Item CF 单路召回

召回几乎是所有推荐系统的基础模块，对应到电商的推荐中，它的作用是从海量的商品池中，筛选出一部分用户可能感兴趣的商品作为上层排序系统的候选集。因此，可以说召回效果的好坏直接决定了推荐效果的上界。召回分为单路召回还有多路召回，常见的有基于用户行为的召回，还有基于协同过滤的召回，和 embedding 向量相似的召回但是新闻是一个随时更新的的场景.赛题以新闻 APP 中的新闻推荐为背景，要求选手根据用户历史浏览点击新闻文章的数据信息预测用户未来点击行为，即用户的最后一次点击的新闻文章，测试集对最后一次点击行为进行了剔除.

推荐系统-新闻推荐的练习
字段
train_click_log.csv：训练集用户点击日志
testA_click_log.csv：测试集用户点击日志
articles.csv：新闻文章信息数据表
articles_emb.csv：新闻文章 embedding 向量表示
sample_submit.csv：提交样例文件
user_id,article_1,article_2,article_3,article_4,article_5

Field	Description
user_id	用户id
click_article_id	点击文章id
click_timestamp	点击时间戳
click_environment	点击环境
click_deviceGroup	点击设备组
click_os	点击操作系统
click_country	点击城市
click_region	点击地区
click_referrer_type	点击来源类型
article_id	文章id，与click_article_id相对应
category_id	文章类型id
created_at_ts	文章创建时间戳
words_count	文章字数
emb_1,emb_2,...,emb_249	文章embedding向量表示

推荐系统索要解决的问题：
就是在海量的 36 万篇的文章，选出 5 篇文章推荐给用户.
预测最后一次点击，就是预测出第一篇文章最好，预测成功就是 1.
目标给用户推荐的是 5 篇没有看过的文章，如何转化为机器学习？
通过用户的点击行为在 36 万篇文章里召回 50 篇，就是看第 51 篇是否被用户点击，但是前提是 51 篇排序在第一位的被用户点击. 所以在机器学习训练之前多出来召回和排序的问题.

物品+用户+Label

先召回物品，比如 30 万文章找 50 个，如果 1000 个人 X50 个等于 50000 个
传统用 TOP10 方法，现在比较流行联邦学习就是 embedding.
乐搜推荐是一种召回方式比如抖音没有注册用户时的推荐.

归纳出新闻召回预测比赛的三个因素：
-ItemCF 物品之间的相似度
-userid
-还有时间戳就是最后一次点击的文章.

代码的实现关键点

新闻的时效性很强，基于近期的历史信息比较好。
一个用户有多次的点击行为.

```
[17]: df_click.head()
```

t[17]:

	user_id	click_article_id	click_timestamp	click_environment	click_de
0	0	30760	1508211672520	4	
1	0	157507	1508211702520	4	
2	1	289197	1508211316889	4	
3	1	63746	1508211346889	4	
4	2	36162	1508211438695	4	

- 机器学习的目的就是通过训练集中的用户 id 还有用户的点击的文章，还有就是用户点击的时长来给测试集中的比如 50000 个用户每个人推荐 5 篇文章.
- Set 的目的是去重
- 首先从训练集里随机采样 1000 份作为验证集样本。
- 然后 groupby，最后一次点击为 label，每个 groupby 有一个 id。然后把每个 groupby 最后一条点击拿出来

```
df_train_click = pd.concat(click_list, sort=False)
df_train_query = pd.concat(train_query_list, sort=False)
```

找出历史点击和最后的点击

召回实现

Itemcf 计算物品的相似度

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

第一步字典的 for 循环，for key，value in 字典被多少个用户喜欢.

```
def cal_item_sim(df):
    user_item_ = df.groupby('user_id')['click_article_id'].agg(
        lambda x: list(x)).reset_index()
    user_item_dict = dict(
        zip(user_item_['user_id'], user_item_['click_article_id']))

    item_cnt = defaultdict(int)
    sim_dict = {}

    for user, items in tqdm(user_item_dict.items()):
        for item in items:
            item_cnt[item] += 1
            sim_dict.setdefault(item, {})
```

第二步计算物品和物品之间的相似度.

```
        for relate_item in items:
            if item == relate_item:
                continue

            sim_dict[item].setdefault(relate_item, 0)

            sim_dict[item][relate_item] += 1 / math.log(1 + len(items))
```

加了一个惩罚项通过 hashmap 方式找相似度

得到用户和点击

```
]: user_item_dict
```

```
    981: [205824, 209122],
    982: [36162, 36385],
    983: [234308, 36162],
    984: [36162, 30760],
    985: [289003, 50644],
    986: [234308, 79851],
    987: [38801, 39001],
    988: [70986, 126011],
    989: [205958, 70986],
    990: [205824, 96877]
```

