1 推荐系统算法

1.1 Top N推荐模型

1.1.1 UserCF

引入

UserCF算法是最古老的推荐系统算法，于1992年被提出并被运用于邮件过滤系统，1994年被GroupLens用于新闻过滤，直到ItemCF算法被提出之前，UserCF算法都是推荐系统领域最著名的算法。

算法

UserCF算法的基本流程：

1. 计算item-user倒排表
2. 计算user-user共轭表
3. 计算user-user相似度矩阵
4. 根据top n的相似用户服务进行推荐

1 计算item-user倒排表

计算商品-用户倒排表就是统计出每种商品对应的购买过的用户，如下表所示：

用户	商品
用户1	a, b, d
用户2	a, c
用户3	b, e
用户4	c, d, e

2 计算user-user共轭表

计算user-user共轭表就是要建立一张(n_users, n_users)维度的矩阵C，其中C[u, v]就表示用户u和用户v所共同购买过的商品数量。该计算可以通过扫描上面求得的item-user倒排表来获得：

用户/用户	用户1	用户2	用户3	用户4
用户1	3	1	1	1
用户2	1	2	0	1
用户3	1	0	2	1
用户4	1	1	1	3

3 计算user-user相似度

根据上面求得的user-user共轭表，可以对其进行归一化计算得到用户的相似度矩阵W。关于相似度的计算公式，比较常用的有：

- Jaccard相似度：

$$w_{uv} = \frac{|N(u) \bigcap N(v)|}{|N(u) \bigcup N(v)|} \tag{21}$$

- 余弦相似度：

$$w_{uv} = \frac{|N(u) \bigcap N(v)|}{\sqrt{|N(u)||N(v)|}} \tag{1}$$

  上面N(u)和N(v)分别表示用户u和用户v喜欢的商品集合。

- 改进的余弦相似度（IIF）：

$$w_{u,v} = \frac{\sum_{i \in N(u) \bigcap N(v)} \frac{1}{log(1+N(i))}}{\sqrt{|N(u)||N(v)|}} \tag{2}$$

  IIF（Inverse Item Frequence）的分子表示用户u和用户v都喜欢的商品热门度对数的倒数加和。如果某商品热门度越大，即N(i)越大，则该商品对用户相似度的贡献越小。换句话说，两个用户都去购买热门商品不能说明两人兴趣相投，反之两个用户都去购买冷门商品才能说明两人兴趣相投。

这里我们选用余弦相似度来计算user-user相似度矩阵：

用户/用户	用户1	用户2	用户3	用户4
用户1	0	0.41	0.41	0.33
用户2	0.41	0	0	0.41
用户3	0.41	0	0	0.41
用户4	0.33	0.41	0.41	0

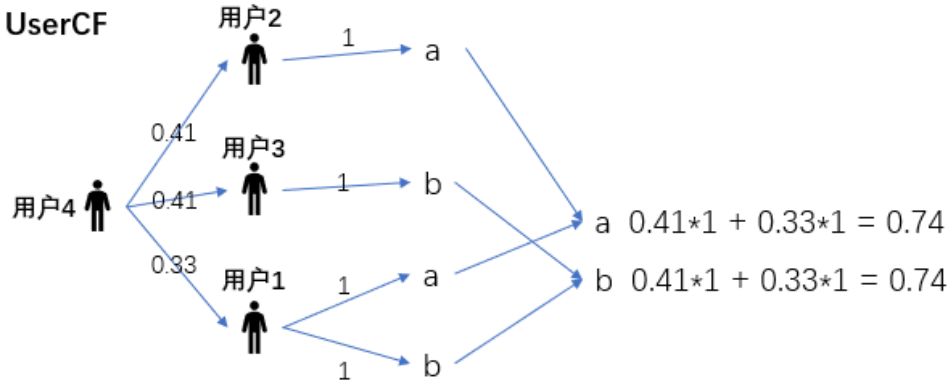
4 商品推荐

计算得到用户的相似度之后，UserCF算法通过如下公式计算用户u对商品i的兴趣：

$$p_{ui} = \sum_{v \in N(i) \bigcap S(u,N)} w_{uv} * r_{vi} \tag{3}$$

其中，N(i)表示购买过商品i的用户集合，S(u, N)是和用户u最为相似的N个用户的集合，w_uv表示用户u和用户v的相似度，r_vi表示用户v对商品i的兴趣评分。下面我们来对上面的例子进行商品推荐。

