实验一 分类技术 —— 二分网络上的链路预测

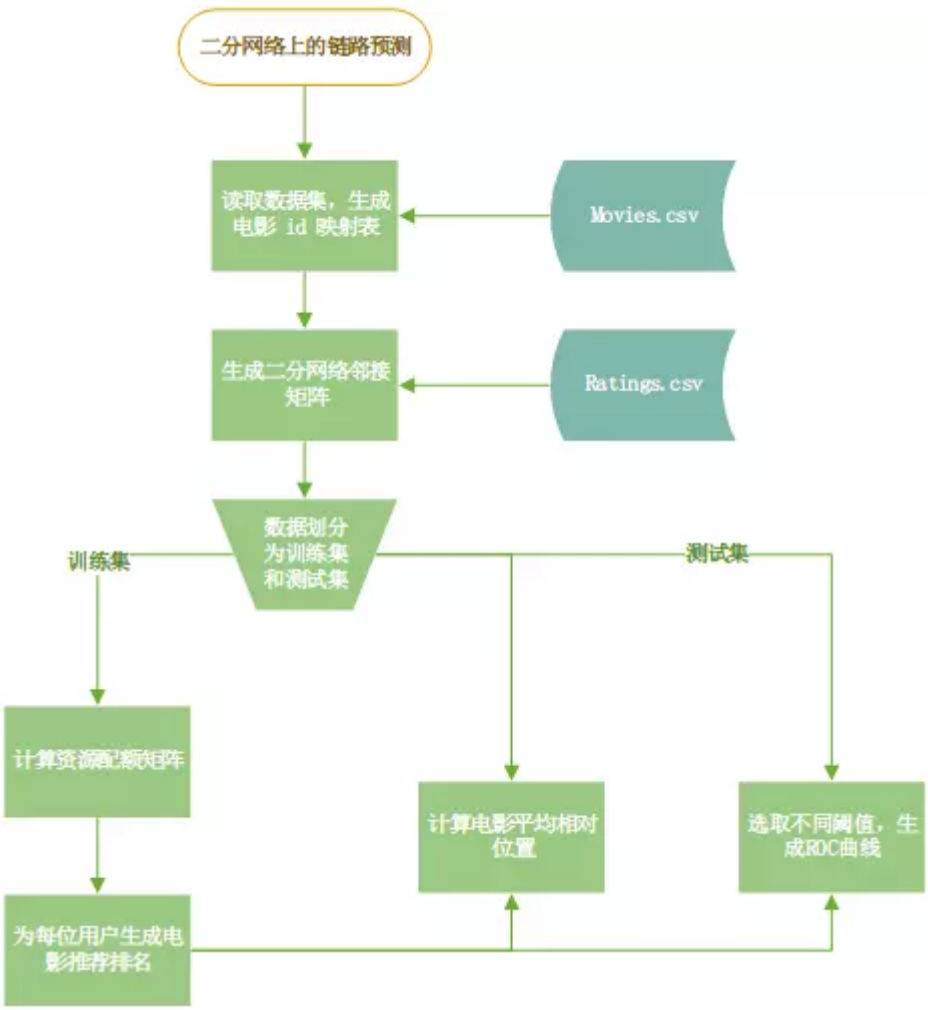
实验内容

> 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。
>
> MovieLens 是历史最悠久的推荐系统。它由美国 Minnesota 大学计算机科学与工程学院的 GroupLens 项目组创办，是一个非商业性质的、以研究为目的的实验性站点。MovieLens 主要使用 Collaborative Filtering 和 Association Rules 相结合的技术，向用户推荐他们感兴趣的电影。https://grouplens.org/datasets/movielens/

实验通过用户对电影的打分数据，生成二分网络模型，在此基础上计算资源配额矩阵，使用资源配额矩阵对所有电影进行打分，将排序靠前的电影推荐给用户。最后使用用户喜欢电影在排名中的相对位置，以及设定不同的阈值绘出 ROC 曲线。来判断预测的准确性。

分析和设计

总设计流程如图所示：



- 步骤1：导入用户对电影的评分数据：

  评分数据通过 csv 的形式给出，读取时，可以使用 python 的 csv 模块直接对 csv 文件进行分析读取。

  之后，将用户对电影的评分数据转换为二部图，采用邻接矩阵进行保存。由m个用户n部电影构成的电影推荐系统，初始化 m * n 二维矩阵为全 0 矩阵，用户 i 对电影 j 打分超过3分，将矩阵元素 a[j][i] 置为 1

