协同过滤推荐算法

原创 wuli小萌哥 算法研习社 2020-01-19

相信大家对推荐系统已经很熟悉了，它通过分析用户的历史行为，挖掘用户的兴趣爱好，预测并推荐给用户其接下来有可能感兴趣的事物，例如商品推荐、音乐推荐、新闻以及最近很火的短视频推荐等等。

协同过滤推荐 作为最为主流与经典的技术之一，它是基于这样的假设：**用户如果在过去对某些项目产生过兴趣，那么将来他很可能依然对其保持热忱。**协同过滤技术又可依据是否采用了机器学习思想建模进一步划分为基于内存的协同过滤（Memory-based CF）与基于模型的协同过滤技术（Model-based CF）。

本文旨在对经典的协同过滤推荐算法进行总结，并通过 Python 代码实现深入理解其算法原理。

目录:

基于内存的协同过滤推荐

基于内存的协同过滤算法是推荐系统中最基本的算法，也叫做基于邻域的协同过滤，该算法不仅在学术界得到了深入研究，而且在业界得到了广泛应用。基于邻域的算法分为两大类，一类是基于用户的协同过滤算法，另一类是基于物品的协同过滤算法。

为了描述简便，下面的算法讲解都是基于我们常见的 topN 推荐场景，而不是评分预测场景。在 topN 场景中，我们只关注用户对物品是否具有行为，而不在乎分数。

数据集采用 MovieLens 中等大小的数据集，包含 6000 多用户对 4000 多部电影的 100 万条评分(1~5 分)。

数据集下载地址：http://files.grouplens.org/datasets/movielens/

输入数据处理为下面的格式：data => {user_id : { item_id ： score}}

基于用户的协同过滤（User CF）

基于用户的协同过滤推荐，一句话概括就是，给用户 A 推荐**与其兴趣相似的朋友们喜欢而用户 A 还没听说过的**物品。

该算法主要包括两个步骤：
(1) 找到和目标用户兴趣相似的用户集合。
(2) 找到这个集合中用户喜欢的，且目标用户没有听说过的物品推荐给目标用户。

1.计算用户相似度

给定用户 u 和用户 v，令 N(u)表示用户 u 曾经有过反馈的物品集合，令 N(v)为用户 v 曾经有过反馈的物品集合。

通过余弦相似度计算用户 u 和 v 的相似度：

$$w_{uv} = \frac{|N(u) \bigcap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

代码实现：

```
def UserSimilarity(train):
    W = dict()
    for u in train.keys():
```

```
            for v in train.keys():
                if u == v:
                    continue
                W[u][v] = len(train[u] & train[v])
                W[u][v] /= np.sqrt(len(train[u]) * len(train[v]) * 1.0)
        return W
```

如果实际应用中用户数过多，上面计算方式是计算每两个用户的相似度，这样很多没必要的时间都浪费了。因此，可以先建立物品到用户的倒排表 item_users，对于每个物品都保存对该物品产生过行为的用户，然后遍历每个 item 的 users。

```
def UserSimilarity(train):
    # 建立item -> users table
    item_users = dict()
    for u, items in train.items():
        for i in items.keys():
            if i not in item_users:
                item_users[i] = set()
            item_users[i].add(u)
    # 记录每个用户反馈过的物品数
    N = dict()
    # 记录每两个用户共同反馈过的物品数
    C = dict()
    for i, users in item_users.items():
        for u in users:
            N[u] += 1
            for v in users:
                if u == v:
                    continue
                C[u][v] += 1
    # 计算用户相似度权重矩阵
    W = dict()
    for u, related_users in C.items():
        for v, cuv in related_users.items():
            W[u][v] = cuv / np.sqrt(N[u] * N[v])

    return W
```

2.基于相似用户推荐物品

得到用户之间的兴趣相似度后，UserCF 算法会给用户推荐和他兴趣最相似的 K 个用户喜欢过但该用户还没见过的物品。如下的公式度量了 UserCF 算法中用户 u 对物品 i 的感兴趣程度:

$$p(u,i) = \sum_{v \in S(u,K), i \in N(v)} w_{uv} r_{vi}$$

其中，S(u, K)包含和用户 u 兴趣最接近的 K 个用户，N(v)是用户 v 喜欢的物品集合，wuv 是用户 u 和用户 v 的兴趣相似度，rvi 代表用户 v 对物品 i 的兴趣，因为使用的是单一行为的隐反馈数 据，所以所有的 rvi=1。

```
def Recommend(user_id, train, W, K):
    rank_rs = dict()
    interacted_items = train[user_id]
    for v, wuv in sorted(W[user_id].items(), key=lambda x:x[1], reverse=True)[0:K]:
        for i, rvi in train[v].items:
            if i in interacted_items:
                # 过滤掉该用户之前反馈过的物品
                continue
            rank_rs[i] += wuv * rvi
    return rank_rs
```