比如，我们要对用户4做推荐。首先选出与用户4最相似的top3用户：用户2（相似度0.41），用户3（相似度0.41）和 用户1（相似度0.33）；并选出这些相似用户喜欢的而用户4没有喜欢的商品：a, b；再计算出用户4对a，b的喜欢程度并做出推荐：对商品a的喜欢程度为0.41 + 0.331 =0.74，对商品b的喜欢程度为0.41 + 0.331 =0.74。计算关系可参考下图：



基于Movielens的UserCF推荐

1 Movielens数据集

下面我们根据Movielens数据集来使用UserCF算法进行电影推荐，Movieslens数据集可以从官网下载。

根据使用需求，我们可以选择ml-1m或者ml-100k或者ml-latest-small（不同的数据集的数据量不同）。在数据集中我们主要使用的是ratings文件，它一共有4列数据，分别是：

- userId，用户id
- movieId，电影id
- rating，用户评分
- timestamp，时间戳（这里用不到）

```
userId,movieId,rating,timestamp
1,1,4,964982703
1,3,4,964981247
1,6,4,964982224
1,47,5,964983815
1,50,5,964982931
1,70,3,964982400
1,101,5,964980868
1,110,4,964982176
1,151,5,964984041
1,157,5,964984100
1,163,5,964983650
1,216,5,964981208
1,223,3,964980985
1,231,5,964981179
1,235,4,964980908
1,260,5,964981680
```

2 算法流程

我们的涉及的推荐流程主要包含以下几个步骤：

1. 对全量数据集划分training set和test set
2. 对training set中的用户进行推荐：
    - 计算item-user倒排表
    - 计算user-user用户共轭表
    - 计算user-user商品相似度矩阵
    - 对用户进行商品推荐
3. 将training set的推荐结果与test set中的结果进行对比，计算出如下的评估指标：
    - precision
    - recall
    - f1 score
    - coverage
    - popularity

3 代码

参见usercf.py

1.1.2 ItemCF

引入

ItemCF算法最早于1998年被亚马逊公司提出，详细内容可以阅读文献Item-Based Collaborative Filtering Recommendation Algorithms。自此之后ItemCF算法被广泛运用于Netflix, YouTube等其他知名网站。

ItemCF算法的核心思想可以用一句话概括：**给用户推荐和他们之前喜欢的商品相似的商品**，比如你喜欢电影复仇者联盟系列，那么itemcf算法很可能给你推荐其他漫威系列的英雄电影，因为这些电影之间是**相似的。**

那么，如何衡量不同商品之间的相似性呢？—— ItemCF算法主要利用用户的消费行为来计算商品之间的相似度（而不是根据商品的本身属性来计算）。具体是如何计算相似度的？—— 简单来说，就是如果同时喜欢A和B商品的用户数越大，那么A和B的相似度就越大。下面我们来介绍ItemCF算法的具体步骤

算法

ItemCF算法的基本流程为：

1. 计算user-item倒排表
2. 计算item-item共轭表
3. 计算item-item相似度矩阵
4. 根据用户消费过的商品，选择与其最为相似的Top N的商品进行推荐

1 计算user-item倒排表

计算用户-商品的倒排表，也就是统计出每个用户都消费过哪些商品，举例如下表所示：

用户	商品
用户1	a, b, c, d
用户2	a, c, e
用户3	a, d, f
用户4	a, b, d
用户5	b, d, f

2 计算item-item的共轭表

也就是根据用户-商品倒排表计算出两两商品共同购买的用户数，比如根据上表，我们可以得到item-item共轭表为：

	A	B	C	D	E	F
a	4	2	2	3	1	1
b	2	3	1	3	0	1
c	2	1	2	1	1	0
d	3	3	1	3	0	2
e	1	0	1	0	1	0

	A	B	C	D	E	F
f	1	1	0	2	0	2

3 计算item-item相似度矩阵

根据上面求得的item-item共轭表，可以对其进行归一化得到物品的相似度矩阵W。关于相似度的计算公式，比较常用的有：

- 余弦相似度

$$w_{i,j} = \frac{|N(i) \bigcap N(j)|}{\sqrt{|N(i)||N(j)|}} \tag{4}$$

  其中N(i)表示喜欢商品i的用户数，N(j)表示喜欢商品j的用户数，分子表示同时喜欢商品i, j的用户数，分母是为了避免非常热门的商品和任何其他商品都很相似，算是对热门商品的一种惩罚。

- 改进的余弦相似度

$$w_{i,j} = \frac{\sum_{u \in N(i) \bigcap N(j)} \frac{1}{log(1+N(u))}}{\sqrt{|N(i)||N(j)|}} \tag{5}$$

