【RS】协同过滤-user_based

原创 机智的叉烧 CS的陋室 2019-02-22

Bump

Eliminate/Hydraulix - Bump

【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

上期谈到了协同过滤的基本概念，这次来讲其中一个重要的分支，基于用户的协同过滤，其核心思想就是，**找和目标用户相似的用户，根据相似用户的喜好来推断目标用户的喜好，从而为目标用户提供更可能是目标用户喜欢的内容。**

说着非常简单，但是做起来其实会有很多的细节，下面我直接上代码，边讲便给大家演示。

数据说明

先来说说数据，用的是比较经典的**movielen数据**，网上直接有下载，这里数据做了预处理，从不同的表中抽取相应的数据后组合成我们需要的特征表。个人认为pandas用起来就和写SQL一样，构建数据表然后取数，还能连表，非常方便（真香）。

```
# data_processing 数据预处理
# # 整合数据，构建基础数据集
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--------------------------
```

这块处理算是基操了，这里写略点哈。

数据导入

上面是一个单独的文件进行处理，对于数据集的操作，毕竟用一次其实就够了，因此有关协同过滤我就单独开一个文件来写。

```
MOVIE_RATING_PATH = "../../data/movie_rating_20190219_1.csv"
# json格式化-user-movie-rating-加载版本
data = {} # DIC用用户-item-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[0] not in data:
            data[ll[0]] = {}
        data[ll[0]][ll[3]] = float(ll[1])
```

数据导入一块，我还是比较喜欢 `withopen + line` 这样的方式，主要原因是这样能在加载数据的同时对每条数据进行处理，这样不同进行两次遍历，而在一次遍历就完成相应的工作，当然由于把这两个个事情耦合在一块也有一些缺点，看个人喜好吧。

上回说到协同过滤分两步走，一个是相似，一个是推断，这块先讲相似。

相似

基于用户的协同过滤，核心就是找相似用户然后看相似用户的喜好，因此，此处相似就是**找相似用户**，说到相似，那就是要算相似度，这里，用与之相反的距离来衡量，距离越小，越相似。

而在这里，判断两个用户的相似度，实质上就是要看两个用户看过的相同电影的打分的差别，如果越多相同的电影打分也相同，说明两个用户比较接近，否则就不相似。

欧氏距离相似

欧氏距离和高中甚至初中谈及的两点间距离相似，而与之不同的就是要取一个均值，这里的均值主要是为了避免不同的两个用户对之间的共同电影数不同，直接上代码。

```
def eu_distance(user1, user2):
    # 用户间距离：欧氏距离
    # user-movie-rating
    distance = 0
    cal = 0
    for user1_key in user1.keys():
        if user1_key in user2.keys():
            distance = distance + pow(user1[user1_key] - user2[user1_key], 2)
            cal = cal + 1
    return (distance ** 0.5) / (cal + 0.001)
```

最后这个 `cal+0.001` 我主要是为了避免cal为0的情况，其他地方其实都比较好理解了。

余弦距离

余弦距离在一些地方和皮尔逊相关系数很类似，他相比欧氏距离的优点在于能忽略量纲因素，而考虑趋势的相似性。在以打分作为喜好程度的衡量的movielen分类项目中，有些人可能对喜欢的电影要求比较松，比较好就能有4分，然后很多时候能达到5分，而有些人则对分数比较严格，即使很喜欢可能也是4分，几乎没有5分，其实说明用户给4分其实也是很好的，欧氏距离会识别其有比较明显的距离，但是余弦距离不会。

另一方面，余弦距离是归一化距离，-1到1的闭区间，此时是具有比较明显的可比性和方向性的，至少知道的是负的说明两个人的意见基本相左，有比较明显的识别性。

不吹了，上代码吧，至少在这里，我似乎更加喜欢余弦距离。

```
def cos_distance(user1, user2):
    # 用户间距离：余弦距离
    distance = 0
    user1_norm = 0
    user2_norm = 0
```

```
    cal = 0
    for user1_key in user1.keys():
        if user1_key in user2.keys():
            distance = distance + user1[user1_key] * user2[user1_key]
            user1_norm = user1_norm + user1[user1_key] * user1[user1_key]
            user2_norm = user2_norm + user2[user1_key] * user2[user1_key]
            cal = cal + 1
    res = distance / ((user1_norm ** 0.5) * (user2_norm ** 0.5) + 0.001)
    return res
```

上方是点积，下面是两个模，具体公式看上一期哈。

最相似

这里需要选择和该用户最为接近的几个用户，这块代码其实也不复杂，这里也用了一些对于数组-数组或者是数组-tuple的排序的技巧，不用自己造轮子。

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

推断

然后就是根据这些用户的喜好来为目标用户推荐了。想简单粗暴，所以把前面的步骤给耦合到这里了，在现实应用建议还是分开的，另外的话，这块注意看注释哈，不拆代码，为了把问题说清楚我注释写的更详细了。

> 代码千万条，注释第一条，代码不规范，被追十条街。

```
def recommend(data, user, user_num=10, rec_num=10):
    # 进行推荐的主函数
    # 获得最接近的几个用户（耦合代码，可自行拿出来）
    user_close = top_similar(data, user, num=user_num)
    # 获取这些用户看过的所有item
    movie = {}
    for item in user_close:
        for user_close_movie in data[item[0]]:
```

```
            if user_close_movie not in movie:
                movie[user_close_movie] = []
            movie[user_close_movie].append(
                (item[1], data[item[0]][user_close_movie]))
    # 给所有相关item进行打分
    movie_cal = []
    for item in movie:
        score = 0
        weight = 0
        # 这块为了看更多相似用户的综合意见，所以求了个均值。
        for cal in movie[item]:
            score = score + cal[1]
            weight = weight + 1
        movie_cal.append((item, score / weight))
    movie_cal.sort(key=lambda val: val[1], reverse=True)
    # 整理推荐结果
    res = []
    USER = data[user]
    for item in movie_cal:
        # 避免推荐用户已经看过的item
        if item[0] in USER:
            continue
        res.append(item)
        # 推荐数量够了即可
        if len(res) >= rec_num:
            break
    if len(res) < rec_num:
        print("合适item不足目标数量")
        return res
    return res[:rec_num]
```

这样，最终结果就已经出来了，自己可以尝试去跑跑哈。

后记

这里有很多细节感觉是很多材料都没有说清楚，尤其是在推断模块，例如集中多个用户的意见是不是只有求均值一种，能不能用一个越接近考虑的越多的权重方式来加权等，不知道是不是我没找到，欢迎提出，但至少这是一个可行的方案吧。

另一方面，值得考虑的是，有关距离，这里考虑的是共同电影的打分，但是共同电影的多少其实也和用户的相似度有关，两个人只看过一部电影，都给5分，其实有很大的随机性，有很多细节的业务需要进行改进和处理，再者，热门效应、冷启动等问题在协同过滤里面都没有考虑，而这些其实都是要考虑的，还有很多问题，这些我会挑选重点且已经被广泛认可的在后续的文章中讨论，敬请期待！

参考文献