基于神经协同过滤的推荐系统

刘炼 1711361
史恩扬 1711373

2019/5/8

摘要

设计实现基于协同过滤神经网络的推荐系统,通过利用训练集中的 $(user, item, score)$ 数据，并结合 $item\_attribute$ 数据，对神经网络进行训练。对于测试集中的数据进行预测。

关键字：协同过滤　神经网络　推荐系统

目录

1 问题描述

1.1 数据集描述

1.1.1 训练集

由 $(user, item, score)$ 数据构成的数据集，其中 $N_{user} = 19835, N_{item} = 624961, N_{rating} = 5001506$，为了便于训练，我们将训练集按照 9:1 的比例分成训练集和验证集，通过验证集的结果来反馈，判断我们算法实现的效果，以防止在调参过程中出现的问题。对数据集进行可视化，我们将所有的评分结果展示如图 1 所示：可以发现，分数的分布不满足我们概率论中所学习的

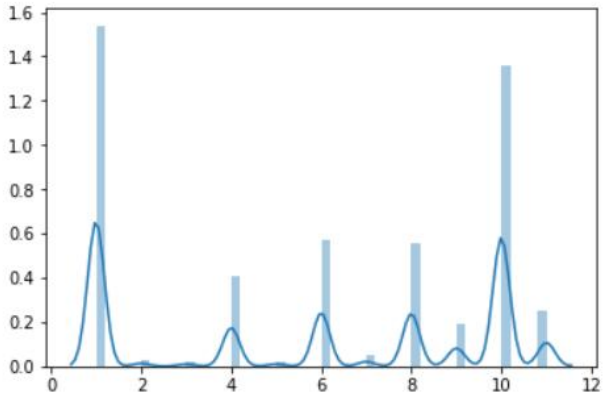


图 1: 将分数缩小十倍后的分布

分布图像，因此，传统的学习方法会有其局限性。同时，由于实际上，打分一般是十的倍数，故实际上我们的处理可以直接考虑除 10。

1.1.2 测试集

由 $(user, item)$ 数据构成的测试集，对于每个 $user$ 都有 6 组 $item$ 测试

1.1.3 item 属性集

对于每个 $item$ 都进行属性描述，包含两个属性，部分 $item$ 属性缺省 $(NONE)$，同时，对于没有标注的 $item$, 将其标注为 nan

1.2 神经网络

近年来，深度学习在许多方面已经取得了极大的成功，人工神经网络在计算机视觉，自然语言处理等方面有长足的进步，深度的神经网络模型在足够的算力支持下，在许多方面，能够比传统的机器学习的算法表现出更好的效果。因此，我们考虑使用神经网络模型来实现此次作业。

关于神经网络，我们通常指的是人工神经网络，它是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络, 它的简单抽象模型如下图所示。神经网络利用反向传播，不断调整模型中的参数，

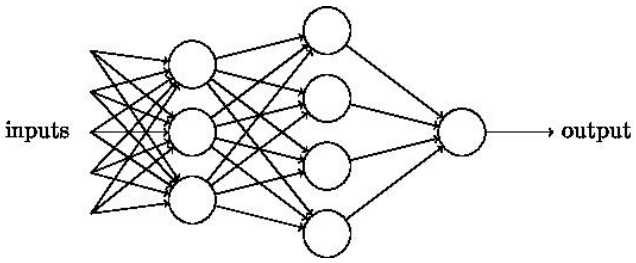


图 2: 简单神经网络模型

来减少目标的 loss 值，使得模型能更好地拟合原始的模型结果。

1.3 协同过滤

协同过滤是机器学习推荐系统中一个极其有效的算法，传统的推荐算法有许多种，包括基于内容的推荐算法，协同过滤推荐算法，混合推荐算法等。

协同过滤算法，原理是用户喜欢那些具有相似兴趣的用户喜欢过的商品，比如你的朋友喜欢电影哈利波特 I，那么就会推荐给你，这是最简单的基于用户的协同过滤算法（user-based collaboratIve filtering），还有一种是基于 Item 的协同过滤算法（item-based collaborative filtering），这两种方法都是将用户的所有数据读入到内存中进行运算的，因此成为 Memory-based Collaborative Filtering，另一种则是 Model-based collaborative filtering，包括 Aspect Model，pLSA，LDA，聚类，SVD，Matrix Factorization 等，这种方法训练过程比较长，但是训练完成后，推荐过程比较快。

