西安電子科技大學
XIDIAN UNIVERSITY

数据挖掘第一次上机实验报告

基于图论与矩阵分解的协同过滤推荐方法对比

姓名：刘臻劼

学号：19030700016

班级：1903071

导师：郭杏莉

目录

1 实验内容

1.1 实验目标

本次实验目标为：在 MovieLens 数据集上构建“用户——电影”评分矩阵，基于评分矩阵对用户与电影进行推荐。

1.2 数据集介绍

MovieLens 数据集包含 6040 个用户关于 3883 部电影的 1000209 条评分信息。数据集分为三个文件：用户数据 users.dat、电影数据 movies.dat 和评分数据 ratings.dat。

（1）用户数据

关于 6040 个用户的个人脱敏数据，格式为

$$UserID :: Gender :: Age :: Occupation :: Zip-code$$

其中：

$UserID$：1 到 6040 顺序编号；

$Gender$：F 为女性，M 为男性；

Age：分为 1,18,25,35,45,50,56 共 7 类，代表用户所属的年龄段；

$Occupation$：0 到 20 的整数，分别代表 21 种职业；

$Zip-code$：邮编。

（2）电影数据

关于 3883 部电影的信息，格式为

$$MovieID :: Title :: Genres$$

其中：

$MovieID$：1 到 3952 无重复的整数，非连续编号；

$Title$：电影名，为电影名与年份的拼接，如“Toy Story (1995)”；

$Genres$：电影所属的类别，如果同属多个类别，则类名间用“|”隔开，如：“Adventure|Children’s”。

（3）评分数据

用户对电影的 1000209 条评分数据，格式为

$$UserID :: MovieID :: Rating :: Timestamp$$

其中：

$UserID$, $MovieID$：同上；

$Rating$：1 到 5 之间（含）的整数；

$Timestamp$：自 1970 年 1 月 1 日零点后到用户提交评价的时间的秒数。

2 实验分析

2.1 题目分析

原问题包含两个要求，一是对题给的用户——电影打分数据集进行建模。再是完善评分矩阵并进行推荐。

（1）用户——电影打分建模

我们考虑用户集 $U = \{u_1, u_2, \cdots u_n\}, n = 6040$，电影集 $V = \{v_1, v_2, \cdots, v_m\}, m = 3883$。则用户对电影的评分可视作以 U, V 构建的二部图 $G =< U, V, E >$ 相应边的权重，其中边集 $E \subseteq U \times V$，则第 i 个用户对第 j 部电影的评分 $r_{ij} = h(u_i, v_j)$, $h : E \to [0, 5]$，此处评分函数的值域为 $[0, 5]$，将原始任务中的多类分类问题转化为了回归问题，这是考虑到评分允许浮点更加自然。

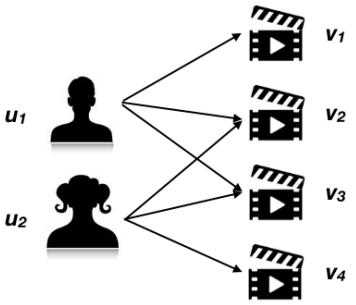

图 1: 二部图模型

（2）评分矩阵

在（1）我们知道边集 $E \subseteq U \times V$，而实际上 $|E| \ll |U| \cdot |V|$，这说明全体用户对所有电影的评价矩阵 $\mathbf{R} = (r_{ij})_{m \times n}$ 是较稀疏的。而本实验的核心问题则是如何通过现有数据补全评价矩阵 $\mathbf{R}$。补全评分矩阵后我们将基于此进行推荐。

2.2 推荐算法简介

2.2.1 推荐算法的分类

所谓推荐算法，即是根据用户行为推测其喜好的一类算法。推荐算法历史悠久，在机器学习还没有兴起的时候就有需求和应用了。概括来说，可以分为以下 5 种[1]：

（1）基于内容的推荐：这一类一般依赖于 NLP 方法，通过挖掘文本的 TF-IDF 特征向量，来得到用户的偏好，进而做推荐。

（2）协调过滤推荐：协调过滤是推荐算法中目前最主流的种类，花样繁多，目前绝大多数实际应用的推荐算法都是协同过滤推荐算法。

（3）混合推荐：类似机器学习中的集成学习，博才众长，通过多个推荐算法的结合，得到一个更好的推荐算法。

（4）基于规则的推荐：这类算法常见的比如基于最多用户点击，最多用户浏览等，属于大众型的推荐方法，在目前的大数据时代并不主流。

（5）基于知识的推荐：在某种程度是可以看成是一种推理技术，它不是建立在用户需要和偏好基础上推荐的。

2.2.2 协同过滤推荐算法

我们接下来重点讨论协同过滤的推荐算法。协同过滤分析用户兴趣，在用户群中找到指定用户的相似（兴趣）用户，综合这些相似用户对某一信息的评价，形成对该指定用户对此信息的喜好程度预测。