- 步骤2：计算资源配额矩阵 W

  > 矩阵 W 中每个元素 w_{ij} 的定义如下
  >
  > $$w_{ij} = \frac{1}{k_j}\sum_{l=1}^{m}\frac{a_{il}a_{jl}}{k_l}$$
  >
  > 其中，k_j 表示产品 j 的度（被多少用户评价过），k_l 表示用户 l 的度（用户选择过多少产品）

  对于 m 个用户 n 部电影构成 的系统，如果直接使用该公式进行计算，其复杂度为：$O(n^2m)$ 计算该矩阵将会花费大量时间，因此，需要使用 numpy 的矩阵运算对资源配额矩阵的计算进行加速，W 的计算公式如下：

  $$W = \frac{A}{j}(\frac{A}{l})^T$$

  其中，A 为矩阵形式的用户评分二部图，j, l 为向量，j_i 表示产品 i 的度，l_i 表示用户 i 的度。将矩阵中每个元素的计算转化为矩阵计算可以大大加快计算速度。

- 步骤3：电影推荐和推荐结果分析

  > 目标用户的资源分配矢量f。初始时，将他选择过的电影对应项资源设置为1，其他为0，得到初始n维0/1向量。则最终的资源分配矢量:
  >
  > $$f' = Wf$$
  >
  > 将用户所有没看过的电影按照 f' 中对应项的得分进行排序，推荐排序靠前的电影给该用户。

  对于每个用户，分别计算 $f' = Wf$ 得到一个一维矩阵 f', f'_i 表示预测的当前用户对电影i 的喜好程度。

  得到 f' 之后，对其进行排序，将排序靠前的电影进行推荐，选定推荐电影有两种方式，第一种是按照排名占所有电影的百分比进行筛选，第二种是按照电影推荐的分数进行筛选，分数大于某一数值的电影认定为用户可能喜欢的电影。

  > 将二部图中的边随机分为两部分，期中90%归为训练集，10%归为测试集。
  >
  > 对给定用户i，假设其有Li个产品是未选择的，如果在测试集中用户i选择的电影j，而电影j依据向量 f' 被排在第Rij位，则计算其相对位置：
  >
  > $$r_{ij} = \frac{R_{ij}}{L_i}$$
  >
  > 越精确的算法，给出的rij越小。对所有用户的rij求平均 $\langle r \rangle$ 来量化评价算法的精确度。

  测试集是二部图中的边的 10%，而二部图中的边表示 a_{ji} 用户 i 对电影 j 的打分超过三分，故测试集中的电影，都是用户喜欢的电影。对于测试集中的每一个数据，都可以求出其电影的相对位置，将这些数据求均值，就可得到平均的相对位置 $\langle r \rangle$

  > 画出ROC曲线来度量预测方法的准确性。
  >
  > 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。

  在上一步中定义的测试集中，只有正样例，即测试集中全部是用户喜欢的电影，为了画出 ROC 曲线，需要在测试集中加入反例 （用户对该电影的评分小于等于 3 分，即用户明确表示不喜欢的电影），这样才能够计算出假阳性率 FP

$$TPR = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

以 TPR 为纵坐标，FPR 为横坐标绘制出 ROC 曲线

详细实现

1. **读取数据集**

   在读取数据时发现给定的数据有一定问题，即电影的 id 编号是不连续的，最大的电影 id 达到了 193609，但实际上总共只有 9743 部电影，如果直接使用电影 id 来构建邻接矩阵，就会使得矩阵的大小异常庞大，也不利于之后的处理和计算。

