**用户相似度计算**

本文档主要是探索用户相似度的计算，并基于商超零售实际数据进行实践，得出用户之间的相似度，为给用户推荐物品做准备。

## 1 计算方法

用户特征矩阵加用户评分矩阵分别计算，再按一定的权重进行融合得到最终用户相似度。

**1.1 评分矩阵**

通过用户历史购买行为构建评分矩阵计算用户间的Jaccard系数来计算相似度。

**1.2特征矩阵**

利用特征工程计算的用户特征来构建特征矩阵，结构为如（user\_id, feature, value）形式。共有88996个用户，时间、购物篮特征共42维特征， 由于特征对数值敏感，利用欧氏距离（Euclidean）计算相似度，spark实现时基于随机分桶+lsh

## 2 遇到的问题

以上的计算方法在用户和物品维度不大时可以正常运行，但当用户、物品维度提升时计算量急剧上升，用户大约超过一万时，单机服务器（8核60g）已无法计算。

## 3 计算优化

**3.1 针对用户两两相似计算优化**

1）按物品分组倒排，在每组内计算用户两两相似，最后再合并

2）过滤掉对计算用户相似度贡献较小的数据，如比较热门的物品、只有一个用户购买的物品。

经过优化可以减少一定幅度的计算量，但对于单机计算全量用户来说，效果不够。

**3.2 spark分布式矩阵RowMatrix**

计算速度有提升，但在单机环境下，效果不明显，内存问题与计算速度问题依然存在。

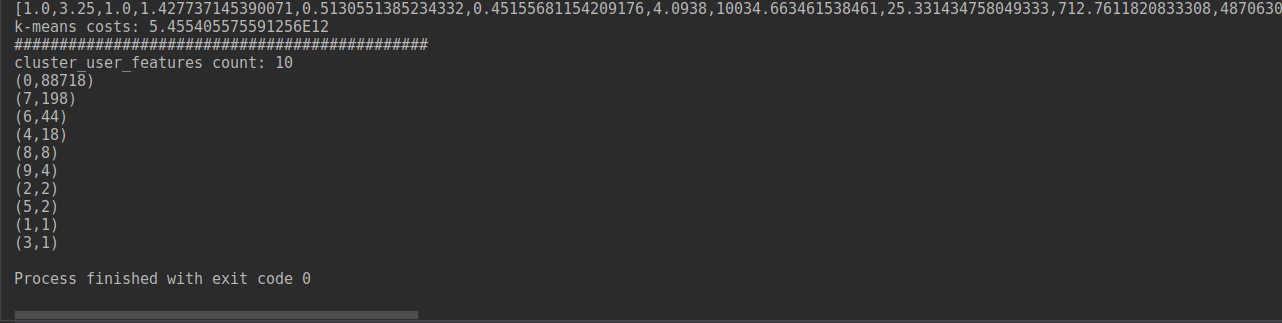
**3.3 minhash（或随机分桶）+ lsh**

基于sparksql ml库实现，优化后单机可以运行，9万用户，3万物品，消耗20g内存一小时内可以运行完成，但会损失一定精度，通过调整相似度计算的阈值可以在精度和计算量之间进行权衡，用户覆盖度通过调整阈值可以超过90%，这种方式比较适合海量数据计算相似度。

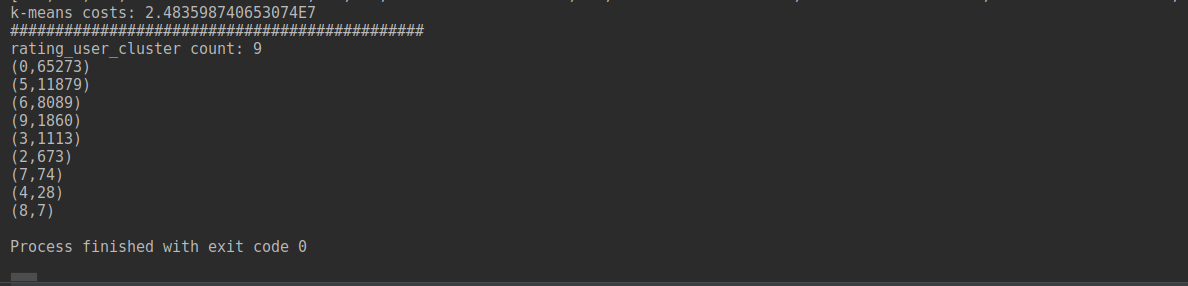
**3.4 聚类+分布式矩阵**

对用户进行kmeans聚类，在每一类中计算用户相似度，问题在于k值的确定，k值太小计算量减少的不明显，太大分的类太多。“手肘”法、bandit算法确定。

用户特征向量（稠密）聚类结果：



用户评分向量（稀疏）聚类效果：



聚类效果不明显，各类用户数不平均，倾斜严重，尝试调整k值优化效果和使用改进的kmeans算法。

**3.5 近似最近邻**

寻找用户的近似最近邻用户，再计算用户的相似度。

两种减少搜索时间的方法。基于哈希的近似最近邻搜索的方法通过设计和优化哈希函数，减少计算的次数，从而缩短搜索时间。基于量化的近似最近邻搜索方法则通过聚类把向量集聚成若干类，每类里面的向量用对应的类中心来近似。

## 4 基于商超零售数据的实现

评分矩阵采取minhash+ lsh计算用户相似度，用户特征矩阵用随机分桶+ lsh计算，两者再分别以0.5的权重进行融合，得到最终的用户相似度，再结合购买历史进行物品推荐。