协同过滤问题一般这样呈现：只有部分用户和部分对象之间是有评分数据的，其它部分评分是空白，此时我们要用已有的部分稀疏数据来预测那些空白的物品和数据之间的评分关系，找到最高评分的物品推荐给用户。很明显本实验属于一协同过滤问题。

2.3 模型设计

明确题意后，我们来讨论解决方法。针对本题考虑使用基于图论的方法与基于矩阵分解及深度学习方法。

2.3.1 图论方法

由 2.1 节，我们将用户——电影建模为二部图，我们可以考虑使用基于图的 SimRank[2] 算法。其核心思想是[3]：如果两个用户相似，则与这两个用户相关联的物品也类似；如果两个物品类似，则与这两个物品相关联的用户也类似。

（1）相似度矩阵

我们期望构建两个相似度矩阵 $\mathbf{X}_{m\times m}, \mathbf{Y}_{n\times n}$。其中 $x_{ij}=s(u_i,u_j)$，$y_{i'j'}=s(v_{i'},v_{j'})$。

考虑我们在 2.1 节中的建模：$G=<U,V,E>$，则用户 $u_i,\ u_j,\ (i,j=1,2,\cdots,n)$ 的相似度为：

$$s(u_i,u_j)=\frac{C}{|V(u_i)||V(u_j)|}\cdot\sum_{p=1}^{|V(u_i)|}\sum_{q=1}^{|V(u_j)|}s(V_p(u_i),V_q(u_j)) \tag{1}$$

其中：C 为阻尼系数；$V(u_i)$ 为用户 u_i 点评过的电影集合，即 V 中与结点 u_i 相连的点的集合，$V(u_j)$ 同理；$s(V_p(u_i),V_q(u_j))$ 为 $V(u_i)$ 中第 p 部电影与 $V(u_j)$ 中第 q 部电影的相似度。

对于电影 $v_{i'},\ v_{j'}\ (i',j'=1,2,\cdots,m)$ 的相似度我们同样有：

$$s(v_{i'},v_{j'})=\frac{C}{|U(v_{i'})||U(v_{j'})|}\cdot\sum_{p=1}^{|U(v_{i'})|}\sum_{q=1}^{|U(v_{j'})|}s(U_p(v_{i'}),U_q(v_{j'})) \tag{2}$$

从中我们可以看到两个体 a,b 间的相似度取决于与他们相连的所有结点间的相似度。

考虑特殊情况：我们定义自身与自身的相似度为 1，且若一用户没有评论任何电影，或一部电影没有任何用户评论，则该用户/电影与其他用户/电影的相似度为 0。

（2）选择矩阵与评分矩阵

在求解相似度矩阵前，有必要对几个概念进行阐述。

选择矩阵 $\mathbf{C}=(c_{ij})_{m\times n}$：刻画用户 u_i 是否对电影 v_j 有评分。即：

$$c_{ij}=\begin{cases}1 & <u_i,v_j>\in E\\ 0 & otherwise\end{cases} \tag{3}$$

评分矩阵 $\mathbf{R}=(r_{ij})_{m\times n}$：刻画用户 u_i 是否对电影 v_j 的评分值。即：

$$r_{ij}=\begin{cases} rating(u_i,v_j) & <u_i,v_j>\in E \\ 0 & otherwise \end{cases} \tag{4}$$

（3）相似度矩阵的求解

我们以用户间相似度为例，求解 $\mathbf{X}_{m\times m}$。考虑迭代求解，则式（1）可写成：

$$s_{k+1}(u_i,u_j)=\frac{C}{|V(u_i)||V(u_j)|}\cdot\sum_{p=1}^{|V(u_i)|}\sum_{q=1}^{|V(u_j)|}s_k(V_p(u_i),V_q(u_j)) \tag{5}$$

其中 k 为迭代轮次数。我们有：

$$s_{k+1}(u_i,u_j)=\frac{C}{|V(u_i)||V(u_j)|}\cdot\sum_{p=1}^{|V|}\sum_{q=1}^{|V|}c_{ip}\cdot c_{jq}\cdot s(v_p,v_q) \tag{6}$$

$$=C\sum_{p=1}^{|V|}\sum_{q=1}^{|V|}(\frac{c_{ip}}{\sum_{e_1=1}^{n}c_{ie_1}})s(v_p,v_q)(\frac{c_{jq}}{\sum_{e_2=1}^{n}c_{je_2}}) \tag{7}$$

注意到 $s(v_p,v_q)=y_{pq}$，所以（7）式可写成矩阵的形式：

$$\mathbf{X}_{k+1}=C\cdot\mathbf{C}^{row}\cdot\mathbf{Y_k}\cdot\mathbf{C}^{row\ T}+\mathbf{E}_{m\times m}-\Lambda(C\cdot\mathbf{C}^{row}\cdot\mathbf{Y_k}\cdot\mathbf{C}^{row\ T}) \tag{8}$$