   因此，在这里我定义了映射表，为每个电影重新分配其 id 值，使其 id 值变得连续，减小了矩阵的大小，使其可以直接在内存中进行计算：

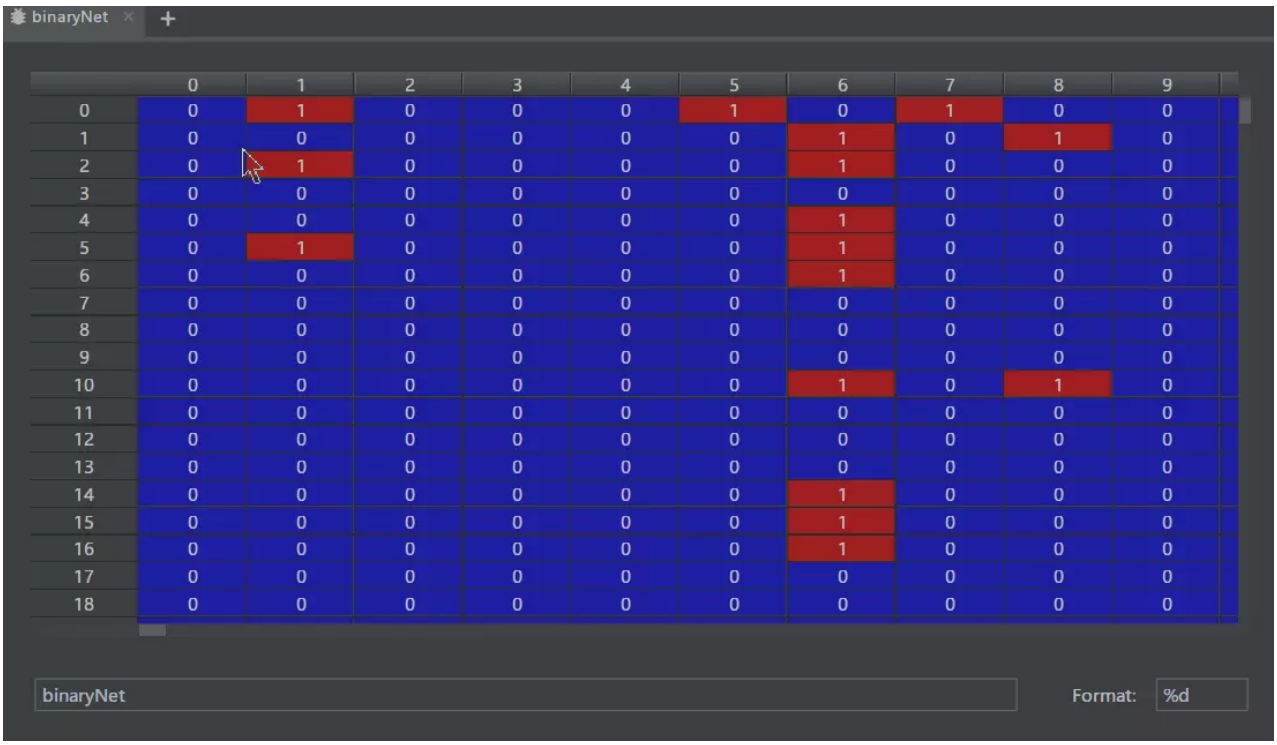
```
def ReadDataSet():
    """
    读取数据集
    :return:
    """
    with open('movies.csv', 'r', encoding='UTF-8') as csvfile:
        # 读取 movies 文件，为每个电影重新分配其 id 值
        reader = csv.reader(csvfile)
        id = 0
        for row in reader:
            try:
                url = int(row[0])
                id2url[id] = url
                url2id[url] = id
                id += 1
            except Exception as e:
                print(e)

    with open('ratings.csv') as csvfile:
        # 读取用户评分表
        reader = csv.reader(csvfile)
        for row in reader:
            try:
                i = int(row[0])
                j = url2id[int(row[1])]
                if int(float(row[2])) > 3: #为打分超过三分的电影添加二部图中的边
                    binaryNet[j][i] = 1
                else:
                    unlike.append((i, j))
                    #打分小于三分的电影视作用户不喜欢的电影，将其记录，用于 ROC曲线绘制
            except Exception as e:
                print(e)
```

我定义了 `ReadDataSet` 函数，分别读取两个 csv 文件，完成 id 映射表和二部图的构建

```
binaryNet = np.zeros((MOVIESNUM, USERNUM), dtype=int)
```

生成的二部图局部如下：



	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	0	1	0	1	0	0
1	0	0	0	0	0	0	1	0	1	0
2	0	1	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0
5	0	1	0	0	0	0	1	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	1	0	1	0
11	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0

2. **计算资源配额矩阵**

只用之前推导的 W 矩阵运算公式完成运算，可以显著缩短运算所用时间：

```
def ResourcesQuota(train_set):
    """
    计算资源配额矩阵
    :param train_set: 训练集
    :return: W 资源配额矩阵
    """
    kl = np.count_nonzero(train_set, axis=0)  # 计算有多少用户
    kj = np.count_nonzero(train_set, axis=1)  # 计算有多少电影

    temp = np.true_divide(train_set, kl)
    temp[np.isnan(temp)] = 0  # 计算过程中会产生 nan 元素，将其置为 0
    temp2 = np.true_divide(temp.T, kj).T
    temp2[np.isnan(temp2)] = 0  # 计算过程中会产生 nan 元素，将其置为 0
    W = np.dot(train_set, temp2.T)  # 完成资源配额矩阵的计算
    return W
```

