据量和数据分布，是影响算法的评分

**程序源代码下载**  
<https://github.com/bsspirit/maven_mahout_template/tree/mahout-0.8/src/main/java/org/conan/mymahout/recommendation/job>

**用Mahout构建职位推荐引擎（单机）**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
<http://blog.fens.me/hadoop-mahout-recommend-job/>

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-recommender-job.png)

**前言**

随着大数据思想实施的落地，推荐系统也开始倍受关注。不光是电商，各种互联网应用都开始应用推荐系统，像搜索，社交网络，音乐，餐饮，地图服务等等。

在以前，我们没有使用推荐算法的时候，我们是通过设置各种约束条件，匹配数据的自然属性呈现给用户，这种就是基于规则的系统。比如，用户购买了一个商品，我们会推荐同类别的其他商品，通过类别属性作为推荐的规则。后来问题就出现了，当用户一次性买了多种类别的不同商品的时候，前一条规则就失败了，我们要进一步设计规则，IT类别优先推荐，价格高的产品优先推荐…..几个回合下来，我们要不停的增加规则，以至于规则有可能的会前后冲突，增加一条新的规则会让推荐结果越来越不好，而且还无法解释是为什么。

推荐算法从另一角度入手，解决了基于规则设置的问题。下面将用Mahout来构建一个职位推荐算法引擎。

**目录**

1. Mahout推荐框架概述
2. 需求分析：职位推荐引擎指标设计
3. 算法模型：推荐算法
4. 架构设计：职位推荐引擎系统架构
5. 程序开发：基于Mahout的推荐算法实现

**1. Mahout推荐系统框架概述**

Mahout框架包含了一套完整的推荐系统引擎，标准化的数据结构，多样的算法实现，简单的开发流程。Mahout推荐的推荐系统引擎是模块化的，分为5个主要部分组成：数据模型，相似度算法，近邻算法，推荐算法，算法评分器。

更详细的介绍，请参考文章：[从源代码剖析Mahout推荐引擎](http://blog.fens.me/mahout-recommend-engine/)

**2. 需求分析：职位推荐引擎指标设计**

下面我们将从一个公司案例出发来全面的解释，如何进行职位推荐引擎指标设计。

案例介绍：  
互联网某职业社交网站，主要产品包括 个人简历展示页，人脉圈，微博及分享链接，职位发布，职位申请，教育培训等。

用户在完成注册后，需要完善自己的个人信息，包括教育背景，工作经历，项目经历，技能专长等等信息。然后，你要告诉网站，你是否想找工作！！当你选择“是”（求职中），网站会从数据库中为你推荐你可能感兴趣的职位。

通过简短的描述，我们可以粗略地看出，这家职业社交网站的定位和主营业务。核心点有2个：

* 用户：尽可能多的保存有效完整的用户资料
* 服务：帮助用户找到工作，帮助猎头和企业找到员工

因此，职位推荐引擎 将成为这个网站的核心功能。

**KPI指标设计**

* 通过推荐带来的职位浏览量: 职位网页的PV
* 通过推荐带来的职位申请量: 职位网页的有效转化

**3. 算法模型：推荐算法**

2个测试数据集：

* pv.csv: 职位被浏览的信息,包括用户ID，职位ID
* job.csv: 职位基本信息,包括职位ID，发布时间，工资标准

**1). pv.csv**

* 2列数据：用户ID，职位ID(userid,jobid)
* 浏览记录:2500条
* 用户数:1000个，用户ID:1-1000
* 职位数:200个，职位ID：1-200

部分数据：

1,11

2,136

2,187

3,165

3,1

3,24

4,8

4,199

5,32

5,100

6,14

7,59

7,147

8,92

9,165

9,80

9,171

10,45

10,31

10,1

10,152

**2). job.csv**

* 3列数据：职位ID，发布时间，工资标准(jobid,create\_date,salary)
* 职位数:200个，职位ID：1-200

部分数据：

1,2013-01-24,5600

2,2011-03-02,5400

3,2011-03-14,8100

4,2012-10-05,2200

5,2011-09-03,14100

6,2011-03-05,6500

7,2012-06-06,37000

8,2013-02-18,5500

9,2010-07-05,7500

10,2010-01-23,6700

11,2011-09-19,5200

12,2010-01-19,29700

13,2013-09-28,6000

14,2013-10-23,3300

15,2010-10-09,2700

16,2010-07-14,5100

17,2010-05-13,29000

18,2010-01-16,21800

19,2013-05-23,5700

20,2011-04-24,5900

为了完成KPI的指标，我们把问题用“技术”语言转化一下：我们需要让职位的推荐结果更准确，从而增加用户的点击。

* 1. 组合使用推荐算法，选出“评估推荐器”验证得分较高的算法
* 2. 人工验证推荐结果
* 3. 职位有时效性，推荐的结果应该是发布半年内的职位
* 4. 工资的标准，应不低于用户浏览职位工资的平均值的80%

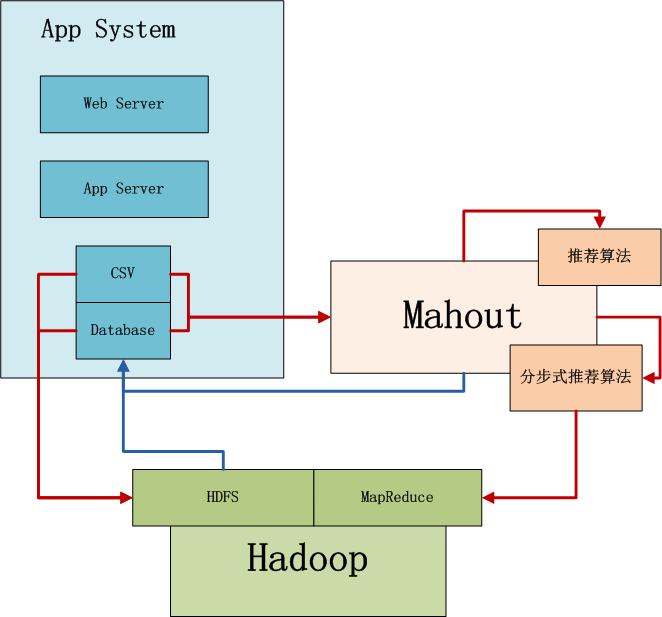
我们选择UserCF,ItemCF,SlopeOne的 3种推荐算法，进行7种组合的测试。

* userCF1: LogLikelihoodSimilarity + NearestNUserNeighborhood + GenericBooleanPrefUserBasedRecommender
* userCF2: CityBlockSimilarity+ NearestNUserNeighborhood + GenericBooleanPrefUserBasedRecommender
* userCF3: UserTanimoto + NearestNUserNeighborhood + GenericBooleanPrefUserBasedRecommender
* itemCF1: LogLikelihoodSimilarity + GenericBooleanPrefItemBasedRecommender
* itemCF2: CityBlockSimilarity+ GenericBooleanPrefItemBasedRecommender
* itemCF3: ItemTanimoto + GenericBooleanPrefItemBasedRecommender
* slopeOne：SlopeOneRecommender

关于的推荐算法的详细介绍，请参考文章：[Mahout推荐算法API详解](http://blog.fens.me/mahout-recommendation-api/)

关于算法的组合的详细介绍，请参考文章：[从源代码剖析Mahout推荐引擎](http://blog.fens.me/mahout-recommend-engine/)

**4. 架构设计：职位推荐引擎系统架构**

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-recommend-job-architect.png)

上图中，左边是Application业务系统，右边是Mahout，下边是Hadoop集群。

* 1. 当数据量不太大时，并且算法复杂，直接选择用Mahout读取CSV或者Database数据，在单机内存中进行计算。Mahout是多线程的应用，会并行使用单机所有系统资源。
* 2. 当数据量很大时，选择并行化算法(ItemCF)，先业务系统的数据导入到Hadoop的HDFS中，然后用Mahout访问HDFS实现算法，这时算法的性能与整个Hadoop集群有关。
* 3. 计算后的结果，保存到数据库中，方便查询

**5. 程序开发：基于Mahout的推荐算法实现**

开发环境mahout版本为0.8。 ，请参考文章：[用Maven构建Mahout项目](http://blog.fens.me/hadoop-mahout-maven-eclipse/)

新建Java类：

* RecommenderEvaluator.java, 选出“评估推荐器”验证得分较高的算法
* RecommenderResult.java, 对指定数量的结果人工比较
* RecommenderFilterOutdateResult.java，排除过期职位
* RecommenderFilterSalaryResult.java，排除工资过低的职位

**1). RecommenderEvaluator.java, 选出“评估推荐器”验证得分较高的算**  
源代码：

public class RecommenderEvaluator {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/job/pv.csv";

DataModel dataModel = RecommendFactory.buildDataModelNoPref(file);

userLoglikelihood(dataModel);

userCityBlock(dataModel);

userTanimoto(dataModel);

itemLoglikelihood(dataModel);

itemCityBlock(dataModel);

itemTanimoto(dataModel);

slopeOne(dataModel);

}

public static RecommenderBuilder userLoglikelihood(DataModel dataModel) throws TasteException, IOException {

System.out.println("userLoglikelihood");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder userCityBlock(DataModel dataModel) throws TasteException, IOException {

System.out.println("userCityBlock");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.CITYBLOCK, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder userTanimoto(DataModel dataModel) throws TasteException, IOException {

System.out.println("userTanimoto");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.TANIMOTO, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemLoglikelihood(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemLoglikelihood");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemCityBlock(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemCityBlock");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.CITYBLOCK, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemTanimoto(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemTanimoto");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.TANIMOTO, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder slopeOne(DataModel dataModel) throws TasteException, IOException {

System.out.println("slopeOne");

RecommenderBuilder recommenderBuilder = RecommendFactory.slopeOneRecommender();

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder knnLoglikelihood(DataModel dataModel) throws TasteException, IOException {

System.out.println("knnLoglikelihood");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemKNNRecommender(itemSimilarity, new NonNegativeQuadraticOptimizer(), 10);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder knnTanimoto(DataModel dataModel) throws TasteException, IOException {

System.out.println("knnTanimoto");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.TANIMOTO, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemKNNRecommender(itemSimilarity, new NonNegativeQuadraticOptimizer(), 10);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder knnCityBlock(DataModel dataModel) throws TasteException, IOException {

System.out.println("knnCityBlock");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.CITYBLOCK, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemKNNRecommender(itemSimilarity, new NonNegativeQuadraticOptimizer(), 10);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder svd(DataModel dataModel) throws TasteException {

System.out.println("svd");

RecommenderBuilder recommenderBuilder = RecommendFactory.svdRecommender(new ALSWRFactorizer(dataModel, 5, 0.05, 10));

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder treeClusterLoglikelihood(DataModel dataModel) throws TasteException {

System.out.println("treeClusterLoglikelihood");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

ClusterSimilarity clusterSimilarity = RecommendFactory.clusterSimilarity(RecommendFactory.SIMILARITY.FARTHEST\_NEIGHBOR\_CLUSTER, userSimilarity);

RecommenderBuilder recommenderBuilder = RecommendFactory.treeClusterRecommender(clusterSimilarity, 3);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

}

运行结果，控制台输出：

userLoglikelihood

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.2741487771272658

Recommender IR Evaluator: [Precision:0.6424242424242422,Recall:0.4098360655737705]

userCityBlock

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.575306732961736

Recommender IR Evaluator: [Precision:0.919580419580419,Recall:0.4371584699453552]

userTanimoto

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.5546485136181523

Recommender IR Evaluator: [Precision:0.6625766871165644,Recall:0.41803278688524603]

itemLoglikelihood

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.5398332608612343

Recommender IR Evaluator: [Precision:0.26229508196721296,Recall:0.26229508196721296]

itemCityBlock

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.9251437840891661

Recommender IR Evaluator: [Precision:0.02185792349726776,Recall:0.02185792349726776]

itemTanimoto

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.9176432856689655

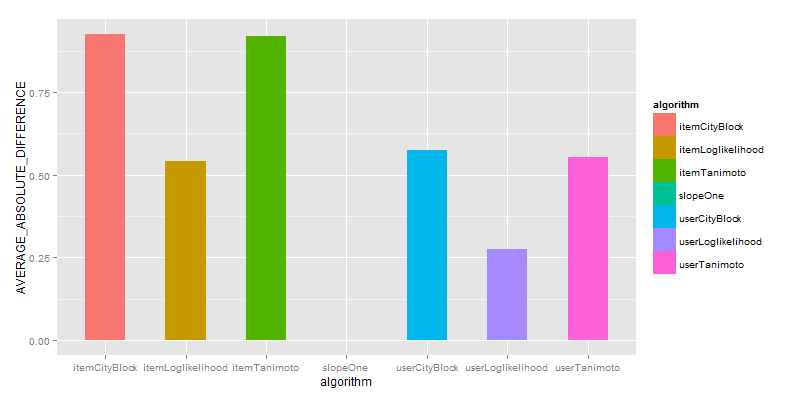
Recommender IR Evaluator: [Precision:0.26229508196721296,Recall:0.26229508196721296]

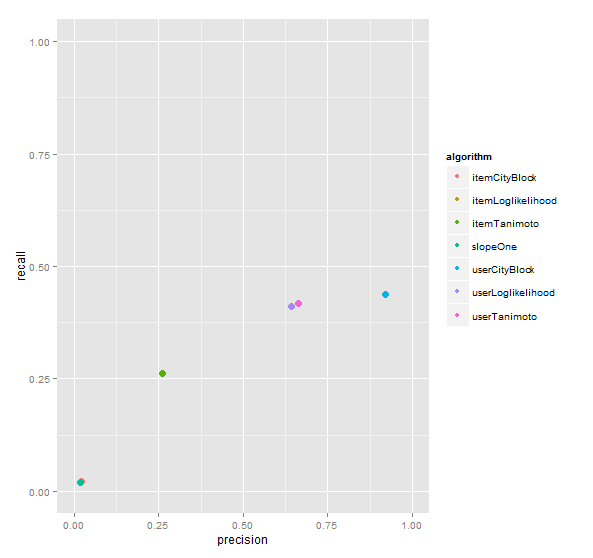
slopeOne

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.0

Recommender IR Evaluator: [Precision:0.01912568306010929,Recall:0.01912568306010929]

可视化“评估推荐器”输出：

[](http://blog.fens.me/wp-content/uploads/2013/10/difference.png)

[](http://blog.fens.me/wp-content/uploads/2013/10/evaluator.png)

UserCityBlock算法评估的结果是最好的，基于UserCF的算法比ItemCF都要好，SlopeOne算法几乎没有得分。

**2). RecommenderResult.java, 对指定数量的结果人工比较**  
为得到差异化结果，我们分别取UserCityBlock,itemLoglikelihood，对推荐结果人工比较。

源代码：

public class RecommenderResult {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/job/pv.csv";

DataModel dataModel = RecommendFactory.buildDataModelNoPref(file);

RecommenderBuilder rb1 = RecommenderEvaluator.userCityBlock(dataModel);

RecommenderBuilder rb2 = RecommenderEvaluator.itemLoglikelihood(dataModel);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

System.out.print("userCityBlock =>");

result(uid, rb1, dataModel);

System.out.print("itemLoglikelihood=>");

result(uid, rb2, dataModel);

}

}

public static void result(long uid, RecommenderBuilder recommenderBuilder, DataModel dataModel) throws TasteException {

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, false);

}

}

控制台输出：只截取部分结果

...

userCityBlock =>uid:968,(61,0.333333)

itemLoglikelihood=>uid:968,(121,1.429362)(153,1.239939)(198,1.207726)

userCityBlock =>uid:969,

itemLoglikelihood=>uid:969,(75,1.326499)(30,0.873100)(85,0.763344)

userCityBlock =>uid:970,

itemLoglikelihood=>uid:970,(13,0.748417)(156,0.748417)(122,0.748417)

userCityBlock =>uid:971,

itemLoglikelihood=>uid:971,(38,2.060951)(104,1.951208)(83,1.941735)

userCityBlock =>uid:972,

itemLoglikelihood=>uid:972,(131,1.378395)(4,1.349386)(87,0.881816)

userCityBlock =>uid:973,

itemLoglikelihood=>uid:973,(196,1.432040)(140,1.398066)(130,1.380335)

userCityBlock =>uid:974,(19,0.200000)

itemLoglikelihood=>uid:974,(145,1.994049)(121,1.794289)(98,1.738027)

...

我们查看uid=974的用户推荐信息：

搜索pv.csv：

> pv[which(pv$userid==974),]

userid jobid

2426 974 106

2427 974 173

2428 974 82

2429 974 188

2430 974 78

搜索job.csv:

> job[job$jobid %in% c(145,121,98,19),]

jobid create\_date salary

19 19 2013-05-23 5700

98 98 2010-01-15 2900

121 121 2010-06-19 5300

145 145 2013-08-02 6800

上面两种算法，推荐的结果都是2010年的职位，这些结果并不是太好，接下来我们要排除过期职位，只保留2013年的职位。

**3).RecommenderFilterOutdateResult.java，排除过期职位**  
源代码：

public class RecommenderFilterOutdateResult {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/job/pv.csv";

DataModel dataModel = RecommendFactory.buildDataModelNoPref(file);

RecommenderBuilder rb1 = RecommenderEvaluator.userCityBlock(dataModel);

RecommenderBuilder rb2 = RecommenderEvaluator.itemLoglikelihood(dataModel);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

System.out.print("userCityBlock =>");

filterOutdate(uid, rb1, dataModel);

System.out.print("itemLoglikelihood=>");

filterOutdate(uid, rb2, dataModel);

}

}

public static void filterOutdate(long uid, RecommenderBuilder recommenderBuilder, DataModel dataModel) throws TasteException, IOException {

Set jobids = getOutdateJobID("datafile/job/job.csv");

IDRescorer rescorer = new JobRescorer(jobids);

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM, rescorer);

RecommendFactory.showItems(uid, list, true);

}

public static Set getOutdateJobID(String file) throws IOException {

BufferedReader br = new BufferedReader(new FileReader(new File(file)));

Set jobids = new HashSet();

String s = null;

while ((s = br.readLine()) != null) {

String[] cols = s.split(",");

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");

Date date = null;

try {

date = df.parse(cols[1]);

if (date.getTime() < df.parse("2013-01-01").getTime()) {

jobids.add(Long.parseLong(cols[0]));

}

} catch (ParseException e) {

e.printStackTrace();

}

}

br.close();

return jobids;

}

}

class JobRescorer implements IDRescorer {

final private Set jobids;

public JobRescorer(Set jobs) {

this.jobids = jobs;

}

@Override

public double rescore(long id, double originalScore) {

return isFiltered(id) ? Double.NaN : originalScore;

}

@Override

public boolean isFiltered(long id) {

return jobids.contains(id);

}

}

控制台输出：只截取部分结果

...

itemLoglikelihood=>uid:965,(200,0.829600)(122,0.748417)(170,0.736340)

userCityBlock =>uid:966,(114,0.250000)

itemLoglikelihood=>uid:966,(114,1.516898)(101,0.864536)(99,0.856057)

userCityBlock =>uid:967,

itemLoglikelihood=>uid:967,(105,0.873100)(114,0.725016)(168,0.707119)

userCityBlock =>uid:968,

itemLoglikelihood=>uid:968,(174,0.735004)(39,0.696716)(185,0.696171)

userCityBlock =>uid:969,

itemLoglikelihood=>uid:969,(197,0.723203)(81,0.710230)(167,0.668358)

userCityBlock =>uid:970,

itemLoglikelihood=>uid:970,(13,0.748417)(122,0.748417)(28,0.736340)

userCityBlock =>uid:971,

itemLoglikelihood=>uid:971,(28,1.540753)(174,1.511881)(39,1.435575)

userCityBlock =>uid:972,

itemLoglikelihood=>uid:972,(14,0.800605)(60,0.794088)(163,0.710230)

userCityBlock =>uid:973,

itemLoglikelihood=>uid:973,(56,0.795529)(13,0.712680)(120,0.701026)

userCityBlock =>uid:974,(19,0.200000)

itemLoglikelihood=>uid:974,(145,1.994049)(89,1.578694)(19,1.435193)

...

