基于深度学习的movielens推荐

## 预处理部分：

先对loaded后的数据进行说明

Users：

Loaded前：

| **UserID** | **Gender** | **Age** | **OccupationID** | **Zip-code** |
| --- | --- | --- | --- | --- |
| **0** | 1 | F | 1 | 10 | 48067 |

Movies

| **MovieID** | **Title** | **Genres** |
| --- | --- | --- |
| **0** | 1 | Toy Story (1995) | Animation|Children's|Comedy |

Ratings:

| **UserID** | **MovieID** | **Rating** |
| --- | --- | --- |
| **0** | 1 | 1193 | 5 |

## 模型解读

title\_count, title\_set, genres2int, features, targets\_values, ratings, users, movies, data, movies\_orig, users\_orig = load\_data()

主要的：

features:

#UserID 2

# MovieID 1193

# Gender 1

# Age 5

# Title [4094, 4032, 1255, 4199, 3605, 90, 2102, 2102,...

# Genres [10,16,16,16，16,16...

# Name: 1, dtype: object

先把文件中出现的所有Ttitle的单词构成一个词库，然后每个title的多个单词构成词库对应单词位置索引的下标构成的向量

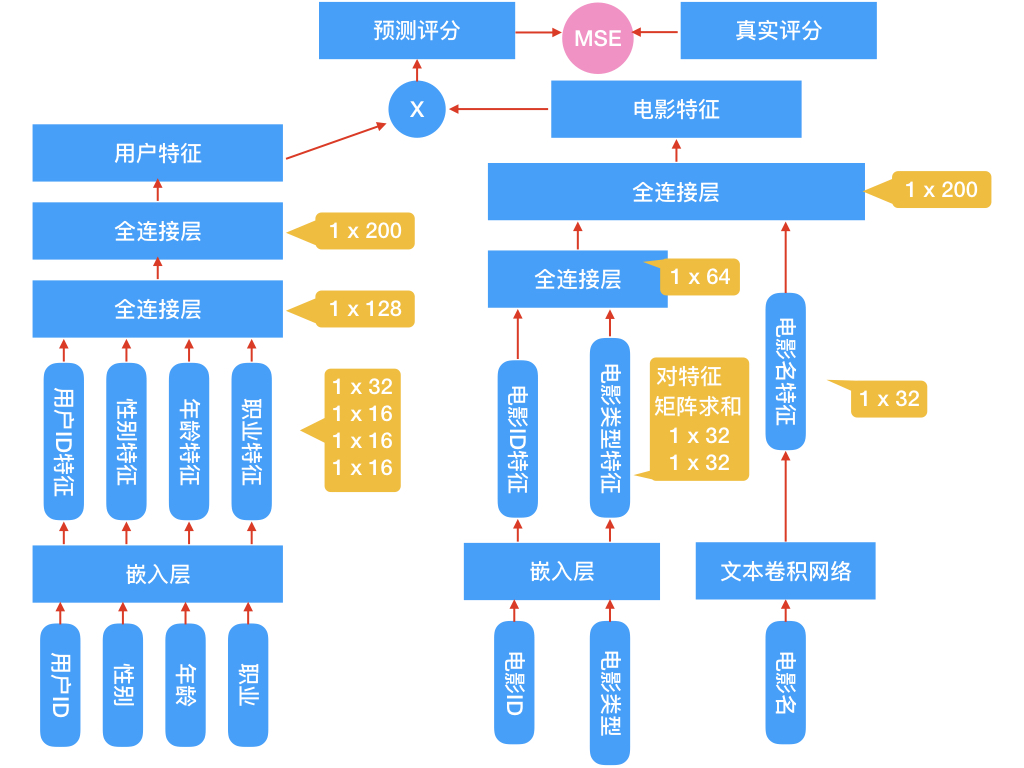
Genres同title，但genres有18种，如果对应位置为None，则以16填充

