实验一 分类技术---二分网络上的链路预测

一. 实验内容

基于网络结构的链路预测算法被广泛的应用于信息推荐系统中。算法不考虑用户和产品的内容特征，把它们看成抽象的节点，利用用户对产品的选择关系构建二部图。为用户评估它从未关注过的产品，预测用户潜在的消费倾向。

简单的讲就通过现有数据集，通过二部图资源分配的算法来为用户推荐和数据集中用户看过后这部电影后联系度排名前几部的电影。

二. 分析及设计

1．数据预处理

我们需要处理数据来预测，就得先分析数据的内容。

先来看数据集 ml-1m，分别有 movies.dat,ratings.dat,users.dat 还有一份说明 README，参考说明可知其中 movies.dat 包含序号，电影名，上映时间，电影类型，导演。Users,dat 包含年龄性别职业等等，可见这两个数据集并不是我们所需要的。下面详细讲解 rating.dat 数据集。

导入数据包见详细实现部分 1，调用 head()函数先看 ratings.dat 数据前几行：(参见代码 print1)

```
C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe F:/数据挖掘上机/exp1/main.py
(1000209, 4)
   UserID  MovieID  Rating   Datetime
0       1     1193       5  978300760
1       1      661       3  978302109
2       1      914       3  978301968
3       1     3408       4  978300275
4       1     2355       5  978824291

Process finished with exit code 0
```

由上图可知 ratings.dat 包含了用户编号，电影编号，评级水平（1-5 星），时间戳。显然我们可以根据此数据集构建二部图并计算资源分配矩阵。

2．二部图的构建

设立阈值 threshold=3，假定该数据由 m 个用户和 n 个电影构成二部图，其中如果用户 i 对电影 j 的评分大于该阈值，就在 i 和 j 之间连接一条边 aji =1(i =1 ,2 , …, m ;j =1 , 2 , … , n)，否则 aji =0。那么这个推荐系统可以用一个具有 m +n 个节点的二部分图表示。

3．基于二部分图资源分配的推荐算法

如何计算资源配额矩阵呢？简单来说是求对看过一部电影的用户，来推荐数据中看过该电影的用户中都看过的电影中前几部电影。

算法如下：

对于任意目标用户 i，推荐算法的目的是把所有 i 没有选择过的产品按照 i 喜欢的程度进行排序，并且把排名靠前的那些产品推荐给 i .假设 i 选择过的所有产品，都具有某种向 i 推荐其他产品的能力.这个抽象的能力可以看做位于相关产品上的某种可分的资源———拥有资源的产品会把更多的资源交给自己更青睐的产品.对于有 m 个用户和 n 个产品的一般的推荐系统，如果用表示 Wij 产品 j 愿意分配给产品 i 的资源配额，可以得到 Wij 的一般表达式：

$$w_{ij}=\frac{1}{k_j}\sum_{l=1}^{m}\frac{a_{il}a_{jl}}{k_l}$$

其中，kj 表示产品 j 的度（被多少用户评价过），kl 表示用户 l 的度（用户选择过多少产品）。

对于上式：若直接按部就班地来计算将循环$m*n^2$次，而 python 中有矩阵相乘优化，所以难点在于如何将上式转化成矩阵形式。

4. 算法改进

对原矩阵 A(m*n)，设其转置矩阵为 B(n*m)。

考虑

$$\sum_{l=1}^{m}a_{il}a_{jl}=\sum_{l=1}^{m}a_{il}b_{lj}$$

即：

$$\begin{pmatrix}a_{01} & a_{02} & \cdots & a_{ij}\\ \cdots & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\end{pmatrix}\times\begin{pmatrix}b_{00} & \cdots & \cdots & \cdots\\ b_{01} & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\\ b_{ji} & \cdots & \cdots & \cdots\end{pmatrix}=\begin{pmatrix}\sum ab & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\\ \cdots & \cdots & \cdots & \cdots\end{pmatrix}$$

那么我们只需要将 A 矩阵与 Kj 矩阵和 Kl 矩阵分别按次序相除即可，最后在乘以原矩阵转置就符合矩阵运算法则。

先看从矩阵 A 按列方向逐个乘，$w_{ij}=\frac{1}{k_j}\sum_{l=1}^{m}\frac{a_{il}a_{jl}}{k_l}$ 中，a_{il}/k_l可以并行运算，按行方向乘，

a_{il}/k_j可以并行运算。最终转置矩阵相乘。

令 Y 矩阵，Y_i表示每个用户 i 选择的电影度数，K 矩阵，K_j表示电影 j 一共被选择的次数。原公式可化为矩阵运算：

$$W = (\sum_{j=1}^{m}(\sum_{i=1}^{n}(A_i/Y))/K) * A^T$$

大致算法如下即：

```
1. for i in range (1,n,1):
2.     temp[:,i]=A[:,i]/Y
3. for j in range (1,m,1):
4.     temp2[j,:]=temp[j,:]/K
5. D=npy.dot(A.T,temp2)
```