我们查看uid=994的用户推荐信息：  
搜索pv.csv：

> pv[which(pv$userid==974),]

userid jobid

2426 974 106

2427 974 173

2428 974 82

2429 974 188

2430 974 78

搜索job.csv:

> job[job$jobid %in% c(19,145,89),]

jobid create\_date salary

19 19 2013-05-23 5700

89 89 2013-06-15 8400

145 145 2013-08-02 6800

排除过期的职位比较，我们发现userCityBlock结果都是19，itemLoglikelihood的第2，3的结果被替换为了得分更低的89和19。

**4).RecommenderFilterSalaryResult.java，排除工资过低的职位**

我们查看uid=994的用户，浏览过的职位。

> job[job$jobid %in% c(106,173,82,188,78),]

jobid create\_date salary

78 78 2012-01-29 6800

82 82 2010-07-05 7500

106 106 2011-04-25 5200

173 173 2013-09-13 5200

188 188 2010-07-14 6000

平均工资为=6140，我们觉得用户的浏览职位的行为，一般不会看比自己现在工资低的职位，因此设计算法，排除工资低于平均工资80%的职位，即排除工资小于4912的推荐职位(6140\*0.8=4912)

大家可以参考上文中RecommenderFilterOutdateResult.java,自行实现。

这样，我们就完成用Mahout构建职位推荐引擎的算法。如果没有Mahout，我们自己写这个算法引擎估计还要花个小半年的时间，善加利用开源技术会帮助我们飞一样的成长！！

原代码下载：  
<https://github.com/bsspirit/maven_mahout_template/tree/mahout-0.8/src/main/java/org/conan/mymahout/recommendation/job>

**用Hadoop构建电影推荐系统（分布式）**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
<http://blog.fens.me/hadoop-mapreduce-recommend/>

[](http://blog.fens.me/wp-content/uploads/2013/10/hadoop-recommand.png)

**前言**

Netflix电影推荐的百万美金比赛，把“推荐”变成了时下最热门的数据挖掘算法之一。也正是由于Netflix的比赛，让企业界和学科界有了更深层次的技术碰撞。引发了各种网站“推荐”热，个性时代已经到来。

**目录**

1. 推荐系统概述
2. 需求分析：推荐系统指标设计
3. 算法模型：Hadoop并行算法
4. 架构设计：推荐系统架构
5. 程序开发：MapReduce程序实现
6. 补充内容：对Step4过程优化

**1. 推荐系统概述**

电子商务网站是个性化推荐系统重要地应用的领域之一，亚马逊就是个性化推荐系统的积极应用者和推广者，亚马逊的推荐系统深入到网站的各类商品，为亚马逊带来了至少30%的销售额。

不光是电商类，推荐系统无处不在。QQ，人人网的好友推荐；新浪微博的你可能感觉兴趣的人；优酷，土豆的电影推荐；豆瓣的图书推荐；大从点评的餐饮推荐；世纪佳缘的相亲推荐；天际网的职业推荐等。

**推荐算法分类：**

按数据使用划分：

* 协同过滤算法：UserCF, ItemCF, ModelCF
* 基于内容的推荐: 用户内容属性和物品内容属性
* 社会化过滤：基于用户的社会网络关系

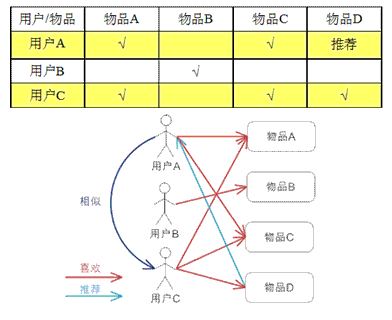
按模型划分：

* 最近邻模型:基于距离的协同过滤算法
* Latent Factor Mode(SVD)：基于矩阵分解的模型
* Graph：图模型，社会网络图模型

**基于用户的协同过滤算法UserCF**

基于用户的协同过滤，通过不同用户对物品的评分来评测用户之间的相似性，基于用户之间的相似性做出推荐。简单来讲就是：给用户推荐和他兴趣相似的其他用户喜欢的物品。

用例说明：

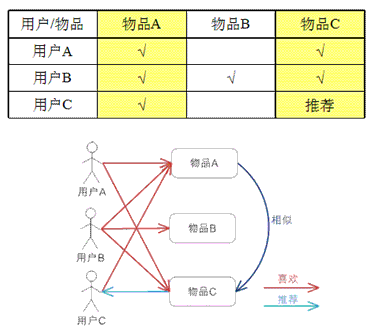
[](http://blog.fens.me/wp-content/uploads/2013/10/image015.gif)

算法实现及使用介绍，请参考文章：[Mahout推荐算法API详解](http://blog.fens.me/hadoop-mapreduce-recommend/mahout-recommendation-api)

**基于物品的协同过滤算法ItemCF**

基于item的协同过滤，通过用户对不同item的评分来评测item之间的相似性，基于item之间的相似性做出推荐。简单来讲就是：给用户推荐和他之前喜欢的物品相似的物品。

用例说明：

[](http://blog.fens.me/wp-content/uploads/2013/10/image017.gif)

算法实现及使用介绍，请参考文章：[Mahout推荐算法API详解](http://blog.fens.me/hadoop-mapreduce-recommend/mahout-recommendation-api)

注：基于物品的协同过滤算法，是目前商用最广泛的推荐算法。

**协同过滤算法实现，分为2个步骤**

* 1. 计算物品之间的相似度
* 2. 根据物品的相似度和用户的历史行为给用户生成推荐列表

有关协同过滤的另一篇文章，请参考：[RHadoop实践系列之三 R实现MapReduce的协同过滤算法](http://blog.fens.me/rhadoop-mapreduce-rmr/)

**2. 需求分析：推荐系统指标设计**

下面我们将从一个公司案例出发来全面的解释，如何进行推荐系统指标设计。

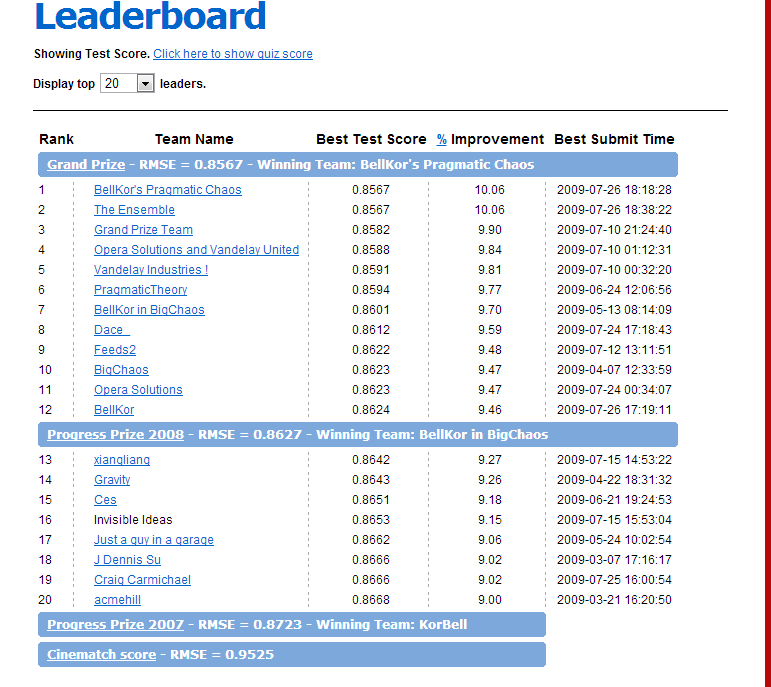
**案例介绍**

Netflix电影推荐百万奖金比赛，<http://www.netflixprize.com/>  
Netflix官方网站：[www.netflix.com](http://www.netflix.com/)

Netflix，2006年组织比赛是的时候，是一家以在线电影租赁为生的公司。他们根据网友对电影的打分来判断用户有可能喜欢什么电影，并结合会员看过的电影以及口味偏好设置做出判断，混搭出各种电影风格的需求。

收集会员的一些信息，为他们指定个性化的电影推荐后，有许多冷门电影竟然进入了候租榜单。从公司的电影资源成本方面考量，热门电影的成本一般较高，如果Netflix公司能够在电影租赁中增加冷门电影的比例，自然能够提升自身盈利能力。

Netflix公司曾宣称60%左右的会员根据推荐名单定制租赁顺序，如果推荐系统不能准确地猜测会员喜欢的电影类型，容易造成多次租借冷门电影而并不符合个人口味的会员流失。为了更高效地为会员推荐电影，Netflix一直致力于不断改进和完善个性化推荐服务，在2006年推出百万美元大奖，无论是谁能最好地优化Netflix推荐算法就可获奖励100万美元。到2009年，奖金被一个7人开发小组夺得，Netflix随后又立即推出第二个百万美金悬赏。这充分说明一套好的推荐算法系统是多么重要，同时又是多么困难。

[](http://blog.fens.me/wp-content/uploads/2013/10/netflix_prize.png)

上图为比赛的各支队伍的排名！

补充说明：

* 1. Netflix的比赛是基于静态数据的，就是给定“训练级”，匹配“结果集”，“结果集”也是提前就做好的，所以这与我们每天运营的系统，其实是不一样的。
* 2. Netflix用于比赛的数据集是小量的，整个全集才666MB，而实际的推荐系统都要基于大量历史数据的，动不动就会上GB,TB等

**Netflix数据下载**  
部分训练集：<http://graphlab.org/wp-content/uploads/2013/07/smallnetflix_mm.train_.gz>  
部分结果集：<http://graphlab.org/wp-content/uploads/2013/07/smallnetflix_mm.validate.gz>  
完整数据集：<http://www.lifecrunch.biz/wp-content/uploads/2011/04/nf_prize_dataset.tar.gz>

所以，我们在真实的环境中设计推荐的时候，要全面考量数据量，算法性能，结果准确度等的指标。

* 推荐算法选型：基于物品的协同过滤算法ItemCF，并行实现
* 数据量：基于Hadoop架构，支持GB,TB,PB级数据量
* 算法检验：可以通过 准确率，召回率，覆盖率，流行度 等指标评判。
* 结果解读：通过ItemCF的定义，合理给出结果解释

**3. 算法模型：Hadoop并行算法**

这里我使用”Mahout In Action”书里，第一章第六节介绍的分步式基于物品的协同过滤算法进行实现。Chapter 6: Distributing recommendation computations

测试数据集:small.csv

1,101,5.0

1,102,3.0

1,103,2.5

2,101,2.0

2,102,2.5

2,103,5.0

2,104,2.0

3,101,2.0

3,104,4.0

3,105,4.5

3,107,5.0

4,101,5.0

4,103,3.0

4,104,4.5

4,106,4.0

5,101,4.0

5,102,3.0

5,103,2.0

5,104,4.0

5,105,3.5

5,106,4.0

每行3个字段，依次是用户ID,电影ID,用户对电影的评分(0-5分，每0.5为一个评分点！)

算法的思想：

* 1. 建立物品的同现矩阵
* 2. 建立用户对物品的评分矩阵
* 3. 矩阵计算推荐结果

**1). 建立物品的同现矩阵**  
按用户分组，找到每个用户所选的物品，单独出现计数及两两一组计数。

[101] [102] [103] [104] [105] [106] [107]

[101] 5 3 4 4 2 2 1

[102] 3 3 3 2 1 1 0

[103] 4 3 4 3 1 2 0

[104] 4 2 3 4 2 2 1

[105] 2 1 1 2 2 1 1

[106] 2 1 2 2 1 2 0

[107] 1 0 0 1 1 0 1

**2). 建立用户对物品的评分矩阵**  
按用户分组，找到每个用户所选的物品及评分

U3

[101] 2.0

[102] 0.0

[103] 0.0

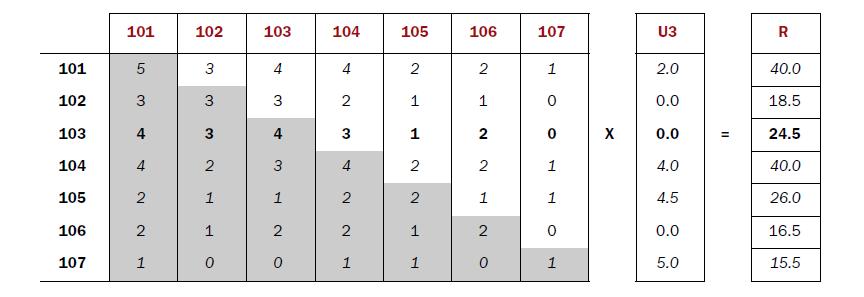
[104] 4.0

[105] 4.5

[106] 0.0

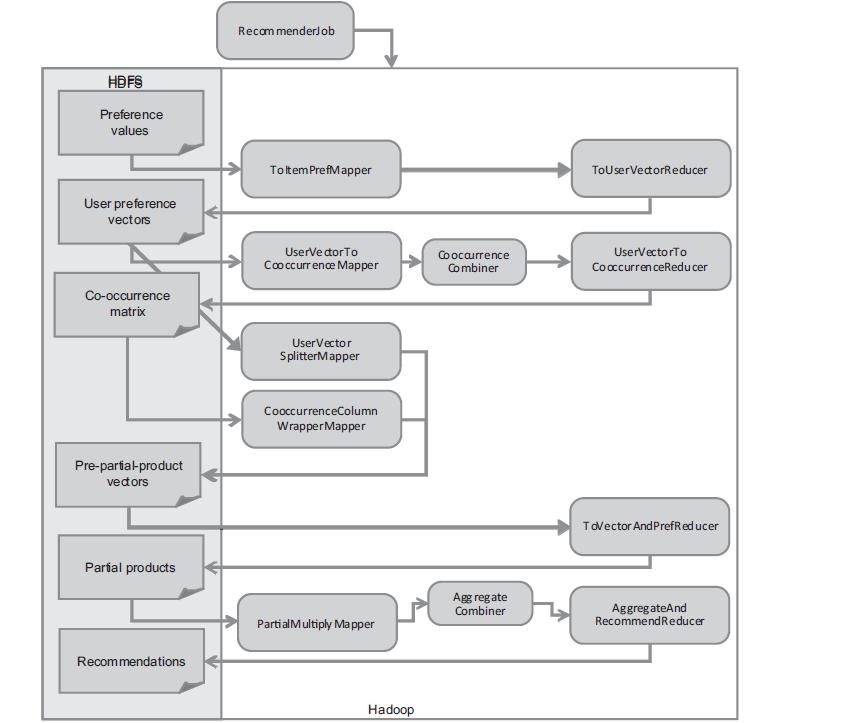
[107] 5.0

**3). 矩阵计算推荐结果**  
同现矩阵\*评分矩阵=推荐结果

[](http://blog.fens.me/wp-content/uploads/2013/10/alogrithm_1.jpg)

图片摘自”Mahout In Action”

**MapReduce任务设计**

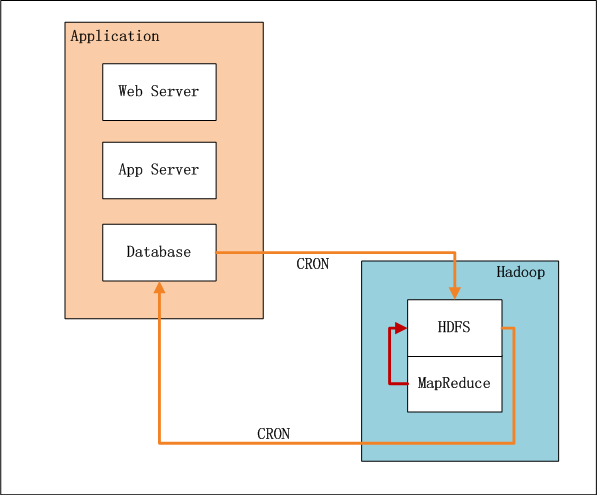
[](http://blog.fens.me/wp-content/uploads/2013/10/aglorithm_2.jpg)

图片摘自”Mahout In Action”

解读MapRduce任务：

* 步骤1: 按用户分组，计算所有物品出现的组合列表，得到用户对物品的评分矩阵
* 步骤2: 对物品组合列表进行计数，建立物品的同现矩阵
* 步骤3: 合并同现矩阵和评分矩阵
* 步骤4: 计算推荐结果列表

**4. 架构设计：推荐系统架构**

[](http://blog.fens.me/wp-content/uploads/2013/10/hadoop-recommand-architect.png)

上图中，左边是Application业务系统，右边是Hadoop的HDFS, MapReduce。

1. 业务系统记录了用户的行为和对物品的打分
2. 设置系统定时器CRON，每xx小时，增量向HDFS导入数据(userid,itemid,value,time)。
3. 完成导入后，设置系统定时器，启动MapReduce程序，运行推荐算法。
4. 完成计算后，设置系统定时器，从HDFS导出推荐结果数据到数据库，方便以后的及时查询。

**5. 程序开发：MapReduce程序实现**

win7的开发环境 和 Hadoop的运行环境 ，请参考文章：[用Maven构建Hadoop项目](http://blog.fens.me/hadoop-maven-eclipse/)

新建Java类：

* Recommend.java，主任务启动程序
* Step1.java，按用户分组，计算所有物品出现的组合列表，得到用户对物品的评分矩阵
* Step2.java，对物品组合列表进行计数，建立物品的同现矩阵
* Step3.java，合并同现矩阵和评分矩阵
* Step4.java，计算推荐结果列表
* HdfsDAO.java，HDFS操作工具类

**1). Recommend.java，主任务启动程序**  
源代码：

package org.conan.myhadoop.recommend;

import java.util.HashMap;

import java.util.Map;

import java.util.regex.Pattern;

import org.apache.hadoop.mapred.JobConf;

public class Recommend {

public static final String HDFS = "hdfs://192.168.1.210:9000";

public static final Pattern DELIMITER = Pattern.compile("[\t,]");

public static void main(String[] args) throws Exception {

Map<String, String> path = new HashMap<String, String>();

path.put("data", "logfile/small.csv");

path.put("Step1Input", HDFS + "/user/hdfs/recommend");

path.put("Step1Output", path.get("Step1Input") + "/step1");

path.put("Step2Input", path.get("Step1Output"));

path.put("Step2Output", path.get("Step1Input") + "/step2");

path.put("Step3Input1", path.get("Step1Output"));

path.put("Step3Output1", path.get("Step1Input") + "/step3\_1");

path.put("Step3Input2", path.get("Step2Output"));

path.put("Step3Output2", path.get("Step1Input") + "/step3\_2");

path.put("Step4Input1", path.get("Step3Output1"));

path.put("Step4Input2", path.get("Step3Output2"));

path.put("Step4Output", path.get("Step1Input") + "/step4");

Step1.run(path);

Step2.run(path);

Step3.run1(path);

Step3.run2(path);

Step4.run(path);

System.exit(0);

}

public static JobConf config() {

JobConf conf = new JobConf(Recommend.class);

conf.setJobName("Recommend");

conf.addResource("classpath:/hadoop/core-site.xml");

conf.addResource("classpath:/hadoop/hdfs-site.xml");

conf.addResource("classpath:/hadoop/mapred-site.xml");

return conf;

}

}

**2). Step1.java，按用户分组，计算所有物品出现的组合列表，得到用户对物品的评分矩阵**

源代码：

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.Iterator;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.RunningJob;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step1 {

public static class Step1\_ToItemPreMapper extends MapReduceBase implements Mapper<Object, Text, IntWritable, Text> {

private final static IntWritable k = new IntWritable();

private final static Text v = new Text();

@Override

public void map(Object key, Text value, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {

String[] tokens = Recommend.DELIMITER.split(value.toString());

int userID = Integer.parseInt(tokens[0]);

String itemID = tokens[1];

String pref = tokens[2];

k.set(userID);

v.set(itemID + ":" + pref);

output.collect(k, v);

}

}

public static class Step1\_ToUserVectorReducer extends MapReduceBase implements Reducer<IntWritable, Text, IntWritable, Text> {

private final static Text v = new Text();

@Override

public void reduce(IntWritable key, Iterator values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {

StringBuilder sb = new StringBuilder();

while (values.hasNext()) {

sb.append("," + values.next());

}

v.set(sb.toString().replaceFirst(",", ""));

output.collect(key, v);

}

}

public static void run(Map<String, String> path) throws IOException {

JobConf conf = Recommend.config();

String input = path.get("Step1Input");

String output = path.get("Step1Output");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(input);

hdfs.mkdirs(input);

hdfs.copyFile(path.get("data"), input);

conf.setMapOutputKeyClass(IntWritable.class);

conf.setMapOutputValueClass(Text.class);

conf.setOutputKeyClass(IntWritable.class);

conf.setOutputValueClass(Text.class);

conf.setMapperClass(Step1\_ToItemPreMapper.class);

conf.setCombinerClass(Step1\_ToUserVectorReducer.class);

conf.setReducerClass(Step1\_ToUserVectorReducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(input));

FileOutputFormat.setOutputPath(conf, new Path(output));

RunningJob job = JobClient.runJob(conf);

while (!job.isComplete()) {

job.waitForCompletion();

}

}

}

计算结果：

~ hadoop fs -cat /user/hdfs/recommend/step1/part-00000

1 102:3.0,103:2.5,101:5.0

2 101:2.0,102:2.5,103:5.0,104:2.0

3 107:5.0,101:2.0,104:4.0,105:4.5

4 101:5.0,103:3.0,104:4.5,106:4.0

5 101:4.0,102:3.0,103:2.0,104:4.0,105:3.5,106:4.0

**3). Step2.java，对物品组合列表进行计数，建立物品的同现矩阵**  
源代码：

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.Iterator;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.RunningJob;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step2 {

public static class Step2\_UserVectorToCooccurrenceMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private final static Text k = new Text();

private final static IntWritable v = new IntWritable(1);

@Override

public void map(LongWritable key, Text values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

for (int i = 1; i < tokens.length; i++) {

String itemID = tokens[i].split(":")[0];

for (int j = 1; j < tokens.length; j++) {

String itemID2 = tokens[j].split(":")[0];

k.set(itemID + ":" + itemID2);

output.collect(k, v);

}

}

}

}

public static class Step2\_UserVectorToConoccurrenceReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

private IntWritable result = new IntWritable();

@Override

public void reduce(Text key, Iterator values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

result.set(sum);

output.collect(key, result);

}

}

public static void run(Map<String, String> path) throws IOException {

JobConf conf = Recommend.config();

