环境

```
OS: Manjaro 18.0.2 Illyria
Kernel: x86_64 Linux 4.19.13-1-MANJARO
python 3.7
numpy 1.15
matplotlib 3.0.2
networkx 2.2
jupyter 4.4.0
```

注：以下的程序划分仅仅是为了符合格式，实际的代码运行使用jupyter notebook. 程序为`Lab2.ipynb`打开后使用`Kernel-->Restart & Run All`。所以查看代码请使用jupyter notebook打开相应codes文件夹中的`src.ipynb`文件，格式更加清晰。

Part II-1

人员

- 董恒 PB16111545
- 张抑淼 PB16111543
- 张莎 PB16111042

预处理

1. 对ratings的预处理如下
    - 抽取成numpy格式

      ```
      ratings=np.loadtxt("./data/ml-1m/ratings.dat",delimiter="::",dtype=int)
      ```

    - 然后获取相关的`rating_matrix`

      ```
      user_num=6040
      movie_num=3952
      rating_matrix=np.zeros((6040+1,3952+1))
      for r in ratings:
          rating_matrix[r[0],r[1]]=r[2] #rating_matrix[userID,movieID]
      ```

    - 分割成训练集和测试集

```
# 分割为train & test
train_matrix=rating_matrix.copy()
test_user_size=rating_matrix.shape[0]//10
test_movie_size=rating_matrix.shape[1]//10

train_matrix[:test_user_size,:test_movie_size]=0

test_matrix=rating_matrix[:test_user_size,:test_movie_size].copy()

print(train_matrix.shape,test_matrix.shape,rating_matrix.shape)
```

2. 对movies的预处理如下
   - 从txt中抽取数据

```
movies=np.zeros((movie_num+1,3),dtype="<100U")
with open("./data/ml-1m/movies.dat",encoding = "ISO-8859-1") as f:
    line=f.readline()
    while line:
        temp=line.strip().split("::")
        movies[int(temp[0]),:]=temp
        line=f.readline()
```

   - 将genre改成int类型的标识符

```
movie_profile=np.zeros((movie_num+1,18))
for i in range(movie_num+1):
    temp=movies[i,2].split('|')
    for g in temp:
        if g in genre:
            movie_profile[i,genre[g]]=1
```

Content-Based Methods

1. 算法描述

   由于仅有ratings矩阵可供使用，所以现在假定item profile $\mathbf{i}$为电影评分向量。而user profile $\mathbf{x}$为$\mathbf{i}$的加权平均和，权重为评分的值。然后所得到的预测值为

$$5 \times \frac{\mathbf{x} \cdot \mathbf{i}}{||\mathbf{x}|| \cdot ||\mathbf{i}||}$$

   伪代码为

```
cb_pred=np.zeros((test_user_size,test_movie_size))
for i in range(1,test_user_size):
    for j in range(1,test_movie_size):
        norm=np.linalg.norm(train_matrix[:,j])*np.linalg.norm(user_profile[i])
        if norm == 0:
            cb_pred[i,j]=0
        else:
            cb_pred[i,j]=((train_matrix[:,j]@user_profile[i])/(norm))*5
```

2. 评价。

使用4种方式进行评价：RMSE, MAE, Precision, Recall

其中

$$RMSE=\sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j-\bar{y}_j)^2}$$

$$MAE=\frac{1}{n}\sum_{j=1}^{n}|y_j-\bar{y}_j|$$

而Precison和Recall使用了TopN的概念，也就是推荐N个，使用的度量方式是，推荐系统推荐的为排名在前N的电影，而ground truth选择评分大于3的电影。

需要强调的是，由于用户仅仅给出了部分电影的得分，有些电影并没有得分，所以在计算RMSE和MAE的值时候，需要把那些原本就没有评分的值去掉，这些值并不知道是否用户会喜欢。