物品的相似度

```
item_sim[233917]
```

```
199198: 0.010102360252528884,
224658: 0.004104346924096312,
198659: 0.009024815676666308,
336476: 0.06662145609475435,
29945: 0.016171971264318213,
158229: 0.00846957257794673,
284474: 0.022976562957490544,
64329: 0.00910399528357524,
236726: 0.024076478685028036
```

最后开始做召回
最后一次点击，做机器学习特征和标签

```
]: def recall(df_query, item_sim, user_item_dict):
    data_list = []

    for user_id, item_id in tqdm(df_query.values):
        rank = defaultdict(int)

        if user_id not in user_item_dict:
            continue

        interacted_items = user_item_dict[user_id]
        interacted_items = interacted_items[::-1]
```

找最后一次的 userid

```
    interacted_items = user_item_dict[user_id]
    interacted_items = interacted_items[::-1]

    for loc, item in enumerate(interacted_items):
        for relate_item, wij in sorted(item_sim[item].items(), key=lambda d: d[1]
            if relate_item not in interacted_items:
                rank.setdefault(relate_item, 0)
                rank[relate_item] += wij

    sim_items = sorted(
        rank.items(), key=lambda d: d[1],  reverse=True)[:50]
    item_ids = [item[0] for item in sim_items]
    item_sim_scores = [item[1] for item in sim_items]

    df_temp = pd.DataFrame()
    df_temp['article_id'] = item_ids
    df_temp['sim_score'] = item_sim_scores
    df_temp['user_id'] = user_id
```

```
        sim_items = sorted(
            rank.items(), key=lambda d: d[1],  reverse=True)[:50]
        item_ids = [item[0] for item in sim_items]
        item_sim_scores = [item[1] for item in sim_items]

        df_temp = pd.DataFrame()
        df_temp['article_id'] = item_ids
        df_temp['sim_score'] = item_sim_scores
        df_temp['user_id'] = user_id

        if item_id == -1:
            df_temp['label'] = np.nan
        else:
            df_temp['label'] = 0
            df_temp.loc[df_temp['article_id'] == item_id, 'label'] = 1

        df_temp = df_temp[[
            'user_id', 'article_id', 'sim_score', 'label'
        ]]
        df_temp['user_id'] = df_temp['user_id'].astype('int')
        df_temp['article_id'] = df_temp['article_id'].astype('int')

        data_list.append(df_temp)

    df_data = pd.concat(data_list, sort=False)
    return df_data
```

转化为机器学习模型

```
df_data.head(10)
```

	user_id	article_id	sim_score	label
0	196387	156560	0.079398	0.0
1	196387	160417	0.053963	0.0
2	196387	272143	0.053529	0.0
3	196387	158229	0.051297	0.0
4	196387	337890	0.050473	0.0
5	196387	64329	0.046571	0.0
6	196387	158536	0.044666	1.0
7	196387	156447	0.043698	0.0
8	196387	157332	0.042521	0.0
9	196387	336476	0.041805	0.0

协同过滤推荐 collaborative Fittering Recommendation

基于一种用户购买行为的推荐算法. 有很多的方发比如邻域方法 neighborhood-based，还有就是隐语义的模型 latent factor model，基于图的随机游走算法 Random walk on graph

协同过滤，就是协同和过滤两个操作. 协同就是利用群体的行动来做出决策推荐
，生物上协同进化的说法， 通过协同的作用， 让群体逐步优化，推荐系统就是达到这个目的， 而过滤就是从可以决策推荐方案中找出用户喜欢的方案过滤出来.

1 基于邻域的模型 第七课内容
简单的归纳为利用余弦相似度计算用户之间的相似都，然后把类似的商品推荐给另外一个没有看过该商品的相似用户.基于商品的就是计算商品之间的相似度，然后再推荐一个商品给点击过相似商品的人，但是没有点击过该推荐商品，以便提高商品的购买力.

2 基于模型的协同过滤矩阵分解
推荐系统的场景主要分为两个：1.Top-N 推荐（item ranking)；2.评分预测（rating prediction)
而评分预测主要用于评测分析来看过的电影评论多少分，听过的音乐打多少颗星，或者用户看自己看过的书评价多少分
用户和电视剧表示为二维矩阵.矩阵中表示用户对电视剧的评价，下图表示用户对电视剧的评价越来越高？表示用户无评分记录。所以评分预测任务可以看做是一个矩阵补全的任务，通过用户已有的评价补全用户没有评分的记录
矩阵补全是需要通过矩阵分解来补全。所以矩阵分解是为了更好的矩阵补全，来补全表格中问好表示未知的评分。

User/Item	小猪佩奇	大秦赋	足球小子	四大才子	汪汪队立大功
1	5	2.2	4	?	4
2	?	2.4	3.5	3.4	4.0
3	2.1	4.7	?	5	?
4	4.1	?	?	2	2
5	?	4	2	2	?
6	?	?	1	4	4
.....	.....	.....	.....	.....	.....