String input = path.get("Step2Input");

String output = path.get("Step2Output");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Step2\_UserVectorToCooccurrenceMapper.class);

conf.setCombinerClass(Step2\_UserVectorToConoccurrenceReducer.class);

conf.setReducerClass(Step2\_UserVectorToConoccurrenceReducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(input));

FileOutputFormat.setOutputPath(conf, new Path(output));

RunningJob job = JobClient.runJob(conf);

while (!job.isComplete()) {

job.waitForCompletion();

}

}

}

计算结果：

~ hadoop fs -cat /user/hdfs/recommend/step2/part-00000

101:101 5

101:102 3

101:103 4

101:104 4

101:105 2

101:106 2

101:107 1

102:101 3

102:102 3

102:103 3

102:104 2

102:105 1

102:106 1

103:101 4

103:102 3

103:103 4

103:104 3

103:105 1

103:106 2

104:101 4

104:102 2

104:103 3

104:104 4

104:105 2

104:106 2

104:107 1

105:101 2

105:102 1

105:103 1

105:104 2

105:105 2

105:106 1

105:107 1

106:101 2

106:102 1

106:103 2

106:104 2

106:105 1

106:106 2

107:101 1

107:104 1

107:105 1

107:107 1

**4). Step3.java，合并同现矩阵和评分矩阵**  
源代码：

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.RunningJob;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step3 {

public static class Step31\_UserVectorSplitterMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text> {

private final static IntWritable k = new IntWritable();

private final static Text v = new Text();

@Override

public void map(LongWritable key, Text values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

for (int i = 1; i < tokens.length; i++) {

String[] vector = tokens[i].split(":");

int itemID = Integer.parseInt(vector[0]);

String pref = vector[1];

k.set(itemID);

v.set(tokens[0] + ":" + pref);

output.collect(k, v);

}

}

}

public static void run1(Map<String, String> path) throws IOException {

JobConf conf = Recommend.config();

String input = path.get("Step3Input1");

String output = path.get("Step3Output1");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

conf.setOutputKeyClass(IntWritable.class);

conf.setOutputValueClass(Text.class);

conf.setMapperClass(Step31\_UserVectorSplitterMapper.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(input));

FileOutputFormat.setOutputPath(conf, new Path(output));

RunningJob job = JobClient.runJob(conf);

while (!job.isComplete()) {

job.waitForCompletion();

}

}

public static class Step32\_CooccurrenceColumnWrapperMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private final static Text k = new Text();

private final static IntWritable v = new IntWritable();

@Override

public void map(LongWritable key, Text values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

k.set(tokens[0]);

v.set(Integer.parseInt(tokens[1]));

output.collect(k, v);

}

}

public static void run2(Map<String, String> path) throws IOException {

JobConf conf = Recommend.config();

String input = path.get("Step3Input2");

String output = path.get("Step3Output2");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Step32\_CooccurrenceColumnWrapperMapper.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(input));

FileOutputFormat.setOutputPath(conf, new Path(output));

RunningJob job = JobClient.runJob(conf);

while (!job.isComplete()) {

job.waitForCompletion();

}

}

}

计算结果：

~ hadoop fs -cat /user/hdfs/recommend/step3\_1/part-00000

101 5:4.0

101 1:5.0

101 2:2.0

101 3:2.0

101 4:5.0

102 1:3.0

102 5:3.0

102 2:2.5

103 2:5.0

103 5:2.0

103 1:2.5

103 4:3.0

104 2:2.0

104 5:4.0

104 3:4.0

104 4:4.5

105 3:4.5

105 5:3.5

106 5:4.0

106 4:4.0

107 3:5.0

~ hadoop fs -cat /user/hdfs/recommend/step3\_2/part-00000

101:101 5

101:102 3

101:103 4

101:104 4

101:105 2

101:106 2

101:107 1

102:101 3

102:102 3

102:103 3

102:104 2

102:105 1

102:106 1

103:101 4

103:102 3

103:103 4

103:104 3

103:105 1

103:106 2

104:101 4

104:102 2

104:103 3

104:104 4

104:105 2

104:106 2

104:107 1

105:101 2

105:102 1

105:103 1

105:104 2

105:105 2

105:106 1

105:107 1

106:101 2

106:102 1

106:103 2

106:104 2

106:105 1

106:106 2

107:101 1

107:104 1

107:105 1

107:107 1

**5). Step4.java，计算推荐结果列表**  
源代码：

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.mapred.RunningJob;

import org.apache.hadoop.mapred.TextInputFormat;

import org.apache.hadoop.mapred.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step4 {

public static class Step4\_PartialMultiplyMapper extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text> {

private final static IntWritable k = new IntWritable();

private final static Text v = new Text();

private final static Map<Integer, List> cooccurrenceMatrix = new HashMap<Integer, List>();

@Override

public void map(LongWritable key, Text values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

String[] v1 = tokens[0].split(":");

String[] v2 = tokens[1].split(":");

if (v1.length > 1) {// cooccurrence

int itemID1 = Integer.parseInt(v1[0]);

int itemID2 = Integer.parseInt(v1[1]);

int num = Integer.parseInt(tokens[1]);

List list = null;

if (!cooccurrenceMatrix.containsKey(itemID1)) {

list = new ArrayList();

} else {

list = cooccurrenceMatrix.get(itemID1);

}

list.add(new Cooccurrence(itemID1, itemID2, num));

cooccurrenceMatrix.put(itemID1, list);

}

if (v2.length > 1) {// userVector

int itemID = Integer.parseInt(tokens[0]);

int userID = Integer.parseInt(v2[0]);

double pref = Double.parseDouble(v2[1]);

k.set(userID);

for (Cooccurrence co : cooccurrenceMatrix.get(itemID)) {

v.set(co.getItemID2() + "," + pref \* co.getNum());

output.collect(k, v);

}

}

}

}

public static class Step4\_AggregateAndRecommendReducer extends MapReduceBase implements Reducer<IntWritable, Text, IntWritable, Text> {

private final static Text v = new Text();

@Override

public void reduce(IntWritable key, Iterator values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {

Map<String, Double> result = new HashMap<String, Double>();

while (values.hasNext()) {

String[] str = values.next().toString().split(",");

if (result.containsKey(str[0])) {

result.put(str[0], result.get(str[0]) + Double.parseDouble(str[1]));

} else {

result.put(str[0], Double.parseDouble(str[1]));

}

}

Iterator iter = result.keySet().iterator();

while (iter.hasNext()) {

String itemID = iter.next();

double score = result.get(itemID);

v.set(itemID + "," + score);

output.collect(key, v);

}

}

}

public static void run(Map<String, String> path) throws IOException {

JobConf conf = Recommend.config();

String input1 = path.get("Step4Input1");

String input2 = path.get("Step4Input2");

String output = path.get("Step4Output");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

conf.setOutputKeyClass(IntWritable.class);

conf.setOutputValueClass(Text.class);

conf.setMapperClass(Step4\_PartialMultiplyMapper.class);

conf.setCombinerClass(Step4\_AggregateAndRecommendReducer.class);

conf.setReducerClass(Step4\_AggregateAndRecommendReducer.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(input1), new Path(input2));

FileOutputFormat.setOutputPath(conf, new Path(output));

RunningJob job = JobClient.runJob(conf);

while (!job.isComplete()) {

job.waitForCompletion();

}

}

}

class Cooccurrence {

private int itemID1;

private int itemID2;

private int num;

public Cooccurrence(int itemID1, int itemID2, int num) {

super();

this.itemID1 = itemID1;

this.itemID2 = itemID2;

this.num = num;

}

public int getItemID1() {

return itemID1;

}

public void setItemID1(int itemID1) {

this.itemID1 = itemID1;

}

public int getItemID2() {

return itemID2;

}

public void setItemID2(int itemID2) {

this.itemID2 = itemID2;

}

public int getNum() {

return num;

}

public void setNum(int num) {

this.num = num;

}

}

计算结果：

~ hadoop fs -cat /user/hdfs/recommend/step4/part-00000

1 107,5.0

1 106,18.0

1 105,15.5

1 104,33.5

1 103,39.0

1 102,31.5

1 101,44.0

2 107,4.0

2 106,20.5

2 105,15.5

2 104,36.0

2 103,41.5

2 102,32.5

2 101,45.5

3 107,15.5

3 106,16.5

3 105,26.0

3 104,38.0

3 103,24.5

3 102,18.5

3 101,40.0

4 107,9.5

4 106,33.0

4 105,26.0

4 104,55.0

4 103,53.5

4 102,37.0

4 101,63.0

5 107,11.5

5 106,34.5

5 105,32.0

5 104,59.0

5 103,56.5

5 102,42.5

5 101,68.0

对Step4过程优化，请参考本文最后的补充内容。

**6). HdfsDAO.java，HDFS操作工具类**  
详细解释，请参考文章：[Hadoop编程调用HDFS](http://blog.fens.me/hadoop-hdfs-api/)

源代码：

package org.conan.myhadoop.hdfs;

import java.io.IOException;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileStatus;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IOUtils;

import org.apache.hadoop.mapred.JobConf;

public class HdfsDAO {

private static final String HDFS = "hdfs://192.168.1.210:9000/";

public HdfsDAO(Configuration conf) {

this(HDFS, conf);

}

public HdfsDAO(String hdfs, Configuration conf) {

this.hdfsPath = hdfs;

this.conf = conf;

}

private String hdfsPath;

private Configuration conf;

public static void main(String[] args) throws IOException {

JobConf conf = config();

HdfsDAO hdfs = new HdfsDAO(conf);

hdfs.copyFile("datafile/item.csv", "/tmp/new");

hdfs.ls("/tmp/new");

}

public static JobConf config(){

JobConf conf = new JobConf(HdfsDAO.class);

conf.setJobName("HdfsDAO");

conf.addResource("classpath:/hadoop/core-site.xml");

conf.addResource("classpath:/hadoop/hdfs-site.xml");

conf.addResource("classpath:/hadoop/mapred-site.xml");

return conf;

}

public void mkdirs(String folder) throws IOException {

Path path = new Path(folder);

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

if (!fs.exists(path)) {

fs.mkdirs(path);

System.out.println("Create: " + folder);

}

fs.close();

}

public void rmr(String folder) throws IOException {

Path path = new Path(folder);

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

fs.deleteOnExit(path);

System.out.println("Delete: " + folder);

fs.close();

}

public void ls(String folder) throws IOException {

Path path = new Path(folder);

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

FileStatus[] list = fs.listStatus(path);

System.out.println("ls: " + folder);

System.out.println("==========================================================");

for (FileStatus f : list) {

System.out.printf("name: %s, folder: %s, size: %d\n", f.getPath(), f.isDir(), f.getLen());

}

System.out.println("==========================================================");

fs.close();

}

public void createFile(String file, String content) throws IOException {

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

byte[] buff = content.getBytes();

FSDataOutputStream os = null;

try {

os = fs.create(new Path(file));

os.write(buff, 0, buff.length);

System.out.println("Create: " + file);

} finally {

if (os != null)

os.close();

}

fs.close();

}

public void copyFile(String local, String remote) throws IOException {

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

fs.copyFromLocalFile(new Path(local), new Path(remote));

System.out.println("copy from: " + local + " to " + remote);

fs.close();

}

public void download(String remote, String local) throws IOException {

Path path = new Path(remote);

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

fs.copyToLocalFile(path, new Path(local));

System.out.println("download: from" + remote + " to " + local);

fs.close();

}

public void cat(String remoteFile) throws IOException {

Path path = new Path(remoteFile);

FileSystem fs = FileSystem.get(URI.create(hdfsPath), conf);

FSDataInputStream fsdis = null;

System.out.println("cat: " + remoteFile);

try {

fsdis =fs.open(path);

IOUtils.copyBytes(fsdis, System.out, 4096, false);

} finally {

IOUtils.closeStream(fsdis);

fs.close();

}

}

}

这样我们就自己编程实现了MapReduce化基于物品的协同过滤算法。

RHadoop的实现方案，请参考文章：[RHadoop实践系列之三 R实现MapReduce的协同过滤算法](http://blog.fens.me/rhadoop-mapreduce-rmr/)

Mahout的实现方案，请参考文章：[Mahout分步式程序开发 基于物品的协同过滤ItemCF](http://blog.fens.me/hadoop-mahout-mapreduce-itemcf/)

我已经把整个MapReduce的实现都放到了github上面：  
<https://github.com/bsspirit/maven_hadoop_template/releases/tag/recommend>

**6. 补充内容：对Step4过程优化**

在Step4.java这一步运行过程中，Mapper过程在Step4\_PartialMultiplyMapper类通过分别读取两个input数据，在内存中进行了计算。

这种方式有明显的限制条件：

* a. 两个输入数据集，有严格的读入顺序。由于Hadoop不能指定读入顺序，因此在多节点的Hadoop集群环境，读入顺序有可能会发生错误，造成程序的空指针错误。
* b. 这个计算过程，在内存中实现。如果矩阵过大，会造成单节点的内存不足。

做为优化的方案，我们需要对Step4的过程，实现MapReduce的矩阵乘法，矩阵算法原理请参考文章：[用MapReduce实现矩阵乘法](http://blog.fens.me/hadoop-mapreduce-matrix/)

对Step4优化的实现：把矩阵计算通过两个MapReduce过程实现。

* 矩阵乘法过程类文件：Step4\_Update.java
* 矩阵加法过程类文件：Step4\_Update2.java
* 修改启动程序：Recommend.java

增加文件：Step4\_Update.java

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step4\_Update {

public static class Step4\_PartialMultiplyMapper extends Mapper {

private String flag;// A同现矩阵 or B评分矩阵

@Override

protected void setup(Context context) throws IOException, InterruptedException {

FileSplit split = (FileSplit) context.getInputSplit();

flag = split.getPath().getParent().getName();// 判断读的数据集

// System.out.println(flag);

}

@Override

public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

if (flag.equals("step3\_2")) {// 同现矩阵

String[] v1 = tokens[0].split(":");

String itemID1 = v1[0];

String itemID2 = v1[1];

String num = tokens[1];

Text k = new Text(itemID1);

Text v = new Text("A:" + itemID2 + "," + num);

context.write(k, v);

// System.out.println(k.toString() + " " + v.toString());

} else if (flag.equals("step3\_1")) {// 评分矩阵

String[] v2 = tokens[1].split(":");

String itemID = tokens[0];

String userID = v2[0];

String pref = v2[1];

Text k = new Text(itemID);

Text v = new Text("B:" + userID + "," + pref);

context.write(k, v);

// System.out.println(k.toString() + " " + v.toString());

}

}

}

public static class Step4\_AggregateReducer extends Reducer {

@Override

public void reduce(Text key, Iterable values, Context context) throws IOException, InterruptedException {

System.out.println(key.toString() + ":");

Map mapA = new HashMap();

Map mapB = new HashMap();

for (Text line : values) {

String val = line.toString();

System.out.println(val);

if (val.startsWith("A:")) {

String[] kv = Recommend.DELIMITER.split(val.substring(2));

mapA.put(kv[0], kv[1]);

} else if (val.startsWith("B:")) {

String[] kv = Recommend.DELIMITER.split(val.substring(2));

mapB.put(kv[0], kv[1]);

}

}

double result = 0;

Iterator iter = mapA.keySet().iterator();

while (iter.hasNext()) {

String mapk = iter.next();// itemID

int num = Integer.parseInt(mapA.get(mapk));

Iterator iterb = mapB.keySet().iterator();

while (iterb.hasNext()) {

String mapkb = iterb.next();// userID

double pref = Double.parseDouble(mapB.get(mapkb));

result = num \* pref;// 矩阵乘法相乘计算

Text k = new Text(mapkb);

Text v = new Text(mapk + "," + result);

context.write(k, v);

System.out.println(k.toString() + " " + v.toString());

}

}

}

}

public static void run(Map path) throws IOException, InterruptedException, ClassNotFoundException {

JobConf conf = Recommend.config();

String input1 = path.get("Step5Input1");

String input2 = path.get("Step5Input2");

String output = path.get("Step5Output");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

Job job = new Job(conf);

job.setJarByClass(Step4\_Update.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Step4\_Update.Step4\_PartialMultiplyMapper.class);

job.setReducerClass(Step4\_Update.Step4\_AggregateReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.setInputPaths(job, new Path(input1), new Path(input2));

FileOutputFormat.setOutputPath(job, new Path(output));

job.waitForCompletion(true);

}

}

增加文件：Step4\_Update2.java

package org.conan.myhadoop.recommend;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Step4\_Update2 {

public static class Step4\_RecommendMapper extends Mapper {

@Override

public void map(LongWritable key, Text values, Context context) throws IOException, InterruptedException {

String[] tokens = Recommend.DELIMITER.split(values.toString());

Text k = new Text(tokens[0]);

Text v = new Text(tokens[1]+","+tokens[2]);

context.write(k, v);

}

}

public static class Step4\_RecommendReducer extends Reducer {

@Override

public void reduce(Text key, Iterable values, Context context) throws IOException, InterruptedException {

System.out.println(key.toString() + ":");

Map map = new HashMap();// 结果

for (Text line : values) {

System.out.println(line.toString());

String[] tokens = Recommend.DELIMITER.split(line.toString());

String itemID = tokens[0];

Double score = Double.parseDouble(tokens[1]);

if (map.containsKey(itemID)) {

map.put(itemID, map.get(itemID) + score);// 矩阵乘法求和计算

} else {

map.put(itemID, score);

}

}

Iterator iter = map.keySet().iterator();

while (iter.hasNext()) {

String itemID = iter.next();

double score = map.get(itemID);

Text v = new Text(itemID + "," + score);

context.write(key, v);

}

}

}

public static void run(Map path) throws IOException, InterruptedException, ClassNotFoundException {

JobConf conf = Recommend.config();

String input = path.get("Step6Input");

String output = path.get("Step6Output");

HdfsDAO hdfs = new HdfsDAO(Recommend.HDFS, conf);

hdfs.rmr(output);

Job job = new Job(conf);

job.setJarByClass(Step4\_Update2.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(Step4\_Update2.Step4\_RecommendMapper.class);

job.setReducerClass(Step4\_Update2.Step4\_RecommendReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.setInputPaths(job, new Path(input));

FileOutputFormat.setOutputPath(job, new Path(output));

job.waitForCompletion(true);

}

}

修改Recommend.java

package org.conan.myhadoop.recommend;

import java.util.HashMap;

import java.util.Map;

import java.util.regex.Pattern;

import org.apache.hadoop.mapred.JobConf;

import org.conan.myhadoop.hdfs.HdfsDAO;

public class Recommend {

public static final String HDFS = "hdfs://192.168.1.210:9000";

public static final Pattern DELIMITER = Pattern.compile("[\t,]");

public static void main(String[] args) throws Exception {

Map path = new HashMap();

path.put("data", "logfile/small.csv");

path.put("Step1Input", HDFS + "/user/hdfs/recommend");

path.put("Step1Output", path.get("Step1Input") + "/step1");

path.put("Step2Input", path.get("Step1Output"));

path.put("Step2Output", path.get("Step1Input") + "/step2");

path.put("Step3Input1", path.get("Step1Output"));

path.put("Step3Output1", path.get("Step1Input") + "/step3\_1");

path.put("Step3Input2", path.get("Step2Output"));

path.put("Step3Output2", path.get("Step1Input") + "/step3\_2");

path.put("Step4Input1", path.get("Step3Output1"));

path.put("Step4Input2", path.get("Step3Output2"));

path.put("Step4Output", path.get("Step1Input") + "/step4");

path.put("Step5Input1", path.get("Step3Output1"));

path.put("Step5Input2", path.get("Step3Output2"));

path.put("Step5Output", path.get("Step1Input") + "/step5");

path.put("Step6Input", path.get("Step5Output"));

path.put("Step6Output", path.get("Step1Input") + "/step6");

Step1.run(path);

Step2.run(path);

Step3.run1(path);

Step3.run2(path);

//Step4.run(path);

Step4\_Update.run(path);

Step4\_Update2.run(path);

System.exit(0);

}

public static JobConf config() {

JobConf conf = new JobConf(Recommend.class);

conf.setJobName("Recommand");

conf.addResource("classpath:/hadoop/core-site.xml");

conf.addResource("classpath:/hadoop/hdfs-site.xml");

conf.addResource("classpath:/hadoop/mapred-site.xml");

conf.set("io.sort.mb", "1024");

return conf;

