实验八　基于Spark MLlib实现音乐推荐

（一）实验目的

1. 熟悉数据的划分：训练集、检验集和测试集；
2. 了解机器学习算法超参数的选择；
3. 熟悉基于MLlib进行数据分析的流程。

（二）实验要求

1. 熟悉Audioscrobbler数据集；
2. 基于该数据集选择合适的MLlib库算法进行数据处理；
3. 进行音乐推荐（或用户推荐）。

（三）实验环境

1. 大数据分析实验系统（FSDP）；
2. CentOS 6.7；
3. Hadoop 2.7.1；
4. Spark 1.4.1。

（四）实验步骤

1. 了解Audioscrobbler数据集；
2. 算法选择及数据准备；
3. 构建模型及模型调优；
4. 得到推荐结果。

1、了解Audioscrobbler数据集

(1) 仅记录了播放数据：如“Bob播放了一首Prince的歌曲”，包含的信息比评分要少。
(2) 隐式反馈数据：虽然单条记录的信息比较少，但此数据集很大，覆盖了很多用户和艺术家，包含了更多总体信息。
(3) 数据集包含文件：

- user_artist_data.txt：（3 columns: userid, artistid, playcount），包含14.1万个用户和160万个艺术家，约2420万条用户播放记录；
- artist_data.txt：（2 columns: artistid, artist_name），184.8万条数据；
- artist_alias.txt：（2 columns: badid, goodid），19.3万条数据，（known incorrectly spelt artists and the correct artist id. you can correct errors in user_artist_data as you read it in using this file.）

2、算法选择--交替最小二乘推荐算法

(1) 要找的推荐算法不需要用户和艺术家的属性信息，这类算法通常称为协同过滤算法；

(2) 数据虽说包含了数千万条播放信息，看起来很大，但从另一方面来看数据集又小而且不充足，因为数据是稀疏的；

(3) 本案例中将用到潜在因素模型中的一种模型；

(4) 矩阵分解模型：把用户和产品数据当成一个大矩阵A，矩阵的第i行和第j列的元素有值，代表用户i播放过艺术家j的音乐；

(5) 算法将A分解为两个小矩阵X和Y的乘积。矩阵X和矩阵Y非常“瘦”，因为A有很多行很多列，但X和YT 行很多而列很少（列数用k表示）。这个k就是潜在的因素，用于解释数据中的交互关系。

(6) 由于k很小，矩阵分解算法只能是近似： $A \approx XY^T$；A可能非常稀疏，但乘积 XY^T是稠密的；

(7) 矩阵X和Y^T，分别有一行对应每个用户和艺术家，每行值很少，只有k个，每个值代表了一个隐含特征；

(8) 不幸的是，$A = XY^T$通常无解，原因就是X和Y通常不够大，无法完美表示A。这其实是件好事。A只是所有可能出现的交互关系的一个微小样本。在某种程度上，我们认为A是对基本事实的一次基本观察，它太稀疏，因此很难解释这个基本事实。但用少数几个因素（k个）就能很好地解释这个基本事实。

(9) XY^T应该尽可能逼近A，毕竟这是所有后续工作的基础，但不能也不应该完全复制A。然而同样不幸的是，想直接同时得到X和Y的最优解是不可能的。但是，如果Y已知，求解X 是非常容易的，反之亦然。但X和事先都是未知的。幸好有算法帮我们摆脱这两难的境地，具体来说，求解X和Y时，使用最小交替二乘算法。

(10) 求解：$A = XY^T$：虽然Y是未知的，但我们可以把它初始为随机行向量矩阵。接着运用简单的线性代数，就能在给定A和Y的条件下求出X的最优解。实际上，X的第i行是A的第i行和Y的函数，因此可以很容易分开计算X的每一行。因为X每一行可以分开计算，所以我们可以将其并行化，而并行化是大规模计算的一大优点。$A_i Y (Y^T Y)^{-1} = X_i$

(11) 我们可以由X计算每个Y_j，然后又可以由Y计算X，这样反复下去，这就是算法名称“交替”的来由。这里有一个问题：Y是“瞎编”的，并且是随机的。不过，随着交替计算的一直继续，X和Y最终会收敛得到一个合适的结果。

