推荐系统

PB16050176 冉礼豪
PB16111485 张劲暾
PB16050193 胡煜霄

实验内容：

实现推荐系统：采用经典的推荐系统数据集：MovieLens 1M Dataset。根据 ratings.dat 建立评分矩阵，分别使用 Content-based Methods 和 Collaborative Filtering Methods 来补全矩阵。在实验的过程中，也可以使用 users.dat 和 movies.dat 中的数据来提高模型的性能。
然后获取测试集上的预测结果，分别使用 RMSE，MAE，Precision，Var 作为标准评估模型再训练集上的性能。

Content-based 部分

数据预处理：

首先读取 movies.dat 中的内容，其按照 MovieID::Title::Genres 的格式组织，其中，Genres 使用对应的英文单词，并用 ‘ | ’ 分隔，分为 Action，Adventure 等共 18 个种类，由于标签是二元的，所以我们采用位储存的方法，为每部电影分配一个整数，其中前十八位每一位对应一个标签，对应位是 1 的时候表示此电影有此标签，否则为 0。建立了电影属性表后，使用此表建立电影之间的 Jaccard 相关性矩阵即共同具有的标签数和分别具有的标签数的比值。

然后读取 rating.dat 中的内容。这是用户对电影的评分信息，按照 UserID::MovieID::Rating::Timestamp 的格式存储，在本模型中只使用了前三个属性，即
userID，movieID 和 rating，在读取的过程中，建立以用户 id 为纵坐标，电影 id 为横坐标的评分矩阵，将数据中的百分之十填入其中，将剩下百分之十存入测试数据集合。以上是最后使用的模型的建立过程

在之前的尝试中，还建立了用户对每个标签的喜好的矩阵，并将电影标签与对应用户的标签喜好进行比较进而得到预测评分。

算法流程：

建立矩阵 genresMatrix 用于存放电影和对应的标签，采用位运算方式
使用 genresMatrix 建立矩阵 movieJaccard 矩阵：

```
For i from 1 to num of movie
    For j from i to num of movie
        m = number of genres i and j both have
        n = number of genres i or j has
        movieJaccard[i][j] = m / n
```

由于训练用数据是稀疏的，所以用 list 仿照邻接链表的方式重新以邻接矩阵的方式建立 rating 的矩阵以提高预测时的运算速度

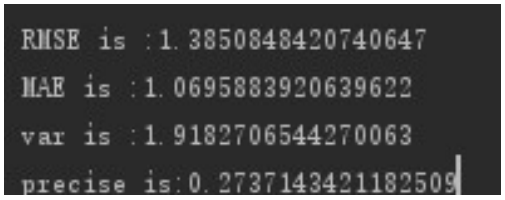
在处理完基本数据后，得到了各个电影之间的相似度，用户对部分电影的评分，这时便可以对测试集进行预测。

```
for each p in testset:
    for each movie m this user rated:
        totalWeight += movieJaccard[p.movieID][m]
        result += movieJaccard[p.movieID][m] * p.rating
    result = result / totalWeight
```

这样便得到了每个测试集结果的预测值，将结果存入 result 文件。

评估结果：

调用评估代码进行评估，得到评估结果截图如下：



Collaboration Filtering Method 部分

算法描述：

```
rating,test,movies = getdata(filename)          #movies 是从 movies.dat 中提取的关于电影题材的信息向量，在计算 item 相似度时不仅用到 rating.dat,还用到 movies.dat 里的信息
rating,test = preprocessing(rating,test)    #预处理
sim_matrix = get_sim_matrix(rating,movies)    #获得相似度矩阵
rating = fullfill_matrix(rating,sim_matrix) #利用相似度矩阵填充 rating
evaluate(test,rating)                       #评估
```