更具体地，我们采用了基于矩阵分解（Matrix Factorization）的协同过滤算法，关于矩阵分解，可以将其理解为对评分矩阵的背后的隐因子的把

握。

假设我们已经有了一个评分矩阵 $R_{m,n}$，m 个用户对 n 个物品的评分全在这个矩阵里，当然这是一个高度稀疏的矩阵，我们用 $r_{u,,i}$ 表示用户 u 对物品 i 的评分。LFM 认为 $R_{m,n} = P_{m,F}\, Q_{F,n}$ 即 R 是两个矩阵的乘积 (所以 LFM 又被称为矩阵分解法，MF，matrix factorization model)，F 是隐因子的个数，P 的每一行代表一个用户对各隐因子的喜欢程序，Q 的每一列代表一个物品在各个隐因子上的概率分布。$\hat{r}_{u,i} = \sum_{f=1}^{F} P_{uf} Q_{fi}$

因此，我们将结合协同过滤的思路，采用神经网络的模型，构建基于神经网络的协同过滤算法，我们将具体的算法分析细节展示在下一节。

2 算法设计及分析

2.1 数据预处理

由于给出的 txt 文档无法直接读出我们想要的数据，我们需要对其进行数据的预处理，这里给出了数据预处理的代码，如下所示

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# train_file='train.txt'
# test_file='test.txt'
# attribute_file='itemAttribute.txt'
# pd.read_csv()
# row = [0,0,1,2]
# col=[344,254,653,213]
# data=[90,80,90,70]
# c=sparse.coo_matrix((data,(row,col)),shape=(10,1000))
# print(c.toarray())

data=open('train.txt')
users=[]
nums=[]
for each_line in data:
    if (not each_line.find('|')==-1):
        (user_id,item_num)= each_line.split('|',1)
        for i in range(int(item_num)):
            users.append(int(user_id))
        #nums.append(int(item_num))
data.close()
users=np.array(users).reshape(len(users),1)
```

```
# print(max(users))
# print(max(nums))


with open('train.txt','r') as f:
    lines=f.readlines()
with open('process_train.txt','w') as fw:
    for line in lines:
        if (not line.find('|')==-1):
            continue
        fw.write(line)

record_data=np.loadtxt('process_train.txt').astype(int)
# items=record_data[:,0]
scores=record_data[:,1]
# # print("items",items[0:10],"max_item",max(items))
# # print("scores",scores[0:10],"min",min(scores),"max",max(scores))
#items=record_data.reshape(119010,1)
# print(record_data.shape)
# print(users.shape)

sns.distplot(scores)
plt.show()

# record=np.column_stack((users,record_data))
# #print(record[0:10])
# pd.DataFrame(record,columns=['user id','item number','score']).to_csv('
    train.csv')
```

language=Python

2.2 算法设计

我们采用深度学习模块 pytorch 作为搭建深度学习网络的基本框架，具体代码展示如下：

```
    import torch.nn as nn
    import pandas as pd
    import torch
    import torch.utils.data as data
    import torch.optim as optim

    EPOCHS = 200
    DEVICE = torch.device("cuda")
    Learn_rate = 1e-3
    Attr_num = 507171
```