基于物品的协同过滤（Item CF）

基于物品的协同过滤(item-based collaborative filtering)算法是目前业界应用最多的算法。Item CF 认为，物品 A 和物品 B 具有很大的相似度是因为喜欢物品 A 的用户大都也喜欢物品 B。就好比大家熟悉的“看了又看”功能，图书网站给用户推荐《天龙八部》可以解释为该用户之前喜欢《射雕英雄传》，而这两本书被很多人共同喜欢过。

基于物品的协同过滤算法主要分为两步。
(1) 计算物品之间的相似度。
(2) 根据物品的相似度和该用户的历史行为给用户生成推荐列表。

1.计算物品相似度

基于余弦相似度，可以计算物品 i 和 j 直接的相似度：

$$w_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

其中，N(i)表示喜欢物品 i 的用户集合，上式分母表示同时喜欢物品 i 和 j 的用户数。

```
def ItemSimilarity(train):
    # 记录每个物品被反馈过的用户数
    N = dict()
    # 记录共同喜欢两个物品的用户数
    C = dict()
    for u, items in train.items():
        for i in items:
            N[i] += 1
            for j in items:
                if i == j:
                    continue
                C[i][j] += 1
    # 计算物品i和j的相似度矩阵
    W = dict()
    for i, related_items in C.items():
        for j, cij in related_items.items():
            W[i][j] = cij / np.sqrt(N[i] * N[j])

    return W
```

2.基于相似用户推荐物品

在得到物品之间的相似度后，ItemCF 通过如下公式计算用户 u 对一个物品 j 的兴趣:

$$p_{uj} = \sum_{i \in N(u), j \in S(i,K)} w_{ij} r_{ui}$$

这里 N(u)是用户喜欢的物品的集合，S(i,K)是和物品 i 最相似的 K 个物品的集合，wij 是物品 j 和 i 的相似度，rui 是用户 u 对物品 i 的兴趣。

代码实现：

```
def Recommendation(train, user_id, W, K):
    rank_rs = dict()
    u_items = train[user_id]
    for i, rui in u_items.items():
        for j, wij in sorted(W[i].items(), key=lambda x: x[1], reverse=True)[0:K]:
            if j in u_items:
                continue
            rank_rs[j] += rui * wij
    return rank_rs
```

UserCF 和 Item CF 比较

- UserCF 给用户推荐那些和他有共同兴趣爱好的用户喜欢的物品，而 ItemCF 给用户推荐那些和他之前喜欢的物品类似的物品。
- UserCF 的推荐结果着重于反映和用户兴趣相似的小群体的热点，而 ItemCF 的推荐结果着重于维系用户的历史兴趣。
- UserCF 的推荐注重社会化，反映了用户所在的小型兴趣群体中物品的热门程度，而 ItemCF 的推荐更加个性化，反映了用户自己的兴趣传承。
- 从技术上考虑，UserCF 需要维护一个用户相似度的矩阵，而 ItemCF 需要维护一个物品相似度矩阵。
- 从存储的角度说，如果用户很多，那么维护用户兴趣相似度矩阵需要很大的空间， 同理，如果物品很多，那么维护物品相似度矩阵代价较大。

	UserCF	ItemCF
性能	适用于用户较少的场合，如果用户很多，计算用户相似度矩阵代价很大	适用于物品数明显小于用户数的场合，如果物品很多（网页），计算物品相似度矩阵代价很大
领域	时效性较强，用户个性化兴趣不太明显的领域	长尾物品丰富，用户个性化需求强烈的领域
实时性	用户有新行为，不一定造成推荐结果的立即变化	用户有新行为，一定会导致推荐结果的实时变化
冷启动	在新用户对很少的物品产生行为后，不能立即对他进行个性化推荐，因为用户相似度表是每隔一段时间离线计算的	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
	新物品上线后一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给和对它产生行为的用户兴趣相似的其他用户	但没有办法在不离线更新物品相似度表的情况下将新物品推荐给用户
推荐理由	很难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