由于 get_sim_matrix 函数涉及矩阵运算，不便描述，故展示代码如下：

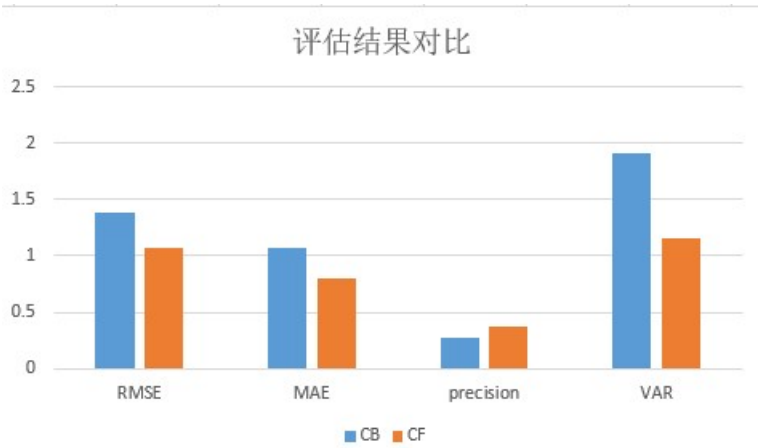
```
def get_sim(i,j,data,movies):
    items = data[:, [i, j]]
    del_inds = np.where(items == 0)[0]
    items = np.delete(items, del_inds, axis=0)
    x = items[:,0]
    y = items[:,1]
    s = time[:,[i,j]]
    dele = np.where(s == 0)[0]
    s = np.delete(s,dele,axis=0)
    tx = s[:,0]
    ty = s[:,1]
    if s.size == 0:
        sim = 0
    if items.size == 0:
        sim = 0
    else:
        sim = ((x @ y)/(np.linalg.norm(x,2) * np.linalg.norm(y,2)))
        sim = sim * (tx @ ty)/(np.linalg.norm(tx,2) * np.linalg.norm(ty,2))
    if i in movies:
        x = np.array(movies[i])
    else:
        sim = 0
        return sim
    if j in movies:
        y = np.array(movies[j])
    else:
        sim = 0
        return sim
    sim = sim * (x @ y)/(np.linalg.norm(x,2) * np.linalg.norm(y,2))
    return sim

def get_sim_matri(data,movies):
    sim_matri = np.zeros((3953,3953))
    v=0
    print("getting simi mat!")
    for i in range(1,3953):
        for j in range(i+1,3953):
            sim_matri[i,j] = get_sim(i,j,data,movies)
            sim_matri[j,i] = sim_matri[i,j]
            v+=1
            if(v==1953276):
                print("25%")
            if(v==3906552):
                print("50%")
    print("done!")
    print("file writing!")
    with open("sim_matrix.dat","w") as w:
        for i in range(3953):
            for j in range(3953):
                w.write(str(sim_matri[i,j])+",")
            w.write("\n")
    print("writing done!")
    return sim_matri
```

评估结果：

```
done.
MAE:            0.7940396635913186
precision:      0.3681139598405138
RMSE:           1.0743667498140819
VAR:            1.1542592970290297
```

结果对比：



可见，在几种评估标准下，Collaboration Filtering Method 的性能均高于 Content-based。下面对比两种方法的优缺点：

Content-based：不需要来自其他用户的数据，可以对针对某个用户的喜好特别进行推荐，可以推荐并不那么流行的内容，可以对推荐理由进行解释。但是有时候比较难以获取推荐内容的特征，且难以理解这些特征之间的关系，同时，它过于个性化，无法推荐和用户喜好相差较大的内容，且难以正确的向新用户进行推荐。

Collaboration Filtering Method：协同过滤没有提取内容特征的必要，但是也有慢启动的特点，且计算耗时长。容易推荐流行内容，对不流行内容几乎不会推荐。

调研与改进：

一、 问题分析

之前实现的推荐系统没有考虑到用户推荐电影的时间,对于推荐效果的影响,显然,最近推荐较多的电影更加热门,其他人推荐的概率也更高。由此我们引入时序多样性。

推荐系统每天推荐结果的变化程度被定义为推荐系统的时间多样性。时间多样性高的推荐系统中用户会经常看到不同的推荐结果。提高推荐结果的时间多样性需要分两步解决:首先,需要保证推荐系统能够在用户有了新的行为后及时调整推荐结果,使推荐结果满足用户最近的兴趣;其次,需要保证推荐系统在用户没有新的行为时也能够经常变化一下结果,具有一定的时间多样性。

给推荐系统提供的信息是有时间效应的,时间信息对用户兴趣的影响表现在以下几个方面:用户兴趣是变化的;物品也是有生命周期和季节效应的。在给定时间信息后,

推荐系统从一个静态系统变成了一个时变的系统，而用户行为数据也变成了时间序列。包含时间信息的用户行为数据集由一系列三元组构成，其中每个三元组（u,I,t）代表了用户 u 在时刻 t 对物品 I 产生过行为。