在计算过程中，会出现某些用户或电影的度为 0 的情况，这样会导致 矩阵中的元素和 0相除，得到结果 nan，后续的计算无法进行，我的解决办法是将这些 nan 按照 0 来进行处理。

3. **进行训练集和测试集的划分**

训练集是二部图中 90% 的边，测试集是剩下的 10%，使用 numpy 的 random.choice 随机从矩阵的非 0 元素中取 10%，作为测试集，并将这些位置的元素置为 0

```python
def getTrainSetandTestSet():
    """
    划分训练集和测试集
    :return:
    """
    # pos记录二部图的边，即对应邻接矩阵元素为 1 的位置
    pos = np.where(binaryNet == 1)
    # 随机选取二部图中边的 10%
    testpos = np.random.choice(len(pos[0]), int(len(pos[0]) * 0.1))
    tp = ([], [])
    # 取得测试集的元素位置
    for i in testpos:
        tp[0].append(pos[0][i])
        tp[1].append(pos[1][i])
    TrainSet = np.copy(binaryNet)
    # 将邻接矩阵这些位置的元素置为 0
    TrainSet[tp] = 0
    return TrainSet, list(zip(tp[1], tp[0]))
```

4. **精确度和 ROC 曲线的绘制**

```python
    pos = np.zeros((USERNUM, MOVIESNUM))  # 存储预测结果 pos[i][j] 表示 用户 i 对电影 j 的预测喜好排名
    mark = np.zeros((USERNUM, MOVIESNUM))  # 存储预测结果 mark[i][j] 表示 用户 i 对电影 j 的预测打分

    print("对所有用户进行预测排名中，时间较长...")
    # 对每个用户分别预测
    for i in range(USERNUM):
        r = recommend(W, train[:, i])
        reslut = zip(r, range(len(r)))
        reslut = sorted(reslut, reverse=True)
        # 对结果完成排序
        j = 0
        for r in reslut:
            k, v = r
            pos[i, v] = j  # 存储排序值
            mark[i, v] = k
            j += 1
        print(f'user:{i}, Finish Recommend')

    r = 0
    kl = np.count_nonzero(train, axis=0)
    kj = np.count_nonzero(train, axis=1)
    for t in test:
        r += pos[t[0]][t[1]] / (MOVIESNUM - kl[t[0]])
        print(f'Test Set: (user {t[0]}, movie {t[1]}), Rank: {pos[t[0]][t[1]]}, UserUnSelectNum: {MOVIESNUM-kl[t[0]]}')
    print("平均 r 为: ", r / len(test))
```

以上代码完成了对平均 r 值的计算

```
    print("选取不同阈值计算 ROC 曲线...")
    x = []
    y = []
    for i in range(100):
        TP = 0
        FP = 0
        FN = 0
        TN = 0
        for t in test:
            if pos[t[0]][t[1]] / (MOVIESNUM - kl[t[0]]) < i / 100:
                # 相对位置在前 百分之i 的电影为推荐电影
                TP += 1
            else:
                FN += 1
        for t in unlike:
            # unlike 集里面是用户不喜欢的电影
            if pos[t[0]][t[1]] / (MOVIESNUM - kl[t[0]]) > i / 100:
                TN += 1
            else:
                FP += 1
        print(np.array([[TP, FN], [FP, TN]]))
        TPR = TP / (TP + FN)
        FPR = FP / (FP + TN)
        print(f"TPR: {TPR},  FPR: {FPR}")
        # 对不同阈值下的 TPR 和 FPR 进行记录
        x.append(FPR)
        y.append(TPR)

    #使用 plt 库完成图像绘制
    plt.title('ROC')
    plt.xlabel('FPR')
    plt.ylabel('TPR')

    plt.plot(x, y, 'r', label='roc')

    plt.legend(bbox_to_anchor=[0.3, 1])
    plt.grid()
    plt.show()
```