由于 python 语言特性，上述 for 循环可以放在一起写,见详细实现 def compute_f_mat();

5．输出结果并绘制 ROC 曲线

为了量化算法的精确性, 把真实数据随机划分为两个部分，一部分看做训练集，另外一部分是隐藏起来用于检测算法准确程度的测试集。构造二部分图和计算 W 矩阵时，只有训练集可以使用。在没有其他已经条件的前提下，只能假设用户已经选择过的产品是他喜欢的，因此，一个好的算法应该要把训练集中已知的用户喜欢的产品排在比较靠前的位置。对于任意一个用户 i，假设他有L_i个产品是没有选择过的，那么算法会给出这L_i个产品一个按照喜好程度的排序(最终资源数量相同的产品被赋予一个随机的序号)。如果在测试集中 i 选择了产品 j(这同时意味着 j 不会出现在训练集中,因此是算法中L_i个没有选择的产品之一),而 j 被算法排在第位,那么认为(i,j)的相对位置是

$$r_{ij} = \frac{R_{ij}}{L_i}$$

越精确的算法，给出的 r_{ij} 越小。对所有用户的 r_{ij} 求平均值 $\langle r \rangle$ 来量化评价算法的精确度。选取不同的算法阈值，计算相应的真阳性率（TP）以及假阳性率（FP），画出 ROC 曲线。

三. 详细实现

备注：共有四次中间结果打印，在源代码中分别以 print Number（1-4）形式注释。注释格式为

```
'''

#print Number

code...

'''
```

如果想要查看中间结果只需要去掉注释号即可。

1. 导入数据包：

```
#导入数据

curpath=os.path.abspath('.')#当前地址

filename = curpath+"\\ml-1m\\ratings.dat"

all_ratings = pd.read_csv(filename, header=None, sep="::", names=["UserId", "MovieId", "Rating", "Datetime"], engine="python")
```

2. 打印 ratings.dat 数据的行数列数，以及前几行

print1:图见分析与设计部分。代码参考文件 main.py 部分

3. 用 train_test_split()函数将数据集分为训练集和测试集：

```
train_ratings, test_ratings, _, _ = train_test_split(all_ratings, all_ratings['UserId'], test_size=0.1)
```

print2:打印原数据集大小，以及训练集与测试集大小之比：

```
#print2

#The ratio of test set to training set

print('tarin_ratings : test_ratings =\n',len(train_ratings),':',len(test_ratings))
```

打印结果：

```
C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe F:/数据挖掘上机/exp1/main.py
(1000209, 5)
tarin_ratings : test_ratings =
 900175 : 100034

Process finished with exit code 0
```

4. 建立二部图

```
userId_col = all_ratings['UserId']
movieId_col = all_ratings['MovieId']

user_count = np.array(userId_col.value_counts())  # count number, every element of array meas number of this ID index
movie_count = np.array(movieId_col.value_counts())  # count number, every element of array meas number of this ID index
movie_index = np.array(movieId_col.value_counts().index)

userId_max = user_count.shape[0]  # all number
movieId_max = movie_count.shape[0]  # all number

mat = np.zeros([userId_max, movieId_max])#create empty matrix

#count the rating of users
for row in train_ratings.itertuples(index=True, name='Pandas'):
    mat[row.UserId - 1, np.where(movie_index == row.MovieId)[0][0]] = row.Rating
```

设立阈值 threshold=3，打分大于 3 建立边。

```
    # set zero when elements smaller that threshold
    threshold=3
    mat_like = (mat > threshold) + 0
    mat_dislike = ((mat > 0) + 0) * ((mat <= threshold)+0)
```

print3：对比前后效果，原电影-用户推荐 ratings 图和建立好的二分网络：

```
main
C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe F:/数据挖掘上机/exp1/main.py
原二维数组：用户电影推荐：
 [[0. 4. 0. ... 0. 0. 0.]
 [4. 0. 5. ... 0. 0. 0.]
 [0. 5. 4. ... 0. 0. 0.]
 [0. 5. 2. ... 0. 0. 0.]]
二分网络图,阈值为3
 [[0 1 0 ... 0 0 0]
 [1 0 1 ... 0 0 0]
 [0 1 1 ... 0 0 0]
 [0 1 0 ... 0 0 0]]

Process finished with exit code 0
```

