Mahout Java 推荐引擎（一）

作者：pencilso
原文链接：https://ld246.com/article/1534307735099
来源网站：链滴

最近产品有一个需求是 给用户推荐 用户可能喜欢的东西

后来找到了Mahout这套引擎 这里主要记录一下 基于用户的协同过滤算法推荐引擎使用

基于用户的协同过滤算法 原理的话 网上有很多资料 这里就不详解算法了 只记录Mahout的使用

引入Maven

Mahout 自己引入了hadoop 而我不需要 就将它给剔除了

```
<dependency>
    <groupId>org.apache.mahout</groupId>
    <artifactId>mahout-core</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.apache.hadoop</groupId>
            <artifactId>hadoop-core</artifactId>
        </exclusion>
    </exclusions>
    <version>0.9</version>
</dependency>
```

数据模型

在Mahout引擎中 当需要给某一个用户进行推荐的时候 肯定要基于现有的数据进行推荐

```
//数据格式
userId,itemId,preference
例如:
    1,2,3.2 //1表示用户ID 2表示为itemID  3.2表示用户对这个item的喜好程度
这里需要注意的是 用户ID  和 ItemId 都只能为整数类型

我们先来随机生成一些用户和Item 并且让他们关联上
ps:这里  我使用了SnowFlake算法的ID生成器（百度上可以找到） 生成的都是长整数的ID

    //以下是生成一些随机关联数据的代码

    Random random = new Random();
    LinkedList<Long> userId = new LinkedList<>();
    LinkedList<Long> itemId = new LinkedList<>();
    //拼接用户关联Item的数据
    StringBuffer data = new StringBuffer();
    //拼接用户
    StringBuffer user = new StringBuffer();
    //生成一千个Item
    for (int i = 0; i < 1000; i++) {
        Long aLong = IdGeneratorCore.generatorId();
        itemId.add(aLong);
    }

    //生成一百个用户
```

```
    for (int i = 0; i < 100; i++) {
        Long uId = IdGeneratorCore.generatorId();
        userId.add(uId);
        //拷贝一份Item列表
        ArrayList<Long> itemArray = new ArrayList<>();
        itemArray.addAll(itemId);
        //给这个用户 随机关联一些Item 随机数量为200以内
        int count = random.nextInt(200);
        for (int j = 0; j < count; j++) {
            //随机取一个角标
            int randomIndex = random.nextInt(itemArray.size());
            Long id = itemArray.get(randomIndex);
            itemArray.remove(randomIndex);
            data.append(uId).append(",").append(id).append(",").append(
                    //随机生成5以内的喜好程度
                    Float.valueOf(random.nextInt(5)
                    )).append("\r\n");
        }
        user.append(uId).append(" count:").append(count).append("\r\n");
    }
    //存储用户关联Item的数据
    FileOutputStream dataOut = new FileOutputStream(new File("D:/data.txt"));
    //存储用户ID 和 他关联的数量  实际开发中 这个不需要存储  我这里存储只是为了后面方便测试
    FileOutputStream userOut = new FileOutputStream(new File("D:/user.txt"));
    dataOut.write(data.toString().getBytes());
    userOut.write(data.toString().getBytes());
```

最后生成的用户关联Item的数据如下

```
479260143402353209,479260143402353204,1.0
479260143402353209,479260143402352676,2.0
479260143402353209,479260143402352897,1.0
479260143402353209,479260143398158518,1.0
479260143402353209,479260143398158594,0.0
479260143402353209,479260143402353063,3.0
..... 很多数据  我就不全贴上来了
```

最后生成的用户ID 和 关联的Item数量如下

```
479261239340434044 count:84
479261239344627712 count:185
479261239353016320 count:117
479261239353016321 count:115
479261239357210624 count:184
479261239365599232 count:75
479261239365599233 count:74
.....
```

构建File数据模型

当我们的用户关联Item的数据 存储在本地文件的时候 可以使用这种方式进行构建

```
//File 数据Model
DataModel dataModel = new FileDataModel(new File("D:/data.txt"));
```

其他方式构建数据模型

基于File文件的话 构建起来比较简单 如果是直接通过数据库 Redis之类读取出来的数据进行构建 会些复杂

```
        //假设这是数据库
        ClassPathResource classPathResource = new ClassPathResource("data.txt");
        File file = classPathResource.getFile();
        //假设lines 是从数据库读取出来的每一行数据
        List<String> lines = Files.readLines(file, Charset.defaultCharset());

        FastByIDMap<LinkedList<Preference>> linkedListFastByIDMap = new FastByIDMap<>();
        int size = lines.size();
        for (int i = 0; i < size; i++) {
            String lien = lines.get(i);
            if (lien.length() == 0) continue;
            //按逗号分隔
            String[] split = lien.split(",");
            //用户ID
            Long uId = Long.valueOf(split[0]);
            //Item Id
            Long itemId = Long.valueOf(split[1]);
            //喜好程度
            Float value = Float.valueOf(split[2]);
            //创建该用户下的item list
            LinkedList<Preference> preferences = linkedListFastByIDMap.get(uId);
            if (preferences == null) preferences = new LinkedList<>();
            //构建偏好
            GenericPreference genericPreference = new GenericPreference(uId, itemId, value);
            preferences.add(genericPreference);
            //重新写入map集合
            linkedListFastByIDMap.put(uId, preferences);
        }
        //遍历map集合
        Set<Map.Entry<Long, LinkedList<Preference>>> entries = linkedListFastByIDMap.entry
et();
        // 构建偏好集合
        FastByIDMap<PreferenceArray> arrayFastByIDMap = new FastByIDMap<>();
        for (Map.Entry<Long, LinkedList<Preference>> entry : entries) {
            Long key = entry.getKey();
            LinkedList<Preference> value = entry.getValue();
            arrayFastByIDMap.put(key, new GenericUserPreferenceArray(value));
        }
        //最后 构建数据模型
        DataModel dataModel = new GenericDataModel(arrayFastByIDMap);
```

构建推荐接口

```
    //用户相似度模型
    UserSimilarity userSimilarity = new PearsonCorrelationSimilarity(dataModel);
    //构建 近邻对象  threshold 是相似阈值 这个数值越高  推荐精准越高  但是推荐的数据也越少 最为 你给用户设置的喜好值最高值 也就是preference的最高值
    float threshold = 0f;
    UserNeighborhood neighborhood = new ThresholdUserNeighborhood(threshold, userSimiarity, dataModel);
    //构建推荐器
    UserBasedRecommender recommender = new GenericUserBasedRecommender(dataModel neighborhood, userSimilarity);
    //开始推荐 一参数为用户ID 二参数为 要推荐Item数量
    //我随便在用户列表里拿一个ID试试
    List<RecommendedItem> recommend = recommender.recommend(479261239365599232, 10);
    for (RecommendedItem recommendedItem : recommend) {
        System.out.println(recommendedItem);
    }
```

最后输出结果为

可以看到的是 它是按照preference的降序进行排序的 第一个是preference 喜好最高的为:4.0

到此 基于用户协同过滤推荐算法 就结束了 这段时间忙完了 再研究研究其他的算法

```
itemId:479261239340434028 preference:4.0
itemId:479261239332044812 preference:3.9843752
itemId:479261239340433731 preference:3.897294
itemId:479261239340433625 preference:3.8967257
itemId:479261239332045030 preference:3.8551462
itemId:479261239340433657 preference:3.8387406
itemId:479261239340433426 preference:3.7195482
itemId:479261239340433980 preference:3.7163606
itemId:479261239340433897 preference:3.7102416
itemId:479261239340433914 preference:3.7042284
```