以上代码完成了 ROC 曲线的绘制

实验结果

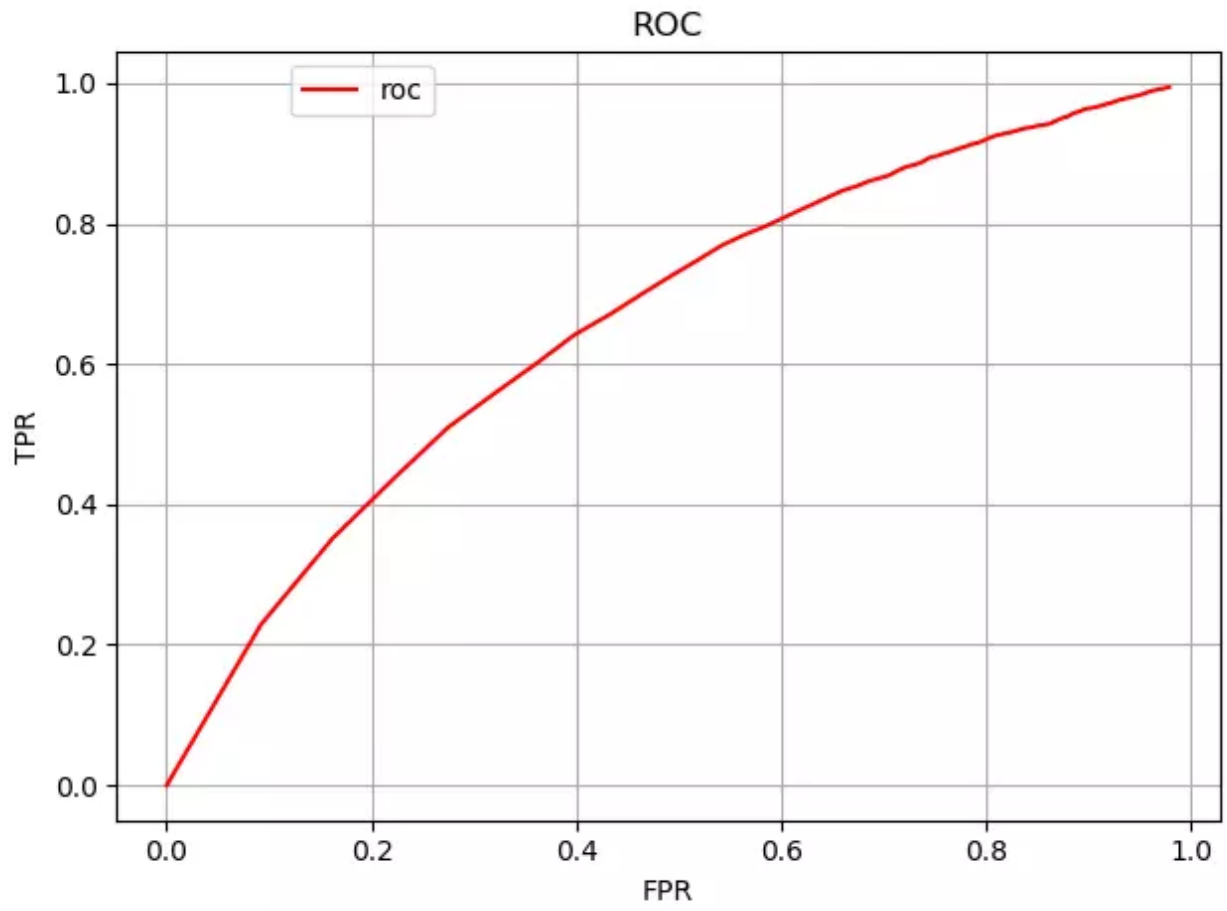
- W 资源配额矩阵的局部计算结果：

binaryNet × W × +

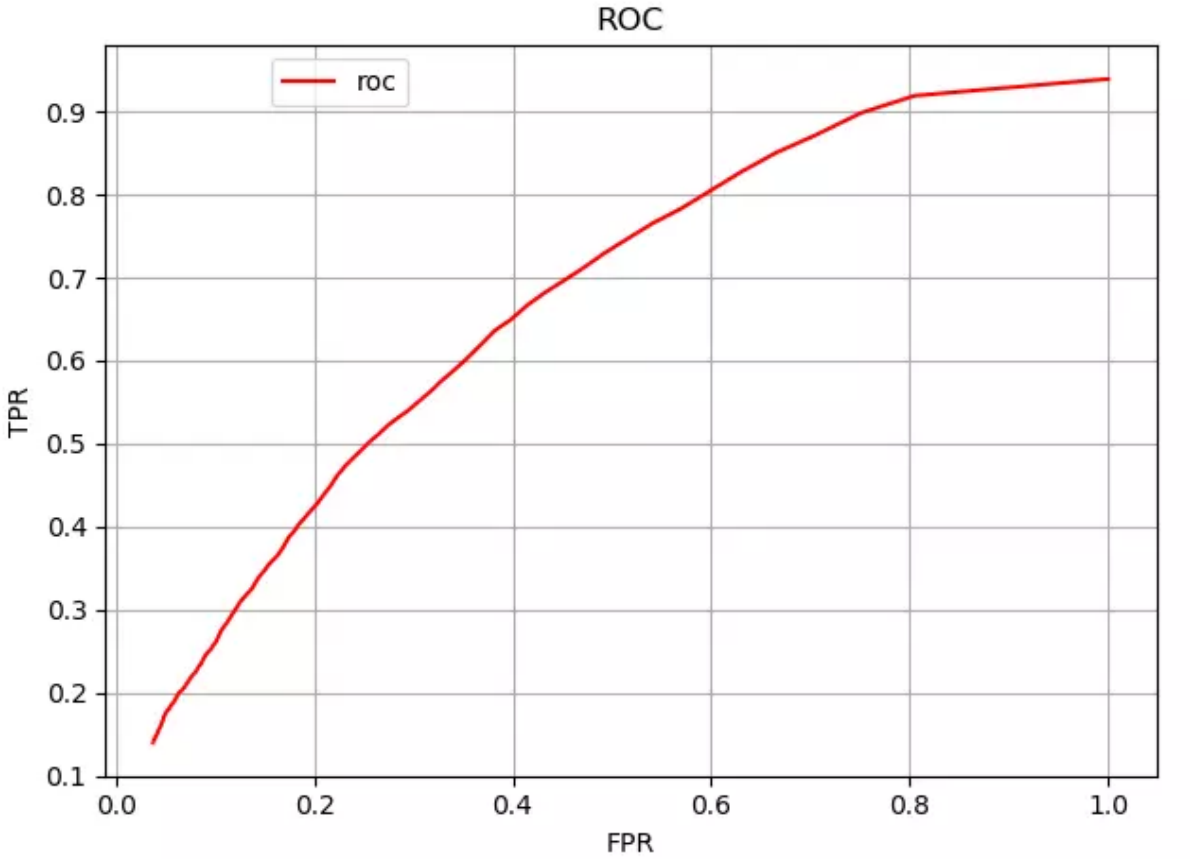
	0	1	2	3	4	5
0	0.021156651904498173	0.005666347960003973	0.011105479150807474	0.0	0.010933894833158542	0.006010110061584511
1	0.0017915658991189034	0.016349458375856902	0.0016800459518906121	0.0	0.00558784723033004	0.001185525982162245
2	0.0012248690239861186	0.0005860625413571902	0.022068381988710313	0.0	0.0028062070821831204	0.001574290822172178
3	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0009647554264551658	0.0015593992270688485	0.002244965665746496	0.0	0.034501680994265586	0.0010043272567127528
5	0.0028282870878044756	0.0017645037874042717	0.006716974174601293	0.0	0.005356412035801349	0.01681673327844834
6	0.0011660798819315738	0.0011315460042078485	0.0010328444536252838	0.0	0.0032292712158865666	0.0001192748091603053
7	7.822277847309137e-05	0.0	0.0007092198581560283	0.0	0.0008865248226950354	0.0
8	0.0008037161300028946	0.0	0.0022537878787878786	0.0	0.0015151515151515152	0.0002840909090909091
9	0.0012826435590215421	0.0008980547529857318	0.006264775413711583	0.0	0.0013520741709259479	0.0012085209923801972
10	0.0010854789510750457	0.002371626439719837	0.0014961441169249472	0.0	0.0019271878825532335	0.0011117150446919928
11	0.0002284165975578369	0.000722433889950368	0.0	0.0	0.0	2.5826446280991735e-0