  这种IUF（Inverse User Frequence）相似度的分子表示同时喜欢商品i,j的用户活跃度对数的倒数，如果用户活跃度越大，即N(u)越大，则该部分用户对商品相似度的贡献值越小。这样做的意义就是尽量削弱过于活跃的用户对推荐系统产生的影响。

我们根据简单余弦相似度的公式，计算出相似度矩阵如下：

	A	B	C	D	E	F
a	0	0.58	0.71	0.87	0.5	0.35
b	0.58	0	0.41	1	0	0.41
c	0.71	0.41	0	0.41	0.71	0
d	0.87	1	0.41	0	0	0.82
e	0.5	0	0.71	0	0	0
f	0.35	0.41	0	0.82	0	0

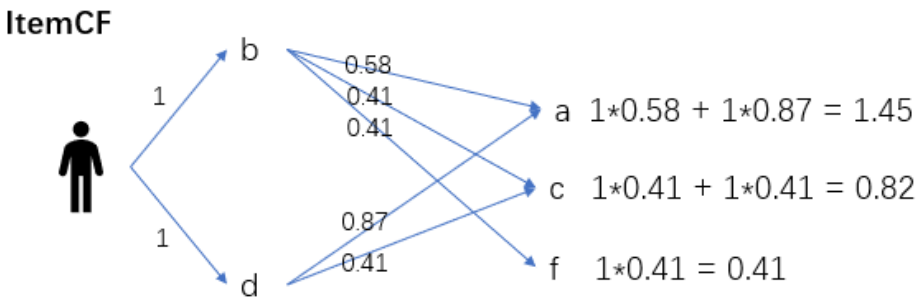
4 商品推荐

计算得到商品的相似度之后，ItemCF算法通过如下公式计算用户u对商品j的兴趣：

$$p_{uj} = \sum_{i \in N(u) \bigcap S(j,N)} w_{ji} * r_{ui} \tag{6}$$

其中，N(u)表示用户u喜欢的商品集合，S(j, N)是和商品j最为相似的N个商品的集合，w_ji表示商品j和商品i的相似度，r_ui表示用户u对商品i的兴趣评分。下面我们来对上面的例子进行商品推荐。

比如，我们要对用户5做推荐，我们使用top3的相似商品来计算。用户5购买过的商品为b, d, f，我们选出除这三个商品之外的，与这三个商品最为相似的三个商品（如果不足三则没有），得到了商品a, c, f。对于商品a，a和b的相似度为0.58，a和d的相似度为0.87，再结合用户5对购买过的商品b, d的评分（对于隐反馈数据集，只要购买过的商品评分就是1，未购买过的商品评分为0），得出用户5对于商品a的综合兴趣为：1 * 0.58 + 1*0.87 = 1.45，用户5对于商品c的综合兴趣为：1 * 0.41 + 1*0.41 = 0.82，用户5对于商品f的综合兴趣为：1 * 0.41 = 0.41。于是最终我们给出的推荐次序就是a, c, f。以上计算过程可以参考下图：



基于Movielens的ItemCF推荐算法

1 Movielens数据集

下面我们根据Movielens数据集来使用ItemCF算法进行电影推荐，Movieslens数据集可以从官网下载。根据使用需求，我们可以选择ml-1m或者ml-100k或者ml-latest-small（不同的数据集的数据量不同）。在数据集中我们主要使用的是ratings文件，它一共有4列数据，分别是：

- userId，用户id
- movieId，电影id
- rating, 用户评分

- timestamp, 时间戳（这里用不到）

```
userId,movieId,rating,timestamp
1,1,4,964982703
1,3,4,964981247
1,6,4,964982224
1,47,5,964983815
1,50,5,964982931
1,70,3,964982400
1,101,5,964980868
1,110,4,964982176
1,151,5,964984041
1,157,5,964984100
1,163,5,964983650
1,216,5,964981208
1,223,3,964980985
1,231,5,964981179
1,235,4,964980908
1,260,5,964981680
```

2 算法流程

我们的涉及的推荐流程主要包含以下几个步骤：

1. 对全量数据集划分training set和test set
2. 对training set中的用户进行推荐：
    - 计算user-item倒排表
    - 计算item-item商品共轭表
    - 计算item-item商品相似度矩阵
    - 对用户进行商品推荐
3. 将training set的推荐结果与test set中的结果进行对比，计算出如下的评估指标：
    - precision
    - recall
    - f1 score
    - coverage
    - popularity

3 代码

参见itemcf.py

1.1.3 UserCF vs ItemCF

表2-11　UserCF和ItemCF优缺点的对比

	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

思想上升一下：

- UserCF算法：**人以群分**，推荐和用户有共同兴趣的用户喜欢的商品（推荐结果**社会化**）
- ItemCF算法：**物以类聚**，推荐和用户喜欢过的商品相类似的商品（推荐结果**个性化**）