}

}

运行Step4\_Update.java，查看输出结果

~ hadoop fs -cat /user/hdfs/recommend/step5/part-r-00000

3 107,2.0

2 107,2.0

1 107,5.0

5 107,4.0

4 107,5.0

3 106,4.0

2 106,4.0

1 106,10.0

5 106,8.0

4 106,10.0

3 105,4.0

2 105,4.0

1 105,10.0

5 105,8.0

4 105,10.0

3 104,8.0

2 104,8.0

1 104,20.0

5 104,16.0

4 104,20.0

3 103,8.0

2 103,8.0

1 103,20.0

5 103,16.0

4 103,20.0

3 102,6.0

2 102,6.0

1 102,15.0

5 102,12.0

4 102,15.0

3 101,10.0

2 101,10.0

1 101,25.0

5 101,20.0

4 101,25.0

2 106,2.5

1 106,3.0

5 106,3.0

2 105,2.5

1 105,3.0

5 105,3.0

2 104,5.0

1 104,6.0

5 104,6.0

2 103,7.5

1 103,9.0

5 103,9.0

2 102,7.5

1 102,9.0

5 102,9.0

2 101,7.5

1 101,9.0

5 101,9.0

2 106,10.0

1 106,5.0

5 106,4.0

4 106,6.0

2 105,5.0

1 105,2.5

5 105,2.0

4 105,3.0

2 104,15.0

1 104,7.5

5 104,6.0

4 104,9.0

2 103,20.0

1 103,10.0

5 103,8.0

4 103,12.0

2 102,15.0

1 102,7.5

5 102,6.0

4 102,9.0

2 101,20.0

1 101,10.0

5 101,8.0

4 101,12.0

3 107,4.0

2 107,2.0

5 107,4.0

4 107,4.5

3 106,8.0

2 106,4.0

5 106,8.0

4 106,9.0

3 105,8.0

2 105,4.0

5 105,8.0

4 105,9.0

3 104,16.0

2 104,8.0

5 104,16.0

4 104,18.0

3 103,12.0

2 103,6.0

5 103,12.0

4 103,13.5

3 102,8.0

2 102,4.0

5 102,8.0

4 102,9.0

3 101,16.0

2 101,8.0

5 101,16.0

4 101,18.0

3 107,4.5

5 107,3.5

3 106,4.5

5 106,3.5

3 105,9.0

5 105,7.0

3 104,9.0

5 104,7.0

3 103,4.5

5 103,3.5

3 102,4.5

5 102,3.5

3 101,9.0

5 101,7.0

5 106,8.0

4 106,8.0

5 105,4.0

4 105,4.0

5 104,8.0

4 104,8.0

5 103,8.0

4 103,8.0

5 102,4.0

4 102,4.0

5 101,8.0

4 101,8.0

3 107,5.0

3 105,5.0

3 104,5.0

3 101,5.0

运行Step4\_Update2.java，查看输出结果

~ hadoop fs -cat /user/hdfs/recommend/step6/part-r-00000

1 107,5.0

1 106,18.0

1 105,15.5

1 104,33.5

1 103,39.0

1 102,31.5

1 101,44.0

2 107,4.0

2 106,20.5

2 105,15.5

2 104,36.0

2 103,41.5

2 102,32.5

2 101,45.5

3 107,15.5

3 106,16.5

3 105,26.0

3 104,38.0

3 103,24.5

3 102,18.5

3 101,40.0

4 107,9.5

4 106,33.0

4 105,26.0

4 104,55.0

4 103,53.5

4 102,37.0

4 101,63.0

5 107,11.5

5 106,34.5

5 105,32.0

5 104,59.0

5 103,56.5

5 102,42.5

5 101,68.0

这样我们就把原来内存中计算的部分，通过MapReduce实现了，结果与之间Step4的结果一致。

代码已经更新到github，请需要的同学更新查看。  
<https://github.com/bsspirit/maven_hadoop_template/tree/master/src/main/java/org/conan/myhadoop/recommend>

#

**Mahout构建图书推荐系统（单机）**

[Hadoop家族系列文章](http://blog.fens.me/series-hadoop-family/)，主要介绍Hadoop家族产品，常用的项目包括Hadoop, Hive, Pig, HBase, Sqoop, Mahout, Zookeeper, Avro, Ambari, Chukwa，新增加的项目包括，YARN, Hcatalog, Oozie, Cassandra, Hama, Whirr, Flume, Bigtop, Crunch, Hue等。

从2011年开始，中国进入大数据风起云涌的时代，以Hadoop为代表的家族软件，占据了大数据处理的广阔地盘。开源界及厂商，所有数据软件，无一不向Hadoop靠拢。Hadoop也从小众的高富帅领域，变成了大数据开发的标准。在Hadoop原有技术基础之上，出现了Hadoop家族产品，通过“大数据”概念不断创新，推出科技进步。

作为IT界的开发人员，我们也要跟上节奏，抓住机遇，跟着Hadoop一起雄起！

**关于作者：**

* 张丹(Conan), 程序员Java,R,PHP,Javascript
* weibo：@Conan\_Z
* blog: [http://blog.fens.me](http://blog.fens.me/)
* email: bsspirit@gmail.com

**转载请注明出处：**  
<http://blog.fens.me/hadoop-mahout-recommend-book/>

[](http://blog.fens.me/wp-content/uploads/2014/03/mahout-recommendation-book.png)

**前言**

本文是Mahout实现推荐系统的又一案例，用Mahout构建图书推荐系统。与之前的两篇文章，思路上面类似，侧重点在于图书的属性如何利用。本文的数据在自于Amazon网站，由爬虫抓取获得。

**目录**

1. 项目背景
2. 需求分析
3. 数据说明
4. 算法模型
5. 程序开发

**1. 项目背景**

Amazon是最早的电子商务网站之一，以网上图书起家，最后发展成为音像，电子消费品，游戏，生活用品等的综合性电子商务平台。Amazon的推荐系统，是互联网上最早的商品推荐系统，它为Amazon带来了至少30%的流量，和可观的销售利润。

如今推荐系统已经成为电子商务网站的标配，如果还没有推荐系统都不好意思，说自己是做电商的。

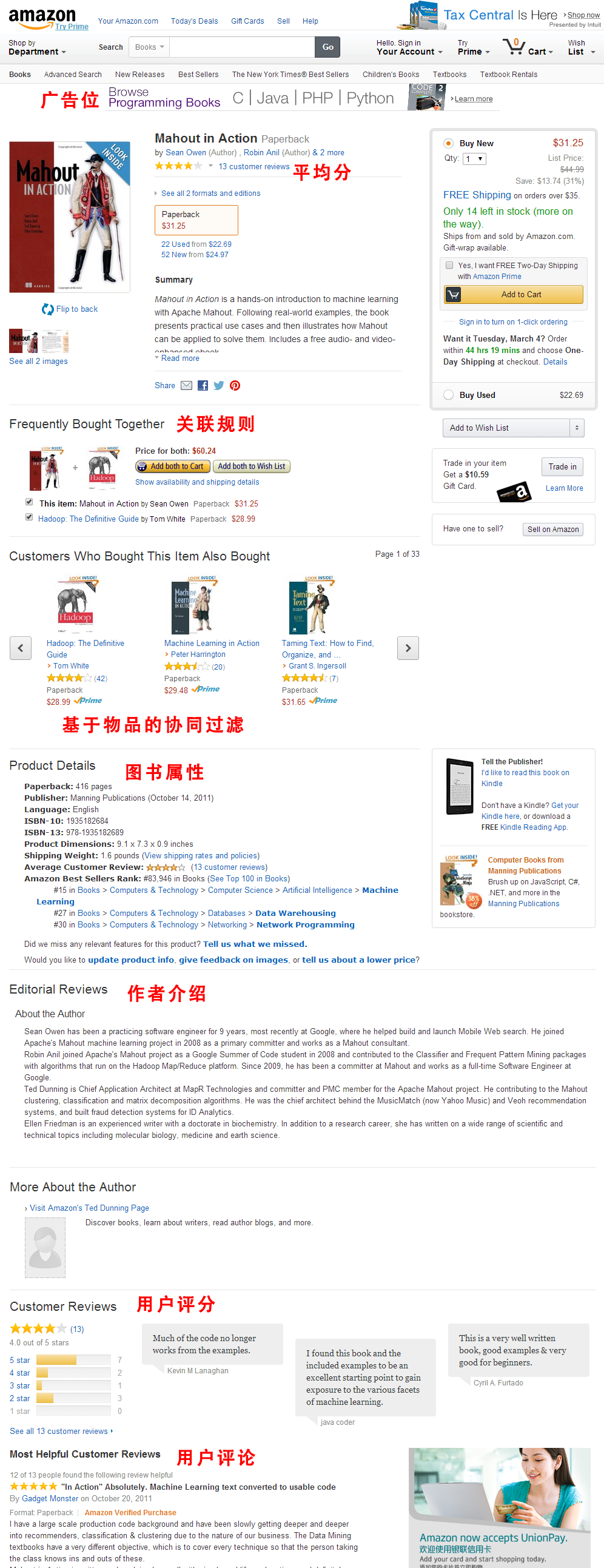
**2. 需求分析**

推荐系统如此重要，我们应该如果理解？

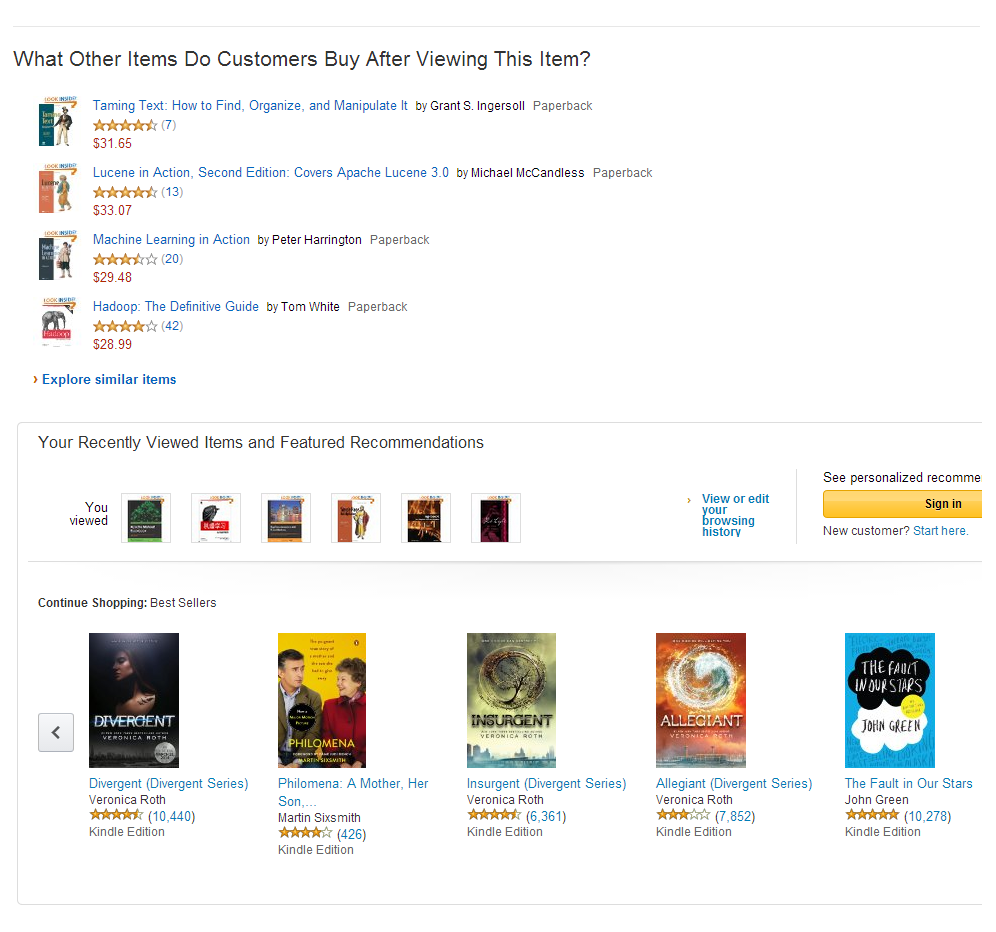
打开Amazon的Mahout In Action图书页面：  
<http://www.amazon.com/Mahout-Action-Sean-Owen/dp/1935182684/ref=pd_sim_b_1?ie=UTF8&refRID=0H4H2NSSR8F34R76E2TP>

网页上的元素：

* 广告位：广告商投放广告的位置，网站可以靠网络广告赚钱，一般是网页最好的位置。
* 平均分：用户对图书的打分
* 关联规则：通过关联规则，推荐位
* 协同过滤：通过基于物品的协同过滤算法的，推荐位
* 图书属性：包括页数，出版社，ISBN，语言等
* 作者介绍：有关作者的介绍，和作者的其他著作
* 用户评分：用户评分行为
* 用户评论：用户评论的内容

[](http://blog.fens.me/wp-content/uploads/2014/03/amazon-book.png)

在网页上，其他的推荐位：

[](http://blog.fens.me/wp-content/uploads/2014/03/amazon-book-2.png)

结合上面2张截图，我们不难发现，推荐对于Amazon的重要性。除了最明显的广告位给了能直接带来利润的广告商，网页中有4处推荐位，分别从不同的维度，用不同的推荐算法，猜用户喜欢的商品。

**3. 数据说明**

2个数据文件：

* rating.csv ：用户评分行为数据
* users.csv ：用户属性数据

**1). book-ratings.csv**

* 3列数据：用户ID，图书ID, 用户对图书的评分
* 记录数: 4000次的图书评分
* 用户数: 200个
* 图书数: 1000个
* 评分：1-10

数据示例

1,565,3

1,807,2

1,201,1

1,557,9

1,987,10

1,59,5

1,305,6

1,153,3

1,139,7

1,875,5

1,722,10

2,977,4

2,806,3

2,654,8

2,21,8

2,662,5

2,437,6

2,576,3

2,141,8

2,311,4

2,101,3

2,540,9

2,87,3

2,65,8

2,501,6

2,710,5

2,331,9

2,542,4

2,757,9

2,590,7

**2). users.csv**

* 3列数据：用户ID，用户性别，用户年龄
* 用户数: 200个
* 用户性别: M为男性，F为女性
* 用户年龄: 11-80岁之间

数据示例

1,M,40

2,M,27

3,M,41

4,F,43

5,F,16

6,M,36

7,F,36

8,F,46

9,M,50

10,M,21

11,F,11

12,M,42

13,F,40

14,F,28

15,M,25

16,M,68

17,M,53

18,F,69

19,F,48

20,F,56

21,F,36

**4. 算法模型**

本文主要介绍Mahout的基于物品的协同过滤模型，其他的算法模型将不再这里解释。

针对上面的数据，我将用7种算法组合进行测试：有关Mahout算法组合的详细解释，请参考文章：[从源代码剖析Mahout推荐引擎](http://blog.fens.me/mahout-recommend-engine/)

7种算法组合

* userCF1: EuclideanSimilarity+ NearestNUserNeighborhood+ GenericUserBasedRecommender
* userCF2: LogLikelihoodSimilarity+ NearestNUserNeighborhood+ GenericUserBasedRecommender
* userCF3: EuclideanSimilarity+ NearestNUserNeighborhood+ GenericBooleanPrefUserBasedRecommender
* itemCF1: EuclideanSimilarity + GenericItemBasedRecommender
* itemCF2: LogLikelihoodSimilarity + GenericItemBasedRecommender
* itemCF3: EuclideanSimilarity + GenericBooleanPrefItemBasedRecommender
* slopeOne：SlopeOneRecommender

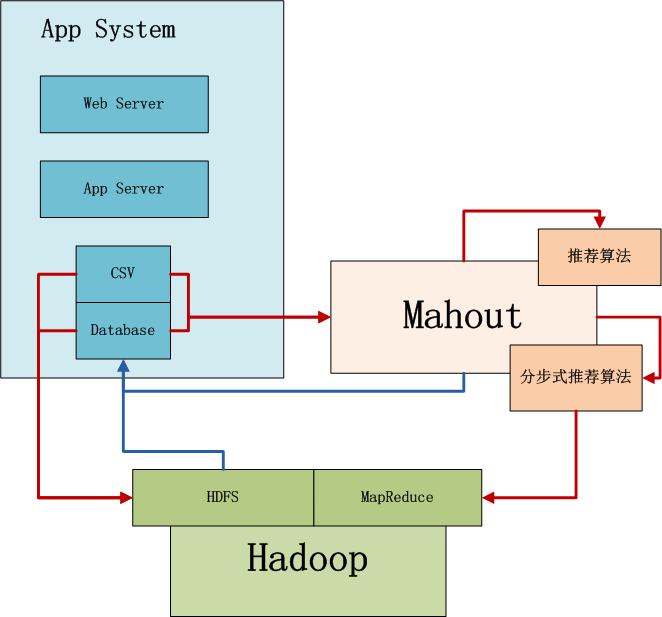
对上面的算法进行算法评估，有关于算法评估的详细解释，请参考文章：[Mahout推荐算法API详解](http://blog.fens.me/mahout-recommendation-api/)

* 查准率:
* 召回率(查全率):

**5. 程序开发**

系统架构：Mahout中推荐过滤算法支持单机算法和分步式算法两种。

* 单机算法: 在单机内存计算，支持多种算法推荐算法，部署运行简单，修正处理数据量有限
* 分步式算法: 基于Hadoop集群运行，支持有限的几种推荐算法，部署运行复杂，支持海量数据

[](http://blog.fens.me/wp-content/uploads/2013/10/mahout-recommend-job-architect.png)

开发环境

* Win7 64bit
* Java 1.6.0\_45
* Maven3
* Eclipse Juno Service Release 2
* Mahout-0.8
* Hadoop-1.1.2

开发环境mahout版本为0.8。 请参考文章：[用Maven构建Mahout项目](http://blog.fens.me/hadoop-mahout-mapreduce-itemcf/)

新建Java类：

* BookEvaluator.java, 选出“评估推荐器”验证得分较高的算法
* BookResult.java, 对指定数量的结果人工比较
* BookFilterGenderResult.java，只保留男性用户的图书列表

**1). BookEvaluator.java, 选出“评估推荐器”验证得分较高的算法**

源代码

package org.conan.mymahout.recommendation.book;

import java.io.IOException;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.eval.RecommenderBuilder;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;

import org.apache.mahout.cf.taste.similarity.ItemSimilarity;

import org.apache.mahout.cf.taste.similarity.UserSimilarity;

public class BookEvaluator {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/book/rating.csv";

DataModel dataModel = RecommendFactory.buildDataModel(file);

userEuclidean(dataModel);

userLoglikelihood(dataModel);

userEuclideanNoPref(dataModel);

itemEuclidean(dataModel);

itemLoglikelihood(dataModel);

itemEuclideanNoPref(dataModel);

slopeOne(dataModel);

}

public static RecommenderBuilder userEuclidean(DataModel dataModel) throws TasteException, IOException {

System.out.println("userEuclidean");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder userLoglikelihood(DataModel dataModel) throws TasteException, IOException {

System.out.println("userLoglikelihood");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder userEuclideanNoPref(DataModel dataModel) throws TasteException, IOException {

System.out.println("userEuclideanNoPref");

UserSimilarity userSimilarity = RecommendFactory.userSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

UserNeighborhood userNeighborhood = RecommendFactory.userNeighborhood(RecommendFactory.NEIGHBORHOOD.NEAREST, userSimilarity, dataModel, NEIGHBORHOOD\_NUM);

RecommenderBuilder recommenderBuilder = RecommendFactory.userRecommender(userSimilarity, userNeighborhood, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemEuclidean(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemEuclidean");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemLoglikelihood(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemLoglikelihood");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.LOGLIKELIHOOD, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, true);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder itemEuclideanNoPref(DataModel dataModel) throws TasteException, IOException {

System.out.println("itemEuclideanNoPref");

ItemSimilarity itemSimilarity = RecommendFactory.itemSimilarity(RecommendFactory.SIMILARITY.EUCLIDEAN, dataModel);

RecommenderBuilder recommenderBuilder = RecommendFactory.itemRecommender(itemSimilarity, false);

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

public static RecommenderBuilder slopeOne(DataModel dataModel) throws TasteException, IOException {

System.out.println("slopeOne");

RecommenderBuilder recommenderBuilder = RecommendFactory.slopeOneRecommender();

RecommendFactory.evaluate(RecommendFactory.EVALUATOR.AVERAGE\_ABSOLUTE\_DIFFERENCE, recommenderBuilder, null, dataModel, 0.7);

RecommendFactory.statsEvaluator(recommenderBuilder, null, dataModel, 2);

return recommenderBuilder;

}

}

控制台输出：

userEuclidean

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:0.33333325386047363

Recommender IR Evaluator: [Precision:0.3010752688172043,Recall:0.08542713567839195]

userLoglikelihood

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:2.5245869159698486

Recommender IR Evaluator: [Precision:0.11764705882352945,Recall:0.017587939698492466]

userEuclideanNoPref

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:4.288461538461536

Recommender IR Evaluator: [Precision:0.09045226130653267,Recall:0.09296482412060306]

itemEuclidean

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:1.408880928305655

Recommender IR Evaluator: [Precision:0.0,Recall:0.0]

itemLoglikelihood

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:2.448554412835434

Recommender IR Evaluator: [Precision:0.0,Recall:0.0]

itemEuclideanNoPref

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:2.5665197873957957

Recommender IR Evaluator: [Precision:0.6005025125628134,Recall:0.6055276381909548]