12	4.9682034976152626e-05	0.0001571338780641106	0.0	0.0	0.0	0.0
13	0.0007791129648340274	0.0005963029218843172	0.0	0.0	0.0	0.0001418038806431663
14	0.0	0.0012828040897267506	0.0	0.0	0.0	0.0
15	0.001090477162526145	0.0003505432802398243	0.0012121212121212121	0.0	0.0025318013344783185	0.00605730355749594
16	0.0025068916251659397	0.0015110754681143463	0.0016754208754208755	0.0	0.005356412035801349	0.0023486149717987874
17	0.0003052273767962778	0.0002114573447915436	0.007264957264957264	0.0	0.0007558928791805504	6.133780991735536e-05
18	0.0006497449729625798	0.002894150684043686	0.0009523809523809524	0.0	0.0	0.0009258629959885945
19	0.0	0.0	0.0	0.0	0.0	0.0
20	0.0016401243080179802	0.0012942696825876923	0.0006045921465125212	0.0	0.00037537537537537537	0.002140560815422893
21	0.0001267748478701825	0.0001775252973548730	0.0	0.0	0.0006361323155216284	0.0001192748091603053
22	7.822277847309137e-05	0.0005963029218843172	0.0007092198581560283	0.0	0.0008865248226950354	0.0
23	0.0002471644816284285	0.0017555492959549462	0.0012121212121212121	0.0	0.0021512838306731435	0.0004033657182512144
24	0.0029285839571774816	0.0008157452988551274	0.0012121212121212121	0.0	0.00472027972027972	0.005922184526880983
25	0.0001668393392664771	0.0002443523489453568	0.0	0.0	0.0006361323155216284	0.0001547861727966689

