【RS】协同过滤-item_based

原创 机智的叉烧 CS的陋室 2019-03-02

Way Back Home

叁/Conor Maynard - Way Back Home (feat. Conor Maynard) (Sam Feldt Edit)

【RS】

本栏目是结合我最近上的七月在线的课、自己自学、以及一些个人的经验推出的专栏，从推荐系统的基础到一些比较好的case，我都会总结发布，当然，按照我往期的风格，更加倾向于去讨论一些网上其实讲得不够的东西，非常推荐大家能多看看并且讨论，欢迎大家给出宝贵意见，觉得不错请点击推文最后的好看，感谢各位的支持。

往期回顾：

- 【NLP.TM】GloVe模型及其Python实现
- 【陋室推荐】| 2018-2-15
- 技术向：推荐学习推荐系统（深度思考，不是广告）
- 【RS】推荐系统的评估
- 【RS】协同过滤-user_based

在上期对user-based协同过滤进行详细讲解后，相信大家都对协同过滤有更加深入的了解，在这次，我们将反过来，从item角度看协同过滤是怎么进行推荐的。

所谓的item-based，就是根据用户所喜好的内容，核心点在内容，去查找和该内容相似的内容，来推荐给用户，知道这点后，我们就可以开始动手了。

数据说明

数据和上一期的类似，是movielens，此处就不赘述啦，代码还是这块。

```
import pandas as pd

MOVIE_PATH = "../../data/ml-20m/movies.csv"
```

```
RATING_PATH = "../../data/ml-20m/ratings.csv"
MOVIE_RATING_PATH = "../../data/movie_rating_20190219_1.csv"

# ----------------------------第一手数据处理--------------------------

# 读取数据
movies = pd.read_csv(MOVIE_PATH)
rating = pd.read_csv(RATING_PATH)

# 数据合并
data = pd.merge(movies, rating, on="movieId")

# 信息组合
data[['userId', 'rating', 'movieId', 'title']].sort_values('userId').to_csv(
    '../../data/movie_rating_20190219_1.csv', index=False)

# ----------------------------第一手数据处理--------------------------
```

数据导入

为了后续更好进行数据处理，即使是导入也不能马虎，在user-based中采用的dic模式是“用户-item-打分”，此处则使用“item-用户-打分”，对数组而言，三者的地位是相同的，但是对于dic，则并非如此，主要原因是从key找value容易，但是从value找key不容易，这个是dic的特点。

```
# json格式化-user-movie-rating-加载版本
data = {}  # DIC用item-用户-打分的形式
with open(MOVIE_RATING_PATH, 'r', encoding='UTF-8') as f:
    idx = 0
    for line in f:
        if idx == 0:
            idx = 1
            continue
        ll = line.strip().split(",")
        if ll[3] not in data:
            data[ll[3]] = {}
        data[ll[3]][ll[0]] = float(ll[1])
```

相似

其实说到相似，在此处就和user-based其实基本无异，抽象出来其实就是相同的，在user-based中衡量用户相似，用的是用户喜欢商品的相似度，那么在item-based，就是喜欢该item的用户相似，来看一下欧式距离和余弦距离的计算方法，公式还是那个公式，只不过是放进去的对象不同了，这就是数学抽象的魅力，当然通过一些模板化也是可以实现两个相似方法的结合。

```
def eu_distance(item1, item2):
    # item间距离：欧氏距离
    distance = 0
    cal = 0
```

```
    for item1_key in item1.keys():
        if item1_key in item2.keys():
            distance = distance + pow(item1[item1_key] - item2[item1_key], 2)
            cal = cal + 1
    return (distance ** 0.5) / (cal + 0.001)

def cos_distance(item1, item2):
    # item间距离：余弦距离
    distance = 0
    item1_norm = 0
    item2_norm = 0
    cal = 0
    for item1_key in item1.keys():
        if item1_key in item2.keys():
            distance = distance + item1[item1_key] * item2[item1_key]
            item1_norm = item1_norm + item1[item1_key] * item1[item1_key]
            item2_norm = item2_norm + item2[item1_key] * item2[item1_key]
            cal = cal + 1
    res = distance / ((item1_norm ** 0.5) * (item2_norm ** 0.5) + 0.001)
    return res
```

最相似

要进行推荐，就是找和该商品最相似的商品了，其实在此处和user-based基本就相同了，主要原因就是我们在数据载入的时候数据格式使用的是“item-用户-打分”，此时用户和商品某种程度就是可替换的。

```
def top_similar(data, item, num=10):
    # 最相似的N个item
    res = []
    for itemid in data.keys():
        if itemid == item:
            continue
        sim = eu_distance(data[item], data[itemid])
        # sim = cos_distance(data[item], data[itemid])
        res.append((itemid, sim))
    res.sort(key=lambda val: val[1])
    return res[:num]
```

推断

下面就是推断了，这里的推断有些许差别。

- 和user-based不同，用户我们是一开始就能拿到，就一个，但是对于用户喜欢的电影而言，就有很多了，所以此处要先取若干用户喜欢的电影。（getMovie）
- 在对相似电影的排序中，采用的是用户的打分情况，即所有看过该电影的平均分

```
def recommend(data, user, item_num_plus=5, item_num_sim=5, rec_num=10):
    # 进行推荐的主函数
```

```
    # 获得用户看过的所有好item
    movie_top = getMovie(data, user, num=item_num_plus)
    # 找出好item相似的item
    movie_close = []
    for movie_item in movie_top:
        movie_get = top_similar(data, movie_item[0], num=item_num_sim)
        for mov_temp in movie_get:
            if mov_temp[0] not in movie_close:
                movie_close.append(mov_temp[0])
    # 相似item的平均分计算
    movie_score = []
    for mov in movie_close:
        movie_score.append((mov, cal_avg(data, mov)))
    # item平均分打分排序
    movie_score.sort(key=lambda val: val[1], reverse=True)
    # 推荐结果整理
    res = []
    USER = getMovie(data, user, num=-1)
    for item in movie_score:
        if item[0] in USER:
            # 跳过用户已经看过的电影
            continue
        res.append(movie_score)
    return res[:rec_num]
```

至此所有的代码就已经开发完成。

后记

在本文中一直在渲染，两者非常相似，然而需要强调的是两者的相似只是从理论和公式角度，但是在实际的应用中，两者的效果千差地别，user-based由于是用户的相似，而更偏向于用户的喜好，因此可能会为用户带来一些新的惊喜，带来更多的多样性，但是对于item-based，则相对稳定，但是对于item-based，是从商品角度出发，因此会为长尾商品带来更多的收益，即使是不太出名的商品也能够得到推荐，简而言之就是各有优缺点，因此大家要注意。

下一期是CF连载的最后一期，主要讨论两种类型的优缺点，以及要解决一些现实应用中会存在的问题，例如哈利波特效应等，还是那句话，我的风格还是旨在和大家去聊其他公众号不会有的一些个人想法和经验，让大家对一个问题有更加深入的了解，而不仅限于公式和代码，毕竟公式只是表达方式和思维抽象，代码只是实现的方法，思想和对额问题的理解才是让大家更加深入的关键。

参考文献

[1] 项亮，《推荐系统实践》

[2] 黄昕，赵伟，王本友等，《推荐系统与深度学习》（这本书强烈推荐）