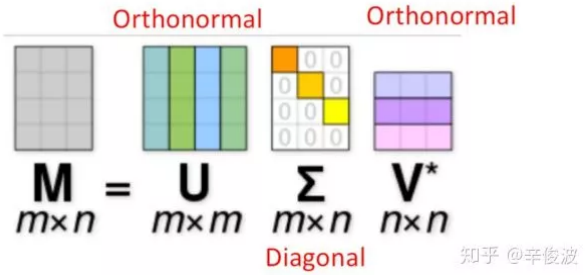
基于模型的协同过滤推荐

在经典的推荐算法中，除了基于邻域的 ItemCF 和 UserCF，提的最多的就是隐语义模型和矩阵分解模型。其实，这两个名词说的是一回事，就是如何通过降维的方法将评分矩阵补全。

若 R[u][i]表示用户 u 对物品 i 的评分，由于用户不会对所有的物品评分，所以真实的 R 矩阵是非常稀疏的，推荐算法要做的就是将这些缺失值补全，预测用户对未反馈过物品的评分，继而对高分物品进行推荐。

经典 SVD

最经典的基于矩阵分解的推荐技术是 SVD（奇异值）分解。



上图左侧 M=m*n 表示用户评分矩阵，m 矩阵的行表示用户数，n 矩阵的列表示 item 数，在大多数推荐系统中 m 和 n 规模都比较大，右侧三个矩阵依次是左奇异矩阵、奇异值矩阵和右奇异矩阵。如果只取前 k 个最大的奇异值组成对角矩阵 $\Sigma_{k\times k}$，并且找到这 k 个奇异值中每个值在 U、V 矩阵中对应的列和行，得到 $U_{m\times k}$、$V^T_{k\times n}$，从而可以得到一个降维后的评分矩阵：

$$M_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n}$$

其中， M (u, i) 就是用户 u 对物品 i 评分的预测值，即完成了填空的任务。

一般来说，SVD 求解可以分为三步：

（1） 对 M 矩阵的 missing data 进行填充，整体均值、用户均值、物品均值或 0

（2） 求解 SVD 问题，得到 U 矩阵和 V 矩阵

（3） 利用 U 和 V 矩阵的低秩 k 维矩阵来估计 M

对于第二步中的 SVD 求解问题，等价于以下的最优化问题：

$$\arg\min_{\mathbf{U},\Sigma,\mathbf{V}}(\mathbf{Y}-\mathbf{U}\Sigma\mathbf{V}^T)^2$$

$$=\arg\min_{\mathbf{U},\Sigma,\mathbf{V}}\sum_{i=1}^{m}\sum_{j=1}^{n}(y_{ij}-(\mathbf{U}\Sigma\mathbf{V}^T)_{ij})^2$$

其中 yij 为用户 i 对物品 j 的真实评分，也就是 label, U 和 V 为模型预估值，求解矩阵 U 和 V 的过程就是最小化用户真实评分矩阵和预测矩阵误差的过程。

这种 SVD 求解方法存在以下问题：

（1） missing data（在数据集占比超过 95%）和 observe data 权重一样

（2） 最小化过程没有正则化（只有最小方差），容易产生过拟合

（3）计算复杂度很高，特别是在稠密的大规模矩阵上更是非常慢。

因此，一般来说针对原始的 SVD 方法会有很多改进方法。

FunkSVD

2006 年 Netflix Prize 开始后，Simon Funk 在博客上公布了一个算法(称为 Funk-SVD)，解决了传统 SVD 的几个缺点。Simon Funk 提出的矩阵分解方法后来被 Netflix Prize 的冠军称为 Latent Factor Model(简称为 LFM)，即隐语义模型。FunkSVD，将矩阵 M 分解为了 2 个低秩的 user，item 矩阵，

$$\hat{y}_{ui} = \mathbf{v}_u^T \mathbf{v}_i$$

User latent vector　Item latent vector

核心思想可以分成两步：

（1）将用户 u 对物品 i 的打分分解成用户的隐向量 vu，以及物品的隐向量 vi

（2）用户 u 和物品 i 的向量点积（inner product）得到的 value，可以用来代表用户 u 对物品 i 的喜好程度，分数越高代表该 item 推荐给用户的概率就越大

同时，FunkSVD 模型引入了 L2 正则来解决过拟合问题

$$L = \sum_u \sum_i w_{ui}(y_{ui} - \hat{y}_{ui})^2 + \lambda(\sum_u ||\mathbf{v}_u||^2 + \sum_i ||\mathbf{v}_i||^2)$$