W Format: %.5f

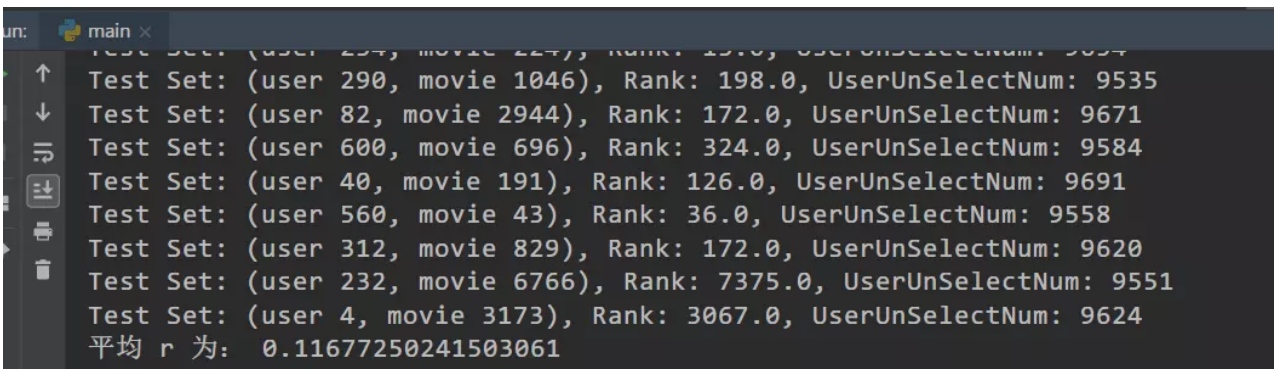
- 绘制出的 ROC 曲线：
  - 以电影推荐分数排名的百分比为阈值绘制 ROC 曲线



- 以电影推荐分数为阈值绘制的 ROC 曲线



- **平均 r 值:**



实验心得

本次实验我使用资源配额矩阵的方法，对二分网络进行预测，完成了电影推荐系统，预测用户可能感兴趣的电影。最后我使用测试集数据，调整推荐阈值，绘制出了以电影推荐分数排名的百分比为阈值绘制 ROC 曲线，以及以电影推荐分数为阈值绘制的 ROC 曲线，并计算了用户感兴趣的电影的平均相对位置，完成了对算法性能的评价。

在本次实验中，我遇到的最大的问题是资源配额矩阵的计算复杂度过高，我通过转换为矩阵运算的方法，大大提高了计算速度，矩阵运算中会产生除 0 运算，我将除 0 运算产生的 nan 值置为 0，使得实验得以顺利进行。

观察实验生成的 ROC 曲线，我认为其推荐结果还不够理想，本实验的模型只是简单的将所有用户喜欢的电影作为边加入二部图，而没有充分利用到对电影的具体评分。并且实验里忽略了用户明确表示不喜欢的电影数据（评分小于等于 3分），该实验算法并没有充分利用所给数据，还有很大的提升空间。