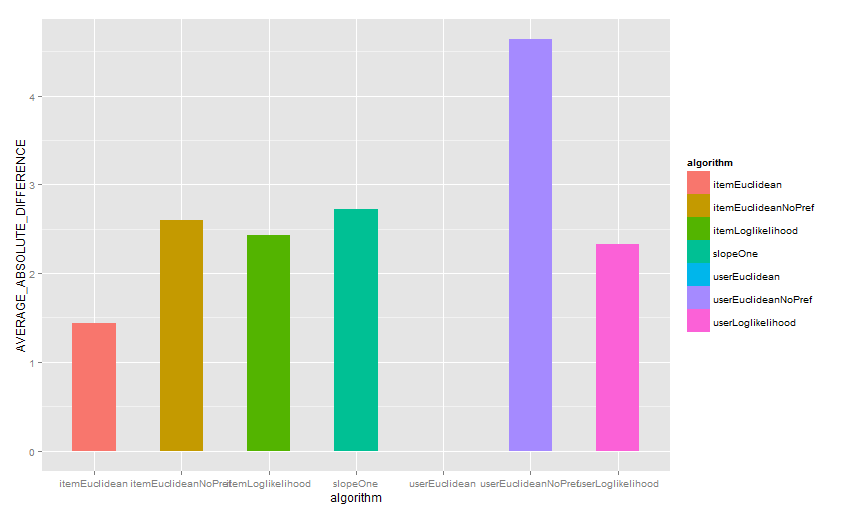
slopeOne

AVERAGE\_ABSOLUTE\_DIFFERENCE Evaluater Score:2.6893078179405814

Recommender IR Evaluator: [Precision:0.0,Recall:0.0]

可视化“评估推荐器”输出：

推荐的结果的平均距离

[](http://blog.fens.me/wp-content/uploads/2014/03/difference.png)

推荐器的评分

[](http://blog.fens.me/wp-content/uploads/2014/03/evaluator.png)

只有itemEuclideanNoPref算法评估的结果是非常好的，其他算法的结果都不太好。

**2). BookResult.java, 对指定数量的结果人工比较**

为得到差异化结果，我们分别取4个算法：userEuclidean,itemEuclidean，userEuclideanNoPref，itemEuclideanNoPref，对推荐结果人工比较。

源代码

package org.conan.mymahout.recommendation.book;

import java.io.IOException;

import java.util.List;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.eval.RecommenderBuilder;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;

public class BookResult {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/book/rating.csv";

DataModel dataModel = RecommendFactory.buildDataModel(file);

RecommenderBuilder rb1 = BookEvaluator.userEuclidean(dataModel);

RecommenderBuilder rb2 = BookEvaluator.itemEuclidean(dataModel);

RecommenderBuilder rb3 = BookEvaluator.userEuclideanNoPref(dataModel);

RecommenderBuilder rb4 = BookEvaluator.itemEuclideanNoPref(dataModel);

LongPrimitiveIterator iter = dataModel.getUserIDs();

while (iter.hasNext()) {

long uid = iter.nextLong();

System.out.print("userEuclidean =>");

result(uid, rb1, dataModel);

System.out.print("itemEuclidean =>");

result(uid, rb2, dataModel);

System.out.print("userEuclideanNoPref =>");

result(uid, rb3, dataModel);

System.out.print("itemEuclideanNoPref =>");

result(uid, rb4, dataModel);

}

}

public static void result(long uid, RecommenderBuilder recommenderBuilder, DataModel dataModel) throws TasteException {

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM);

RecommendFactory.showItems(uid, list, false);

}

}

控制台输出：只截取部分结果

...

userEuclidean =>uid:63,

itemEuclidean =>uid:63,(984,9.000000)(690,9.000000)(943,8.875000)

userEuclideanNoPref =>uid:63,(4,1.000000)(723,1.000000)(300,1.000000)

itemEuclideanNoPref =>uid:63,(867,3.791667)(947,3.083333)(28,2.750000)

userEuclidean =>uid:64,

itemEuclidean =>uid:64,(368,8.615385)(714,8.200000)(290,8.142858)

userEuclideanNoPref =>uid:64,(860,1.000000)(490,1.000000)(64,1.000000)

itemEuclideanNoPref =>uid:64,(409,3.950000)(715,3.830627)(901,3.444048)

userEuclidean =>uid:65,(939,7.000000)

itemEuclidean =>uid:65,(550,9.000000)(334,9.000000)(469,9.000000)

userEuclideanNoPref =>uid:65,(939,2.000000)(185,1.000000)(736,1.000000)

itemEuclideanNoPref =>uid:65,(666,4.166667)(96,3.093931)(345,2.958333)

userEuclidean =>uid:66,

itemEuclidean =>uid:66,(971,9.900000)(656,9.600000)(918,9.577709)

userEuclideanNoPref =>uid:66,(6,1.000000)(492,1.000000)(676,1.000000)

itemEuclideanNoPref =>uid:66,(185,3.650000)(533,3.617307)(172,3.500000)

userEuclidean =>uid:67,

itemEuclidean =>uid:67,(663,9.700000)(987,9.625000)(486,9.600000)

userEuclideanNoPref =>uid:67,(732,1.000000)(828,1.000000)(113,1.000000)

itemEuclideanNoPref =>uid:67,(724,3.000000)(279,2.950000)(890,2.750000)

...

我们查看uid=65的用户推荐信息：

查看user.csv数据集

> user[65,]

userid gender age

65 65 M 14

用户65，男性，14岁。

以itemEuclideanNoPref的算法的推荐结果，查看bookid=666的图书评分情况

> rating[which(rating$bookid==666),]

userid bookid pref

646 44 666 10

1327 89 666 7

2470 165 666 3

2697 179 666 7

发现有4个用户对666的图书评分，查看这4个用户的属性数据

> user[c(44,89,165,179),]

userid gender age

44 44 F 76

89 89 M 40

165 165 F 59

179 179 F 68

这4个用户，3女1男。

我们假设男性和男性有相同的图书兴趣，女性和女性有相同的图书偏好。因为用户65是男性，所以我们接下来排除女性的评分者，只保留男性评分者的评分记录。

**3). BookFilterGenderResult.java，只保留男性用户的图书列表**

源代码

package org.conan.mymahout.recommendation.book;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.eval.RecommenderBuilder;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.model.DataModel;

import org.apache.mahout.cf.taste.recommender.IDRescorer;

import org.apache.mahout.cf.taste.recommender.RecommendedItem;

public class BookFilterGenderResult {

final static int NEIGHBORHOOD\_NUM = 2;

final static int RECOMMENDER\_NUM = 3;

public static void main(String[] args) throws TasteException, IOException {

String file = "datafile/book/rating.csv";

DataModel dataModel = RecommendFactory.buildDataModel(file);

RecommenderBuilder rb1 = BookEvaluator.userEuclidean(dataModel);

RecommenderBuilder rb2 = BookEvaluator.itemEuclidean(dataModel);

RecommenderBuilder rb3 = BookEvaluator.userEuclideanNoPref(dataModel);

RecommenderBuilder rb4 = BookEvaluator.itemEuclideanNoPref(dataModel);

long uid = 65;

System.out.print("userEuclidean =>");

filterGender(uid, rb1, dataModel);

System.out.print("itemEuclidean =>");

filterGender(uid, rb2, dataModel);

System.out.print("userEuclideanNoPref =>");

filterGender(uid, rb3, dataModel);

System.out.print("itemEuclideanNoPref =>");

filterGender(uid, rb4, dataModel);

}

/\*\*

\* 对用户性别进行过滤

\*/

public static void filterGender(long uid, RecommenderBuilder recommenderBuilder, DataModel dataModel) throws TasteException, IOException {

Set userids = getMale("datafile/book/user.csv");

//计算男性用户打分过的图书

Set bookids = new HashSet();

for (long uids : userids) {

LongPrimitiveIterator iter = dataModel.getItemIDsFromUser(uids).iterator();

while (iter.hasNext()) {

long bookid = iter.next();

bookids.add(bookid);

}

}

IDRescorer rescorer = new FilterRescorer(bookids);

List list = recommenderBuilder.buildRecommender(dataModel).recommend(uid, RECOMMENDER\_NUM, rescorer);

RecommendFactory.showItems(uid, list, false);

}

/\*\*

\* 获得男性用户ID

\*/

public static Set getMale(String file) throws IOException {

BufferedReader br = new BufferedReader(new FileReader(new File(file)));

Set userids = new HashSet();

String s = null;

while ((s = br.readLine()) != null) {

String[] cols = s.split(",");

if (cols[1].equals("M")) {// 判断男性用户

userids.add(Long.parseLong(cols[0]));

}

}

br.close();

return userids;

}

}

/\*\*

\* 对结果重计算

\*/

class FilterRescorer implements IDRescorer {

final private Set userids;

public FilterRescorer(Set userids) {

this.userids = userids;

}

@Override

public double rescore(long id, double originalScore) {

return isFiltered(id) ? Double.NaN : originalScore;

}

@Override

public boolean isFiltered(long id) {

return userids.contains(id);

}

}

控制台输出:

userEuclidean =>uid:65,

itemEuclidean =>uid:65,(784,8.090909)(276,8.000000)(476,7.666667)

userEuclideanNoPref =>uid:65,

itemEuclideanNoPref =>uid:65,(887,2.250000)(356,2.166667)(430,1.866667)

我们发现，由于只保留男性的评分记录，数据量就变得比较少了，基于用户的协同过滤算法，已经没有输出的结果了。基于物品的协同过滤算法，结果集也有所变化。

对于itemEuclideanNoPref算法，输出排名第一条为ID为887的图书。

我再进一步向下追踪：查询哪些用户对图书887进行了打分。

> rating[which(rating$bookid==887),]

userid bookid pref

1280 85 887 2

1743 119 887 8

2757 184 887 4

2791 186 887 5

有4个用户对图书887评分，再分别查看这个用户的属性

> user[c(85,119,184,186),]

userid gender age

85 85 F 31

119 119 F 49

184 184 M 27

186 186 M 35

其中2男，2女。由于我们的算法，已经排除了女性的评分，我们可以推断图书887的推荐应该来自于2个男性的评分者的推荐。

分别计算用户65，与用户184和用户186的评分的图书交集。

rat65<-rating[which(rating$userid==65),]

rat184<-rating[which(rating$userid==184),]

rat186<-rating[which(rating$userid==186),]

> intersect(rat65$bookid ,rat184$bookid)

integer(0)

> intersect(rat65$bookid ,rat186$bookid)

[1] 65 375

最后发现，用户65与用户186都给图书65和图书375打过分。我们再打分出用户186的评分记录。

> rat186

userid bookid pref

2790 186 65 7

2791 186 887 5

2792 186 529 3

2793 186 375 6

2794 186 566 7

2795 186 169 4

2796 186 907 1

2797 186 821 2

2798 186 720 5

2799 186 642 5

2800 186 137 3

2801 186 744 1

2802 186 896 2

2803 186 156 6

2804 186 392 3

2805 186 386 3

2806 186 901 7

2807 186 69 6

2808 186 845 6

2809 186 998 3

用户186，还给图书887打过分，所以对于给65用户推荐图书887，是合理的。

我们通过一个实际的图书推荐的案例，更进一步地了解了如何用Mahout构建推荐系统。

**用Mahout搭建推荐系统之入门篇**

## 1.基本内容

    1. 加载数据: 判断userID和itemID的大小关系

    2. 过滤数据: 评分较少的用户直接过滤掉, 那些评分均一致且评分数量多的用户过滤掉. 计算过滤百分比, 如果过滤过多, 则需要考虑其它方法了.

    3. DataModel选择: 选择数据库存储还是文件存储； 选择GenericDataModel还是GenericBooleanDataModel

    4. 选择相似矩阵和参数, 如N值和门限值； 可视化(可选).

## 2、运行环境

     JAVA MYSQL等配置参考"最美的词" [基于mahout的电影推荐系统](http://blog.csdn.net/huhui_cs/article/details/8596388)

Mahout环境搭建

     本篇使用mahout 0.8的taste等相关jar包进行开发, jar包可以从 [http://mirror.bit.edu.cn/apache/mahout/mahout-distribution-0.8.tar.gz](http://mirror.bit.edu.cn/apache/mahout/mahout-distribution-0.8.tar.gz%E4%B8%AD%E6%91%98%E5%8F%96%EF%BC%8C%E4%B9%9F%E5%8F%AF%E4%BB%A5%E5%9C%A8%E7%99%BE%E5%BA%A6%E7%BD%91%E7%9B%98%E4%B8%8A%E4%B8%8B%E8%BD%BD)中摘取，也可以在百度网盘上下载 http://pan.baidu.com/s/1iSOWk.

     与上次不同, 0.8版本的distribution合并了两个包, 上次漏了两个log包, 最终只需要引入7个包即可.

mahout核心类不变: 提供推荐Model等核心类

    mahout-core-0.8.jar

    mahout-math-0.8.jar

辅助类: 提供Log和部分数学公式类.

    slf4j-api-1.7.5.jar commons-logging-1.1.1.jar slf4j-jcl-1.7.5.jar提供Log服务

    guava-14.1.0.jar合并了两个google相关的数学类google-collections.jar和guava.jar

    commons-math3-3.2.jar包取代了uncommons-maths-1.2.jar类

## 3、程序运行

搭建基本框架并进行简单测试

     我在博文1的框架下做了一点小改动, 从而说明推荐算法算法的结果不稳定性以及调参的重要性. 推荐系统不像一般的业务逻辑, 搭建好系统只完成了极小的一部分, 重点在于调参和响应速度.

类似于博客1中叙述所述, 搭建基本的框架, 并引入movielens 100K中的u.data数据,运行成功.

工程目录结构:



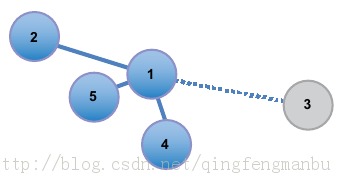
[数据格式说明: movielens u.data数据格式为"244     51     2 880606923", 以tab隔开. 表示ID为244的用户对ID为51的物品打分为2分, 时间为880606923, 猜测类似于从1970年1月1日开始记的秒数, 数量级差不多, 暂时不使用此参数.]

首先介绍User-based和Item-based的方法.

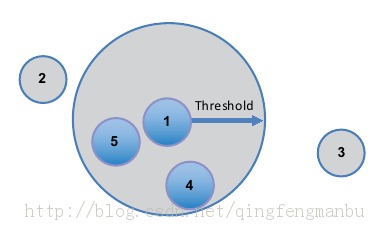
     以User-based为例, 将每一个物品表示为一个维度, 那么每个用户都可以表示为一个向量. 如果一个有{101, 102, 103, 104, 105}五个物品, 用户1对101评分为2.0, 对105评分为3.0, 那么用户1可以表示为[2.0, 0, 0, 0, 3.0]. 那么用户之间就有距离, 距离由Similarity相似性决定, 常见的如欧拉距离. 如果我们确定了所有用户间的距离, 那么可以使用N近邻法或者门限法确定每个人的相邻圈子, 如下所示.

     如何选择每个item或者user响铃圈子:

    常见的有N近邻法和门限值法. 如下面2图所示:



此图表示N = 3时,选择与1最近的前三位2, 4, 5而排除3.  1的圈子由2, 4, 5组成.



此图表示门限(Threshold)选择法, 4, 5 在门限之内, 而2. 3在门限之外. 1的圈子由4, 5组成.

总结: 那么接下来的问题就是如何定义相似性, 即计算距离了.

### 3.1 调整N值和Threshold值对推荐结果的影响：

重要代码片段如下:

public static void main(String[] args) throws Exception {

int userId = 1;

int rankNum = 2;

QingRS qingRS = new QingRS();

for(int neighberNum = 2; neighberNum < 10; neighberNum++) {

System.out.println("neigherNum=" + neighberNum);

qingRS.initRecommenderIntro(filename, neighberNum);

String resultStr = qingRS.getRecommender(userId, rankNum);

System.out.println(resultStr);

}

}

**运行结果：**

A. 当neigherhood从2到9变化时, 推荐的物品前期在变化, 后期趋于稳定.

neigherNum=2

Recommend=313 4.5  
neigherNum=3  
Recommend=286 5.0  
neigherNum=4  
Recommend=286 5.0  
neigherNum=5  
Recommend=990 5.0  
neigherNum=6  
Recommend=990 5.0  
neigherNum=7  
Recommend=990 5.0  
neigherNum=8  
Recommend=990 5.0  
neigherNum=9  
Recommend=990 5.0

解释: neigherhood一开始变化时, 参考的人数增多了, 所谓三个臭皮匠顶过一个诸葛亮, 推荐将会变化, 但是随着neigherhood的变大, 加再多的人进来也只是凑人数而已没有多大的决定能力.

B. 当rankNum从2到10变化时, 感觉上rankNum的改变不应该影响推荐结果.

List<RecommendedItem> recommendations = recommender.recommend(userid,  
rankNum);

但是: 我们发现除了neigherNum = 2以外, 推荐结果均发生了变化, 而且数据开始震荡, 如果将neigherNum放大到30, 推荐结果依旧不停地震荡.

neigherNum=2

Recommend=313 4.5  
neigherNum=3  
Recommend=323 5.0  
neigherNum=4  
Recommend=898 5.0  
neigherNum=5  
Recommend=323 5.0  
neigherNum=6  
Recommend=323 5.0  
neigherNum=7  
Recommend=898 5.0  
neigherNum=8  
Recommend=326 5.0  
neigherNum=9  
Recommend=326 5.0

解释???: 问题应该出在排序算法上, Mahout为了节约内存使用了qSort, 因此排序算法不稳定. 但是我去查看Mahout源代码发现GenericUserBasedRecommender中使用了Collections.sort(), sort默认使用的是MergeSort, 所以排序应该是稳定的. 依旧存在着疑问.

### 3.2. 针对DataModel做一些数据分析,

### 类似于博文2, 判断item和user数量, value范围, 方差等.

代码如下:

package com.qingfeng.rs.test;

import java.io.File;

import java.io.IOException;

import org.apache.mahout.cf.taste.common.TasteException;

import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;

import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;

import org.apache.mahout.cf.taste.model.DataModel;

public class QingDataModelTest {

private final static String filename = "data/u.data";

public static void main(String[] args) throws IOException, TasteException {

DataModel dataModel = new FileDataModel(new File(filename));

// compute the max and min value

// 计算最大最小值

float maxValue = dataModel.getMaxPreference();

float minValue = dataModel.getMinPreference();

// compute the number of usersNum and itemsNum

// 计算用户和物品总数

int usersNum = dataModel.getNumUsers();

int itemsNum = dataModel.getNumItems();

int[] itemsNumForUsers = new int[usersNum];

int[] usersNumForItems = new int[itemsNum];

LongPrimitiveIterator userIDs = dataModel.getUserIDs();

int i = 0;

while (userIDs.hasNext()) {

itemsNumForUsers[i++] = dataModel.getPreferencesFromUser(

userIDs.next()).length();

}

assert (i == usersNum);

LongPrimitiveIterator itemIDs = dataModel.getItemIDs();

i = 0;

while (itemIDs.hasNext()) {

usersNumForItems[i++] = dataModel.getPreferencesForItem(

itemIDs.next()).length();

}

assert (i == itemsNum);

// compute mean and variance

// 计算平均值和方差

double usersMean;

double usersVar;

int sum = 0;

int sqSum = 0;

for (int num : itemsNumForUsers) {

sum += num;

sqSum += num \* num;

}

usersMean = (double) sum / usersNum;

double userSqMean = (double) sqSum / usersNum;

usersVar = Math.sqrt(userSqMean - usersMean \* usersMean);

double itemsMean;

double itemsVar;

sum = 0;

sqSum = 0;

for (int num : usersNumForItems) {

sum += num;

sqSum += num \* num;

}

itemsMean = (double) sum / itemsNum;

double itemsSqMean = (double) sqSum / itemsNum;

itemsVar = Math.sqrt(itemsSqMean - itemsMean \* itemsMean);

System.out.println("Preference=(" + minValue + ", " + maxValue + ")");

System.out.println("usersNum=" + usersNum + ", userMean=" + usersMean

+ ", userVar=" + usersVar);

System.out.println("itemsNum=" + itemsNum + ", itemsMean=" + itemsMean

+ ", itemsVar=" + itemsVar);

}

}

**设置门限过滤数据**

在代码中加入过滤模块

for (int num : itemsNumForUsers) {

sum += num;

if (num < 20) {

countLower++;

// System.out.println("user warning(" + countLower + ")=" + num);

}

sqSum += num \* num;

}

System.out.println("user warning(" + countLower + ")");

for (int num : usersNumForItems) {

sum += num;

if (num < 20) {

countLower++;

//System.out.println("item warning(" + countLower + ")=" + num);

}

sqSum += num \* num;

}

System.out.println("item warning(" + countLower + ")");

运行结果如下

user warning(0)   
item warning(743)   
Preference=(1.0, 5.0)   
usersNum=943, userMean=106.04453870625663, userVar=100.87821227051644   
itemsNum=1682, itemsMean=59.45303210463734, itemsVar=80.3599467406018

分析：与官方的1000个用户, 1700部电影的说法一致.  <http://www.grouplens.org/datasets/movielens/>

user warning(0)   
item warning(743) 表示有743个item评分个数小于20.

     物品评分较为稀疏程度和物品总数大小是一致的. 使用user-based则用户少,节约内存, 且矩阵致密。

设置门限为20时, 发现物品矩阵稀疏、方差大和过滤器的统计结果item warning(743)大是一致, 此处先不过滤数据, 后期再说.

注：当然优秀的过滤器需要改变门限值来不停的调试

### 3.3 选择DataModel, 并计算内存使用情况

     由于数据有rate, 所以不使用Boolean形式的存储.

