基于 Hadoop Map-Reduce 的推荐系统

1. 设计思路

通过分析题目要求，我们首先关注到在评判标准中采用了均方根误差和平均绝对误差，这两项标准为评分预测问题的常用指标，并且由于最终进行十折交叉交叉验证是将数据随机分成不相交的集合，最后的评判没有准确率、覆盖率和召回率等推荐评判指标，因此我们认为这是一个评分预测问题。

如果是传统的推荐，我们只会推荐“预测分数”较高的电影给相关用户，但实际上，测试集中的数据也有评分很低的。为了应对这种问题，有两种方法：一是采用基于用户相似度的推荐方法；二是进行全样本的打分预测；这里我们采用了第二种方法，因为测试数据和训练数据之间没有固定的时序关系，是完全随机的。该种方法其实就是一种基于物品的协同过滤算法。

采用全样本打分可以尽可能多的涵盖测试集中的数据，即我们会给每一个训练集的用户对每一个电影产生一个我们预测的评分，这样保证我们不遗漏所有可以进行有效评判的评分数据。

2. 实现方案

为了实现上述设计，我们需要依赖初始数据，生成如下几种数据结构。

2.1 用户评分矩阵

第一个 Map 和 Reduce 的任务就是对输入文件进行基本的扫描和统计，生成这个矩阵。该矩阵的输出设计的示例如下所示：

1 101:5.0,102:3.0,103:2.5

2 101:2.0,102:2.5,103:5.0,104:2.0

……

8 104:4.0,105:3.5,103:2.0

第一行表示用户1看过了101、102和103这三个电影，评分分别为5.0、3.0和2.5。其余行以此类推。

2.2 共现矩阵

生成共现矩阵是必须的。该矩阵表达了用户看过的电影中，两两之间的共同出现次数。比如，若3个用户都给122和185电影有过打分，则122和185的共现次数为3。第二个 Map 和 Reduce 就是以第一次 Reduce 的输出作为输入，完成从用户评分矩阵到共现矩阵的转换。输出的数据格式如下所示。

101@101 5

101@102 3

……

122@122 1

第一行表示101和101（即101本身）这个电影一共出现了5次。第二行表示101和102这两个电影共同出现了3次，以此类推。

2.3 评分矩阵

第三个 Map 依然以用户评分矩阵作为输入，对其继续转换，生成以电影 ID 作为 Key，用户 ID 和评分作为 value 的输出，为最终的矩阵相乘做好准备，其格式如下所示：

101 1@5.0

101 5@4.0

……

132 8@3.5

第一行即表示对于电影101，用户1的评分是5.0。

2.4 生成所有评分

通过之前生成的共现矩阵和评分矩阵，对着两个矩阵做乘法运算，我们可以产生一个“未归一化”的评分结果，该结果包含了一个用户对所有电影（训练集中出现的）的“评分”。矩阵的乘法通过两组 Map 和 Reduce 完成，第一个 Map 和 Reduce 完成每一项的乘法，第二个 Map 和 Reduce 将每一项的乘法结果进行相加，即完成矩阵的乘法。下表展示了这个过程：

	101	102	103	104		1		Result
101	3	3	2	1		0.0		11.0
102	3	4	3	1	×	0.0	=	14.5
103	2	3	4	2		3.5		22.0
104	1	1	2	2		4.0		15.0

上图中，左侧为电影的共现矩阵，右侧只单列出了用户1的评分向量作为示例，两个矩阵相乘的结果即为一个未归一化的评分结果——用户1对所有电影的评分偏好。

2.5 评分原理

上述结果中（Result），对于103和104的评分可以忽略，因为用户1已经看过他们了，这个评分值不具有实际意义。对于101和102这两个电影，102的评分要比101高，这两个结果分别是共现矩阵的第一行和第二行与用户评分向量的点积结果。

对于第一行而言，其包含了101电影与其它电影的共现次数，我们的计算方法将共现次数和用户的已有评分进行乘积，并进行相加，那么如果101电影和用户看过的电影总是出现（即累加项多或/和每个累加项的次数多），那么最终的结果也会越大。

2.4节的例子中，102的评分结果比101大，也正是由于102和103的共现次数比101和103的共现次数大，表明了102比101更符合用户的偏好，所以在最终的评分结果中102的分数比101高。

2.6 结果归一化

最终的输出中，我们不能直接把11.0和14.5这样的评分结果输出，因为用户的实际评分在0~5之间，所以需要借助现有的信息对评分结果归一化，让其“返回”0~5这个区间。

如果采用常见的线性转换算法，来将最终的评分向量进行直接归一化，则我们会从结果中选择一个最小值和最大值，这样使得最终归一化的结果中必然有评分值为0或5，这显然比较明显地是去了分数本身的一些精确度。因此我们采用了另外一种思路。

以2.4节中，102电影的评分产生为例。我们产生14.5分的计算过程为：

$3\times0 + 4\times0 + 3\times3.5 + 1\times4.0 = 14.5$

归一化的方法即为：

$14.5 / (3 + 4 + 3 + 1) = 1.31818181\cdots$

也就是说我们将共现次数视为用户实际评分这个权重的参数，这个未归一化结果的加权除以这个参数和，得到其实际应有的权重，该权重必在0~5之间，首先对其保留一位小数。由此得到的结果为1.3。然后对其乘2并取上整，即2.6取上整为3，再除以2，即得到最终的评分值为1.5。采用这种方法，对于保留一位小数不为整数的值，在(x.0, x.5)区间的将被归一化为 x.5 分；在(x.5, x+1.0)区间的将被归一化为 x+1。如果保留一位小数是整数值，则转换后结果不变。

