ItemCF 推荐系统的 MapReduce 实现

劳马东 16337113
计算机科学与技术（超算方向）

2018 年 7 月 22 日

1 ItemCF 的基本思想

1. 计算物与物之前的相似度
2. 根据用户的行为历史，给出和历史列表中的物品相似度最高的推荐

通俗的来讲就是：

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。

2 算法实现步骤

2.1 数据集

数据格式：$User\_id, Item\_id, preference$ 数据集字段：

- $User\_id$: 用户 ID
- $Item\_id$: 物品 ID
- $preference$: 用户对该物品的评分

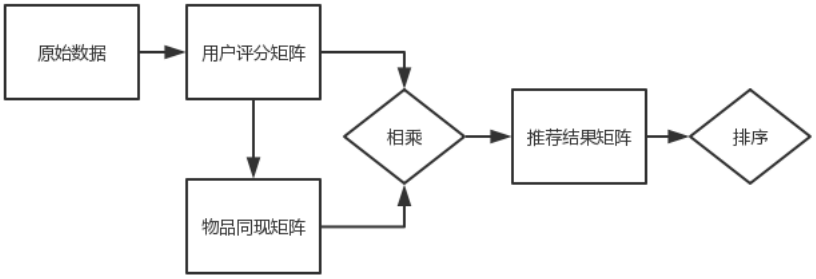

图 1: ItemCF 实现步骤

2.2 计算用户评分矩阵 P

假设有 n 个用户和 m 个物品，则

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1m} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2m} \\ p_{31} & p_{32} & p_{33} & \cdots & p_{3m} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nm} \end{pmatrix} \tag{1}$$

其中 p_{ij}（$i \in [1,n], j \in [1,m]$）代表用户 i 对物品 j 的评分。

从行的方向看，S 的每一个行在 MapReduce 程序中可以用一行简单的字符串来表示：

$$User\_id_i \qquad Item\_id_1 : p_{i1}, Item\_id_2 : p_{i2}, ..., Item\_id_j : p_{ij}, ... \tag{2}$$

因此，该步骤可以简单归纳如下，代码如图 2、3：

1. 在 map 阶段，分割每行内容，输出的 key 为 $User\_id_i$，value 为 $Item\_id_j : p_{ij}$

2. 在 reduce 阶段，将 User_id 相同的所有评分记录进行汇总，输出的 key 仍然为 $User\_id_i$，value 形如：$Item\_id_1 : p_{i1}, Item\_id_2 : p_{i2}, ..., Item\_id_j : p_{ij}, ...$

```java
public class UserScoreMapper extends Mapper<LongWritable, Text, Text, Text>{
    Text k = new Text();
    Text v = new Text();
    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String line = value.toString();
        String[] str = line.split(",");
        k.set(str[0]);
        v.set(str[1]+":"+str[2]);
        context.write(k, v);
    }
}
```

图 2: 用户评分矩阵-map

```java
public class UserScoreReducer extends Reducer<Text, Text, Text, Text>{

    Text v = new Text();
    @Override
    public void reduce(Text key, Iterable<Text> value, Context context)
            throws IOException, InterruptedException {

        String str = new String();
        for(Text v : value)
        {
            str += "," + v.toString();
        }
        v.set(str.replaceFirst(",", ""));
        context.write(key, v);
    }
}
```

图 3: 用户评分矩阵-reduce

2.3 计算物品同现矩阵 T

T 是由上一步得到的结果计算而来的，具体而言就是将文件中格式为“$User\_id_i$ $Item\_id_1 : p_{i1}, Item\_id_2 : p_{i2}, ..., Item\_id_j : p_{ij}, ...$”的行转化成格式为“$Item\_id_k : Item\_id_l$ 次数”的行。

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} & ... & t_{1m} \\ t_{21} & t_{22} & t_{23} & ... & t_{2m} \\ t_{31} & t_{32} & t_{33} & ... & t_{3m} \\ ... & ... & ... & ... & ... \\ t_{m1} & t_{m2} & t_{m3} & ... & t_{mm} \end{pmatrix} \tag{3}$$

t_{ij} $(i \in [1, m], j \in [1, m])$ 代表物品 i 和物品 j 同时被用户使用过的次

数和，即：

$$t_{ij} = \sum_{k=1}^{n} Item\_id_i \in a_k \wedge Item\_id_i \in a_k \tag{4}$$

其中 a_k 表示用户 k 使用过的物品的集合。这个矩阵的意义就是各个物品之间的相似度，为什么可以这么说？如果两个物品经常同时被很多用户喜欢，那么可以说这两个物品是相似的，同时被越多的用户喜欢，T 中相应的值就越大，这两个物品的相似度就越高。

因此，该步骤可以简单归纳如下：

1. 在 map 阶段，枚举每行内容中的 Item_id，输出的 key 为 $Item\_id_i$: $Item\_id_j$，value 为 1，代码如图 4