同样对于Precison和Recall值，确实可以推荐那些没有看过的电影，但是这样也没法判断是否推荐正确了，所以仍然需要去除。

最终的计算方式为

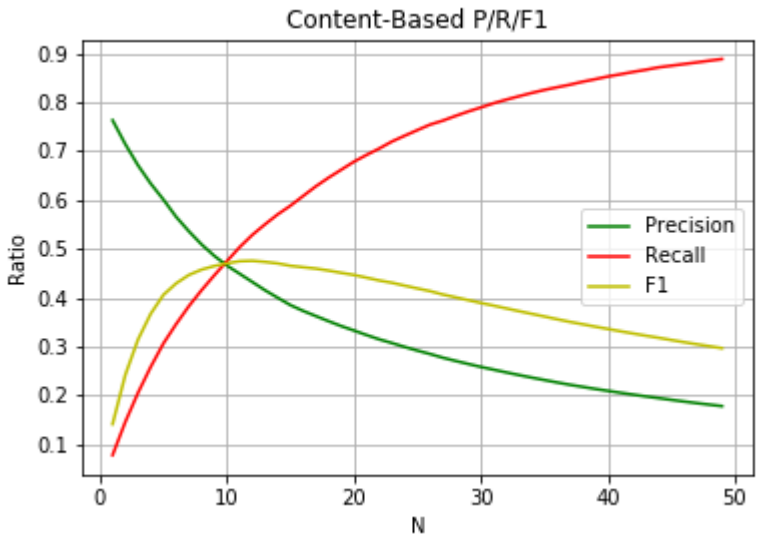
```
result_matrix2=cb_pred.copy()
result_matrix2[test_matrix==0]=0
result_matrix=result_matrix2-test_matrix

#RMSE
RMSE=np.linalg.norm(result_matrix)/np.count_nonzero(test_matrix)
#MAE
MAE=np.sum(np.absolute(result_matrix))/np.count_nonzero(test_matrix)
N=10 #TopN
recommend_rate=3
pred=np.argsort(result_matrix2,axis=1)[:,-N:]
#truth=np.argsort(test_matrix,axis=1)[:,-N:]

Total_Right=0
index=np.arange(test_movie_size)
for i in range(test_user_size):
    #Total_Right+=np.intersect1d(pred[i],truth[i]).shape[0]


 Total_Right+=np.intersect1d(pred[i],index[test_matrix[i]>recommend_rate]).shape[0]

precision=Total_Right/(N*test_user_size)
recall=Total_Right/test_matrix[test_matrix>recommend_rate].shape[0]
```

下面的几个方法的评测方式类似，不再详细给出代码。

在这样的定义下得到的结果如下(N=10)

```
RMSE=0.016676901400536383
MAE=1.4344089529739494
Precision=0.4870860927152318
Recall=0.49628879892037786
```

为了判断不同的N对于Precison 和Recall以及F1值的影响，做了另外的测试。得到的结果如下



可以发现正好差不多是10的时候就兼顾了Precison和Recall

Collaborative Filtering Methods

1. 算法描述

   这里使用的算法为PPT给出的

CF: Common Practice

- Define similarity s_{ij} of item *i* and *j*
- Select *k* nearest neighbors *N(i; x)*
  - Items most similar to *i*, that were rated by *x*
- Estimate rating as the weighted average:

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

  - b_{xi}: baseline estimate for r_{xi}
  - $b_{xi} = \mu + b_x + b_i$
  - μ=overall mean movie rating
  - b_x=rating deviation of user *x* = (average rating of user *x*)-μ
  - b_i=rating deviation of movie *i*

相应的伪代码过程为

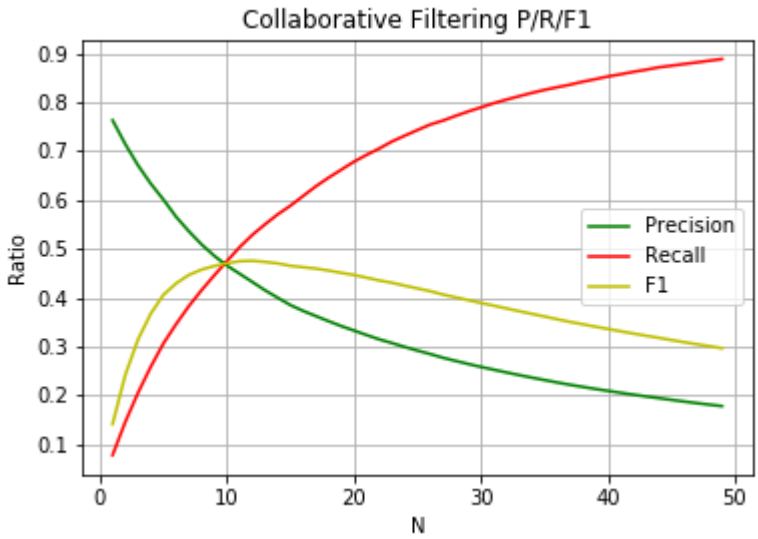
> 1. 计算item相似性movie_sim[i,j]
>
>   公式为
>
> $$Sim_{ij} = \frac{\sum_{x \in S_{ij}} (r_{xi} - \bar{r}_i)(r_{xj} - \bar{r}_j)}{\sqrt{\sum_{x \in S_{ij}} (r_{xi} - \bar{r}_i)^2 (r_{xj} - \bar{r}_j)^2}}$$
>
>   其中S_{ij}为同时给i, j评分过的人，r_{xi}表示用户x给电影i的评分，$\bar{r}_i$为电影i得分的平均分
>
> 2. 计算用户x对未知的i的得分，公式为
>
> $$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} Sim_{ij}(r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} Sim_{ij}}$$
>
>   其中各种符号的定义见上面的PPT