实验八　基于Spark MLlib实现音乐推荐

3、数据准备

(1)将三个数据文件均放到了HDFS上；
(2)由于运算非常占用内存，启动Spark-shell时指定参数--driver-memory　6g
(3)ALS算法要求用户和产品ID必须是数值型，并且是32位非负整数，通过对数据的分析，发现ID都是合法的。

4、构建模型及查看推荐结果

(1)Rating对象是ALS算法实现对“用户-产品-值”的抽象，这里“产品”指的是艺术家

```
import org.apache.spark.mllib.recommendation._
val bArtistAlias = sc.broadcast(artistAlias)
val trainData = rawUserArtistData.map { line =>
val Array(userID, artistID, count)=line.split(' ').map(_.toInt)
val finalArtistID=bArtistAlias.value.getOrElse(artistID,artistID)Rating(userID,finalArtistID,count)}.cache()
val model = ALS.trainImplicit(trainData, 10 , 5, 0.01, 1.0)
model.userFeatures.mapValues(_.mkString(",")).first()
```

(2)查看推荐结果是否合理，如用户2093760

```
Val rawArtistForUser=rawUserArtistData.map(_.split(' ')).filter{ case Array(user,_,_) => user.toInt == 2093760 }
val existingProducts = rawArtistForUser.map{ case Array(_,artist,_) => artist.toInt }.collect().toSet
artistByID.filter { case(id,name) => existingProducts.contains(id)}.values.collect.foreach(println)
Valrecommendations=model.recommendProducts(2093760, 5) recommendations.foreach(println)
ValrecommendedProuctIDs=recommendations.map(_.product).toSetartistByID.filter { case (id,name) =>recommendedProuctIDs.contains(id)}.values.collect().foreach(println)
```

5、评价推荐质量

(1)AUC：可以看成是随机选择的好推荐比随机选择的差推荐排名高的概率；
(2)取出一部分数据来选择模型并评估模型准确度是所有机器学习的通用做法。通常数据被分为三个子集：训练集、检验（集和测试集。本案例只用到了训练集和检验集。
(3)详见代码（其中AUC计算：areaUnderCurve函数）

6、模型调优--选择超参数

(1)超参数的值不是由算法学习得到的，而是由调用者指定的；
(2)rank = 10 模型潜在因素个数，即“用户-特征”和“产品-特征”矩阵的列数；
(3)iterations = 5 矩阵分解迭代的次数；迭代次数越多，花费的时间越长，但是分解的结果可能更好；
(4)lambda = 0.01 标准的过拟合参数；值越大越不容易产生过拟合，但值太大会降低分解的准确度；
(5)alpha =1.0 控制分解矩阵时，被观察到的“用户-产品”交互相对没被观察到的交互的权重
(6)可将rank、lambda和alpha看作为模型的超参数（iterations更像是对分解过程使用资源的一种约束）；
(7)我们使用的超参数不一定是最优的，如何选择好的超参数是一个普遍性的问题，最基本的方法是尝试不同值的组合并对每个组合评估某个指标，然后挑选指标值最好的组合。

7、产生推荐

(1)用前面获取的最优参数对ID为2093760用户给出推荐：[unknown]、The Beatles、Coldplay、U2、Green Day；
(2)对100个用户进行推荐的结果（详见代码）。

```
val someUsers =
allData.map(_.user).distinct().take(100)
val someRecommendations = someUsers.map(userID => model.recommendProducts(userID,5))
someRecommendations.map(recs => recs.head.user + " -> " + recs.map(_.product).mkString(", ")).foreach(println)
```

（3）整个流程也可以用于向艺术家推荐用户，可用于回答类似如下问题：“艺术家X的新专辑哪100个用户可能最感兴趣？”在向艺术家推荐用户时，只需要在解析输入的时候对换用户和艺术家字段就可以了。

```
rawUserArtistData.map { line =>
            ...
            val userID = tokens(1).toInt
            val artistID = tokens(0).toInt
            ...
}
```

8、备注

① 代码见 musicrecommend.scala；
② 代码有详细注解。