Targets\_value: 每个用户对所看过电影评分向量

神经网络嵌入图细节：

用户：uid, gender,age,JobID 都是单个数字 1\*1矩阵

电影：movie\_title(电影名)：一个向量 ，电影类型：一个向量，电影ID：一个数字



用户特征向量构建过程：

嵌入层有四个矩阵分别为：

1\*16

1\*16

2\*16

6040\*32

1\*1

4个 的矩阵分别进入对应层lookup寻找对应行向量，得到

1\*16

1\*16

1\*16

1\*32

然后concat得到矩阵

1\*80

然后进入第一个全连接层，这层有128个神经元，即权重矩阵为80\*128，激活函数为relu

第二层全连接层有200个神经元，权重矩阵为128\*200，激活函数为tanh

输出得到用户的特征矩阵（大小：1\*200）

电影特征向量构建过程：

电影ID 和 电影类型特征构建

输入层：

电影ID： 电影类型：

1\*19

1\*1

嵌入层构建：

嵌入层矩阵为：

3952\*32

19\*32

输入层电影ID对嵌入层矩阵lookup，电影类型向量对嵌入层矩阵做矩阵乘法分别得到：

1\*32

1\*32

两个矩阵拼接得到

1\*64

进入全连接神经元 ，本层神经元个数也为64个

电影名特征构建：

输入层：(a为不确定的整数，电影的名字长度不一样)

1\*15

嵌入层：

5216\*32

输入层对嵌入层lookup得到

15\*32

然后进入卷积层和池化层，一共3层

每一个卷积层的卷积核为8个  
每层的卷积核对应维度分别为2\*32, 4\*32 5\*32

池化为max\_pool

之后连接droupout层

输出电影名的特征向量为：

1\*32

1\*64

然后与电影ID和电影类型形成的 作为输入进入全连接层，

最终得到 的电影特征向量。

1\*200

用户和电影的汇总：

用户对应向量和电影对应向量转置相乘得到一个预测评分，优化MSE得到最终模型

训练方式：随机分成训练集和测试集

训练集和测试集每256个为一个batch

每个batch进行一次记录loss，写入dictionary.txt

rating\_movie（uid,mid）

指定用户对指定电影的预测评分

对新数据使用本模型进行评估和推荐：

利用模型计算电影特征和用户

1、推荐同类型的电影：

思路是计算当前看的电影特征向量与整个电影特征矩阵的余弦相似度，取相似度最大top\_k个，这里加了些随机选择在里面，保证每次的推荐稍稍有些不同

2、推荐您喜欢的电影

思路是使用用户特征向量与电影特征矩阵计算所有电影的评分，取评分最高的top\_k个，同样加了些随机选择部分。

3、协同过滤（基于电影的协同过滤）

看过这个电影的人还看了（喜欢）哪些电影

#首先选出喜欢某个电影的top\_k个人，得到这几个人的用户特征向量。

#然后计算这几个人对所有电影的评分

#选择每个人评分最高的电影作为推荐

#同样加入了随机选择

代码注释解析见jupyter

详细执行细节见jupyter

## 相关推荐的函数：

#指定用户和电影进行预测评分 ：rating\_movie

#推荐同类型的电影 ：recommend\_same\_type\_movie

#推荐您喜欢的电影 ：recommend\_your\_favorite\_movie

#看过这个电影的人还看了（喜欢）哪些电影：recommend\_other\_favorite\_movie

执行，得到全部见结果：

#recommend\_same\_type\_movie(list(movies['MovieID']), top\_k = 5)

#recommend\_your\_favorite\_movie(list(users['UserID']), top\_k = 5)

#recommend\_other\_favorite\_movie(list(movies['MovieID']), top\_k = 5)

相关文件说明：

部分的执行结果分别保存在：

结果文件：

same\_type\_recommend.txt推荐同类型的电影

recommend\_your\_favorite\_movie.txt推荐您喜欢的电影

recommend\_other\_favorite\_movie.txt看过这个电影的人还看了（喜欢）哪些电影

dictionary.txt为模型训练日志

模型文件：

checkpoint（一个读取的必要文件）

save.data-00000-of-00001（参数值）

save.index（参数名）

save.meta（图结构）

movie\_matrics.p(模型电影特征矩阵的P文件)

users\_matrics.p(模型用户特征矩阵的P文件)

代码文件：

DL\_recommend.ipynb

数据文件：

users.dat

movies.dat

ratings.dat