2. 在 reduce 阶段所做的就是根据 key 对 value 进行累加输出

map 阶段两两枚举某个用户使用过的物品，生成 $(item_i, item_j, 1)$ 项。

```
public class ItemOccurenceMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
    final static IntWritable one = new IntWritable(1);
    Text k = new Text();

    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        String[] str = value.toString().split("\t");

        String[] tmps = str[1].split(",");
        for(int i = 0; i < tmps.length; i++)
        {
            String item_i = tmps[i].split(":")[0];
            for(int j = 0; j < tmps.length; j++)
            {
                String item_j = tmps[j].split(":")[0];
                k.set(item_i+":"+item_j);
                context.write(k, one);
            }
        }

    }
}
```

图 4: 物品同现矩阵-map

2.4 计算推荐结果矩阵 R

$$R = P \times T \tag{5}$$

R_{ij} 表示用户 i 对物品 j 的喜好度（推荐结果）。为什么两个矩阵相乘可以得到推荐结果？因为其他用户对物品 1 的评价 * 物品 1 与物品 2 的相似度，可以大致反映出用户对物品 2 的喜好度。

该步骤可以简单归纳如下，代码如图 5、6：

1. 在 map 阶段，输入的 value 为 S 中的一行，将其与 T 中的每一列相乘得到 R 中的一行，并筛选出未被其使用过的物品及对应的喜好度

2. 在 reduce 阶段，输出 map 阶段筛选的结果，key 为 $User\_id_i$，value 为 $Item\_id_j : R_{ij}$

```
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
    String[] str = value.toString().split("\t");
    String userId = str[0];

    // matrix production
    for(Map.Entry<String, Map<String, Integer>> row : t_matrix.entrySet())
    {
        String item = row.getKey();
        // filter items that have been used.
        if(!value.toString().contains(item))
        {
            double weight = 0.0;
            Map<String, Integer> t_i = row.getValue();
            String[] tmps = str[1].split(",");
            for(int i = 0; i < tmps.length; i++)
            {
                try {
                    String[] item_score = tmps[i].split(":");
                    double score = Double.parseDouble(item_score[1]);
                    weight += score * t_i.get(item_score[0]);
                }
                catch (Exception e) {

                }
            }
            k.set(userId + ":" + item);
            v.set(weight);
            context.write(k, v);
        }
    }

}
```

图 5: 推荐结果矩阵-map

```java
@Override
public void reduce(Text key, Iterable<DoubleWritable> values,
        Context context) throws IOException, InterruptedException {
    double totalScore = 0.0;
    for(DoubleWritable d : values)
    {
        totalScore += d.get();
    }

    String str[] = key.toString().split(":");
    if (!r_matrix.containsKey(str[0])) {
        r_matrix.put(str[0], new TreeMap<Double, String>(new Comparator(){
                public int compare(Object o1, Object o2)
                {
                    return ((Comparable)o2).compareTo((Double)o1);
                }
            }
        ));
    }
    r_matrix.get(str[0]).put(totalScore, str[1]);
}
```

图 6: reduce

2.5 对推荐结果按推荐分值从高到低排序

排序是为了方便挑选出用户最有可能喜欢的一些物品，因为物品的总数很多，不可能把所有物品都推荐给用户。推荐物品个数可以是一个设定的值，由于上一步的 map 阶段已经筛选出了用户所有未使用过的物品，可以考虑在 reduce 将结果存入一个从大到小排序的 TreeMap 中。代码如图 6、7:

```java
@Override
public void cleanup(Context context) throws IOException, InterruptedException {
    super.cleanup(context);
    final int cnt = 10;
    for(Map.Entry<String, Map<Double, String>> entry : r_matrix.entrySet()) {
        k.set(entry.getKey());
        int tmp = cnt;
        for (Map.Entry<Double, String> e : entry.getValue().entrySet()) {
            v.set(e.getValue() + ":" + e.getKey());
            context.write(k, v);
            tmp--;
            if (tmp <= 0)   break;
        }
    }
}
```

图 7: 选择 top10 推荐结果

3 测试结果

测试数据集：ratings.csv(下载自 https://grouplens.org/datasets/movielens/)

测试环境：Ubuntu18.04 64 位 hadoop 2.9.1

结果：output 文件夹中, user-score-matrix.txt 为用户评分矩阵 P, item-occurrence-matrix.txt 为物品共现矩阵 T, result.txt 为推荐结果。

参考文献

[1] 87hbteo, *Ubuntu16.04 下 hadoop 的安装与配置（伪分布式环境）*. https://www.cnblogs.com/87hbteo/p/7606012.html

[2] liushahe2012, *Hadoop 案例之基于物品的协同过滤算法 ItemCF*. https://blog.csdn.net/liushahe2012/article/details/54122080

[3] FreeBird, *基于物品的协同过滤 ItemCF 的 mapreduce 实现*. https://www.cnblogs.com/anny-1980/articles/3519555.html