预估内存开销:

     由上文分析可知: Preference ~= usersNum \* userMean ~= 100K, 每个Preference消耗28bytes,

预估内存开销= 28bytes \* 100K = 2.8 Mbytes. 此外相似矩阵如果使用邻接矩阵方式存储, max{usersNum, itemsNum}\*\*2 \* 4bytes(float) = 8Mbytes左右. 因此内存总结开销在10M左右.

    [但是查看Mahout源代码org.apache.mahout.cf.taste.impl中相关文件发现, 相似矩阵是临时计算的, 每次recommend时通过重写Estimator接口的estimate方法来具体实现. 可以mahout还是考虑到内存开销, 牺牲了计算速度吧. 所以估计程序运行内存开销依旧在2.8Mbytes左右. 究竟哪个是正确的理解呢?]

     因此我使用in-memory形式的GenericDataModel将数据直接加载到内存中.

实验测试内存开销:

通过多次调用System.gc()来回收内存, 通过Rumtime.totalMemory和Runtime.freeMemory()查看内存使用状态.

<http://docs.oracle.com/javase/6/docs/api/>

代码如下:

public class QingMemoryTest {

private static final String filename = "data/u.data";

public static void main(String[] args) throws Exception {

DataModel dataModel = new FileDataModel(new File(filename));

UserSimilarity similarity = new PearsonCorrelationSimilarity(dataModel);

UserNeighborhood neighborhood = new NearestNUserNeighborhood(5,

similarity, dataModel);

Recommender recommender = new GenericUserBasedRecommender(dataModel,

neighborhood, similarity);

System.out.println("1: jvm free-memory= "

+ Runtime.getRuntime().freeMemory() + "Bytes");

System.gc();

System.out.println("2: jvm free-memory= "

+ Runtime.getRuntime().freeMemory() + "Bytes");

// dataModel被回收, 所以推荐结果错误.

System.out.println(recommender.recommend(1, 2).get(1).getValue());

}

}

运行结果如下:

start: jvm used-memory= 0.5967178344726562MB

after dataModel: jvm used-memory= 19.2872314453125MB  
after similarity: jvm used-memory= 19.2872314453125MB  
after neighborhood: jvm used-memory= 19.58240509033203MB  
after recommender: jvm used-memory= 19.58240509033203MB  
recommend=340  
after recommend first: jvm used-memory= 19.877883911132812MB  
after gc: jvm used-memory= 9.829483032226562MB  
recommend=340  
after recommend second: jvm used-memory= 9.829483032226562MB

分析: 由上述数据可见,gc回收内存后, JVM内存消耗回收了10Mbytes, 与猜测一致.

问题: 回收完数据后, 为什么recommender还可以进行推荐, 而且没有额外的内存开销???

数据增长10倍, 即使用1M数据进行测试

简单统计分析结果:

user warning(0)  
item warning(663)  
Preference=(1.0, 5.0)  
usersNum=6040, userMean=165.5975165562914, userVar=192.73107252940773  
itemsNum=3706, itemsMean=269.88909875876953, itemsVar=383.9960197430679

估计内存消耗: usersNum和itemsNum增长了3到6倍, 而相似矩阵消耗内存为平方级别, 那么内存消耗上线为9到36倍; 此外数据增长10倍, DataModel内存消耗为线性增长, 增长10倍内存消耗. 那么估计内存消耗= 2.8M \* 10 + (9~36)\*8M = 100M ~ 316M内存之间. 如果不存储相似矩阵, 那么内存消耗为28M左右.

由于数据以"::"作为分割符, 用python简单处理一下,替换为\t

f = open("result.dat", "w")

for line in open("ratings.dat", "r"):

newLine = line.replace("::", "\t")

f.write(newLine)

运行结果如下

start: jvm used-memory= 0.5967178344726562MB

after dataModel: jvm used-memory= 204.9770050048828MB  
after similarity: jvm used-memory= 204.9770050048828MB  
after neighborhood: jvm used-memory= 204.9770050048828MB  
after recommender: jvm used-memory= 204.9770050048828MB  
recommend=2908  
after recommend first: jvm used-memory= 208.10643768310547MB  
after gc: jvm used-memory= 76.12030029296875MB  
recommend=2908  
after recommend second: jvm used-memory= 76.12030029296875MB

分析: 由上述数据可以: 数据回收了132Mbytes, 76M为运行开销. 与估计内存消耗移植. DataModel线性增长, 相似矩阵平方级别增长.

结论: 如果评分数增加到10M级别, 用户或者物品数增长3~10倍, 那么需要4G到40G的内存才能快速的计算出推荐结果, 需要增加内存条, 设置JVM配置以及使用hadoop来实现. 另外真实的数据用户数达到GB级别, 总数达到TB级别, 需要的内存数量和运算量是十分恐怖的. 传统地算法已经无法满足要求, 需要借助Hadoop这种分布式来实现运算.

     当然内存不够大, 硬盘可以很大, 处理10M级别以上的推荐数据时, 选择使用MysqlJDBCDataModel来实现存储.

另外: 据数盟的一位Q友说, "淘宝有8kw的商品（记忆也许有出入），用户2亿，多大的矩阵啊". 每次想到这里, 都会默默地闭上双眼, 遥想远方的宇宙, 数据又是多么地浩淼. 在上帝眼中, 我们也许还只是玩过家家, 学1+1的小孩子吧.

### 3.4. 选择相似性矩阵和调参

此外， 后期希望考虑user-based, item-based, slope-one算法的比较, 同时参考运行时间.

相似矩阵选择下面4种

PearsonCorrelationSimilarity   EuclideanDistanceSimilarity  TanimotoCoefficientSimilarity  LogLikeLihoodSimilarity

[ ~~注:其中EuclideanDistanceSimilarity比较特殊, 它没有实现UserSimilarity接口, 所以不能放到一个Collection<UserSimilarity>容器中~~]

[注: 勿看了org.apache.mahout.math.hadoop.similarity.cooccurrence.measures文件]

参数调整只选择近邻N和threashold

这里给出代码原型, 但是在普通PC上跑100K的数据集都太慢了, 使用intro.csv这个toy数据跑一跑.

N选择[2, 4, 8, ... 64], Threshold选择[0.9, 0.85, ... 0.7];

代码如下:

public class QingParaTest {

private final String filename = "data/intro.csv";

private double threshold = 0.95;

private int neighborNum = 2;

private ArrayList<UserSimilarity> userSims;

private final int SIM\_NUM = 4;

private final int NEIGHBOR\_NUM = 64;

private final double THRESHOLD\_LOW = 0.7;

public static void main(String[] args) throws IOException, TasteException {

new QingParaTest().valuate();

}

public QingParaTest() {

super();

this.userSims = new ArrayList<UserSimilarity>();

}

private void valuate() throws IOException, TasteException {

DataModel dataModel = new FileDataModel(new File(filename));

RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

// populate Similarity

populateUserSims(dataModel);

int simBest = -1;

double scoreBest = 5.0;

int neighborBest = -1;

double thresholdBest = -1;

System.out.println("SIM\tNeighborNum\t\tThreshold\tscore");

for (int i = 0; i < SIM\_NUM; i++) {

for (neighborNum = 2; neighborNum <= NEIGHBOR\_NUM; neighborNum \*= 2) {

for (threshold = 0.75; threshold >= THRESHOLD\_LOW; threshold -= 0.05) {

double score = 5.0;

QingRecommenderBuilder qRcommenderBuilder = new QingRecommenderBuilder(

userSims.get(i), neighborNum, threshold);

// Use 70% of the data to train; test using the other 30%.

score = evaluator.evaluate(qRcommenderBuilder, null,

dataModel, 0.7, 1.0);

System.out.println((i + 1) + "\t" + neighborNum + "\t"

+ threshold + "\t" + score);

if (score < scoreBest) {

scoreBest = score;

simBest = i + 1;

neighborBest = neighborNum;

thresholdBest = threshold;

}

}

}

}

System.out.println("The best parameter");

System.out.println(simBest + "\t" + neighborBest + "\t" + thresholdBest

+ "\t" + scoreBest);

}

private void populateUserSims(DataModel dataModel) throws TasteException {

UserSimilarity userSimilarity = new PearsonCorrelationSimilarity(

dataModel);

userSims.add(userSimilarity);

userSimilarity = new TanimotoCoefficientSimilarity(dataModel);

userSims.add(userSimilarity);

userSimilarity = new LogLikelihoodSimilarity(dataModel);

userSims.add(userSimilarity);

userSimilarity = new EuclideanDistanceSimilarity(dataModel);

userSims.add(userSimilarity);

}

}

class QingRecommenderBuilder implements RecommenderBuilder {

private UserSimilarity userSimilarity;

private int neighborNum;

private double threshold;

public QingRecommenderBuilder(UserSimilarity userSimilarity,

int neighborNum, double threshold) {

super();

this.userSimilarity = userSimilarity;

this.neighborNum = neighborNum;

this.threshold = threshold;

}

@Override

public Recommender buildRecommender(DataModel dataModel)

throws TasteException {

UserNeighborhood neighborhood = new NearestNUserNeighborhood(

neighborNum, threshold, userSimilarity, dataModel);

return new GenericUserBasedRecommender(dataModel, neighborhood,

userSimilarity);

}

}

运行结果如下:

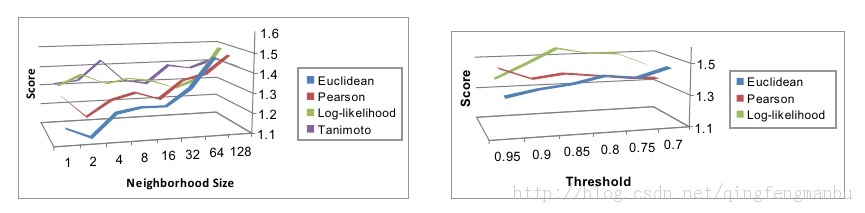
SIM NeighborNum Threshold score  
1 2 0.75 0.4858379364013672  
1 2 0.7 NaN  
1 4 0.75 0.4676065444946289  
1 4 0.7 NaN  
1 8 0.75 0.8704338073730469  
1 8 0.7 0.014162302017211914  
1 16 0.75 NaN  
1 16 0.7 0.7338032722473145  
1 32 0.75 0.7338032722473145  
1 32 0.7 0.4858379364013672  
1 64 0.75 NaN  
1 64 0.7 1.0

The best parameter

1 8 0.7 0.014162302017211914

分析: 运行最佳的结果为N = 8, Threshold = 0.7 当然, 这个方法, 十分的粗糙, 但是也说明了参数的重要性, 毕竟推荐系统上线了必须有优秀的A\B Test结果, 要不然还不如使用打折, 优惠券来的简单实在.

顺便截一张Mahout in Action上一个真实案例的数据, 如下图所示



 item-based与user\_based一致, 基本上就是就Similarity, Neighborhood和Recommender的User换成Item即可.

### 3.5 slope-one

public class SlopeOne {

public static void main(String[] args) throws IOException, TasteException {

DataModel dataModel = new FileDataModel(new File("data/intro.csv"));

RecommenderEvaluator evaluator = new AverageAbsoluteDifferenceRecommenderEvaluator();

double score = evaluator.evaluate(new SlopeOneNoWeighting(), null,

dataModel, 0.7, 1.0);

System.out.println(score);

}

}

class SlopeOneNoWeighting implements RecommenderBuilder {

public Recommender buildRecommender(DataModel model) throws TasteException {

DiffStorage diffStorage = new MemoryDiffStorage(model,

Weighting.UNWEIGHTED, Long.MAX\_VALUE);

return new SlopeOneRecommender(model, Weighting.UNWEIGHTED,

Weighting.UNWEIGHTED, diffStorage);

}

}

运行结果为: 1.3571428571428572 当然这个结果意义不大, 因为数据集很小.

## 4、总结

     推荐系统的难点在于各种参数、算法的选择，以及推荐系统整体架构的测试；如果希望搭建商业级别的应用，在数据和架构上所花的时间要比算法调参多一些。

## 5、Similarity和Algorithm相关总结

如何计算相似性:

    常见的方法如下表所示: Similarity只是描述计算方法, 并不计算并保存相似矩阵.

    相似性的基本思路就是不适用欧式距离的, 都得加上权重或者门限来防止交集较小的相似距离.

|  |  |  |  |
| --- | --- | --- | --- |
| 相似距离(距离越小值越大) | 优点 | 缺点 | 取值范围 |
| PearsonCorrelation  类似于计算两个矩阵的协方差 | 不受用户评分偏高  或者偏低习惯影响的影响 | 1. 如果两个item相似个数小于2时  无法计算相似距离.  [可以使用item相似个数门限来解决.]  没有考虑两个用户之间的交集大小[使用weight参数来解决]  2. 无法计算两个完全相同的items | [-1, 1] |
| EuclideanDistanceSimilarity  计算欧氏距离, 使用1/(1+d) | 使用与评分大小较  重要的场合 | 如果评分不重要则需要归一化,  计算量大  同时每次有数据更新时麻烦 | [-1, 1] |
| CosineMeasureSimilarity  计算角度 | 与PearsonCorrelation一致 |  | [-1, 1] |
| SpearmanCorrelationSimilarity  使用ranking来取代评分的  PearsonCorrelation | 完全依赖评分和完全放弃评分之间的平衡 | 计算rank消耗时间过大  不利于数据更新 | [-1, 1] |
| CacheUserSimilarity  保存了一些tag, reference | 缓存经常查询的user-similarity | 额外的内存开销 |  |
| TanimotoCoefficientSimilarity  统计两个向量的交集占并集的比例  同时并集个数越多, 越相近. | 适合只有相关性  而没有评分的情况 | 没有考虑评分,信息丢失了 | [-1,1] |
| LogLikeLihoodSimilarity  是TanimoteCoefficientSimilarity  的一种基于概率论改进 | 计算两者重合的偶然性  考虑了两个item相邻的独特性 | 计算复杂 | [-1,1] |

     如何选择推荐算法:

    user-based算法: 最古老的算法, 计算相似的人群, 最大的问题是存储相似矩阵, 由于每个用户喜欢的物品在变化, 导致相似矩阵不停的变化. 更新相似矩阵计算量可能较大. 针对搜索引擎来说, 搜索词如果比用户数目多的话,可以考虑user-based.

    item-based算法: 与user-based类似, 每个物品被喜欢的用户个数不停地变化, 相似矩阵持续地更新. 在互联网时代,商品上百万, 用户上亿. 那么使用item-based比较靠谱, 物品相似矩阵变化较小, Amazon的推荐算法就是使用item-based为基础的.

    SVD: 现在比较流行的算法, 因为可以进行降维. 发掘有价值的特征维度来取代用户维度或者商品维度. 举个例子: 例如两个人分别喜欢保时捷和法拉利, user-based和item-based计算的相似性都很低, 但是SVD引入跑车或者奢侈品这种潜在的特征后, 两者就有相似性了. 当然缺点在于, SVD需要将整个矩阵加载到内存进行矩阵分解, 对内存消耗大, 不知道SVD的矩阵分解有没有Map-Reduce实现方法.

    Slope-One算法: 上述三种算法都不太适合作为在线算法和更新数据, 但是Slope-One可以. 举个例子, 假设所有用户评价电影A比电影B高1.0分, 评价电影C和电影A一致. 如果一个用户评价电影B为2.0分, 评价电影C为4.0分, 那么用户评价电影A为3.0分或者4.0分, 最佳的方法的取两者的加权平均值, 权重由同时出现次数决定. Slope-One可以离线计算所有的n\*(n-1)/2中相关性, 当一个用户更新了电影时, 相关性更新快捷； 通过遍历一遍电影即可获得所有电影的评分,从而排序给出推荐. 缺点是相关性计算复杂. [个人觉得这个计算量也不小, 取决于电影个数以及用户评分电影个数]

# 基于MapReduce的ItemBase推荐算法的共现矩阵实现

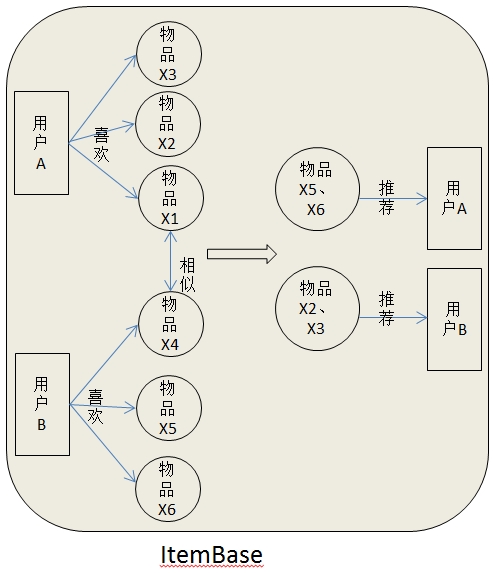
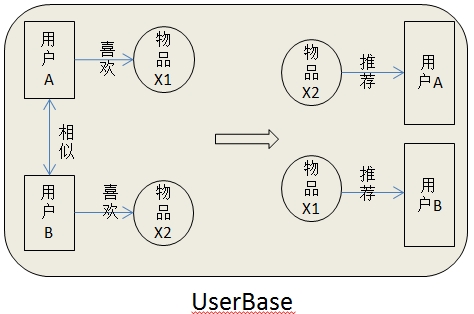
## 1、概述

    这2个月研究根据用户标签情况对用户的相似度进行评估，其中涉及一些推荐算法知识，在这段时间研究了一遍《推荐算法实践》和《Mahout in action》，在这里主要是根据这两本书的一些思想和自己的一些理解对分布式基于ItemBase的推荐算法进行实现。其中分两部分，第一部分是根据共现矩阵的方式来简单的推算出用户的推荐项，第二部分则是通过传统的相似度矩阵的方法来实践ItemBase推荐算法。这篇blog主要记录第一部分的内容，并且利用MapReduce进行实现，下一篇blog则是记录第二部分的内容和实现。

## 2、算法原理

    协同推荐算法，作为众多推荐算法中的一种已经被广泛的应用。其主要分为2种，第一种就是基于用户的协同过滤，第二种就是基于物品的协同过滤。

    所谓的itemBase推荐算法简单直白的描述就是：用户A喜欢物品X1，用户B喜欢物品X2，如果X1和X2相似则，将A之前喜欢过的物品推荐给B，或者B之前喜欢过的物品推荐给A。这种算法是完全依赖于用户的历史喜欢物品的；所谓的UserBase推荐算法直白地说就是：用户A喜欢物品X1，用户B喜欢物品X2，如果用户A和用户B相似则将物品X1推荐给用户B，将物品X2推荐给用户A。简单的示意图：

[](http://s3.51cto.com/wyfs02/M02/49/EA/wKioL1Qe6-7gZbvgAAF26e9WFgA063.jpg)

至于选择哪种要看自己的实际情况，如果用户量比物品种类多得多那么就采用ItemBase的协同过滤推荐算法，如果是用户量比物品种类少的多则采用UserBase的协同顾虑推荐算，这样选择的一个原因是为了让物品的相似度矩阵或者用户相似度矩阵或者共现矩阵的规模最小化。

## 3、数据建模

    基本的算法上面已经大概说了一下，对于算法来说，对数据建模使之运用在算法之上是重点也是难点。这小节主要根据自己相关项目的经验和《推荐引擎实践》的一些观点来讨论一些。分开2部分说，一是根据共现矩阵推荐、而是根据相似度算法进行推荐。

(1)共现矩阵方式：

第一步：转换成用户向量

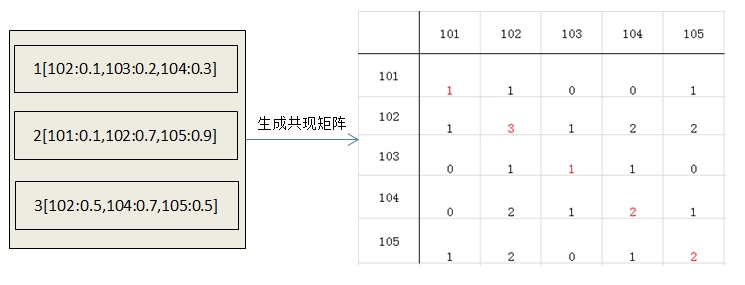
1[102:0.1,103:0.2,104:0.3]：表示用户1喜欢的物品列表，以及他们对应的喜好评分。

2[101:0.1,102:0.7,105:0.9]：表示用户2喜欢的物品列表，以及他们对应的喜好评分。

3[102:0.1,103:0.7,104:0.2]：表示用户3喜欢的物品列表，以及他们对应的喜好评分。

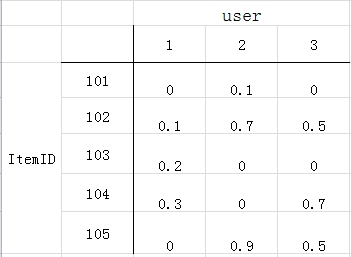
第二步：计算共现矩阵

简单地说就是将同时喜欢物品x1和x2的用户数组成矩阵。

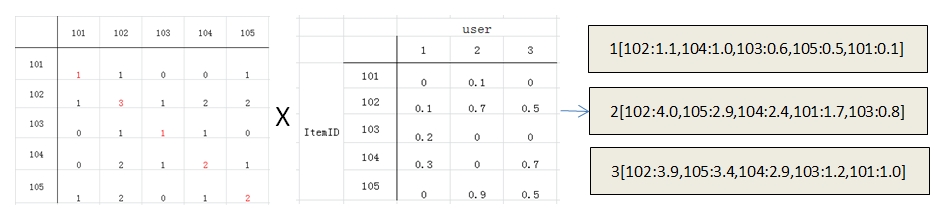
[](http://s3.51cto.com/wyfs02/M01/49/E9/wKiom1QfAT_xHw_EAAD7ZBEpZ4E049.jpg)