```
64      Class_size = 11
65      Train_data_num = 4951491
66      Test_data_num = 50015
67      User_num = 19835
68      Item_num = 624961
69      Batch_size = 16
70      Max_attr1 = 624944
71      Max_attr2 = 624952
72      Embedding_dim = 64
73      Each_user_test_size = 6
74      Weight_decay = 0.001
75
76
77      class Mynet(nn.Module):
78          def __init__(self, embedding_dim, attr):
79              super(Mynet, self).__init__()
80              self.attr = attr
81              self.default1 = nn.Parameter(torch.FloatTensor([0]),
        requires_grad=True)
82              self.default2 = nn.Parameter(torch.FloatTensor([0]),
        requires_grad=True)
83              self.default3 = nn.Parameter(torch.FloatTensor([0]),
        requires_grad=True)
84              self.default4 = nn.Parameter(torch.FloatTensor([0]),
        requires_grad=True)
85              self.embedding_user = nn.Embedding(User_num, embedding_dim)
86              self.embedding_item = nn.Embedding(Item_num, embedding_dim)
87              self.embedding_attr1 = nn.Embedding(Max_attr1, embedding_dim)
88              self.embedding_attr2 = nn.Embedding(Max_attr2, embedding_dim)
89              self.act_layer1 = nn.ReLU()
90              self.act_layer2 = nn.ReLU()
91              self.act_layer3 = nn.ReLU()
92              self.act_layer4 = nn.ReLU()
93              self.CF_layer1 = nn.Linear(4*embedding_dim, 40)
94              self.CF_layer2 = nn.Linear(40, 30)
95              self.CF_layer3 = nn.Linear(30, 20)
96              self.CF_layer4 = nn.Linear(20, 10)
97              self.CF_layer5 = nn.Linear(10, 1)
98
99          def forward(self, user, item, attr1, attr2):
100             for i in range(attr1.size()[0]):
101                 if(attr1[i]==-1):
102                     attr1[i] = self.default1
103                 if (attr1[i] == -2):
104                     attr1[i] = self.default3
105                 if (attr2[i] == -1):
```

```
106                     attr2[i] = self.default2
107                 if (attr2[i] == -2):
108                     attr2[i] = self.default4
109             attr1 = self.embedding_attr1(attr1.long())
110             attr2 = self.embedding_attr2(attr2.long())
111             user = self.embedding_user(user)
112             item = self.embedding_item(item)
113             attr1.squeeze_()
114             attr2.squeeze_()
115             user.squeeze_()
116             item.squeeze_()
117             x = torch.cat((user, item, attr1, attr2), 1)
118             x = x.to(DEVICE)
119             x = self.CF_layer1(x)
120             x = self.act_layer1(x)
121             x = self.CF_layer2(x)
122             x = self.act_layer2(x)
123             x = self.CF_layer3(x)
124             x = self.act_layer3(x)
125             x = self.CF_layer4(x)
126             x = self.act_layer4(x)
127             x = self.CF_layer5(x)
128             return x
129
130
131     class Dataset(data.Dataset):
132         def __init__(self, filename, attr, type):
133             self.frame = pd.read_csv(filename)
134             self.type = type
135             self.attr = attr
136
137         def __len__(self):
138             if(self.type=='train'):
139                 return Train_data_num
140             if(self.type=='test'):
141                 return Test_data_num
142
143         def __getitem__(self, idx):
144             if (self.type == 'train'):
145                 user = self.frame.iloc[idx, 1]
146                 item = self.frame.iloc[idx, 2]
147                 target = self.frame.iloc[idx, 3]
148             if (self.type == 'test'):
149                 user = self.frame.iloc[idx+Train_data_num, 1]
150                 item = self.frame.iloc[idx+Train_data_num, 2]
151                 target = self.frame.iloc[idx+Train_data_num, 3]
```

```
152             try:
153                 attr1, attr2 = self.attr[item][0], self.attr[item][1]
154             except KeyError:
155                 attr1, attr2 = torch.FloatTensor([-2]), torch.FloatTensor
    ([-2])
156             else:
157                 if(attr1=='None'):
158                     attr1 = torch.FloatTensor([-1])
159                 else:
160                     attr1 = torch.FloatTensor([float(attr1)])
161                 if(attr2=='None'):
162                     attr2 = torch.FloatTensor([-1])
163                 else:
164                     attr2 = torch.FloatTensor([float(attr2)])
165             return user, item, target, attr1, attr2
166
167
168     class Test_Dataset(data.Dataset):
169         def __init__(self, filename, attr):
170             self.frame = pd.read_csv(filename)
171             self.attr = attr
172
173         def __len__(self):
174             return User_num*Each_user_test_size
175
176         def __getitem__(self, idx):
177             user = self.frame.iloc[idx, 1]
178             item = self.frame.iloc[idx, 2]
179             try:
180                 attr1, attr2 = self.attr[item][0], self.attr[item][1]
181             except KeyError:
182                 attr1, attr2 = torch.FloatTensor([-2]), torch.FloatTensor
    ([-2])
183             else:
184                 if(attr1=='None'):
185                     attr1 = torch.FloatTensor([-1])
186                 else:
187                     attr1 = torch.FloatTensor([float(attr1)])
188                 if(attr2=='None'):
189                     attr2 = torch.FloatTensor([-1])
190                 else:
191                     attr2 = torch.FloatTensor([float(attr2)])
192             return user, item, attr1, attr2
193
194
195     def attr_init(attr_filename):
```