其中：C 为阻尼系数；$\mathbf{C}^{row}$ 为选择矩阵按行归一化；$\mathbf{E}_{m\times m}$ 为 $m\times m$ 的单位矩阵；Λ 为只保留矩阵的对角元素而其他元素置 0。公式后半部分的 $\mathbf{E}_{m\times m}-\Lambda(C\cdot\mathbf{C}^{row}\cdot\mathbf{Y_k}\cdot\mathbf{C}^{row\ T})$ 为将矩阵主对角线元素置 1，这是考虑到任意用户与自身的相似度为 1。

同理，电影相似度矩阵也有迭代公式：

$$\mathbf{Y}_{k+1}=C\cdot\mathbf{C}^{row\ T}\cdot\mathbf{X_k}\cdot\mathbf{C}^{row}+\mathbf{E}_{n\times n}-\Lambda(C\cdot\mathbf{C}^{row\ T}\cdot\mathbf{X_k}\cdot\mathbf{C}^{row}) \tag{9}$$

（4）评分矩阵的补全

根据（8）、（9）式经过多次迭代基本稳定后我们即得到了 $\mathbf{X},\mathbf{Y}$，接下来我们基于此补全评分矩阵 $\mathbf{R}_{m\times n}$。为了充分利用已知信息，对于 $r_{ij},<u_i,v_j>\notin E$，同时使用用户 u_i 与电影 v_j 的所有评分。于是有：

$$r_{ij}=\frac{\sum_{p=1}^{n}c_{ip}\cdot r_{ip}\cdot s(v_p,v_j)+\sum_{q=1}^{m}c_{qj}\cdot r_{qj}\cdot s(u_q,u_i)}{\sum_{e_1=1}^{n}c_{ie_1}+\sum_{e_2=1}^{m}c_{e_2j}} \tag{10}$$

2.3.2 矩阵分解结合深度学习

协同过滤中另一经典算法为矩阵分解。我们希望评分矩阵有如下分解：

$$\mathbf{R}_{m\times n}=\mathbf{P}_{m\times k}\mathbf{Q}_{k\times n} \tag{11}$$

而在此处我们不使用传统矩阵分解方法来求解矩阵 $\mathbf{P}$ 与 $\mathbf{Q}$，而是通过深度学习方法得到。

传统方法以及上文提到的图论方法仅利用了用户对电影的评分数据，其他诸如用户性别，电影名及类别等等信息则没有利用。通过深度学习方法可以更充分地利用全部信息。对于式（1），我们可以认为 $\mathbf{P}$ 为用户特征矩阵，$\mathbf{Q}$ 为电影特征矩阵。

（1）模型概览模型的概览图如下：

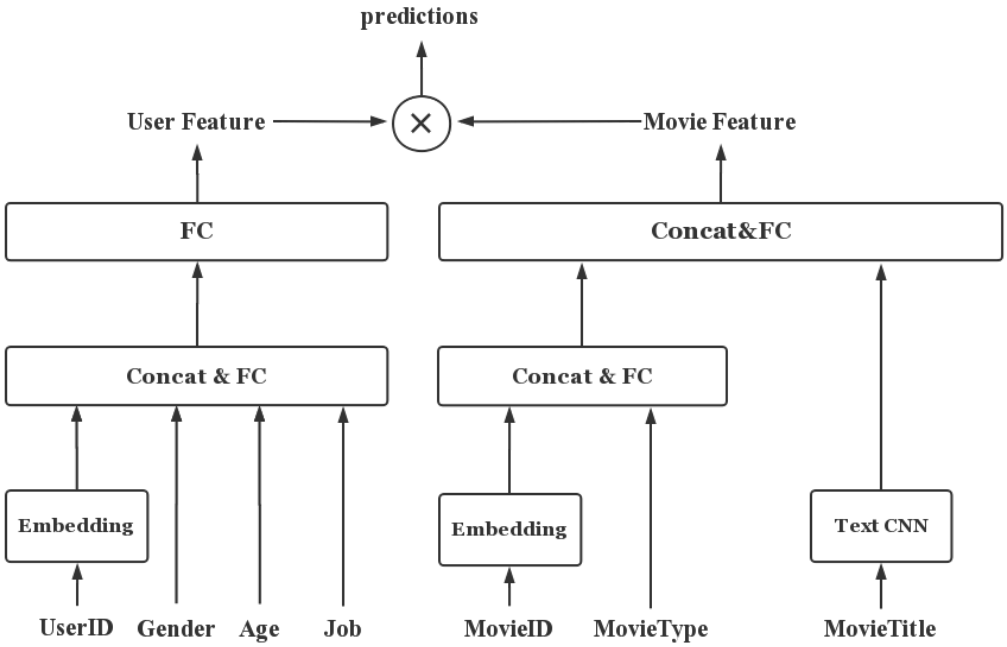

图 2: 模型示意图

（2）用户特征提取

用户信息中，对于 $Gender$, Age, Job 三个字段的信息，由于类别数较少可以采用 One-Hot 编码。而 $UserID$ 由于类别太多故而此处采用 Embedding 方法，用 $embedding\_dim = 32$ 维向量表示用户的 ID 值。