1.2 LFM隐语义推荐算法

2.2.1引入

LFM, latent factor model，基于隐含属性的推荐模型。

什么是“隐含属性”？看似深奥，其实很容易理解：我们喜欢一件商品，通常是因为这个商品的某些属性非常吸引我们。比如，我们要选购一款智能手机，如果我们喜欢的是夜拍牛逼，续航持久，充电飞快，屏幕高清的手机，那么我们很可能选中HUAWEI P30（如果不差钱的话）。那么在这里，夜拍效果，续航时间，充电速度，屏幕分辨率等等就是智能手机的“隐含属性”。

我们可以把这个喜欢的过程进一步拆解，并用数学方法分析：

1. 对于一款智能手机，我们本身对它的隐含属性就有一定的**个人偏好**。我们可以用一个长度为5的向量来表示对夜拍效果，续航时间，充电速度，屏幕分辨率，外观颜值这4种属性的偏好（其中每种偏好的分值为0~1，分值越高表示我们对属性的偏好程度越高）：

$$p = (0.9, 1.0, 0.7, 0.8, 0.5) \tag{7}$$

2. 某款智能手机本身在它的隐含属性上就有一定的**属性分值**。我们同样可以用一个长度为5的向量来表示某款手机在夜拍效果，续航时间，充电速度，屏幕分辨率，外观颜值这5种属性上的分值（其中每种属性的分值为0~1，分值越高表示属性越牛逼）：

$$q = (1.0, 0.9, 1.0, 0.8, 0.7) \tag{8}$$

2.2.2 数学模型

基本思想

说回LFM推荐系统，如果我们有m个用户，n个商品，假如用户会对购买过的商品进行打分（1~5分，没有购买过的先记为0分），那么我们可以搜集到一个维度为（m, n）的矩阵A。那么如何对用户进行商品推荐呢？借助上面个人“个人偏好”和“属性分值”的思想，我们可以借助数学方法把A矩阵分解为（m, k）维和（k, n）维的两个矩阵的乘积，其中$P[i,k]$表示第i个用户的“个人偏好”，$Q[k,j]$表示第j个商品的“属性分值”，k为商品的属性种类：

$$A_{m\times n} = P_{m\times k} \times Q_{k\times n} \tag{9}$$

那么，如果我们能够求得P和Q，我们也就知道了每一位用户的“个人偏好”和每一件商品的“属性分值”，也就能够计算每个用户对每件商品的综合评分，从而就能根据评分的高低为用户推荐商品了，这也就是LFM的**数学思想**，这种矩阵分解的方法也叫做**低秩矩阵分解（Low Rank Matrix Factorization）**。

求解方法

了解了LFM的数学思想之后，我们要解决的问题就变成了 —— 如何求解矩阵P和Q使得两者相乘的结果与矩阵A最接近（差距越小）？ 这时候，我们可以借鉴一下机器学习中线性回归问题的求解方法：

1. 利用所有矩阵元素的差值构造一个损失函数J（这里用的是SSE指标）：

   $$J(P,Q|A) = \sum_{i=1}^{m}\sum_{j=1}^{n}(a_{ij} - \sum_{r=1}^{k} p_{ir} * q_{rj})^2 + \lambda * (\sum_{i=1}^{m}\sum_{r=1}^{k} p_{ir}^2 + \sum_{j=1}^{n}\sum_{r=1}^{k} q_{rj}^2) \tag{10}$$