```
196         attr = pd.read_csv(attr_filename)
197         size = attr.shape
198         attrdataset = {}
199         for i in range(size[0]):
200             attrdataset[attr.iloc[i, 1]] = [attr.iloc[i, 2], attr.iloc[i,
    3]]
201         return attrdataset
202 
203 
204     def train(model, device, train_loader, optimizer):
205         model.train()
206         loss_fn = nn.MSELoss(reduction='mean')
207         for batch_idx, (user, item, target, attr1, attr2) in enumerate(
    train_loader):
208             user, item, target = user.to(device), item.to(device), target.to
    (device).float()
209             attr1, attr2 = attr1.to(device), attr2.to(device)
210             optimizer.zero_grad()
211             target = target / 10
212             output = model(user, item, attr1, attr2)
213             loss = loss_fn(output, target)
214             loss.backward()
215             optimizer.step()
216 
217 
218     def test(model, device, test_loader):
219         model.eval()
220         result = torch.LongTensor(0, 2)
221         for batch_idx, (user, item, attr1, attr2) in enumerate(test_loader):
222             user, item = user.to(device), item.to(device)
223             attr1, attr2 = attr1.to(device), attr2.to(device)
224             output = model(user, item, attr1, attr2)
225             w = torch.stack((item, output.long().squeeze_()),1)
226             result = torch.cat((result.long(), w), 0)
227         with open('result.txt', 'w') as f:
228             for i in range(User_num*Each_user_test_size):
229                 if(i%6==0):
230                     f.writelines(str(int(i/6)) + '|' + str(
    Each_user_test_size)+'\n')
231                 f.writelines(str(int(result[i][0])) + ' ' + str(int(result[i
    ][1]*10))+'\n')
232             f.close()
233 
234 
235     if __name__ =='__main__':
236         attr_filename = 'ItemAttribute.csv '
```