上述示例仅仅是为了说明计算过程，因为样本数较少，所以使得看起来似乎是符合用户偏好的电影，评分依然不高。实际测试中，该值会更加接近真实评分值。这种归一化的结果除了涵盖了物品相似性信息，也包含了用户打分的习惯，因为除以参数的总和会使得评分更趋向于用户对其它喜好电影打分的均值。

3. 测试结果

为了验证评分的准确性，我们随机选择了官方数据集的10w 行数据，将其随机拆分为10组不相交的训练集和测试集，分别为 r1.train，r2.train，…，r10.train 和 r1.test，r2.test，…，r10.test。输入文件即为 *.train 文件，然后利用 *.test 文件评判输出文件，计算其 RMSE 和 MAE值。

3.1 运行方法

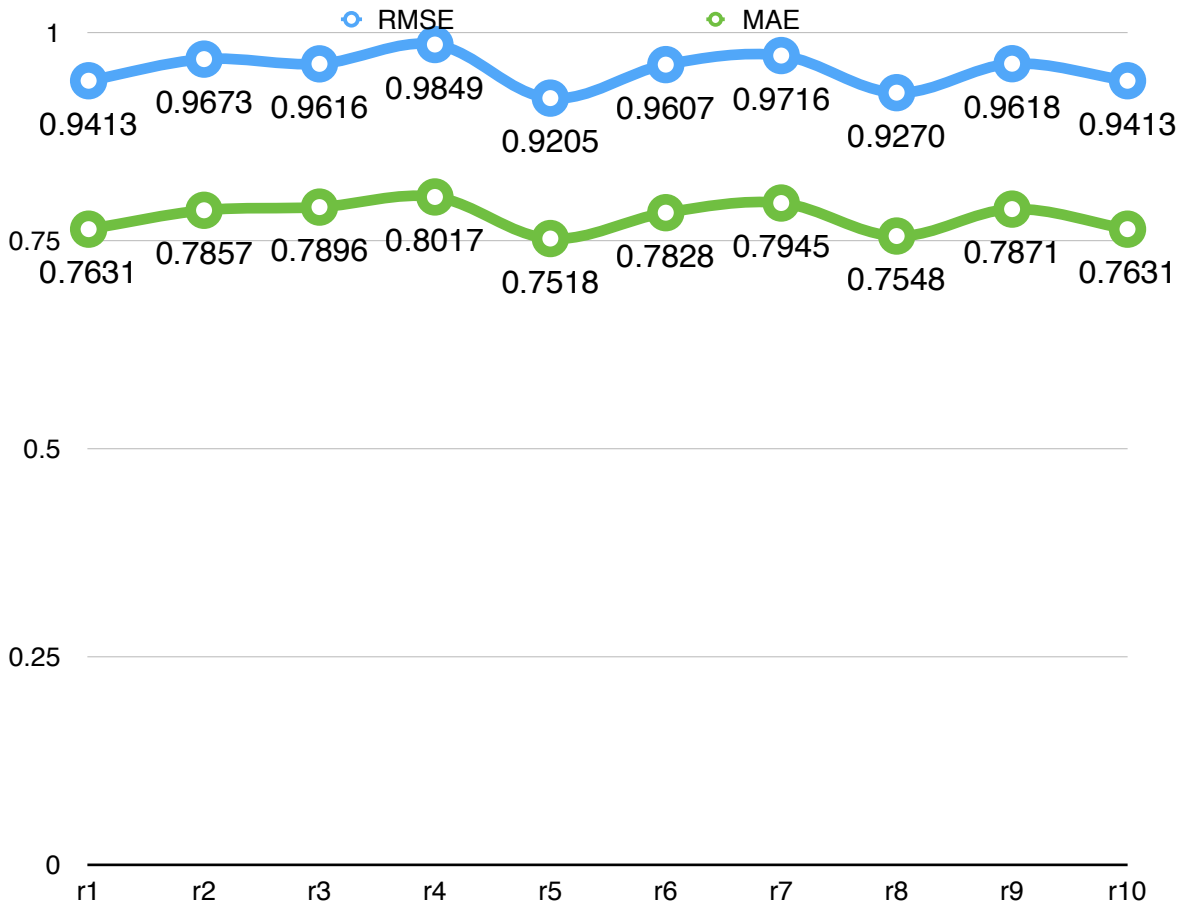
程序的 jar 包为 DataSpark_problem2.jar。本程序在 Hadoop 2.4.1环境下分布式运行，使用方法为：

hadoop jar DataSpark_problem2.jar <local_filesystem_path> <output_path> [IP:PORT]

第三个参数为可选的，因为程序在运行过程中需要反复读写 HDFS，我们的测试集群的 HDFS 地址为：hdfs://namenode:9000 。因此，若在其它集群中运行，可能需要更改该地址，只需在第三个参数处以 IP:PORT 的形式传入 IP 地址（或机器名）和 HDFS 端口即可。

3.2 评判结果

为了评判程序运行的结果，我们用 Python 编写的评判程序（详见附件），分别用对应的 test 文件计算每一组输出文件的 RMSE 和 MAE 的值，再计算10次结果的平均值。评分结果如下所示：



由此得到的十折交叉的均值分别为：

RMSE = 0.953809

MAE = 0.777414