   等号左半部分就是矩阵元素差值平方的累加，右半部分是对P和Q的L2正则项

2. 对损失函数J实施梯度下降来求解未知参数：

   先求出梯度值：

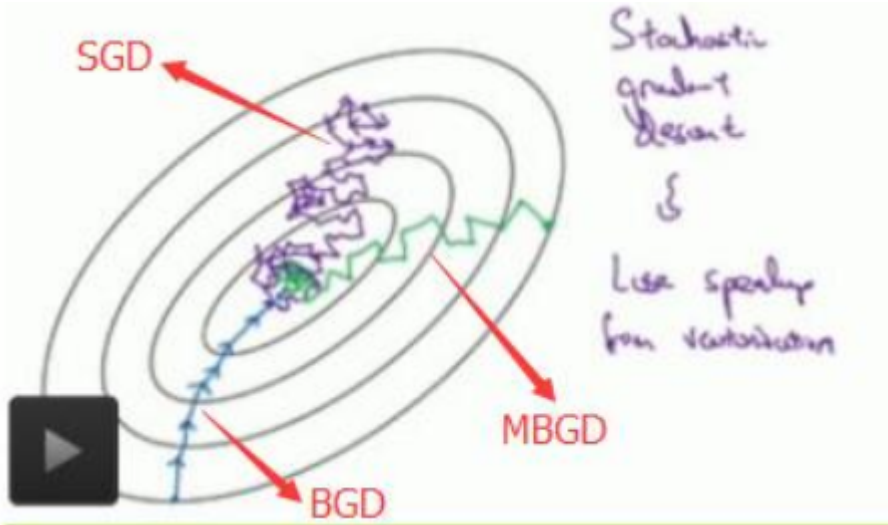
$$\begin{cases} \dfrac{\partial J}{\partial p_{ir}} = -2(a_{ij} - \sum_{r=1}^{k} p_{ir} * q_{rj}) * q_{rj} + 2\lambda * p_{ir} \\ \dfrac{\partial J}{\partial q_{rj}} = -2(a_{ij} - \sum_{r=1}^{k} p_{ir} * q_{rj}) * p_{ir} + 2\lambda * q_{rj} \end{cases}$$

再进行梯度下降：

$$\begin{cases} p_{ir} = p_{ir} + \alpha * [2(a_{ij} - \sum_{r=1}^{k} p_{ir} * q_{rj}) * q_{rj} + 2\lambda * p_{ir}] \\ q_{rj} = q_{rj} + \alpha * [2(a_{ij} - \sum_{r=1}^{k} p_{ir} * q_{rj}) * p_{ir} + 2\lambda * q_{rj}] \end{cases}$$

注：

这里使用的是**批量梯度下降 Batch Gradient Descent**，每一次梯度下降更新参数需要遍历所有的用户评分。 如果用户评分矩阵非常庞大，这种梯度下降的速度可能会比较慢，这时我们可以考虑使用**随机梯度下降 Stochastic Gradient Descent**，这种方法每次梯度下降更新参数只使用一条样本。 除此以外，还有一种方法是**小批量梯度下降Mini-Batch Gradient Descent**，它综合了BGD和SGD的优势，是一种折衷的梯度下降方法，也就是每次梯度下降更新参数的时候使用一小部分样本。 关于BGD, SGD, MBGD的收敛区别见下图，详情可以阅读博客



2.2.3 举例

下面我们举个例子，并进行求解： 假如我们有4位用户，5部电影（每位用户会对自己看过的电影打分1~5分，没看过的电影记为0分），用户打分情况（即矩阵A）如下：

	电影A	电影B	电影3	电影4	电影5
哈哈	5	0	1	0	5
大白	4	2	0	0	5
王老师	0	5	1	2	4

	电影A	电影B	电影3	电影4	电影5
佳晏	5	0	5	3	2

于是，我们编写如下代码求解A的分解矩阵P和Q

```
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()


def lfm(A, k):
    """
    :param A: 表示需要分解的评价矩阵
    :param k: 分解的属性（隐变量）个数
    :return:
    """
    assert type(A) == np.ndarray

    alpha = 0.01                    # 学习率
    lambda_ = 0.01                  # 正则化参数
    epochs = 10000                  # 随机梯度下降的次数
    print_step = 200                # 记录代价的间隔
    cost_list = []                  # 记录代价的list
    epoch_list = []

    m, n = A.shape                  # 用户数m，商品数n
    P = np.random.rand(m, k)        # "个人口味"矩阵
    Q = np.random.randn(k, n)       # "属性分值"矩阵

    for epoch in range(epochs):  # 进行epochs轮随机梯度下降
        for i in range(m):  # 遍历m个用户
            for j in range(n):  # 遍历n个商品
                if math.fabs(A[i][j]) > 1e-4:
                    err = A[i][j] - np.dot(P[i, :], Q[:, j])
 # 真实评分和预测平分的差值
                    # 遍历商品的k个属性
                    for r in range(k):
                        grad_P = err * Q[r][j] - lambda_ *
P[i][r]   # J对P[i][r]的负梯度
                        grad_Q = err * P[i][r] - lambda_ *
Q[r][j]  # J对Q[r][j]的负梯度
                        P[i][r] += alpha * grad_P
            # P梯度下降
```

```
                    Q[r][j] += alpha * grad_Q
            # Q梯度下降
        # 计算代价
        cost = calc_cost(A, P, Q, k, lambda_)
        if epoch % print_step == 0:
            print("epoch {}, cost = {}...".format(epoch,
cost))
            cost_list.append(cost)
            epoch_list.append(epoch)
    # 绘制learning curve
    plot_cost(epoch_list, cost_list)
    return P, Q, cost_list, epoch_list


def calc_cost(A, P, Q, k, lambda_reg):
    """计算代价值"""
    cost_err = np.sum(np.power(A - np.dot(P, Q), 2))
    cost_reg_P = lambda_reg * np.sum(np.power(P, 2))
    cost_reg_Q = lambda_reg * np.sum(np.power(Q, 2))
    cost = cost_err + cost_reg_P + cost_reg_Q
    return cost


def plot_cost(epoch_list, cost_list):
    """绘制学习曲线"""
    plt.plot(epoch_list, cost_list)
    plt.xlabel("epoch")
    plt.ylabel("cost")
    plt.title("learning curve")
    plt.show()


if __name__ == "__main__":
    A = np.array([[5, 0, 1, 0, 5],
                  [4, 2, 0, 0, 5],
                  [0, 5, 1, 2, 4],
                 [5, 0, 5, 3, 2]])
    P, Q, cost_list, epoch_list = lfm(A, 3)

    # 查看结果
    print(P)              # "个人口味"
    print(Q)              # "属性分值"
    print(np.dot(P, Q))  # 预测评分
    print(cost_list)
    print(epoch_list)
```

