# 2. item-based CF的Python实现

Lance LanceDaily 2016-07-31

本篇是1. user/item-based协同过滤中提到的item-based协同过滤算法的具体实现说明。

代码用Python实现，使用的推荐系统引擎为implicit。https://github.com/benfred/implicit

使用的数据集为某店2016年6月1日至2016年6月27日的数据，共222481条。特征包括：购买日期、商品ID、商品名称、订单ID、用户ID等。取6月1日至6月20日数据为训练集，6月21日至6月27日数据为测试集。

首先使用BM25对数据进行预处理。

```
def bm25_weight(amount, K1=10):
    return amount * (K1 + 1.0) / (K1 + amount)
```

原数据是一段时间内某一个用户对于某件商品的购买量，通过BM25处理后，amount趋于无穷时，结果将趋近于K1。可以防止某一特征值过大对最终结果产生影响。

在本测试集中，K1=10附近时可得到较优的结果。

得到处理后的矩阵后，将其进行分解。

矩阵分解时推荐系统中常用的处理方法。数据集是一个用户*物品的矩阵，值为经过处理后的购买数量值。通过SVD等处理方法将此矩阵分解为两个向量user_factor和item_factor，长度分别为用户数量和物品数量。

在本次使用的推荐系统引擎implicit中，矩阵分解使用交替式最小二乘法(Alternating Least square)实现：

$$C = \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum \|y_i\|^2)$$



目标函数如上，Pui为原矩阵，Xu和Yi为分解后的向量。常见的目标函数多为凸函数，但是这个函数中有X和Y两个向量，故先将其中一个变量固定，求另一个变量的最优值，再将优化后的变量固定，求另一个的最优值，如此交替求解，最终可以逼近最优值。

得到user_factor和item_factor后即可根据其值寻找和每个物品最相近的物品，最终我们得到一个dict，key为每个物品商品的id，value为和这个商品最相近的k个商品的id。但是我们实际场景中面对的是向每一个用户推荐。本次使用的方法是，统计用户购买过的所有物品，将对应每一个物品相似的商品进行排序，去重，得出相似度最高的TopK个物品。例如某用户共购买了n个物品，对于每个物品有k个与其相近的物品，相似度为SIM[i, j]。SIM中共有n*k个值，将这n*k个值从大到小排序，得出最相近的Topk个物品ID。但是这Topk个物品中可能存在重复，例如用户购买了物品A和物品B，而物品C同时与物品A和物品B相关，则在最终的推荐中，物品C只出现一次。

另外，和一个物品最相近的物品是它本身，所以上述算法在推荐中可能不可避免的出现推荐之前买过的商品的情况。在实现中也尝试将其相似度减小，乘一个参数p处理。但是在实际测试中不甚理想，当p取0.999时，推荐结果的准确率和召回率相较之前即有下降，减小p值后下降更明显，p取0.99时准确率和召回率均

低于原结果的1/2。尝试增加p的值，准确率和召回率均有极小幅度的提升，在p=1.5附近时达到顶峰，再增大p的值时则有下降。

最终结果：

第一个参数为Topk，第二个参数为p

测试样例略