实现推荐系统的实时性除了对用户行为的存取有实时性要求，还要求推荐算法本身具有实时性，而推荐算法本身的实时性意味着：

（1）实时推荐系统不能每天都给所有用户离线计算推荐结果，然后在线展示昨天计算出来的结果。所以，要求在每个用户访问推荐系统时，都根据用户这个时间点前的行为实时计算推荐列表。

（2）推荐算法需要平衡考虑用户的近期行为和长期行为，即要让推荐列表反应出用户近期行为所体现的兴趣变化，又不能让推荐列表完全受用户近期行为的影响，要保证推荐列表对用户兴趣预测的延续性。

二、 实现方法——基于时间上下文的 itemCF 算法

基本的 itemCF 是先计算物品之间的相似度：

$$sim(i,j)=\frac{\sum_{u\in N(i)\cap N(j)}1}{\sqrt{|N(i)||N(j)|}}$$

然后利用公式：

$$p(u,i)=\sum_{j\in N(u)\cap S(i,K)}sim(i,j)$$

来计算用户 u 对物品 i 的兴趣，从而补全 rating 矩阵

然而，在得到时间信息后（用户对物品产生行为的时间）后，我们可以通过如下公式改进相似度计算和修正预测公式：

$$sim(i,j)=\frac{\sum_{u\in N(i)\cap N(j)}f(|t_{ui}-t_{uj}|)}{\sqrt{|N(i)||N(j)|}}\qquad f(|t_{ui}-t_{uj}|)=\frac{1}{1+\alpha|t_{ui}-t_{uj}|}$$

$$p(u,i)=\sum_{j\in N(u)\cap S(i,K)}sim(i,j)\frac{1}{1+\beta|t_0-t_{uj}|}$$

在计算物品间相似度 sim（I，j）中引入了和时间有关的衰减项 f（|t（ui）- t（uj）|），其中 t（ui）是用户 u 对物品 I 产生行为的时间。f 函数的含义是，用户对物品 I 和物品 j 产生行为的时间越远，则 f 函数值越小。alpha 是时间衰减参数，它的取值在不同系统中不同。如果一个系统用户兴趣变化很快，就应该取比较大的 alpha，反之需要取比较小的 alpha。P（u，I）中，t0 是当前时间,公式表明，t（uj）越靠近 t0，和物品 j 相似的物品就会在用户 u 的推荐列表中获得越高的排名。beta 是时间衰减参数，需要根据不同的数据集选择合适的值。

三、 具体实现

首先在从文件中读取信息的时候，建立 time 矩阵，储存时间戳，time[u , l] 代表用户 u 对 l 打分的时间。然后在计算物品间相似度时考虑上时间因素：

```
s = time[:,[i,j]]
dele = np.where(s == 0)[0]
s = np.delete(s,dele,axis=0)
tx = s[:,0]
ty = s[:,1]


sim = sim * (tx @ ty)/(np.linalg.norm(tx,2) * np.linalg.norm(ty,2))
```

还要建立物品 i，j 时间上影响度矩阵：

```
C = dict()
N = dict()
for u, items in train.items():
    for i,tui in items.items():
        N[i] += 1
        for j,tuj in items.items():
            if i == j:
                continue
            C[i][j] += 1 / (1 + alpha * abs(tui - tuj))
W = dict()
for i,related_items in C.items():
    for j, cij in related_items.items():
        W[u] = cij / math.sqrt(N[i] * N[j])
return W
```

在计算物品相似度时，利用物品时间影响力矩阵，对两物品不同用户对其评分的相似度进行加权计算，以表现时间多样性。

除此之外，还要考虑时新性，离当前时间越近的电影越新，越有可能推荐。

四、 分析

测试结果:

```
MAE:            0.7651251215852
precision:      0.37651252525545
RMSE:           1.0554585524423
VAR:            1.112545565455
```

在与基础版本的相同的训练集上训练，相同的测试集上测试，结果稍好于基础版本，但性能提升的效果很有限，考虑原因可能与时间衰减参数有关，由于对时间参数没有做过多调研，想当然的选择了一个较小的数，首先，这个数值不一定合适，其次，这个数值不会随着不同用户而改变，是一个定值，因此使得基于上下文的 itemCF 算法性能优化很有限，