5. 计算资源分配矢量

```
f_mat = compute_f_mat(mat_like,user_count,movie_count)
```

compute_f_mat()函数实现：

```
def compute_f_mat(mat_rat,user_count,movie_count):
    """
    compute the f matrix
    :param mat_rat: user`s rating matrix([user number,movie number]) where 1 means user likes the index movie.
    :param user_count: statistics of moive numbers that user have watch.
    :param movie_count: statistics of user numbers that movie have been rated.
    :return: f matrix
    """
    temp = (mat_rat / user_count.reshape([-1,1]) )/ movie_count.reshape([1,-1])
    W = np.dot(mat_rat.T, temp)

    f = np.dot(W, mat_rat.T).T

    return f
```

详细原理见分析与设计中 3．基于二部分图资源分配的推荐算法和 4．算法改进部分。

print4:

打印用户数量和电影数量、F 矩阵大小。

以及查看前 4 名用户对各个电影推荐 f 系数。

```
C:\Users\lenovo\AppData\Local\Programs\Python\Python37\python.exe F:/数据挖掘上机/exp1/main.py
number of users: 6040
number of movies 3706
(6040, 3706)
[[0.17138475 0.20008812 0.19435547 ... 0.         0.         0.        ]
 [0.31250454 0.33910457 0.33833678 ... 0.         0.         0.        ]
 [0.12002193 0.14477875 0.14960456 ... 0.         0.         0.        ]
 [0.08603781 0.11673456 0.11231918 ... 0.         0.         0.        ]]
```

6. 绘制 ROC 曲线：

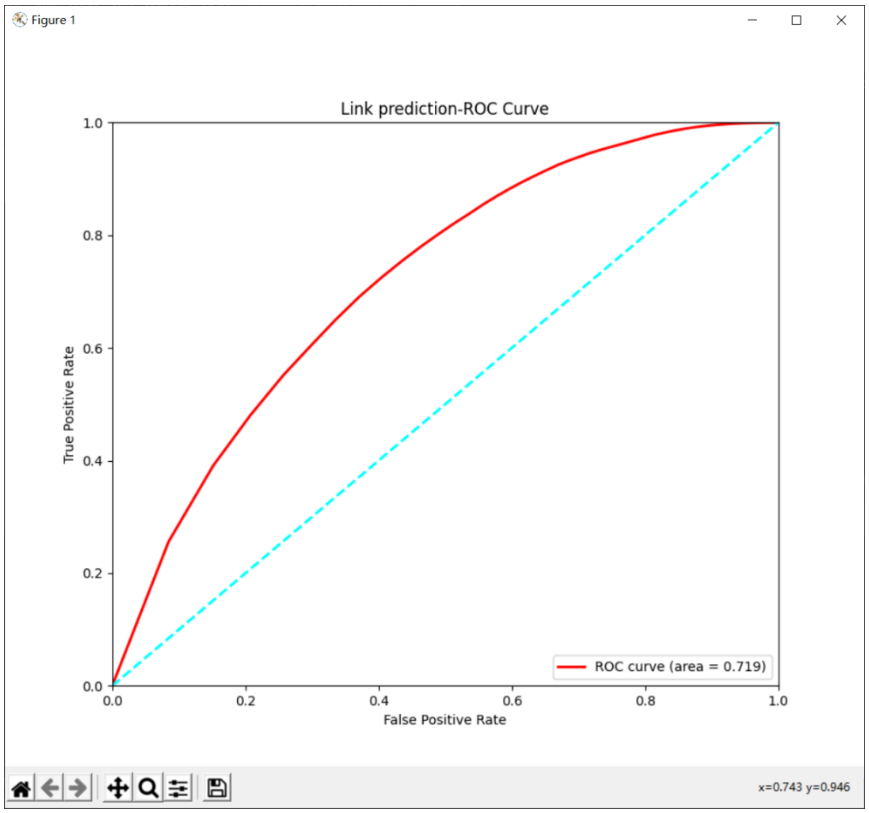
```
roc_pic(f_mat, user_count, mat_like, mat_dislike)
```

roc_pic()函数见源码部分。

四. 实验结果

绘制 roc 曲线如下：

算法阈值的划分排名等级 num=80，值越大效果越好。



五. 心得体会

通过本次实验，我学到很多，包括论文排版，Latex 公式写法，以及通过中间变量查错。二分网络上链路预测的原理并不难，不过处理矩阵运算是需要用数学知识，来让行列间矩阵并行运算。此外也明白了 ROC 曲线绘制原理。最后最重要的一点，本次实验让我感受到了把所学知识真正应用到实际的快乐，终于让我觉得有不再做题的感觉，逐渐清晰未来数据挖掘需要处理的工作，以及应用数据挖掘知识能解决的问题。