2. 评测

依然使用上面提到的4种方法，得到的结果为

```
RMSE=0.010776504401721263
MAE=0.8694166710904666
Precision=0.46589403973509935
Recll=0.47469635627530365
```

同样为了比较不同的TopN的N值对结果的影响，另做额外的测试，得到结果如下



通过上面的数据可以看到，CF的结果与CB结果相近。也同样得出了使用N=10比较均衡的结论。

Improved Methods

1. 初衷

   注意到数据集还有其他的数据可以使用 ，比如电影的流派/种类，于是，可以考虑使用流派建立用户的profile，然后来预测。

2. 算法

   算法本身是基于Collaborative Filtering的，巨大的不同点在于如下的公式

$$r_{xi} = b_{xi} + (1-\alpha)\frac{\sum_{j\in N(i;x)} Sim_{ij}(r_{xj}-b_{xj})}{\sum_{j\in N(i;x)} Sim_{ij}} + \alpha \mathrm{LIKE}(x,i)$$

   其中α为预先设定的参数，相当于权重。而$\mathrm{LIKE}(x,i)$表示用户x对i的喜爱程度，这里同样使用了movie_profile和user_profile, 不同点在于，movie_profile使用18个电影流派的one-hot码的加权表示，而user_profile使用movie的profile向量的加权和来表示。

   > 1. 计算movie_sim[i,j]与CF相同
   > 2. 计算movie_profile[movie_num,18]使用one-hot码的加权和
   >
   > ```
   > movie_profile=np.zeros((movie_num+1,18))
   > for i in range(movie_num+1):
   >     temp=movies[i,2].split('|')
   >     for g in temp:
   >         if g in genre:
   >             movie_profile[i,genre[g]]=1
   > ```
   >
   > 3. 计算user_profile, 使用movie_profile的加权和(权重为给电影的评分)
   >
   > ```
   > user_profile=np.zeros((test_user_size,18))
   > for i in range(test_user_size):
   >     user_profile[i]=np.sum(movie_profile.T*train_matrix[i],axis=1)
   > ```
   >
   > 4. 根据上面的公式计算r_{xi}.