将 $UserID$ 的 Embedding 向量与其他信息 One-Hot 向量拼接并通过全连接层并通过 ReLU 激活，而后再次通过全连接层并进行批归一化。至此得到了用户信息的特征矩阵。

（3）电影特征提取

对于电影信息中 $MovieID$ 与 $MovieType$ 字段，类似用户信息的处理。ID 通过 Embedding 得到，电影类型为 Multi-Hot 向量。

而对于电影名称，此处采用 CNN 提取特征。CNN 在 NLP 的应用最早来源于 Kim 在 2014 年的工作[4]。相比 CV 领域，TextCNN 结构较为简单，一般仅有词嵌入、卷积、池化与全连接四部分。而本文用其提取电影文本特征，仅包含词嵌入、卷积与池化三个部分。最后再将电影名称信息与电影信息拼接通过全连接层，得到电影信息的特征矩阵。

（4）拟合评分上述步骤完成后，我们得到了用户特征矩阵 $\mathbf{P} \in \mathbb{R}^{m \times hidden\_size}$ 与电影特征矩阵 $\mathbf{Q} \in \mathbb{R}^{n \times hidden\_size}$。则评分矩阵为：

$$\mathbf{R}_{m \times n} = \mathbf{P}_{m \times h} \mathbf{Q}_{n \times h}^{T} \tag{12}$$

其中 h 为隐藏层维度。之后通过计算与目标值的 MSE 误差得到梯度并反向传播进行优化。

3 具体实现

3.1 初步数据预处理

首先读取并加载数据：

```
# 加载电影数据
movie_names = ['movie_id', 'movie_title', 'movie_type']
movie = pd.read_table('movies.dat', sep='::', header=None,
                      names=movie_names, engine='python')
# 加载用户数据
user_names = ['user_id', 'user_gender', 'user_age', 'user_job', 'zip']
user = pd.read_table('users.dat', sep='::', header=None,
                     names=user_names, engine='python')
# 加载评分数据
rate_names = ['user_id', 'movie_id', 'rank', 'timestamp']
rating = pd.read_table('ratings.dat', sep='::',
                       header=None, names=rate_names, engine='python')
```

而后剔除无关列：邮编 $Zip-code$ 与时间戳 $Timestamp$ 是我们不需要的。

```
user = user.drop(['zip'], axis=1)
rating = rating.drop(['timestamp'], axis=1)
```

再然后处理用户数据，将 $Gender$，Age，Job 转换为 One-Hot 向量。

```
# User 相关数据处理
user['user_gender'] = user['user_gender'].apply(lambda x: [1, 0] if x
    == 'F' else [0, 1])

def convert_age_to_One_Hot(age):
    if age == 1:
        return [1, 0, 0, 0, 0, 0, 0]
    elif age == 18:
        return [0, 1, 0, 0, 0, 0, 0]
    elif age == 25:
        return [0, 0, 1, 0, 0, 0, 0]
    elif age == 35:
        return [0, 0, 0, 1, 0, 0, 0]
    elif age == 45:
```

```
        return [0, 0, 0, 0, 1, 0, 0]
    elif age == 50:
        return [0, 0, 0, 0, 0, 1, 0]
    else:
        return [0, 0, 0, 0, 0, 0, 1]

def convert_job_to_One_Hot(job):
    jobs = [0] * 21
    jobs[job] += 1
    return jobs

user['user_age'] = user['user_age'].apply(convert_age_to_One_Hot)
user['user_job'] = user['user_job'].apply(convert_job_to_One_Hot)
```

接下来处理电影数据。将电影类别转换为 Multi-Hot 向量，而后构建电影标题文本词典，将其索引化。

```
# Movie 相关数据处理
# 电影名称索引化
args.max_length = 16
movie_title_word2id = {'pad': 0}
for i in range(len(movie['movie_title'])):
    words = movie['movie_title'][i].split(' ')
    del words[-1]  # 去除年份
    movie_title_id = []
    for word in words:
        if word not in movie_title_word2id:
            movie_title_word2id[word] = len(movie_title_word2id)
        movie_title_id.append(movie_title_word2id[word])
    movie_title_id.extend([0] * (args.max_length - len(words)))  # 填充
    movie['movie_title'].loc[i] = movie_title_id
args.vocabulary_size = len(movie_title_word2id)

# 电影类型
for i in range(len(movie['movie_type'])):
    types = movie['movie_type'][i].split('|')
```

```
    type_id = []
    for j in range(len(movie_types)):
        if movie_types[j] in types:
            type_id.append(1)
        else:
            type_id.append(0)
    movie['movie_type'].loc[i] = type_id
```

之后融合三个 DataFrame 中的数据作为初步预处理的结果。

```
tmp = pd.merge(rating, user)
data = pd.merge(tmp, movie)
```