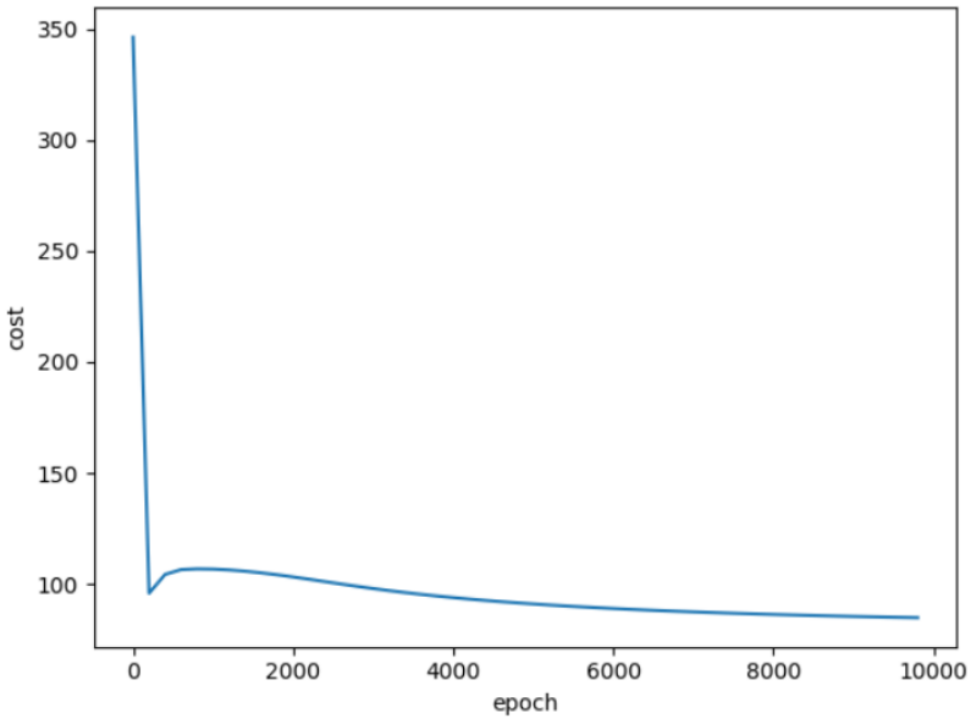
输出结果为:

P，“个人口味”矩阵： [[2.04865262 0.46595477 0.47920973] [1.92621974 -0.36010038 0.88606726] [1.56630487 1.41066346 0.36889496] [1.63047114 0.47425637 -1.76764177]]

Q，“属性分值”矩阵： [[2.3957747 1.34087647 0.89303134 0.96514552 2.18449315] [0.60735005 1.9949505 0.22303096 0.51658636 0.15997061] [-0.44800607 0.16196098 -1.92892288 -0.66355098 0.92592942]]

$P * Q$，**预测评分矩阵：** [[4.97641889 3.75416007 1.00907473 1.89997369 4.99352106] [3.99911803 2.00794863 -0.06929434 1.08510861 4.97064409] [4.44401292 4.97417171 1.00181104 1.99566102 3.98881765] [4.98619537 2.84608937 4.97148033 2.99155672 2.0009086]]