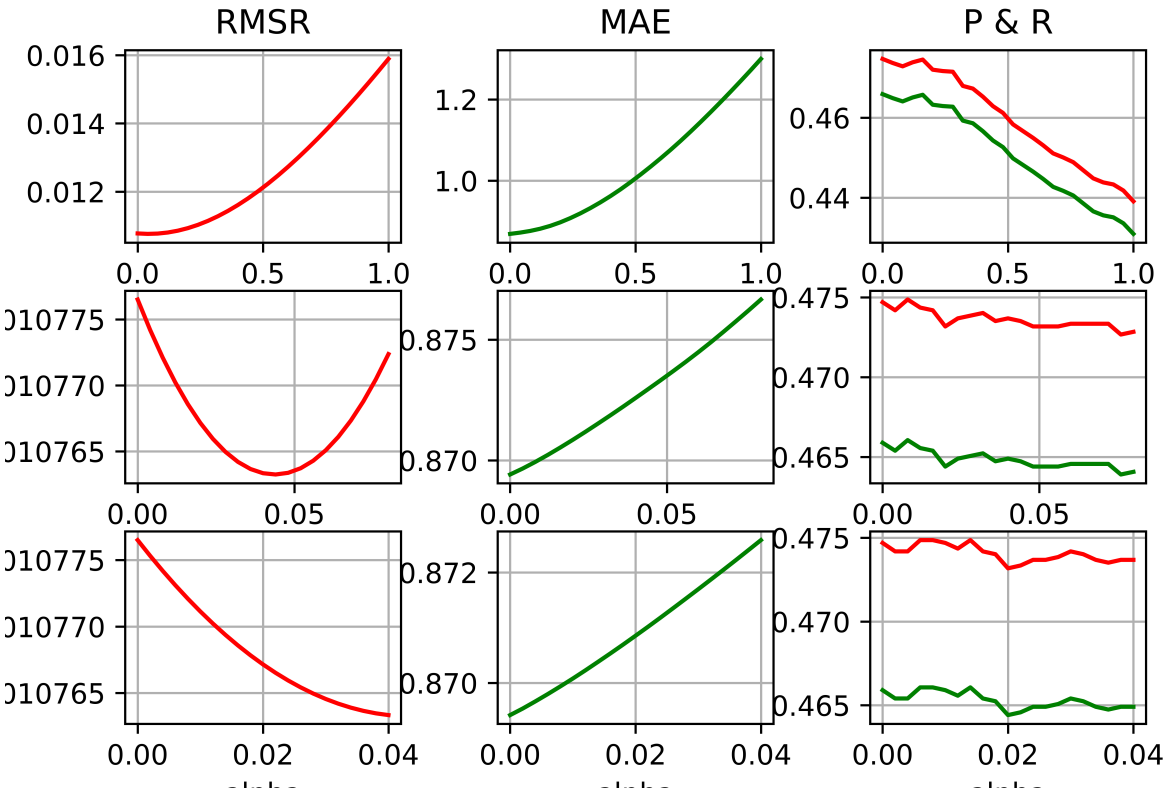
3. 测试

   初始随机选定$\alpha = 0.01$, 得到的结果如下

```
RMSE=0.01077113454270768
MAE=0.8700781457473302
Precision=0.46589403973509935
Recall=0.47469635627530365
```

   对比可以看到有很小的改进，但几乎可以忽略不计。

   上面使用了预先设定的参数α,于是理所当然需要测试不同的值对结果的影响。于是现在逐步缩小区间，得到一系列的值，选定的区间为[0,1]/[0,0.08]/[0,0.04]得到结果如下



   所以可以看到结果是取$\alpha = 0.05$结果会比较好。但事实上改进的很小。这也有可能是Collaborative Filtering方法已经发掘了比较多的信息，再使用基于流派的信息得到的额外信息并不是很多，也有另外的一种解释是：某个用户喜欢看某种流派的东西，并不代表他喜欢这种电影流派所有的东西，也有一些不好的电影会给很差的评分。所以如果需要取得更好的结果，就需要更加细致化的用户/流派/电影建模。这一点在下面的基于矩阵分解的方法中有一些涉及。

CB与CF方法的优缺点

1. CB

   Pros

   - 不需要使用其他用户的数据，没有cold-start或稀疏性的问题
   - 能够给用户推荐个性化的东西
   - 能够推荐比较新以及不是流行的东西。没有first-rater的问题
   - 能够提供一定的解释。

   Cons

- 找到合适的features很困难。比如images/movies/music
- 过于特性化。从来不推荐其他用户也喜欢的东西。一个用户也可能有多种兴趣。
- 对于新用户，没办法给出profile

2. CF

   Pros

   - 对于任何的item都适合，不用提前选定features

   Cons

   - Cold Start. 需要足够多的用户数据才能推荐。
   - 稀疏性问题。矩阵很稀疏的话，就很难找到相似的用户或者电影
   - First Rater. 不能去推荐那些还没有评分的电影
   - 容易推荐流行的东西而不是个性化的东西

调研-推荐系统的冷启动问题

1. 冷启动问题定义

> 推荐系统需要根据用户的历史行为和兴趣预测用户未来的行为和兴趣，对于BAT这类大公司来说，它们已经积累了大量的用户数据，不发愁。但是对于很多做纯粹推荐系统的网站或者很多在开始阶段就希望有个性化推荐应用的网站来说，如何在对用户一无所知（即没有用户行为数据）的情况下进行最有效的推荐呢？这就衍生了冷启动问题。

2. 冷启动的分类

冷启动问题主要分为3类：

- **用户冷启动**，即如何给新用户做个性化推荐
- **物品冷启动**，即如何将新的物品推荐给可能对它感兴趣的用户
- **系统冷启动**，即如何在一个新开发的网站（没有用户，没有用户行为，只有部分物品信息）上设计个性化推荐系统，从而在网站刚发布时就让用户体会到个性化推荐