3.2 数据集探索与划分

我们注意到一共有 3883 部电影但有用户评论的电影数仅 3706 部，亦即有 177 部电影无用户评论。在本题的二分图建模中，这些电影对应结点为孤立节点，此处将其删去。故而 n 值改变为 3706。

```
movies = {}
users = {}
for i in range(len(data)):
    sample = data.iloc[i]
    if sample['user_id'] not in users.keys():
        users[sample['user_id']] = {'uid': sample['user_id']}
    if sample['movie_id'] not in movies.keys():
        movies[sample['movie_id']] = {'mid': sample['movie_id']}
mid_rated = movies.keys() # 3706
total_mid = movie['movie_id'].values # 3883
mid_not_rated = [x for x in total_mid if x not in mid_rated] # 177
```

再从中抽取 10000 条数据用于模型测试并根据训练集构建选择矩阵 $\mathbf{C}_{6040\times3706}$ 与初始评分矩阵 $\mathbf{R}_{6040\times3706}$。

```
choice_matrix = pd.DataFrame(np.zeros([6040, 3706], dtype=float))
rank_matrix_initial = pd.DataFrame(np.zeros([6040, 3706], dtype=float))

for i in range(len(train)):
    sample = train.iloc[i]
    uid = sample['user_id']
    mid = sample['movie_id']
```

```
    rank = sample['rank']
    user_index = user_index_to_uid.index(uid)
    movie_index = movie_index_to_mid.index(mid)
    choice_matrix[movie_index][user_index] = 1
    rank_matrix_initial[movie_index][user_index] = rank

pkl.dump(choice_matrix, open('choice_matrix.pkl', 'wb'))
pkl.dump(rank_matrix_initial, open('rank_matrix_initial.pkl', 'wb'))
```

3.3 矩阵分解结合深度学习方法

构建学习用户特征矩阵与电影特征矩阵的深度学习模型。

```
class MovieLens(nn.Module):
    def __init__(self, user_max_dict, movie_max_dict, args):
        super(MovieLens, self).__init__()
        self.args = args
        # ------------ user channel ------------
        # user embeddings
        self.embedding_uid = nn.Embedding(user_max_dict['uid'], args.
            embedding_dim, device=args.device)

        # user info NN
        self.user_layer = nn.Sequential(
            nn.Dropout(args.dropout),
            nn.Linear(
                args.embedding_dim + user_max_dict['gender'] +
                    user_max_dict['age'] + user_max_dict['job'],
                args.hidden_dim, device=args.device),
            nn.ReLU(),

            nn.Linear(args.hidden_dim, args.hidden_dim, device=args.
                device),
            nn.Tanh(),
            nn.BatchNorm1d(args.hidden_dim, device=args.device)
        )

```

```
22          # ------------ movie channel ------------
23          # movie embeddings
24          self.embedding_mid = nn.Embedding(movie_max_dict['mid'], args.
                embedding_dim, device=args.device)  # normally 32
25
26          # movie info NN
27          self.movie_layer = nn.Sequential(
28              nn.Linear(args.embedding_dim + movie_max_dict['mtype'],
                    args.hidden_dim, device=args.device),
29              nn.ReLU(),
30              nn.BatchNorm1d(args.hidden_dim, device=args.device)
31          )
32
33          # movie title text
34          self.text_embedding = get_movie_text_embedding(args)
35          self.convs = nn.ModuleList(
36              [nn.Conv2d(1, args.hidden_dim, (size, args.embedding_dim),
                    device=args.device) for size in
37               args.filter_sizes])
38          self.dropout = nn.Dropout(args.dropout)
39
40          self.movie_combine_layer = nn.Sequential(
41              nn.Linear(args.hidden_dim + len(args.filter_sizes) * args.
                    hidden_dim, args.hidden_dim, device=args.device),
42              nn.Tanh()
43          )
44
45      def forward(self, user_input, movie_input):
46          # ------------ user channel ------------
47          uid = torch.squeeze(
48              self.embedding_uid(user_input['uid'].to(self.args.device)),
                    dim=1)
49          uid = torch.squeeze(uid, dim=1)  # [batch_size, embedding_dim]
50          gender = torch.squeeze(user_input['gender'].to(self.args.device
                ), dim=1)  # [batch_size, gender_types (2)]
```

```
51          age = torch.squeeze(user_input['age'].to(self.args.device), dim
                =1)  # [batch_size, age_types (7)]
52          job = torch.squeeze(user_input['job'].to(self.args.device), dim
                =1)  # [batch_size, job_types (21)]
53          user_info = torch.cat([uid, gender, age, job],
54                                dim=1)  # concat of user info tensors  [
                                    batch_size, sum of dimensions]
55          user_feature = self.user_layer(user_info)  # user feature  [
                batch_size, hidden_dim]
56
57          # ------------ movie channel ------------
58          mid = torch.squeeze(
59              self.embedding_mid(movie_input['mid'].to(self.args.device))
                    , dim=1)
60          mid = torch.squeeze(mid, dim=1)  # [batch_size, embedding_dim]
61          mtype = movie_input['mtype'].to(self.args.device)  # [
                batch_size, movie_types (18)]
62          movie_info = torch.cat([mid, mtype], 1)  # concat of movie_id
                tensor and movie_type tensor
63          movie_info = self.movie_layer(movie_info)  # movie info  [
                batch_size, hidden_dim]
64
65          # movie title text
66          mtext = movie_input['mtext'].to(self.args.device)  # [
                batch_size, seq_len (16)]
67          text_embedding = self.text_embedding(mtext).unsqueeze(1)  # [
                batch_size, seq_len, text_embedding_dim]
68          text_info = [F.relu(conv(text_embedding)).squeeze(3) for conv
                in self.convs]
69          text_info = [F.max_pool1d(item, item.size(2)).squeeze(2) for
                item in text_info]
70          text_info = self.dropout(
71              torch.cat(text_info, 1))  # movie text info  [batch_size,
                    hidden_dim * len(filter_sizes)]
72
```

```
        movie_info = torch.cat([movie_info, text_info], dim=1)
        movie_feature = self.movie_combine_layer(movie_info)
        output = torch.sum(user_feature * movie_feature, 1)  # [
            batch_size, 1]

        return output, user_feature, movie_feature
```