Prediction error　L2 regularizer

可以采用随机梯度下降法优化求解。

BiasSVD

实际情况下，一个评分系统有些固有属性和用户物品无关，而用户也有些属性和物品无关，物品也有些属性和用户无关。因此，Netflix Prize 中提出了另一种 LFM，其预测公式如下:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \cdot q_i$$

这个预测公式中加入了 3 项 μ 、 bu 、 bi ,这个模型被称为 BiasSVD，也叫做加了偏置项的 LFM。

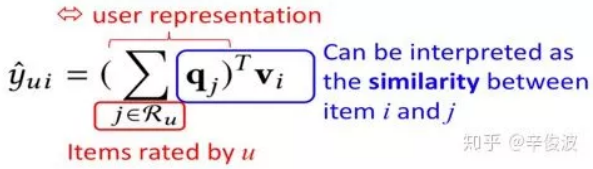
增加的三个参数的含义如下：

- μ：训练集中所有记录的评分的全局平均数。比如有的网站的评分普遍很高，有的普遍很低。
- bu：用户偏置(user bias)项。这一项表示了用户的评分习惯中和物品没有关系的那种因素。比如有些用户就是比较苛刻，对什么东西要求都很高，那么他的评分就会偏低，而有的用户恰恰相反。
- bi 物品偏置(item bias)项。这一项表示了物品接受的评分中和用户没有什么关系的因素。比如有些物品本身质量就很高，因此获得的评分相对都比较高。

要注意，3 个参数中只有 bu 、bi 是要通过机器学习训练出来的，可以通过求导然后用梯度下降法求解这两个参数。

FISM 模型

上述提到的模型方法都只是简单利用了 user- item 的交互信息，对于用户本身的表达是 userid 也就是用户本身。2014 年 KDD 上提出了一种更加能够表达用户信息的方法，Factored Item Similarity Model，简称 FISM，顾名思义，就是将用户喜欢过的 item 作为用户的表达来刻画用户，用数据公式表示如下：



注意到用户表达不再是独立的隐向量，而是用用户喜欢过的所有 item 的累加求和得到作为 user 的表达；而 item 本身的隐向量 vi 是另一套表示，两者最终同样用向量内积表示。

SVD++

FunkSVD 模型和 BiasSVD 都可以看成是 user-based 的 CF 模型，直接将用户 id 映射成隐向量，而 FISM 模型可以看成是 item- based 的 CF 模型，将用户交户过的 item 的集合映射成隐向量。一个是 userid 本身的信息，一个是 user 过去交互过的 item 的信息，如何结合 user-base 和 item-base 这两者本身的优势呢？

SVD++方法正是这两者的结合，其核心数学表达如下：

$$\hat{y}_{ui} = (\mathbf{v}_u + \underbrace{\sum_{j \in \mathcal{R}_u} \mathbf{q}_j}_{\text{User representation in latent space}})^T \mathbf{v}_i$$

其中，每个用户表达分成两个部分，左边 vu 表示用户 id 映射的隐向量（user-based CF 思想），右边是用户交互过的 item 集合的求和（item- based CF 思想）。User 和 item 的相似度还是用向量点击来表达。

完整的 SVD++公式如下，可以看做是前面 BiasSVD 和 FISM 的融合：

$$\hat{r}_{ui} = \mu + b_u + b_i + \left(p_u + \frac{1}{\sqrt{|R(u)|}} \sum_{j \in R(u)} y_j \right)^T \cdot q_i$$

其中，前三项是全局偏置、用户偏置和物品偏置，括号里面是用户 u 的兴趣偏好向量，包括 FunkSVD 学出来的 user 隐向量和 FISM 中学到的基于用户历史反馈物品生成的向量，用户 u 的兴趣偏好被建模为两者相加。R(u)表示用户反馈的所有物品集合，qi 则代表物品 i 的隐向量，整个公式计算的是 u 对 i 的预测评分数。

需要注意的是，上式包含三个参数矩阵，P，Q 和 Y，P 和 Q 分别是 FunkSVD 中的 user 参数矩阵和 item 参数矩阵，而 Y 是 FISM 中 item 参数矩阵，每个隐向量的维度都是一致的，这样才能进行求和和内积计算。

参考：

1，项亮，推荐系统实践

2，辛俊波，推荐系统中的深度匹配模型，https://zhuanlan.zhihu.com/p/101136699

3，推荐系统 SVD 和 SVD++算法，https://www.cnblogs.com/shenxiaolin/p/9494579.html

推荐阅读：

- SVD矩阵分解
- Python面试题系列
- PCA原理
- 机器学习-手撕推导系列