3 隐语义模型算法

LFM 隐语义模型早期用于文本的挖掘找出文本的隐含语义，利用隐含的特性联系用户兴趣和相关物品.
LFM 降维方法--矩阵因子分解

假设用户物品评分矩阵为 R，现在有 m 个用户，n 个物品
K 个隐类，找到两个矩阵 P 和 Q， 使这两个矩阵的乘积近似于 R
即将用户物品评分矩阵 R 分解为两个低维度矩阵相乘

$$\hat{R}_{m\times n} = p^{T}_{m\times k} \cdot Q_{k\times n} \approx R$$

LFM(Lat
ent Factor Model,隐语义模型)的主要思想是将原始评分矩阵 M（mn）分解成两个矩阵 P（mk）和 Q（kn），同时仅考察原始评分矩阵中有评分的项分解结果是否准确，而判别标准则是均方差(MSE)。即对于矩阵 M(mn)，我们想办法将其分解为 P(mk)、Q(kn)，此时对于原始矩阵中有评分的位置 MUI 来说，其在分解后矩阵中对应的值就是

$$M_{UI} = \sum_{k-1}^{k} P_{uk} Q_{ki}$$

通过训练 user 和 item 在每个隐类别上的“兴趣”或者“比重”，进而恢复出 UI 矩阵的思想

遇到的 UI（User-Item)矩阵都是稀疏的，即用户基本和绝大多数的 item 都没有交互过，体现在隐式反馈的数据中则是一大片的空缺的。所以希望利用模型恢复出（i,j）位置用户 i 对物品 j 的打分（主要这个位置用户 i 没有对物品 j 进行评分）
使用了 LFM 之后，将 UI 矩阵抽象到了两个矩阵 P 和 Q 上，这两个矩阵一个代表用户兴趣矩阵，一个代表物品矩阵，两者相乘恢复出 UI 矩阵。只要这个用户或者物品与其他物品或者用户交互过，其矩阵就能得到训练，所以不需要矩阵稠密，即可训练。

特殊的情况：
当然，用户没有告诉系统为什么会打出这样的分数，所以需要挖掘用户的打分的隐藏因素，同时通过未被评分的与这些隐藏的关联度，得出未评分的物品的得分（预测值）
隐藏因素：如年代，动画类型，题材等等找到合适的隐藏因子可以对 user_item 进行关联，推测一个用户对某一件商品的评分.
这样评分预测任务其实是可以看成为一个矩阵补全的任务，矩阵补全是推荐系统的任务，但是矩阵补全是需要通过矩阵分解来补全。所以矩阵分解是为了更好的矩阵补全，来补全表格中问好表示未知的评分。

User/Item	小猪佩奇	大秦赋	足球小子	四大才子	汪汪队立大功
1	5	2.2	4	?	4
2	?	2.4	3.5	3.4	4.0
3	2.1	4.7	?	5	?
4	4.1	?	?	2	2
5	?	4	2	2	?
6	?	?	1	4	4
.....	.....	.....	.....	.....	.....

LFM 模型的求解
矩阵的分解

m×n的评分矩阵R可以通过两个小矩阵$P_{m\times k}$和
$Q_{k\times n}$的乘积R来近似表示

	item1	item2	item3	item4
user1	R11	R12	R13	R14
user2	R21	R22	R23	R24
user3	R31	R32	R33	R34

$\hat{R}$ =

	K1	K2	K3
user1	P11	P12	P13
user2	P21	P22	R23
user3	P31	P32	P33

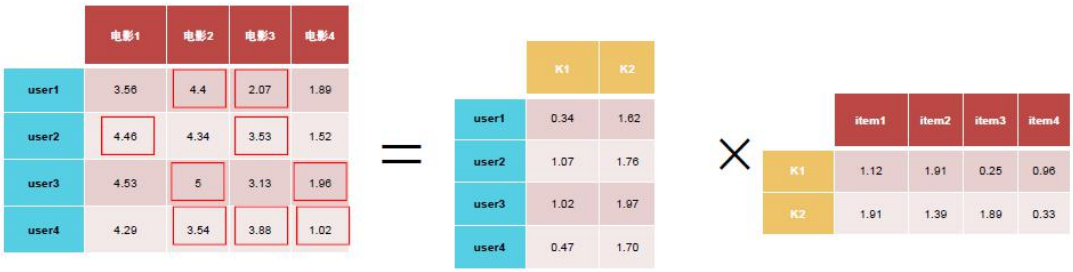
P ×

	item1	item2	item3	item4
K1	Q11	Q12	Q13	Q14
K2	Q21	Q22	R23	Q24
K3	Q31	Q32	Q33	Q34

Q

如果近似矩阵R与原始矩阵R在已知打分的矩阵上一致，则认为在预测的位置上二者也是相似的

- 相似矩阵



*内容来自开课吧第八课课件

模型的求解

LFM 其实是就是求相似的矩阵 $\hat{R}$

原始的矩阵 R 与相似矩阵 $\hat{R}$ 在已知的 ui 用户-物品评分上存在一定的误差，如何找到一个好的矩阵分解的方式，就是找到于是矩阵和相似预测的矩阵之间的误差值最小.
可以用损失函数选择的为平方差，Ω 其实就是一个正则项.