```
237         train_filename = 'pd_data.csv '
238         test_filename = 'test.csv '
239         attr = attr_init(attr_filename)
240         traindataset = Dataset(filename=train_filename, attr=attr, type='
    train')
241         testdataset = Test_Dataset(filename=test_filename, attr=attr)
242         train_loader = torch.utils.data.DataLoader(traindataset, batch_size=
    Batch_size, shuffle=True)
243         test_loader = torch.utils.data.DataLoader(testdataset, batch_size=
    Batch_size)
244         model = Mynet(embedding_dim=Embedding_dim, attr=attr).to(DEVICE)
245         optimizer = optim.Adam(model.parameters(), lr=Learn_rate,
    weight_decay=Weight_decay)
246         for epoch in range(1, EPOCHS + 1):
247             train(model, DEVICE, train_loader, optimizer)
248         test(model, DEVICE, test_loader)
```

2.3 算法分析

我们的算法的基本思路展示如图 2 所示：粗略地，我们可以将我们的神

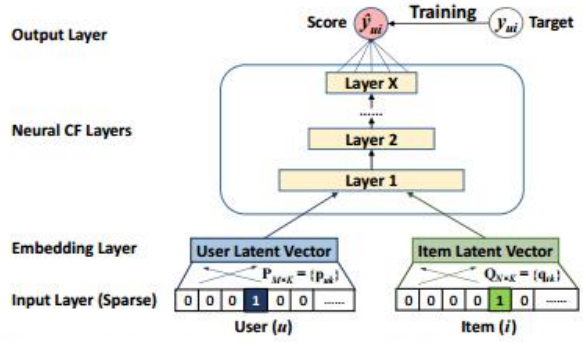


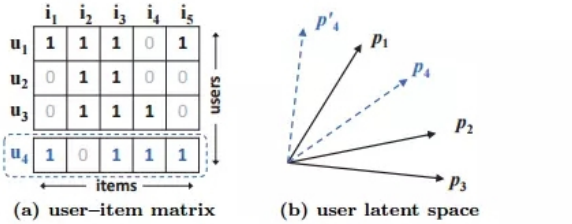
Figure 2: Neural collaborative filtering framework

图 3: 基于神经网络的矩阵分解框架

经网络分为以下几个部分：1. 输入层和 embedding 层 2. 利用多层神经网络搭建的矩阵分解协同过滤算法 3. 输出层。这里主要对第一部分和第二部分进行具体的算法分析。主要分为以下几个方面，同时也对训练时间进行分析。

2.3.1 one-hot

什么是 one-hot 编码？one-hot 编码，又称独热编码、一位有效编码。其方法是使用 N 位状态寄存器来对 N 个状态进行编码，每个状态都有它独立的寄存器位，并且在任意时候，其中只有一位有效。采用 one-hot 编码能够较好地反映数据的特征，并对后续的 Embedding 的处理有较好的结果。但是，如果只是采用 one-hot 编码的话，我们可以明显发现，这样的编码结果使得数据量急剧变大，会导致我们无法对其进行处理。



(a) user–item matrix (b) user latent space

Figure 1: An example illustrates MF's limitation. From data matrix (a), u_4 is most similar to u_1, followed by u_3, and lastly u_2. However in the latent space (b), placing p_4 closest to p_1 makes p_4 closer to p_2 than p_3, incurring a large ranking loss.

图 4: one-hot 编码的 Embedding 表示

2.3.2 Embedding

在进行 one-hot 编码时，编码得到的向量会很高维也会很稀疏，同时，我们无法有效地判断两个不同的编码对之间的相似度，这会使得我们的协同过滤过程无法有效进行。实际上，Embedding 这一思路来源于自然语言处理中的 word2vec 的方法，从本质上而言，Embedding 是在捕捉向量中的隐因子（潜在因子），而我们实际上可以利用相似度的计算来得到两个不同向量之间的相似度。这样，我们就可以使用嵌入矩阵来而不是庞大的 one-hot 编码向量来保持每个向量更小。