深度学习模型的训练：

```
def load_model(args):
    set_seed(args)
    train_datasets = MovieRankDataset(pkl_file=args.path + 'train.pkl',
        args=args)
    test_datasets = MovieRankDataset(pkl_file=args.path + 'test.pkl',
       args=args)
    train_iter = DataLoader(train_datasets, batch_size=args.batch_size,
        drop_last=True)
    test_iter = DataLoader(test_datasets, batch_size=args.batch_size)
    model = MovieLens(user_max_dict=user_max_dict, movie_max_dict=
       movie_max_dict, args=args).to(args.device)
    return train_iter, test_iter, model


def train(args, model, train_iter):
    # optimizer
    no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
        {'params': [p for n, p in model.named_parameters() if not any(
           nd in n for nd in no_decay)],
         'weight_decay': 0.01},
        {'params': [p for n, p in model.named_parameters() if any(nd in
            n for nd in no_decay)], 'weight_decay': 0.0}
    ]
    optimizer = Adam(optimizer_grouped_parameters, lr=args.lr)
    # criterion
    criterion = nn.MSELoss()
```

```
    writer = SummaryWriter(logdir='/root/tf-logs')
    for epoch in range(args.epochs):
        print('Epoch [{}/{}]'.format(epoch + 1, args.epochs))
        model = model.train()
        losses = []
        for i_batch, sample_batch in enumerate(train_iter):
            user_inputs = sample_batch['user_inputs']
            movie_inputs = sample_batch['movie_inputs']
            target = torch.squeeze(sample_batch['target'].to(args.
                device))

            rank, _, _ = model(user_inputs, movie_inputs)
            loss = criterion(rank, target)
            losses.append(loss.item())
            loss.backward()
            if i_batch % 100 == 0:
                writer.add_scalar('data/loss', loss, i_batch * 20)
                print('Steps: {}\{}  Loss: {}'.format(i_batch, len(
                    train_iter), loss.item()))

            # nn.utils.clip_grad_norm_(model.parameters(), max_norm
                =1.0)
            optimizer.step()
            optimizer.zero_grad()
        # 一个Epoch训练完毕, 输出train_loss
        print('Epoch: {0}    Train Loss: {1:>5.6}'.format(epoch + 1, np.
            mean(losses)))

    writer.export_scalars_to_json("./test.json")
    writer.close()
```

3.4 SimRank 算法

```
def simrank(args):
    user_index_to_uid = pkl.load(open(args.path + 'user_index_to_uid.
        pkl', 'rb'))
```

```
    movie_index_to_mid = pkl.load(open(args.path + 'movie_index_to_mid.pkl', 'rb'))
    R = pkl.load(open(args.path + 'rank_matrix_initial.pkl', 'rb'))
    C = pkl.load(open(args.path + 'choice_matrix.pkl', 'rb'))  # Choice Matrix
    C_col = C.apply(normalization, axis=0)  # 列归一化
    C_row = C.apply(normalization, axis=1)  # 行归一化
    X = pd.DataFrame(np.zeros([6040, 6040], dtype=int))
    Y = pd.DataFrame(np.zeros([3706, 3706], dtype=int))

    for k in range(args.k):
        X_new = args.c * np.dot(np.dot(C_row, Y), np.matrix(C_row).T) + np.eye(6040) - np.diag(
            np.diagonal(args.c * np.dot(np.dot(C_row, Y), np.matrix(C_row).T)))
        Y_new = args.c * np.dot(np.dot(C_col.T, X), C_col) + np.eye(3706) - np.diag(
            np.diagonal(args.c * np.dot(np.dot(C_col.T, X), C_col)))
        X = X_new
        Y = Y_new

    def series_notzero_values(series):
        val = 0
        for i in series.index:
            if series[i] > 0:
                val += 1
        return val

    for i in R.index:
        for j in R.columns:
            if R.iloc[i][j] > 0:
                pass
            else:
                val = 0
                for p in R.columns:
```

```
                    val += C.iloc[i][p] * R.iloc[i][p] * Y[p, j]
                for q in R.index:
                    val += C.iloc[q][j] * R.iloc[q][j] * X[q, i]
                not_zero_val = series_notzero_values(C[j]) +
                   series_notzero_values(C.iloc[i])
                # not_zero_val = series_notzero_values(C.iloc[i])
                R[j][i] = val / not_zero_val

    return R
```