第三步：

生成用户对物品的评分矩阵

[](http://s3.51cto.com/wyfs02/M02/49/E9/wKiom1QfA5_xQYhxAAB86TsiCxE249.jpg)

第四步：物品共现矩阵和用户对物品的评分矩阵相乘得到推荐结果

[](http://s3.51cto.com/wyfs02/M01/49/E9/wKiom1QfDkfDDcVWAAFPZSfTYP4878.jpg)

举个例子计算用户1的推荐列表过程：

用户1对物品101的总评分计算：

1\*0+1\*0.1+0\*0.2+0\*0.3+1\*0=0.1

用户1对物品102的总评分计算：

1\*0+3\*0.1+1\*0.2+2\*0.3+2\*0=1.1

用户1对物品103的总评分计算：

0\*0+1\*0.1+1\*0.2+1\*0.3+0\*0=0.6

用户1对物品104的总评分计算：

0\*0+2\*0.1+1\*0.2+2\*0.3+1\*0=1.0

用户1对物品105的总评分计算：

1\*0+2\*0.1+0\*0.2+1\*0.3+2\*0=0.5

从而得到用户1的推荐列表为1[101:0.1,102:1.1,103:0.6,104:1.0,105:0.5]再经过排序得到最终推荐列表1[102:1.1,104:1.0,103:0.6,105:0.5,101:0.1]。

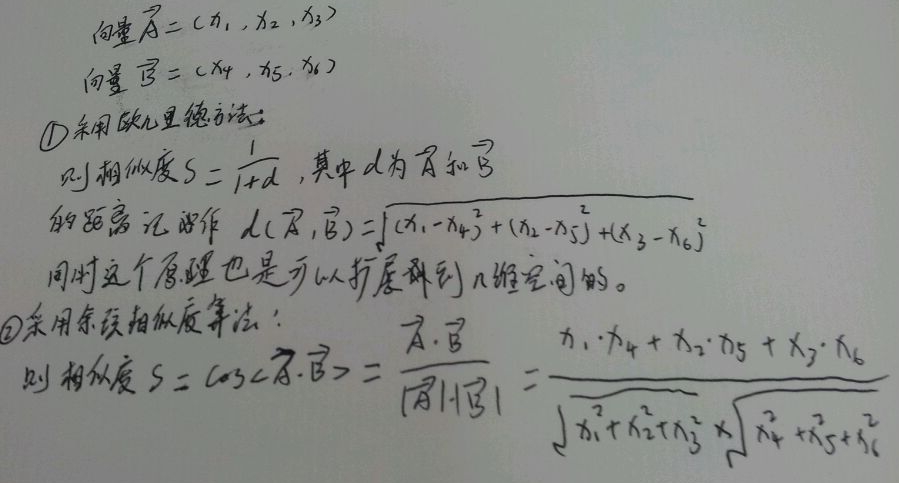
(2)通过计算机物品相似度方式计算用户的推荐向量。

    通过计算机物品相似度方式计算用户的推荐向量和上面通过共现矩阵的方式差不多，就是将物品相似度矩阵代替掉共现矩阵和用户对物品的评分矩阵相乘，然后在计算推荐向量。

计算相似度矩阵：

在计算之前我们先了解一下物品相似度相关的计算方法。

对于计算物品相似度的算法有很多，要根据自己的数据模型进行选择。基于皮尔逊相关系数计算、欧几里德定理（实际上是算两点距离）、基于余弦相似度计算斯皮尔曼相关系数计算、基于谷本系数计算、基于对数似然比计算。其中谷本系数和对数似然比这两种方式主要是针对那些没有指名对物品喜欢度的数据模型进行相似度计算，也就是mahout中所指的Boolean数据模型。下面主要介绍2种，欧几里德和余弦相似度算法。

[](http://s3.51cto.com/wyfs02/M02/49/F2/wKioL1Qfw8GgP-nIAAIircdsK3M870.jpg)现在关键是怎么将现有数据转化成对应的空间向量模型使之适用这些定理，这是个关键点。下面我以欧几里德定理作为例子看看那如何建立模型：

第一步：将用户向量转化为物品向量

用户向量：

1[102:0.1,103:0.2,104:0.3]

2[101:0.1,102:0.7,105:0.9]

3[102:0.1,103:0.7,104:0.2]

转为为物品向量：

101[2:0.1]

102[1:0.1,2:0.7,3:0.1]

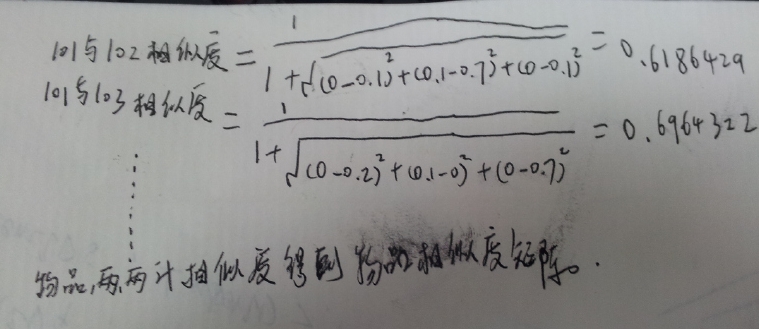
103[1:0.2,3:0.7]

104[1:0.3,3:0.2]

105[2:0.9]

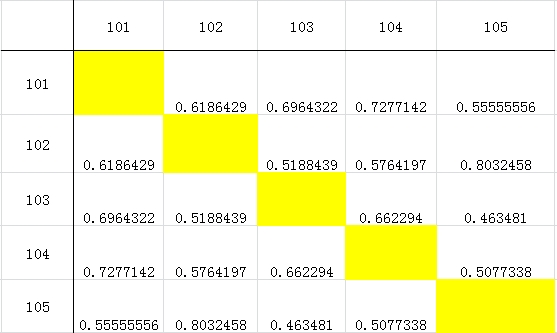
第二步：

那么物品相似度计算为：

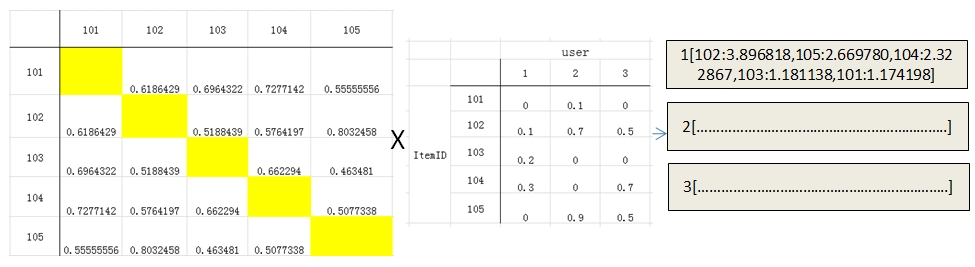
[](http://s3.51cto.com/wyfs02/M02/49/F2/wKiom1Qf0FyAsRypAAJKQFVRufY677.jpg)

第三步：

最终得到物品相似度矩阵为：(这里省略掉没有意义的自关联相似度)

[](http://s3.51cto.com/wyfs02/M00/49/F3/wKiom1Qf1svzxfXzAAEMhjBgreE838.jpg)

第四步：物品相似度矩阵和用户对物品的评分矩阵相乘得到推荐结果：

[](http://s3.51cto.com/wyfs02/M00/49/F5/wKioL1Qf3dqiMzOKAAF8WAjWLEo861.jpg)

举个例子计算用户1的类似推荐列表过程：

用户1对物品101的总评分计算：

1\*0+1\*0.6186429+0\*0.6964322+0\*0.7277142+1\*0.55555556=1.174198

用户1对物品102的总评分计算：

1\*0.6186429+3\*0+1\*0.5188439+2\*0.5764197+2\*0.8032458=3.896818

用户1对物品103的总评分计算：

0\*0.6964322+1\*0.5188439+1\*0+1\*0.662294+0\*0.463481=1.181138

用户1对物品104的总评分计算：

0\*0.7277142+2\*0.5764197+1\*0.662294+2\*0+1\*0.5077338=2.322867

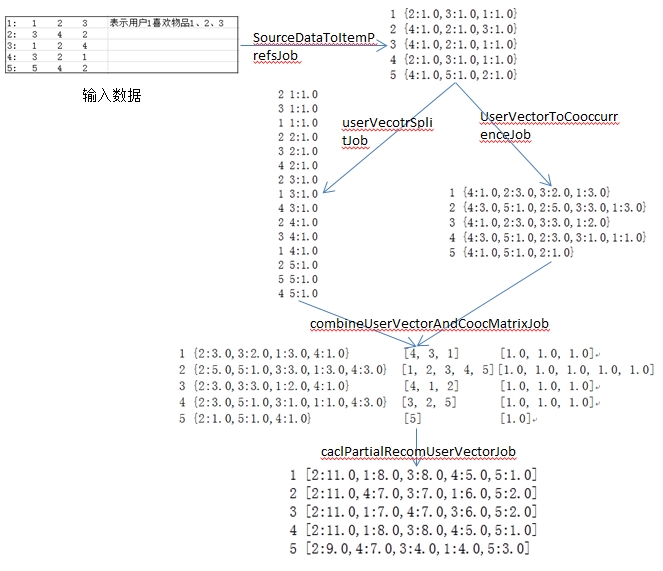
用户1对物品105的总评分计算：

1\*0.55555556+2\*0.8032458+0\*0.463481+1\*0.5077338=2.669780

## 4、共现矩阵方式的MapReduce实现

这里主要是利用MapReduce结合Mahout连的一些数据类型对共现矩阵方式的推荐方法进行实现,至于相似度矩阵方式进行推荐的在下一篇blog写。这里采用Boolean数据模型，即用户是没有对喜欢的物品进行初始打分的，我们在程序中默认都为1。

先看看整个MapReduce的数据流向图：

[](http://s3.51cto.com/wyfs02/M00/49/F6/wKioL1Qf8tnQZNJUAALGyzmu93M351.jpg)

|  |
| --- |
| package com.util;    import java.io.IOException;  import java.util.Arrays;  import java.util.Iterator;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.fs.PathFilter;  import org.apache.hadoop.io.Writable;  import org.apache.hadoop.mapreduce.InputFormat;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.JobContext;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.hadoop.mapreduce.OutputFormat;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.common.iterator.sequencefile.PathType;  import org.apache.mahout.common.iterator.sequencefile.SequenceFileDirValueIterator;  import org.apache.mahout.common.iterator.sequencefile.SequenceFileValueIterator;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;    public final class HadoopUtil {      private static final Logger log = LoggerFactory.getLogger(HadoopUtil.class);      private HadoopUtil() { }      public static Job prepareJob(String jobName,                             String[] inputPath,                             String outputPath,                             Class<? extends InputFormat> inputFormat,                             Class<? extends Mapper> mapper,                             Class<? extends Writable> mapperKey,                             Class<? extends Writable> mapperValue,                             Class<? extends OutputFormat> outputFormat, Configuration conf) throws IOException {        Job job = new Job(new Configuration(conf));      job.setJobName(jobName);      Configuration jobConf = job.getConfiguration();        if (mapper.equals(Mapper.class)) {        throw new IllegalStateException("Can't figure out the user class jar file from mapper/reducer");      }      job.setJarByClass(mapper);        job.setInputFormatClass(inputFormat);      job.setInputFormatClass(inputFormat);      StringBuilder inputPathsStringBuilder =new StringBuilder();      for(String p : inputPath){          inputPathsStringBuilder.append(",").append(p);      }      inputPathsStringBuilder.deleteCharAt(0);      jobConf.set("mapred.input.dir", inputPathsStringBuilder.toString());        job.setMapperClass(mapper);      job.setMapOutputKeyClass(mapperKey);      job.setMapOutputValueClass(mapperValue);      job.setOutputKeyClass(mapperKey);      job.setOutputValueClass(mapperValue);      jobConf.setBoolean("mapred.compress.map.output", true);      job.setNumReduceTasks(0);        job.setOutputFormatClass(outputFormat);      jobConf.set("mapred.output.dir", outputPath);        return job;    }      public static Job prepareJob(String jobName,                               String[] inputPath,                             String outputPath,                             Class<? extends InputFormat> inputFormat,                             Class<? extends Mapper> mapper,                             Class<? extends Writable> mapperKey,                             Class<? extends Writable> mapperValue,                             Class<? extends Reducer> reducer,                             Class<? extends Writable> reducerKey,                             Class<? extends Writable> reducerValue,                             Class<? extends OutputFormat> outputFormat,                             Configuration conf) throws IOException {        Job job = new Job(new Configuration(conf));      job.setJobName(jobName);      Configuration jobConf = job.getConfiguration();        if (reducer.equals(Reducer.class)) {        if (mapper.equals(Mapper.class)) {          throw new IllegalStateException("Can't figure out the user class jar file from mapper/reducer");        }        job.setJarByClass(mapper);      } else {        job.setJarByClass(reducer);      }        job.setInputFormatClass(inputFormat);      StringBuilder inputPathsStringBuilder =new StringBuilder();      for(String p : inputPath){          inputPathsStringBuilder.append(",").append(p);      }      inputPathsStringBuilder.deleteCharAt(0);      jobConf.set("mapred.input.dir", inputPathsStringBuilder.toString());        job.setMapperClass(mapper);      if (mapperKey != null) {        job.setMapOutputKeyClass(mapperKey);      }      if (mapperValue != null) {        job.setMapOutputValueClass(mapperValue);      }        jobConf.setBoolean("mapred.compress.map.output", true);        job.setReducerClass(reducer);      job.setOutputKeyClass(reducerKey);      job.setOutputValueClass(reducerValue);        job.setOutputFormatClass(outputFormat);      jobConf.set("mapred.output.dir", outputPath);        return job;    }              public static Job prepareJob(String jobName, String[] inputPath,              String outputPath, Class<? extends InputFormat> inputFormat,              Class<? extends Mapper> mapper,              Class<? extends Writable> mapperKey,              Class<? extends Writable> mapperValue,              Class<? extends Reducer> combiner,              Class<? extends Reducer> reducer,              Class<? extends Writable> reducerKey,              Class<? extends Writable> reducerValue,              Class<? extends OutputFormat> outputFormat, Configuration conf)              throws IOException {            Job job = new Job(new Configuration(conf));          job.setJobName(jobName);          Configuration jobConf = job.getConfiguration();            if (reducer.equals(Reducer.class)) {              if (mapper.equals(Mapper.class)) {                  throw new IllegalStateException(                          "Can't figure out the user class jar file from mapper/reducer");              }              job.setJarByClass(mapper);          } else {              job.setJarByClass(reducer);          }            job.setInputFormatClass(inputFormat);          StringBuilder inputPathsStringBuilder = new StringBuilder();          for (String p : inputPath) {              inputPathsStringBuilder.append(",").append(p);          }          inputPathsStringBuilder.deleteCharAt(0);          jobConf.set("mapred.input.dir", inputPathsStringBuilder.toString());            job.setMapperClass(mapper);          if (mapperKey != null) {              job.setMapOutputKeyClass(mapperKey);          }          if (mapperValue != null) {              job.setMapOutputValueClass(mapperValue);          }            jobConf.setBoolean("mapred.compress.map.output", true);            job.setCombinerClass(combiner);            job.setReducerClass(reducer);          job.setOutputKeyClass(reducerKey);          job.setOutputValueClass(reducerValue);            job.setOutputFormatClass(outputFormat);          jobConf.set("mapred.output.dir", outputPath);            return job;      }      public static String getCustomJobName(String className, JobContext job,                                    Class<? extends Mapper> mapper,                                    Class<? extends Reducer> reducer) {      StringBuilder name = new StringBuilder(100);      String customJobName = job.getJobName();      if (customJobName == null || customJobName.trim().isEmpty()) {        name.append(className);      } else {        name.append(customJobName);      }      name.append('-').append(mapper.getSimpleName());      name.append('-').append(reducer.getSimpleName());      return name.toString();    }        public static void delete(Configuration conf, Iterable<Path> paths) throws IOException {      if (conf == null) {        conf = new Configuration();      }      for (Path path : paths) {        FileSystem fs = path.getFileSystem(conf);        if (fs.exists(path)) {          log.info("Deleting {}", path);          fs.delete(path, true);        }      }    }      public static void delete(Configuration conf, Path... paths) throws IOException {      delete(conf, Arrays.asList(paths));    }      public static long countRecords(Path path, Configuration conf) throws IOException {      long count = 0;      Iterator<?> iterator = new SequenceFileValueIterator<Writable>(path, true, conf);      while (iterator.hasNext()) {        iterator.next();        count++;      }      return count;    }      public static long countRecords(Path path, PathType pt, PathFilter filter, Configuration conf) throws IOException {      long count = 0;      Iterator<?> iterator = new SequenceFileDirValueIterator<Writable>(path, pt, filter, null, true, conf);      while (iterator.hasNext()) {        iterator.next();        count++;      }      return count;    }  } |

先看看写的工具类：

第一步：处理原始输入数据

处理原始数据的SourceDataToItemPrefsJob作业的mapper：SourceDataToItemPrefsMapper

|  |
| --- |
| package com.mapper;    import java.io.IOException;  import java.util.regex.Matcher;  import java.util.regex.Pattern;    import org.apache.hadoop.io.LongWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.mahout.math.VarLongWritable;      /\*\*   \* mapper输入格式：userID:itemID1 itemID2 itemID3....   \* mapper输出格式:<userID,itemID>   \* @author 曾昭正   \*/  public class SourceDataToItemPrefsMapper extends Mapper<LongWritable, Text, VarLongWritable, VarLongWritable>{      //private static final Logger logger = LoggerFactory.getLogger(SourceDataToItemPrefsMapper.class);      private static final Pattern NUMBERS = Pattern.compile("(\\d+)");      private String line = null;        @Override      protected void map(LongWritable key, Text value,Context context)              throws IOException, InterruptedException {           line = value.toString();           if(line == null) return ;          // logger.info("line:"+line);           Matcher matcher = NUMBERS.matcher(line);           matcher.find();//寻找第一个分组，即userID           VarLongWritable userID = new VarLongWritable(Long.parseLong(matcher.group()));//这个类型是在mahout中独立进行封装的           VarLongWritable itemID = new VarLongWritable();           while(matcher.find()){               itemID.set(Long.parseLong(matcher.group()));          //   logger.info(userID + " " + itemID);               context.write(userID, itemID);           }      }  } |

处理原始数据的SourceDataToItemPrefsJob作业的reducer：SourceDataToItemPrefsMapper

|  |
| --- |
| package com.reducer;    import java.io.IOException;    import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.math.RandomAccessSparseVector;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.VectorWritable;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;    /\*\*   \* reducer输入：<userID,Iterable<itemID>>   \* reducer输出:<userID,VecotrWriable<index=itemID,valuce=pres>....>   \* @author 曾昭正   \*/  public class SourceDataToUserVectorReducer extends Reducer<VarLongWritable, VarLongWritable, VarLongWritable, VectorWritable>{      private static final Logger logger = LoggerFactory.getLogger(SourceDataToUserVectorReducer.class);      @Override      protected void reduce(VarLongWritable userID, Iterable<VarLongWritable> itemPrefs,Context context)              throws IOException, InterruptedException {          /\*\*           \*  DenseVector，它的实现就是一个浮点数数组，对向量里所有域都进行存储，适合用于存储密集向量。              RandomAccessSparseVector 基于浮点数的 HashMap 实现的，key 是整形 (int) 类型，value 是浮点数 (double) 类型，它只存储向量中不为空的值，并提供随机访问。              SequentialAccessVector 实现为整形 (int) 类型和浮点数 (double) 类型的并行数组，它也只存储向量中不为空的值，但只提供顺序访问。              用户可以根据自己算法的需求选择合适的向量实现类，如果算法需要很多随机访问，应该选择 DenseVector 或者 RandomAccessSparseVector，如果大部分都是顺序访问，SequentialAccessVector 的效果应该更好。              介绍了向量的实现，下面我们看看如何将现有的数据建模成向量，术语就是“如何对数据进行向量化”，以便采用 Mahout 的各种高效的聚类算法。           \*/          Vector userVector = new RandomAccessSparseVector(Integer.MAX\_VALUE, 100);          for(VarLongWritable itemPref : itemPrefs){              userVector.set((int)itemPref.get(), 1.0f);//RandomAccessSparseVector.set(index,value),用户偏好类型为boolean类型，将偏好值默认都为1.0f          }          logger.info(userID+" "+new VectorWritable(userVector));          context.write(userID, new VectorWritable(userVector));      }  } |