对于我们原来构建的神经网络模型中，我们只对 user 和 item 进行了 Embedding，但实际上，我们所给出的 item_attribute 是对 item 本身的补充，因此对其进行 Embedding 能使得我们学到更多的 item 的性质，得到更好的结果，故在此处，我们将我们的神经网络模型修改如图 4 所示：

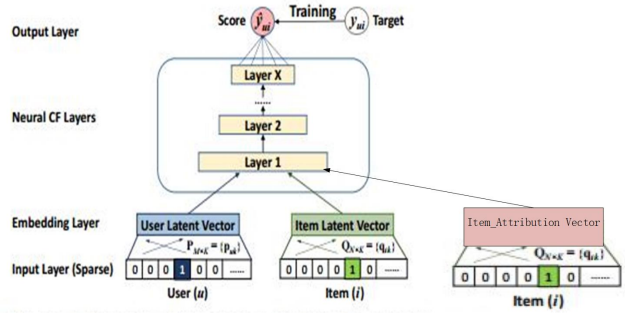


图 5: 基于 item_attribute 的新的框架

2.3.3 Nerual CF Layers

这里主要是运用了多层的深度神经网络来做协同过滤的处理，那么究竟这里的多层神经网络到底实现了什么？

我们首先将 item_attribution 的 Embedding 层的结果映射到 item 的 Embedding 层，将 item 的 Embedding 的 vector 和 user 的 Embedding 的 vector 进行 concat 操作进行合并，得到一个新的矩阵 K。我们首先来关注 Embedding 层的作用，实际上，可以明显的看到，Embedding 层作了一个对于原矩阵 R 的压缩，而这样的压缩是基于 Embedding 的，更确切的说，是基于 vector 的相似度的。这样一种相似度，利用的是 cos 的相似度的计算方式，并将其压缩到规定的隐因子数目中。

而对于后续压缩的矩阵，其和 svd 等传统机器学习算法中的矩阵分解是不同的。传统算法中，以 svd 为例，将评分矩阵分解为 $R_{U\times I} = P_{U\times K}Q_{K\times I}$，将矩阵分解之后，未知的评分可以用 $\hat{r}_{ui} = p_u^T q_i$，因此，我们可以用真实值与预测值之间的误差来进行训练 $e_{ui} = r_{ui} - \hat{r}_{ui}$, 继而算出总的误差平方和：$SSE = \sum_{u,i} e_{ui}^2 = \sum_{u,i}(r_{ui} - \sum_{k=1}^{K} p_{uk}q_{ki})^2$。对于测试的预测结果如图 6 所示，从中可以观察可以发现预测结果集中于 53 左右，其原因为 SVD 作为一种传统的线性模型，故而其拟合结果将服从正态分布

然而在 Nerual CF Layers 中，利用多层神经网络搭建的模型，因其结构中的 Relu 层，解决了线性模型预测结果服从正态分布的问题，更好地学习到压缩后的矩阵的 feature。并且通过多层神经网络的学习，利用反向传播，我们能得到更好的结果。

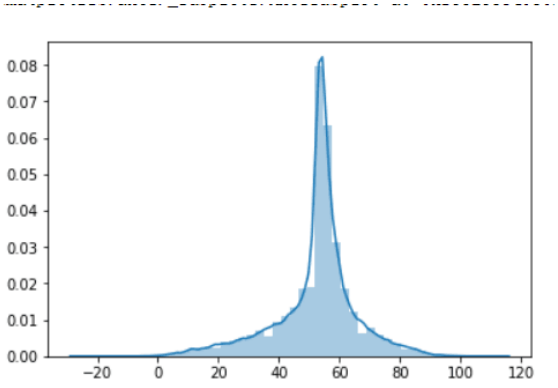


图 6: 基于传统 SVD 模型的预测结果

2.3.4 训练

mini_batch=512, 采用 Adam 进行梯度下降，利用 l2 正则化方法防止过拟合。

3 实证结果及分析

3.1 结果

基于 NCF 神经网络模型的预测分数分布图表示如下所示：通过对其 RMSE 的计算,得到验证集的 RMSE 的结果为:$RMSE = 25.937700279450134$，而通过 SVD 进行训练的结果为 $RMSE = 28.3264$, 故实际上应该有所提升，但是提升不是很大。训练时间：由于采用了神经网络进行学习，需要大量的计算资源，在 k40 的 GPU 上运行了 6 小时 23 分钟，而消耗的空间资源也较为巨大，在笔记本中运行时会出现显存不足的问题。

3.2 对结果的分析

由于计算资源的限制，我们无法进行有效地调参，在对于参数的设定中，很大程度上依赖于自己的经验和对于其它论文的借鉴，因此，如何调整参数是改进本实验结果的关键之一。

同时，如果考虑基于混合模型的协同过滤方法的启发，我们可以利用自编码器，多层感知机等其它模型来进行模型之间的融合，搭建更有效的神经

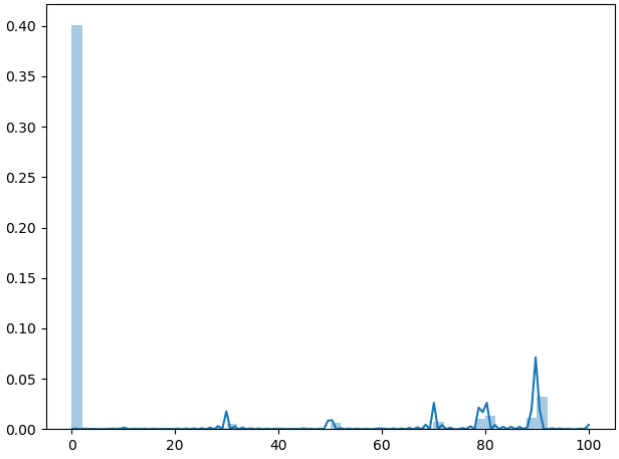


图 7: 基于 NCF 神经网络模型的预测结果

网络模型。同时，对于隐因子的分析，也是此次试验的关键所在，以及，如何利用激活层的非线性性来得到较好的分布结果，依然是十分重要的一个问题。同时，在对于 attribute 的处理上，如何有效地处理空值，依然是一个值得探讨的问题。