学习曲线



2.2.4 拓展知识：LFM与SVD

1. 低秩矩阵

以上的推荐算法，都有一个基本假设，就是假设评分矩阵是**低秩矩阵 Low Rank Matrix。**

秩的数学定义

从数学定义上看，如果一个m行n列矩阵A，如果它的秩满足：

$$rank(A) < m, n \tag{11}$$

那么矩阵A就是低秩矩阵。如何求矩阵A的秩呢？大学时候线性代数老师教过：对矩阵A初等变换为阶梯矩阵，矩阵A的秩就等于阶梯矩阵的非零行数（在python中可以通过np.linalg.matrix_rank(A)来求矩阵A的秩）。

秩的数学意义

我们再来回顾一下，秩代表什么意义呢？—— 秩的大小跟矩阵行（或者列）线性相关的程度有关：

- 秩越低（小）表示线性相关的行（或者列）越多
- 秩越高（大）表示线性相关的行（或者列）越少
- 如果一个矩阵是满秩矩阵，那么它的所有行都是线性无关的

举个例子，如下三元一次方程组：

$$\begin{cases} 3x + y + 4z = 5 \\ x + y + z = 3 \\ 2x + 2y + 2z = 6 \\ 3x + 3y + 3z = 9 \end{cases}$$

一共有4个方程，但是明显第2、第3、第4个方程的解是一样的，也就是说第3、第4个方程是没有意义的，可以直接忽略。从矩阵的角度来看，方程组的增广矩阵A的秩就是2，矩阵A的第2、第3、第4行是**线性相关**的。

评分矩阵低秩的意义

再来回到推荐系统，LFM的算法前提是用户评分矩阵是“低秩矩阵”，也就是说评分矩阵的部分行（或者列）是线性相关的，用人话讲也就是：

- 部分用户对不同电影的评分是相似的
- 或者，部分电影的不同用户评分是相似的

从常理上我们也能够理解：对于一部电影，肯定有一部分观众对于电影的理解程度是类似的，否则的话，用户评分也就没有意义了。

思想上升

再升华一下这种思想：**人以类聚，物以群分**。其实我们老祖先总结的道理还是非常牛逼的。很多数学问题都可以上升到哲学高度。

2. SVD

其实上面的低秩矩阵分解是由奇异值分解方法推导过来的。SVD定理可以证明，对任意一个矩阵A，都有它的满秩分解：

$$A_{m\times n} = P_{m\times k} \times Q_{k\times n} \tag{12}$$

其中k = Rank(A)。注：推导过程可以参考博客

因此，上面LFM模型的低秩矩阵分解实质上也就是SVD分解，加入正则化之后也被称为RSVD。

此外，还有很多SVD方法的扩展方法：

- ASVD, Asymmetric-SVD
- SVD++
- 对偶的SVD算法

详情可以阅读Yehuda Koren博士的论文*Matrix Factorization Techniques for Recommender Systems*，以及《推荐系统实战》的第8章内容。

2 推荐系统评价

2.1 实验方法

- 离线实验（依靠评价指标）
- 用户调查
- 在线实验（如ABtest）

2.2 评价指标

1 满意度

- 购买率
- 点击率
- 用户停留时间
- 用户反馈信息

2 准确度

- RMSE

$$RMSE = \frac{\sqrt{\sum_{u,i \in T}(r_{ui} - \hat{r}_{ui})^2}}{|T|} \tag{13}$$

- MAE

$$MAE = \frac{\sum_{u,i \in T}|r_{ui} - \hat{r}_{ui}|}{|T|} \tag{14}$$

  相比MAE，RMSE 加大了对预测不准的用户物品评分的惩罚（平方项的惩罚），因而对系统评测更加苛刻。

- Recall

$$Recall = \frac{\sum_{u \in U}|R(u) \bigcap T(U)|}{\sum_{u \in U} T(u)} \tag{15}$$

  召回率的含义：用户购买过的所有商品中，我能推荐出来百分之多少。

- Precision

$$Precision = \frac{\sum_{u \in U}|R(u) \bigcap T(U)|}{\sum_{u \in U} R(u)} \tag{16}$$

  精准率的含义：我推荐的结果中，有百分之多少是用户实际购买过的。

3 覆盖率

- Coverage

$$Coverage = \frac{U_{u \in U} R(u)}{|I|} \tag{17}$$

  覆盖率（coverage）描述一个推荐系统对物品长尾的发掘能力，可以简单定义为推荐系统推荐出来的物品集合占总物品集合的比例。覆盖率越高说明推荐算法越能够把长尾中的物品推荐给用户。这是一个内容提供商会关心的指标。