3.5 推荐

```
def recommend_Movies(args, uid, rank_matrix):
    """
    根据 UserID 为其推荐电影与也喜欢看这些电影的用户 (相似用户)
    """
    users_raw, movies_raw, _, _ = load_preprocessed_data(args)
    user_ratings = rank_matrix.loc[uid].sort_values(ascending=False)
    top_k_movies = list(user_ratings.index)[0:args.recommend_num]
    similar_users = []
    print('该用户信息: ' + 'UserID: {0}    Gender:{1}    Age:{2}
       Occupation:{3}'.format(
        uid, user_gender[users_raw[uid]['Gender']], user_age[users_raw[
           uid]['Age']], user_job[users_raw[uid]['Job']]
    ))
    print('为该用户推荐的电影: ')
    for i, mid in enumerate(top_k_movies):
        # 推荐电影
        movie_recommend = movies_raw[mid]
        print('{0}: MovieID: {1}    Movie Title: {2}    Movie Type: {3}
           '.format(
            i + 1, movie_recommend['Movie_ID'], movie_recommend['
               Movie_Title'], movie_recommend['Movie_Type']
        ))
        # 获取相似用户 (对推荐电影评分最高者)
        movie_ratings = rank_matrix[mid].sort_values(ascending=False)
```

```
21          for j in range(len(movie_ratings)):
22              similar_user = movie_ratings.index[j]
23              if similar_user in similar_users:
24                  continue
25              else:
26                  similar_users.append(similar_user)
27                  break
28      print('为该用户推荐的相似用户：')
29      for i, similar_user_id in enumerate(similar_users):
30          user_recommend = users_raw[similar_user_id]
31          print('{0}: UserID: {1}    Gender:{2}    Age:{3}    Occupation
                :{4}'.format(
32              i + 1, similar_user_id, user_gender[user_recommend['Gender'
                    ]],
33              user_age[user_recommend['Age']], user_job[user_recommend['
                    Job']]
34          ))
35
36
37  def recommend_Users(args, mid, rank_matrix):
38      """
39      根据 MovieID 得到喜欢看该电影的用户与他们喜欢看的电影（相似电影）
40      """
41      users_raw, movies_raw, _, _ = load_preprocessed_data(args)
42      movie_ratings = rank_matrix[mid].sort_values(ascending=False)
43      top_k_users = list(movie_ratings.index)[0:args.recommend_num]
44      similar_movies = []
45      print('该电影信息：' + 'MovieID: {0}    Movie Title: {1}    Movie
            Type: {2}'.format(
46          mid, movies_raw[mid]['Movie_Title'], movies_raw[mid]['
                Movie_Type']
47      ))
48      print('喜欢看这部电影的用户：')
49      for i, uid in enumerate(top_k_users):
50          user_recommend = users_raw[uid]
```

```
51          print('{0}: UserID: {1}    Gender:{2}    Age:{3}    Occupation
                :{4}'.format(
52              i + 1, uid, user_gender[user_recommend['Gender']],
53              user_age[user_recommend['Age']], user_job[user_recommend['
                    Job']]
54          ))
55          user_ratings = rank_matrix.loc[uid].sort_values(ascending=False
                )
56          for j in range(len(user_ratings)):
57              similar_movie = movie_ratings.index[j]
58              if similar_movie in similar_movies:
59                  continue
60              else:
61                  similar_movies.append(similar_movie)
62                  break
63      print('他们也在看: ')
64      for i, similar_movie_id in enumerate(similar_movies):
65          movie_recommend = movies_raw[similar_movie_id]
66          print('{0}: MovieID: {1}    Movie Title: {2}    Movie Type: {3}
                '.format(
67              i + 1, movie_recommend['Movie_ID'], movie_recommend['
                    Movie_Title'], movie_recommend['Movie_Type']
68          ))
```

4 实验结果

4.1 推荐结果

（1）矩阵分解结合深度学习方法

为编号为 2021 的用户推荐电影与相似用户：

```
[8]: recommend_Movies(args, uid=2021, rank_matrix=rank_matrix)
```

```
该用户信息：UserID: 2021    Gender:Male    Age:25-34    Occupation:other or not specified
为该用户推荐的电影：
1: MovieID: 318    Movie Title: Shawshank Redemption, The (1994)    Movie Type: Drama
2: MovieID: 1212    Movie Title: Third Man, The (1949)    Movie Type: Mystery|Thriller
3: MovieID: 750    Movie Title: Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)
Movie Type: Sci-Fi|War
4: MovieID: 912    Movie Title: Casablanca (1942)    Movie Type: Drama|Romance|War
5: MovieID: 260    Movie Title: Star Wars: Episode IV - A New Hope (1977)    Movie Type: Action|Adventure|Fan
tasy|Sci-Fi
为该用户推荐的相似用户：
1: UserID: 714    Gender:Male    Age:18-24    Occupation:college/grad student
2: UserID: 4342    Gender:Male    Age:25-34    Occupation:other or not specified
3: UserID: 53    Gender:Male    Age:25-34    Occupation:other or not specified
4: UserID: 2793    Gender:Male    Age:35-44    Occupation:executive/managerial
5: UserID: 5165    Gender:Male    Age:35-44    Occupation:doctor/health care
```