3. 冷启动问题的解决方案

3.1 提供非个性化的推荐

最简单的例子就是提供热门排行榜，可以给用户推荐热门排行榜，等到用户数据收集到一定的时候，再切换为个性化推荐。Netflix的研究也表明新用户在冷启动阶段确实是更倾向于热门排行榜的，老用户会更加需要长尾推荐。

3.2 利用用户注册信息

用户的注册信息主要分为3种：

- 人口统计学信息，包括年龄、性别、职业、民族、学历和居住地
- 用户兴趣的描述，部分网站会让用户用文字来描述兴趣
- 从其他网站导入的用户站外行为，比如用户利用社交网站账号登录，就可以在获得用户授权的情况下导入用户在该社交网站的部分行为数据和社交网络数据。

这种个性化的粒度很粗，假设性别作为一个粒度来推荐，那么所有刚注册的女性看到的都是同样的结果，但是相对于男女不区分的方式，这种推荐精度已经大大提高了。

3.3 选择合适的物品启动用户的兴趣

用户在登录时对一些物品进行反馈，收集用户对这些物品的兴趣信息，然后给用户推荐那些和这些物品相似的物品。一般来说，能够用来启动用户兴趣的物品需要具有以下特点：

- 比较热门，如果要让用户对物品进行反馈，前提是用户得知道这是什么东西；
- 具有代表性和区分性，启动用户兴趣的物品不能是大众化或老少咸宜的，因为这样的物品对用户的兴趣没有区分性；
- 启动物品集合需要有多样性，在冷启动时，我们不知道用户的兴趣，而用户兴趣的可能性非常多，为了匹配多样的兴趣，我们需要提供具有很高覆盖率的启动物品集合，这些物品能覆盖几乎所有主流的用户兴趣。

3.4 利用物品的内容信息

物品冷启动问题在新闻网站等时效性很强的网站中非常重要，因为这些网站时时刻刻都有新物品加入，而且每个物品必须能够再第一时间展现给用户，否则经过一段时间后，物品的价值就大大降低了。

对UserCF算法来说，针对推荐列表并不是给用户展示内容的唯一列表（大多网站都是这样的）的网站。当新物品加入时，总会有用户通过某些途径看到，那么当一个用户对其产生反馈后，和他历史兴趣相似的用户的推荐列表中就有可能出现该物品，从而更多的人对该物品做出反馈，导致更多的人的推荐列表中出现该物品。

因此，该物品就能不断扩散开来，从而逐步展示到对它感兴趣用户的推荐列表中针对推荐列表是用户获取信息的主要途径（例如豆瓣网络电台）的网站UserCF算法就需要解决第一推动力的问题，即第一个用户从哪儿发现新物品。最简单的方法是将新的物品随机战士给用户，但是太不个性化。因此可以考虑利用物品的内容信息，将新物品先投放给曾经喜欢过和它内容相似的其他物品的用户。

对ItemCF算法来说，物品冷启动就是很严重的问题了。因为该算法的基础是通过用户对物品产生的行为来计算物品之间的相似度，当新物品还未展示给用户时，用户就无法产生行为。为此，只能利用物品的内容信息计算物品的相关程度。基本思路就是将物品转换成关键词向量，通过计算向量之间的相似度（例如计算余弦相似度），得到物品的相关程度。

3.5 采用专家标注

很多系统在建立的时候，既没有用户的行为数据，也没有充足的物品内容信息来计算物品相似度。这种情况下，很多系统都利用专家进行标注。

代表系统：个性化网络电台Pandora、电影推荐网站Jinni。

以Pandora电台为例，Pandora雇用了一批音乐人对几万名歌手的歌曲进行各个维度的标注，最终选定了400多个特征。每首歌都可以标识为一个400维的向量，然后通过常见的向量相似度算法计算出歌曲的相似度。

Extended Methods

1. 原理

   这里使用的方法是带正则化的评分矩分解。也就是

   $$R_{m\times n} \approx P_{m\times k} \times Q_{k\times n} = \hat{R}_{m\times n}$$