第二步：将SourceDataToItemPrefsJob作业的reduce输出结果组合成共现矩阵

UserVectorToCooccurrenceJob作业的mapper：UserVectorToCooccurrenceMapper

|  |
| --- |
| package com.mapper;    import java.io.IOException;  import java.util.Iterator;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.VectorWritable;    /\*\*   \* mapper输入：<userID,VecotrWriable<index=itemID,valuce=pres>....>   \* mapper输出:<itemID,itemID>(共现物品id对)   \* @author 曾昭正   \*/  public class UserVectorToCooccurrenceMapper extends Mapper<VarLongWritable, VectorWritable, IntWritable, IntWritable>{      @Override      protected void map(VarLongWritable userID, VectorWritable userVector,Context context)              throws IOException, InterruptedException {          Iterator<Vector.Element> it = userVector.get().nonZeroes().iterator();//过滤掉非空元素          while(it.hasNext()){              int index1 = it.next().index();              Iterator<Vector.Element> it2 = userVector.get().nonZeroes().iterator();              while(it2.hasNext()){                  int index2  = it2.next().index();                  context.write(new IntWritable(index1), new IntWritable(index2));              }          }        }  } |

UserVectorToCooccurrenceJob作业的reducer：UserVectorToCoocurrenceReducer

|  |
| --- |
| package com.reducer;    import java.io.IOException;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.cf.taste.hadoop.item.VectorOrPrefWritable;  import org.apache.mahout.math.RandomAccessSparseVector;  import org.apache.mahout.math.Vector;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  /\*\*   \* reducer输入:<itemID,Iterable<itemIDs>>   \* reducer输出:<mainItemID,Vector<coocItemID,coocTime(共现次数)>....>   \* @author 曾昭正   \*/  public class UserVectorToCoocurrenceReducer extends Reducer<IntWritable, IntWritable, IntWritable, VectorOrPrefWritable>{      private static final Logger logger = LoggerFactory.getLogger(UserVectorToCoocurrenceReducer.class);      @Override      protected void reduce(IntWritable mainItemID, Iterable<IntWritable> coocItemIDs,Context context)              throws IOException, InterruptedException {          Vector coocItemIDVectorRow = new RandomAccessSparseVector(Integer.MAX\_VALUE,100);          for(IntWritable coocItem : coocItemIDs){              int itemCoocTime = coocItem.get();              coocItemIDVectorRow.set(itemCoocTime,coocItemIDVectorRow.get(itemCoocTime)+1.0);//将共现次数累加          }          logger.info(mainItemID +" "+new VectorOrPrefWritable(coocItemIDVectorRow));          context.write(mainItemID, new VectorOrPrefWritable(coocItemIDVectorRow));//记录mainItemID的完整共现关系      }  } |

第三步：将SourceDataToItemPrefsJob作业的reduce输出结果进行分割

userVecotrSplitJob作业的mapper：UserVecotrSplitMapper

|  |
| --- |
| package com.mapper;    import java.io.IOException;  import java.util.Iterator;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.mahout.cf.taste.hadoop.item.VectorOrPrefWritable;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.Vector.Element;  import org.apache.mahout.math.VectorWritable;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;      /\*\*   \* 将用户向量分割，以便和物品的共现向量进行合并   \* mapper输入:<userID,Vector<itemIDIndex,preferenceValuce>....>   \* reducer输出:<itemID,Vecotor<userID,preferenceValuce>....>   \* @author 曾昭正   \*/  public class UserVecotrSplitMapper extends Mapper<VarLongWritable, VectorWritable, IntWritable, VectorOrPrefWritable>{      private static final Logger logger = LoggerFactory.getLogger(UserVecotrSplitMapper.class);      @Override      protected void map(VarLongWritable userIDWritable, VectorWritable value,Context context)              throws IOException, InterruptedException {          IntWritable itemIDIndex = new IntWritable();          long userID = userIDWritable.get();          Vector userVector = value.get();          Iterator<Element> it = userVector.nonZeroes().iterator();//只取非空用户向量          while(it.hasNext()){              Element e = it.next();              int itemID = e.index();              itemIDIndex.set(itemID);              float preferenceValuce = (float) e.get();              logger.info(itemIDIndex +" "+new VectorOrPrefWritable(userID,preferenceValuce));              context.write(itemIDIndex, new VectorOrPrefWritable(userID,preferenceValuce));          }        }  } |

第四步：将userVecotrSplitJob和UserVectorToCooccurrenceJob作业的输出结果合并

combineUserVectorAndCoocMatrixJob作业的mapper：CombineUserVectorAndCoocMatrixMapper

|  |
| --- |
| package com.mapper;    import java.io.IOException;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.mahout.cf.taste.hadoop.item.VectorOrPrefWritable;    /\*\*   \* 将共现矩阵和分割后的用户向量进行合并，以便计算部分的推荐向量   \* 这个mapper其实没有什么逻辑处理功能，只是将数据按照输入格式输出   \* 注意：这里的mapper输入为共现矩阵和分割后的用户向量计算过程中的共同输出的2个目录   \* mapper输入：<itemID,Vecotor<userID,preferenceValuce>> or <itemID,Vecotor<coocItemID,coocTimes>>   \* mapper输出:<itemID,Vecotor<userID,preferenceValuce>/Vecotor<coocItemID,coocTimes>>   \* @author 曾昭正   \*/  public class CombineUserVectorAndCoocMatrixMapper extends Mapper<IntWritable, VectorOrPrefWritable, IntWritable, VectorOrPrefWritable>{      @Override      protected void map(IntWritable itemID, VectorOrPrefWritable value,Context context)              throws IOException, InterruptedException {          context.write(itemID, value);      }    } |

combineUserVectorAndCoocMatrixJob作业的CombineUserVectorAndCoocMatrixReducer

|  |
| --- |
| package com.reducer;    import java.io.IOException;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.cf.taste.hadoop.item.VectorAndPrefsWritable;  import org.apache.mahout.cf.taste.hadoop.item.VectorOrPrefWritable;  import org.apache.mahout.math.Vector;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;    /\*\*   \* 将共现矩阵和分割后的用户向量进行合并，以便计算部分的推荐向量   \* @author 曾昭正   \*/  public class CombineUserVectorAndCoocMatrixReducer extends Reducer<IntWritable, VectorOrPrefWritable, IntWritable, VectorAndPrefsWritable>{      private static final Logger logger = LoggerFactory.getLogger(CombineUserVectorAndCoocMatrixReducer.class);      @Override      protected void reduce(IntWritable itemID, Iterable<VectorOrPrefWritable> values,Context context)              throws IOException, InterruptedException {          VectorAndPrefsWritable vectorAndPrefsWritable = new VectorAndPrefsWritable();          List<Long> userIDs = new ArrayList<Long>();          List<Float> preferenceValues = new ArrayList<Float>();          Vector coocVector = null;          Vector coocVectorTemp = null;          Iterator<VectorOrPrefWritable> it = values.iterator();          while(it.hasNext()){              VectorOrPrefWritable e = it.next();              coocVectorTemp = e.getVector() ;              if(coocVectorTemp == null){                  userIDs.add(e.getUserID());                  preferenceValues.add(e.getValue());              }else{                  coocVector = coocVectorTemp;              }          }          if(coocVector != null){              //这个需要注意，根据共现矩阵的计算reduce聚合之后，到了这个一个Reudce分组就有且只有一个vecotr(即共现矩阵的一列或者一行，这里行和列是一样的)了。              vectorAndPrefsWritable.set(coocVector, userIDs, preferenceValues);              logger.info(itemID+" "+vectorAndPrefsWritable);              context.write(itemID, vectorAndPrefsWritable);          }      }  } |

第五步：将combineUserVectorAndCoocMatrixJob作业的输出结果生成推荐列表

caclPartialRecomUserVectorJob作业的mapper：CaclPartialRecomUserVectorMapper

|  |
| --- |
| package com.mapper;    import java.io.IOException;  import java.util.List;    import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Mapper;  import org.apache.mahout.cf.taste.hadoop.item.VectorAndPrefsWritable;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.VectorWritable;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;    /\*\*   \* 计算部分用户推荐向量   \* @author 曾昭正   \*/  public class CaclPartialRecomUserVectorMapper extends Mapper<IntWritable,VectorAndPrefsWritable, VarLongWritable, VectorWritable>{      private static final Logger logger = LoggerFactory.getLogger(CaclPartialRecomUserVectorMapper.class);      @Override      protected void map(IntWritable itemID, VectorAndPrefsWritable values,Context context)              throws IOException, InterruptedException {          Vector coocVectorColumn = values.getVector();          List<Long> userIDs = values.getUserIDs();          List<Float> preferenceValues = values.getValues();          for(int i = 0; i< userIDs.size(); i++){              long userID = userIDs.get(i);              float preferenceValue = preferenceValues.get(i);              logger.info("userID:" + userID);              logger.info("preferenceValue:"+preferenceValue);              //将共现矩阵中userID对应的列相乘，算出部分用户对应的推荐列表分数              Vector preferenceParScores = coocVectorColumn.times(preferenceValue);              context.write(new VarLongWritable(userID), new VectorWritable(preferenceParScores));          }      }  } |

caclPartialRecomUserVectorJob作业的combiner：ParRecomUserVectorCombiner

|  |
| --- |
| package com.reducer;    import java.io.IOException;    import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.VectorWritable;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  /\*\*   \* 将计算部分用户推荐向量的结果进行合并，将userID对应的贡现向量的分值进行相加(注意：这个只是将一个map的输出进行合并，所以这个也是只部分结果)   \* @author 曾昭正   \*/  public class ParRecomUserVectorCombiner extends Reducer<VarLongWritable, VectorWritable, VarLongWritable, VectorWritable>{      private static final Logger logger = LoggerFactory.getLogger(ParRecomUserVectorCombiner.class);      @Override      protected void reduce(VarLongWritable userID, Iterable<VectorWritable> coocVectorColunms,Context context)              throws IOException, InterruptedException {            Vector vectorColunms = null;            for(VectorWritable  coocVectorColunm : coocVectorColunms){              vectorColunms = vectorColunms == null ? coocVectorColunm.get() : vectorColunms.plus(coocVectorColunm.get());          }          logger.info(userID +" " + new VectorWritable(vectorColunms));          context.write(userID, new VectorWritable(vectorColunms));      }  } |

caclPartialRecomUserVectorJob作业的reducer：MergeAndGenerateRecommendReducer

|  |
| --- |
| package com.reducer;    import java.io.IOException;  import java.util.ArrayList;  import java.util.Collections;  import java.util.Iterator;  import java.util.List;  import java.util.PriorityQueue;  import java.util.Queue;    import org.apache.hadoop.mapreduce.Reducer;  import org.apache.mahout.cf.taste.hadoop.RecommendedItemsWritable;  import org.apache.mahout.cf.taste.impl.recommender.ByValueRecommendedItemComparator;  import org.apache.mahout.cf.taste.impl.recommender.GenericRecommendedItem;  import org.apache.mahout.cf.taste.recommender.RecommendedItem;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.Vector;  import org.apache.mahout.math.Vector.Element;  import org.apache.mahout.math.VectorWritable;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;    /\*\*   \* 合并所有已经评分的共现矩阵   \* @author 曾昭正   \*/  public class MergeAndGenerateRecommendReducer extends Reducer<VarLongWritable, VectorWritable, VarLongWritable, RecommendedItemsWritable>{      private static final Logger logger = LoggerFactory.getLogger(MergeAndGenerateRecommendReducer.class);      private int recommendationsPerUser;      @Override      protected void setup(Context context)              throws IOException, InterruptedException {          recommendationsPerUser = context.getConfiguration().getInt("recomandItems.recommendationsPerUser", 5);      }      @Override      protected void reduce(VarLongWritable userID, Iterable<VectorWritable> cooVectorColumn,Context context)              throws IOException, InterruptedException {          //分数求和合并          Vector recommdVector = null;          for(VectorWritable vector : cooVectorColumn){              recommdVector = recommdVector == null ? vector.get() : recommdVector.plus(vector.get());          }          //对推荐向量进行排序，为每个UserID找出topM个推荐选项(默认找出5个)，此队列按照item对应的分数进行排序          //注意下：PriorityQueue队列的头一定是最小的元素,另外这个队列容量增加1是为了为添加更大的新元素时使用的临时空间          Queue<RecommendedItem> topItems = new PriorityQueue<RecommendedItem>(recommendationsPerUser+1, ByValueRecommendedItemComparator.getInstance());            Iterator<Element> it = recommdVector.nonZeroes().iterator();          while(it.hasNext()){              Element e = it.next();              int itemID = e.index();              float preValue = (float) e.get();              //当队列容量小于推荐个数，往队列中填item和分数              if(topItems.size() < recommendationsPerUser){                  topItems.add(new GenericRecommendedItem(itemID, preValue));              }              //当前item对应的分数比队列中的item的最小分数大，则将队列头原始（即最小元素）弹出，并且将当前item：分数加入队列              else if(preValue > topItems.peek().getValue()){                  topItems.add(new GenericRecommendedItem(itemID, preValue));                  //弹出头元素（最小元素）                  topItems.poll();              }          }            //重新调整队列的元素的顺序          List<RecommendedItem> recommdations = new ArrayList<RecommendedItem>(topItems.size());          recommdations.addAll(topItems);//将队列中所有元素添加即将排序的集合          Collections.sort(recommdations,ByValueRecommendedItemComparator.getInstance());//排序            //输出推荐向量信息          logger.info(userID+" "+ new RecommendedItemsWritable(recommdations));          context.write(userID, new RecommendedItemsWritable(recommdations));        }  } |

第六步：组装各个作业关系

PackageRecomendJob

|  |
| --- |
| package com.mapreduceMain;    import java.io.IOException;  import java.net.URI;    import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.conf.Configured;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.IntWritable;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;  import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;  import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;  import org.apache.hadoop.util.Tool;  import org.apache.hadoop.util.ToolRunner;  import org.apache.mahout.cf.taste.hadoop.RecommendedItemsWritable;  import org.apache.mahout.cf.taste.hadoop.item.VectorAndPrefsWritable;  import org.apache.mahout.cf.taste.hadoop.item.VectorOrPrefWritable;  import org.apache.mahout.math.VarLongWritable;  import org.apache.mahout.math.VectorWritable;  import com.mapper.CaclPartialRecomUserVectorMapper;  import com.mapper.CombineUserVectorAndCoocMatrixMapper;  import com.mapper.UserVecotrSplitMapper;  import com.mapper.UserVectorToCooccurrenceMapper;  import com.mapper.SourceDataToItemPrefsMapper;  import com.reducer.CombineUserVectorAndCoocMatrixReducer;  import com.reducer.MergeAndGenerateRecommendReducer;  import com.reducer.ParRecomUserVectorCombiner;  import com.reducer.UserVectorToCoocurrenceReducer;  import com.reducer.SourceDataToUserVectorReducer;  import com.util.HadoopUtil;      /\*\*   \* 组装各个作业组件，完成推荐作业   \* @author 曾昭正   \*/  public class PackageRecomendJob extends Configured implements Tool{      String[] dataSourceInputPath = {"/user/hadoop/z.zeng/distruteItemCF/dataSourceInput"};      String[] uesrVectorOutput = {"/user/hadoop/z.zeng/distruteItemCF/uesrVectorOutput/"};      String[] userVectorSpliltOutPut = {"/user/hadoop/z.zeng/distruteItemCF/userVectorSpliltOutPut"};      String[] cooccurrenceMatrixOuptPath = {"/user/hadoop/z.zeng/distruteItemCF/CooccurrenceMatrixOuptPath"};      String[] combineUserVectorAndCoocMatrixOutPutPath = {"/user/hadoop/z.zeng/distruteItemCF/combineUserVectorAndCoocMatrixOutPutPath"};      String[] caclPartialRecomUserVectorOutPutPath = {"/user/hadoop/z.zeng/distruteItemCF/CaclPartialRecomUserVectorOutPutPath"};        protected void setup(Configuration configuration)              throws IOException, InterruptedException {          FileSystem hdfs = FileSystem.get(URI.create("hdfs://cluster-master"), configuration);          Path p1 = new Path(uesrVectorOutput[0]);          Path p2 = new Path(userVectorSpliltOutPut[0]);          Path p3 = new Path(cooccurrenceMatrixOuptPath[0]);          Path p4 = new Path(combineUserVectorAndCoocMatrixOutPutPath[0]);          Path p5 = new Path(caclPartialRecomUserVectorOutPutPath[0]);            if (hdfs.exists(p1)) {              hdfs.delete(p1, true);          }          if (hdfs.exists(p2)) {              hdfs.delete(p2, true);          }          if (hdfs.exists(p3)) {              hdfs.delete(p3, true);          }          if (hdfs.exists(p4)) {              hdfs.delete(p4, true);          }          if (hdfs.exists(p5)) {              hdfs.delete(p5, true);          }      }      @Override      public int run(String[] args) throws Exception {                Configuration conf=getConf(); //获得配置文件对象                setup(conf);            //  DistributedCache.addArchiveToClassPath(new Path("/user/hadoop/z.zeng/distruteItemCF/lib"), conf);            //配置计算用户向量作业            Job wikipediaToItemPrefsJob = HadoopUtil.prepareJob(                      "WikipediaToItemPrefsJob",                        dataSourceInputPath,                        uesrVectorOutput[0],                        TextInputFormat.class,                        SourceDataToItemPrefsMapper.class,                        VarLongWritable.class,                        VarLongWritable.class,                        SourceDataToUserVectorReducer.class,                        VarLongWritable.class,                        VectorWritable.class,                        SequenceFileOutputFormat.class,                        conf);            //配置计算共现向量作业            Job userVectorToCooccurrenceJob = HadoopUtil.prepareJob(                      "UserVectorToCooccurrenceJob",                        uesrVectorOutput,                        cooccurrenceMatrixOuptPath[0],                        SequenceFileInputFormat.class,                        UserVectorToCooccurrenceMapper.class,                        IntWritable.class,                        IntWritable.class,                        UserVectorToCoocurrenceReducer.class,                        IntWritable.class,                        VectorOrPrefWritable.class,                        SequenceFileOutputFormat.class,                        conf);            //配置分割用户向量作业            Job userVecotrSplitJob = HadoopUtil.prepareJob(                      "userVecotrSplitJob",                        uesrVectorOutput,                        userVectorSpliltOutPut[0],                        SequenceFileInputFormat.class,                        UserVecotrSplitMapper.class,                        IntWritable.class,                        VectorOrPrefWritable.class,                        SequenceFileOutputFormat.class,                        conf);            //合并共现向量和分割之后的用户向量作业            //这个主意要将分割用户向量和共现向量的输出结果一起作为输入            String[] combineUserVectorAndCoocMatrixIutPutPath = {cooccurrenceMatrixOuptPath[0],userVectorSpliltOutPut[0]};            Job combineUserVectorAndCoocMatrixJob = HadoopUtil.prepareJob(                      "combineUserVectorAndCoocMatrixJob",                      combineUserVectorAndCoocMatrixIutPutPath,                        combineUserVectorAndCoocMatrixOutPutPath[0],                        SequenceFileInputFormat.class,                        CombineUserVectorAndCoocMatrixMapper.class,                        IntWritable.class,                        VectorOrPrefWritable.class,                        CombineUserVectorAndCoocMatrixReducer.class,                        IntWritable.class,                        VectorAndPrefsWritable.class,                        SequenceFileOutputFormat.class,                        conf);            //计算用户推荐向量            Job caclPartialRecomUserVectorJob= HadoopUtil.prepareJob(                      "caclPartialRecomUserVectorJob",                      combineUserVectorAndCoocMatrixOutPutPath,                      caclPartialRecomUserVectorOutPutPath[0],                      SequenceFileInputFormat.class,                        CaclPartialRecomUserVectorMapper.class,                        VarLongWritable.class,                        VectorWritable.class,                        ParRecomUserVectorCombiner.class,//为map设置combiner减少网络IO                        MergeAndGenerateRecommendReducer.class,                        VarLongWritable.class,                        RecommendedItemsWritable.class,                        TextOutputFormat.class,                        conf);              //串联各个job            if(wikipediaToItemPrefsJob.waitForCompletion(true)){                if(userVectorToCooccurrenceJob.waitForCompletion(true)){                    if(userVecotrSplitJob.waitForCompletion(true)){                        if(combineUserVectorAndCoocMatrixJob.waitForCompletion(true)){                             int rs = caclPartialRecomUserVectorJob.waitForCompletion(true) ? 1 :0;                            return rs;                        }else{                            throw new Exception("合并共现向量和分割之后的用户向量作业失败！！");                        }                    }else{                        throw new Exception("分割用户向量作业失败！！");                    }                }else{                    throw new Exception("计算共现向量作业失败！！");                }            }else{                throw new Exception("计算用户向量作业失败！！");            }      }      public static void main(String[] args) throws IOException,              ClassNotFoundException, InterruptedException {          try {              int returnCode =  ToolRunner.run(new PackageRecomendJob(),args);              System.exit(returnCode);          } catch (Exception e) {          }      }    } |

## 5、总结

    本blog主要说了下itemBase推荐算法的一些概念，以及如何多现有数据进行建模。其中对共现矩阵方式的推荐用MapReduce结合Mahout的内置数据类型进行了实现。写完这篇blog和对算法实现完毕后，发现Mapreduce编程虽然数据模型非常简单，只有2个过程：数据的分散与合并，但是在分散与合并的过程中可以使用自定义的各种数据组合类型使其能够完成很多复杂的功能。