为编号为 2021 的电影相似电影与相关用户：

```
[9]: recommend_Users(args, mid=2021, rank_matrix=rank_matrix)
```

```
该电影信息：MovieID: 2021    Movie Title: Dune (1984)    Movie Type: Fantasy|Sci-Fi
喜欢看这部电影的用户：
1: UserID: 5165    Gender:Male    Age:35-44    Occupation:doctor/health care
2: UserID: 2203    Gender:Male    Age:45-49    Occupation:programmer
3: UserID: 5103    Gender:Female    Age:35-44    Occupation:self-employed
4: UserID: 3540    Gender:Male    Age:18-24    Occupation:technician/engineer
5: UserID: 714    Gender:Male    Age:18-24    Occupation:college/grad student
他们也在看：
1: MovieID: 318    Movie Title: Shawshank Redemption, The (1994)    Movie Type: Drama
2: MovieID: 1212    Movie Title: Third Man, The (1949)    Movie Type: Mystery|Thriller
3: MovieID: 912    Movie Title: Casablanca (1942)    Movie Type: Drama|Romance|War
4: MovieID: 904    Movie Title: Rear Window (1954)    Movie Type: Mystery|Thriller
5: MovieID: 750    Movie Title: Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1963)
Movie Type: Sci-Fi|War
```

（2）SimRank 算法

为编号为 2021 的用户推荐电影与相似用户：

```
recommend_Similar_Users(args, uid=2021, rank_matrix=rank_matrix)
```

```
该用户信息：UserID: 2021    Gender:Male    Age:25-34    Occupation:other or not specified
为该用户推荐的相似用户：
1: UserID: 53    Gender:Male    Age:25-34    Occupation:other or not specified
2: UserID: 1034    Gender:Female    Age:35-44    Occupation:academic/educator
3: UserID: 317    Gender:Male    Age:35-44    Occupation:executive/managerial
4: UserID: 4512    Gender:Male    Age:25-34    Occupation:academic/educator
5: UserID: 4342    Gender:Male    Age:25-34    Occupation:other or not specified
```

为编号为 2021 的电影推荐相似电影：

```
recommend_Similar_Movies(args, mid=2021, rank_matrix=rank_matrix)
```

```
该电影信息：MovieID: 2021    Movie Title: Dune (1984)    Movie Type: Fantasy|Sci-Fi
该电影的相似电影：
1: MovieID: 260    Movie Title: Star Wars: Episode IV - A New Hope (1977)    Movie Type: Action|Adventure|Fan
tasy|Sci-Fi
2: MovieID: 2622    Movie Title: Midsummer Night's Dream, A (1999)    Movie Type: Comedy|Fantasy
3: MovieID: 2900    Movie Title: Monkey Shines (1988)    Movie Type: Horror|Sci-Fi
4: MovieID: 671    Movie Title: Mystery Science Theater 3000: The Movie (1996)    Movie Type: Comedy|Sci-Fi
5: MovieID: 674    Movie Title: Barbarella (1968)    Movie Type: Adventure|Sci-Fi
```

4.2 模型对比

在预先划分的 10000 条数据构成的测试集上比较 SimRank 算法与矩阵分解算法的 MSE 误差，结果如下：

表 1: 各模型基本信息与 MSE 指标

Models	Parameters	Time Cost	MSE
SimRank	K=10, C=1	14m37.2s	2.083
SimRank	K=10, C=0.8	15m32.6s	**1.657**
FunkSVD+DL	4epochs, lr=5e-4, bs=512	49m26.7s	0.899
FunkSVD+DL	4epochs, lr=3e-4, bs=512	51m18.2s	0.880
FunkSVD+DL	5epochs, lr=3e-4, bs=512	61m63.1s	**0.865**

参考文献

[1] 刘建平 Pinard. 协同过滤推荐算法总结[Z]. https://www.cnblogs.com/pinard/p/6349233.html. 2017.

[2] JEH G, WIDOM J. Simrank: a measure of structural-context similarity[C]/ /Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. 2002: 538-543.

[3] 刘建平 Pinard. SimRank 协同过滤推荐算法[Z]. https://www.cnblogs.com/pinard/p/6362647.html. 2017.

[4] KIM Y. Convolutional Neural Networks for Sentence Classification[Z]. 2014. arXiv: 1408 . 5882 [cs.CL].