$$loss = \Sigma_{(u,i)\in R_0}(R_{ui} - \hat{R}_{ui})^2 = \Sigma_{(u,i)\in R_0}(R_{ui} - P_U^T Q_i)^2 + \Omega$$

ALS 算法
目标是找到一个最好的矩阵分解方式，得到的相似矩阵和原始矩阵相互之间的总误差最小，损失函数选择的为平方差，求解出 P 和 Q 误差最小化的过程，一般采用随机梯度法或者交替最小二乘法 ALS-P/Q 均未知，可以先固定住 Q，把 P 当成变量，通过损失函数最小化求出 P，反过来，先固定住 P，把 Q 当成变量，通过损失函数最小化求出 Q;如此交替，求得满足最小误差要求的 P 和 Q

1.随机生成 Q0,一般为 0 或者全局平均
2.固定 Q0,求解 P0
3.固定 P0,求解 Q1
4.固定 Q1,求解 P1
5.重复以上过程
6.直到损失函数达到目标值，或者迭代次数

ALS 求解的过程

求解过程以固定 Q，求解 P 为例由于每个用户 u 都是相互独立的，当 Q 固定时，用户特征向量 Pu 应该取得与其他特征向量无关，所以每一个 Pu 都可以单独求解

loss 损失函数最小化公式

$$l = \min_{P}[\sum_{u,i}(R_{ui} - P_u^T \cdot Q_i)^2 + \lambda \sum_{u} \| P_u \|^2]$$

$$= \sum_{u} \min_{P}[\sum_{i}(R_{ui} - P_u^T \cdot Q_i)^2 + \lambda \| P_u \|^2]$$

我们只需求每个用户的 Pu,使得 loss 最小

$$l = \sum_{i}(R_{ui} - P_u^T \cdot Q_i)^2 + \lambda \| P_u \|^2$$

对 Pu 求导

$$\frac{\partial l}{\partial \mathrm{P_u}} = \sum_{i}(R_{ui} - P_u^T \cdot Q_i)^2 + \lambda \| P_u \|^2$$

$$= \sum_{i} 2(P_u^T \cdot Q_i - R_{ui})Q_i + 2\lambda P_u = 0$$

$$\sum_{i} 2(P_u^T \cdot Q_i - R_{ui})Q_i + 2\lambda P_u = 0$$

$$P_u^T \cdot Q_i = \sum_{k} P_{uk} Q_{ki} = Q_i^T P_u$$

$$\sum_{i}(Q_i^T P_u - R_{ui})Q_i + \lambda P_u = 0$$

$$(\sum_{i} Q_i^T Q_i + \lambda I)P_u = \sum_{i} R_{ui} Q_i$$

$$[(Q_1, Q_2, Q_3, \cdots, Q_n)(Q_1^T, Q_2^T, Q_3^T, \cdots, Q_n^T) + \lambda I]P_u$$

$$= (Q_1, Q_2, Q_3, \cdots, Q_n)(R_{u1}, R_{u2}, R_{u3}, \cdots, R_{un})^T$$

$$(Q_1, Q_2, Q_3, \cdots, Q_n) = Q$$

$$(QQ^T + \lambda I) P_u = QR_u^T$$

$$P_u = (QQ^T + \lambda I)^{-1} QR_u^T$$

LFM-梯度下降法

$$\frac{\partial l}{\partial \mathrm{P_u}} = \sum_i (R_{ui} - P_u^T \cdot Q_i)^2 + \lambda \| P_u \|^2$$

$$= \sum_i 2(P_u^T \cdot Q_i - R_{ui}) Q_i + 2\lambda P_u$$

梯度下降算法

$$P_u := P_u - \alpha \frac{\partial loss}{\partial P_u} = P_u - \alpha \sum_i 2(P_u^T \cdot Q_i - R_{ui}) Q_i + 2\lambda P_u$$

$$Q_i := Q_i - \alpha \frac{\partial loss}{\partial Q_i} = Q_i - \alpha \sum_u 2(P_u^T \cdot Q_i - R_{ui}) P_u + 2\lambda Q_i$$