4 新颖度

新颖性（noverty）可以用推荐结果的平均流行程度来表示，平均流行程度越低，推荐结果新颖性越高。

5 惊喜度

惊喜度（ serendipity ）表示推荐结果和用户的历史兴趣不相似，但能让用户满意的程度。目前没有公认的定义公式。

6 信任度

信任度（trust）只能通过问卷调查来获取。提高信任度的方法有增加推荐系统的透明度（提供更可靠的推荐理由），还有通过用户的好友做推荐。

7 实时性

在新闻，微博等媒体网站中的推荐结果必须要有很强的时效性。实时性可以通过推荐结果的变化速率来衡量。

8 健壮性

健壮性（robust）指标衡量了一个推荐系统抗击作弊的能力。 通过注入噪声前后的推荐结果对比就可以评价推荐系统的健壮性。

2.3 评测维度

一个推荐算法，虽然整体性能不好， 但可能在某种情况下性能比较好， 而增加评测维度的目的就是知道一个算法在什么情况下性能最好。这样可以为融合不同推荐算法取得最好的整体性能带来参考。

- 用户维度：主要包括用户的人口统计学信息、活跃度以及是不是新用户等
- 物品维度：包括物品的属性信息、流行度、平均分以及是不是新加入的物品等
- 时间维度：包括季节，是工作日还是周末，是白天还是晚上等

3 推荐系统冷启动

3.1 冷启动问题

冷启动对于一个推荐系统来说其实是非常重要的。打个比喻：

- 推荐模型 —— 老朋友，"嘘寒问暖"
- 冷启动 —— 陌生人，"第一印象"

就像如果第一印象不好，你可能不会跟他做朋友一样，如果冷启动给用户的第一印象不好的话，用户很可能再安装App之后就立马卸载。

冷启动问题可以归结为三个类别：

- 系统冷启动

- 用户冷启动
- 物品冷启动

3.2 冷启动方法

3.2.1 用户冷启动

信息采集

利用新用户的注册信息：

- 人口统计学信息
    - 年龄
    - 性别
    - 职业
    - 学历
    - ...
- 用户兴趣预采集
- 从其他社交账号导入信息
- 通过注册手机/邮箱等购买用户画像（黑产业😔）

推荐策略

可以针对用户的某个特征推荐对应的**热门商品**：比如利用**人口统计信息**中的年龄、性别、职业三个特征，我们获取到一位用户的信息是：30岁、男、程序员，如果我们要给他推荐一本书的话，应该如何做呢？我们需要首先获取三张表格：

- 年龄 —— 图书表
- 性别 —— 图书表
- 职业 —— 图书表

然后从这三张表中分别查询出30岁、男、程序员喜欢的图书，并按照一定的权重加和，最终得到这位用户的推荐结果。

计算方法

那么如何得到这三张表呢？关键在于**如何计算每种特征的用户喜欢的物品。**也就是说，对于每种特征f，计算具有这种特征f的用户对商品i的喜欢程度$p(f,i)$：

$$p(f,i)=\frac{|N(i)\bigcap U(f)|}{|N(i)|+\alpha} \tag{18}$$

其中$N(i)$是喜欢用品i的用户集合，$U(f)$是具有特征f的用户集合，α是一个比较大的数字，用于避免一个商品只被一个用户喜欢而且这个用户恰好具有特征f（因为此时会导致$p=1$）的情况发生。

3.2.2 商品冷启动

推荐策略

如果推荐平台中不断有新的商品加入，那么如何将这些新的商品推荐给用户呢？我们可以利用商品的**内容信息**计算商品-商品的相似度矩阵，同时进行频繁更新（如30min更新一次），然后将新加入的商品推荐给喜欢与之类似商品的用户。

计算方法

如何利用商品的内容信息计算商品之间的相似度？—— 主要利用提前搜集到的商品信息关键词来组成**关键词向量**。比如一部手机，它的关键词可能有：中国红（颜色），轻薄（外观），大电池（续航）等等。

对于商品d_i，它的关键词向量可以表示为：

$$d_i=\{(e_1,w_1),(e_2,w_2),\dots\} \tag{19}$$

其中e_i就代表商品d_i的关键词对应的数字，w_i就是该关键词对应的权重。

在给出所有商品的关键词向量之后就可计算任意两商品的相似度了，这里可以用简单的余弦相似度来计算：

$$w_{i,j}=\frac{d_i*d_j}{\sqrt{||d_i||*||d_j||}} \tag{20}$$

有了商品的相似度矩阵之后，可以借鉴ItemCF算法思想来进行推荐，这种基于内容向量的推荐方法也叫做**内容过滤算法ContentItemKnn**。

3.2.3 系统冷启动

系统冷启动说的通俗点就是推荐系统在刚刚建立时既没有充足的用户，也没有充足的商品，"一穷二白"的时候。这个不多说了，先请个靠谱的架构师或者产品经理最重要。