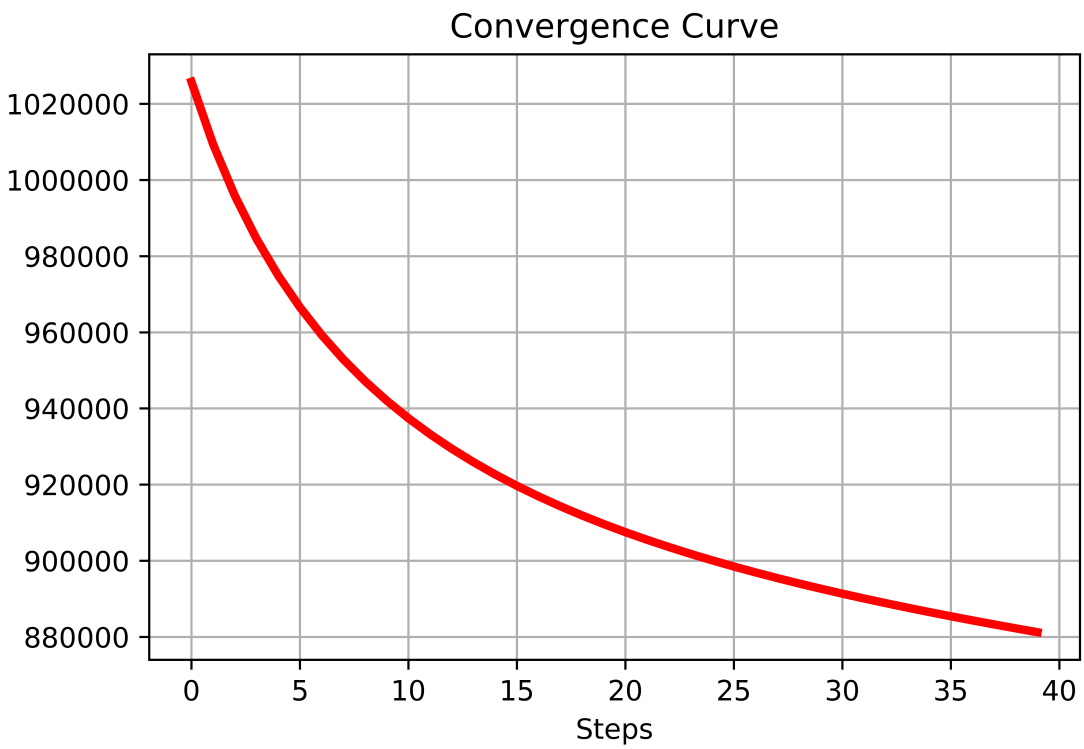
   应用到评分矩阵上，可以解释为，$P_{m\times k}$为用户m对流派k的喜爱值，$Q_{k\times n}$为流派k对于电影n的推荐系数。然后使用一系列的梯度下降法得到迭代的式子，同时为了防止过拟合加上正则项。最终得到的矩阵分解的代码为

   ```
   def matrix_factorization(R,P,Q,K,steps=5000,alpha=0.0002,beta=0.02):
       Q=Q.T  # .T操作表示矩阵的转置
       result=[]
       for step in range(steps):
           print(step)
           for i in range(len(R)):
               for j in range(len(R[i])):
                   if R[i][j]>0:
   ```

```
                    eij=R[i][j]-np.dot(P[i,:],Q[:,j]) # .dot(P,Q) 表示矩阵内积
                    for k in range(K):
                        P[i][k]=P[i][k]+alpha*(2*eij*Q[k][j]-beta*P[i][k])
                        Q[k][j]=Q[k][j]+alpha*(2*eij*P[i][k]-beta*Q[k][j])
        eR=np.dot(P,Q)
        e=0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j]>0:
                    e=e+pow(R[i][j]-np.dot(P[i,:],Q[:,j]),2)
                    for k in range(K):
                        e=e+(beta/2)*(pow(P[i][k],2)+pow(Q[k][j],2))
        result.append(e)
        if e<0.001:
            break
    return P,Q.T,result
```

2. 测试

由于k的强烈含义，这里设置$k = 18$, 同时由于迭代的速度实在是太慢了，大概一分钟一轮迭代，所以设置迭代次数为40, 同时得到loss的收敛曲线为



可以看到接近收敛了，但是没有完全。

得到的结果如下

```
RMSE=0.009260923901617401
MAE=0.7447313937537074
Precision=0.4995033112582781
Recall=0.5089406207827261
```

对比CF的结果

```
# RMSE 0.010776504401721263
# MAE 0.8694166710904666
# P 0.46589403973509935
# R 0.47469635627530365
```

可以看到还是有比较明显的提高的，特别是在Precision(同比增长7.21%),Recall(7.21%)

需要注意的是，这里仅仅是欠拟合的结果，考虑到时间问题，没有再继续运行，但是可以猜测，完全收敛后得到的解肯定有更大的提高。