Web信息处理与应用实验报告

lab3-推荐系统实践

曾勇程 PB18000268
张弛 PB18000269

实验目的

本实验要求以给定的豆瓣电影打分数据集为基础，实现一个电影推荐系统。

该系统的主要功能是从给定的2173个用户对58431个电影所打出的227万多个评分中学习或训练出每个用户的观影偏好打分偏好、每部电影的隐含特征，从而对于测试集中给定的目标用户和电影给出预测的评分值，便于我们向用户推荐影片。

实验内容

数据来源为豆瓣电影的评分记录，包含用户ID，电影ID，打分，打分时间，电影标签。需要运用个性化推荐技术。

本次实验中我们小组实现了基于**内容**和基于**模型**的协同过滤推荐，即**用户协同过滤**和**SVD**算法。我们的工作可以分为以下几个步骤：

1. 对文档数据集进行**预处理**。这一步骤包括逐行读取文档内容，获得用户和电影信息，并进行重新编码处理，将打分数据**存入矩阵**数组中。注意，为了区分评分为0的和用户未打分两种情况，录入评分时我们**对所有评分加1**，得以和矩阵中未评分项0区分。
2. 实现**基于用户的协同过滤算法**。首先根据矩阵计算余弦相似度。具体计算方式即每个评分减去该用户的平均评分后，计算余弦相似度。需要注意的是，用户和电影的平均得分都**只考虑矩阵中的非0项**。然后对于每一个用户，搜索其**K个最近邻**的其他对目标电影评分的用户，以相似度**加权**偏置打分再加上用户偏置评分计算预测评分。注意，这里我们只采用相似度为正的近邻用户。为了用对协同过滤方法中**矩阵稀疏**的问题，我们尝试了先通过电影过滤进行**数据补足**再用户过滤的方法，即**混合协同过滤**。
3. 实现**SVD矩阵分解**算法。我们通过特定公式来预测打分，并通过不断地**梯度下降**学习，得到分解的两个用于打分的**特征矩阵**。
4. 思考并使用更多的**特征信息**。我们对于给定的**社交网络**，做出了各种尝试，以提升系统的准确性，如**冷启动**和**参与加权评分**。

实验条件

1. 硬件条件：一台PC机。
2. 软件条件：

   系统Windows10

   语言python3.6

   编辑器pycharm2018.3.4

关键代码讲解

本次实验用到的库有：

```
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
```

1. 读取数据并对用户和电影编码

```
trainDF = pd.read_csv("data/training.dat", sep=",", header=None,
    names=["UserID", "MovieID", "Rating", "Timestamp", "Tag1"])
# 可以继续加上Tag2, ...
testDF = pd.read_csv("data/testing.dat", sep=",", header=None,
    names=["UserID", "MovieID", "Timestamp", "Rating"])

Num2UserID = dict(enumerate(list(trainDF["UserID"].unique())))
UserID2Num = {key: value for value, key in Num2UserID.items()}
UserNum = len(UserID2Num)       # 2173

Num2MovieID = dict(enumerate(list(trainDF["MovieID"].unique())))
MovieID2Num = {key: value for value, key in Num2MovieID.items()}
MovieNum = len(MovieID2Num)     # 58431
```

2. 建立评分矩阵

注意，在矩阵数组中，我们把所有用户对电影的评分+1，这样得以把评分为0的项和未评分项区分开。

```
DataMatrix = np.zeros((UserNum, MovieNum))
for line in trainDF.itertuples():
    DataMatrix[UserID2Num[line[1]], MovieID2Num[line[2]]] = line[3]+1
    # 将评分为0项与矩阵未评分项区分
```

同时，我们也计算出每个电影和用户的平均得分，这里只统计矩阵中非零项的均值。

```
DataExist = (DataMatrix != 0)
MovieMeanScore = DataMatrix.sum(axis=0) / DataExist.sum(axis=0)
# print(MovieMeanScore)
UserBias = DataMatrix.sum(axis=1) / DataExist.sum(axis=1)
```

3. 获取社交关系

我们将社交关系存储在字典中，每个用户ID对应的关注账号以集合存储。

```
SocialRelation = {}
readrel = open("data/relation.txt", "r")
attention = readrel.readline()
while attention:
    attention = attention.strip("\n")
    userid = int(attention.split(":", 1)[0])
    SocialRelation[userid] = set(map(int, attention.split(":", 1)[1].split(",")))
    attention = readrel.readline()
readrel.close()
# print(SocialRelation)
```

基于用户的协同过滤

4. 计算用户间的余弦相似度

先对每个评分减去用户偏置分数，同样，代表未评分的0项不做处理。

```
BiasMatrix = DataExist * UserBias[:, np.newaxis]
# print(BiasMatrix)
NormMatrix = DataMatrix - BiasMatrix
UserSimilarity = cosine_similarity(NormMatrix)
# print(UserSimilarity)
```

5. 建立用户协同过滤评分预测系统

这一步中，我们不断调整近邻数目K的值，以找到预测效果最佳的模型。

以下是基础预测模型函数，我们对所有相似度为正的已评分用户加权平均。

```
def UserCFPredict(userid, movieid):
    useri = UserID2Num[userid]
    movie = MovieID2Num[movieid]
    weight = 0
    rating = 0
    # no k-nearest
    for userj in range(UserNum):
        if (DataMatrix[userj, movie] != 0) and (UserSimilarity[useri, userj] > 0):
            weight += UserSimilarity[useri, userj]
            rating += UserSimilarity[useri, userj]*NormMatrix[userj, movie]
    if weight == 0:
        return UserBias[useri]
    return UserBias[useri] + rating/weight
```

以下是K近邻改进后的系统。经过尝试，我们最终认为K=50时效果较好。

```
def UserCFPredict(userid, movieid):
    useri = UserID2Num[userid]
    movie = MovieID2Num[movieid]
    weight = 0
    rating = 0
    # k-nearest
    K = 25
    TopK = []
    for index in range(K):
```

```
        TopK.append([0, 0])     # userid, similarity
    for userj in range(UserNum):
        if DataMatrix[userj, movie] != 0:
            ptr = K-1
            if UserSimilarity[useri, userj] > TopK[ptr][1]:
                ptr = ptr-1
                while (ptr >= 0) and (UserSimilarity[useri, userj] > TopK[ptr][1]):
                    TopK[ptr+1] = TopK[ptr]
                    ptr = ptr-1
                TopK[ptr+1] = [userj, UserSimilarity[useri, userj]]
    for index in range(K):
        if TopK[index][1] > 0:
            weight += TopK[index][1]
            rating += TopK[index][1]*NormMatrix[TopK[index][0], movie]
    if weight == 0:
        return UserBias[useri]
    return UserBias[useri] + rating/weight
```

6. 生成预测数据

对于预测分数，我们四舍五入化为整数后还要减1，同时将过大和过小的数据修正回[0, 5]范围内。

```
def PrintResult(filename):
    writeres = open(filename, "w")
    for line in testDF.itertuples():
        userid = line[1]
        movieid = line[2]
        predscore = round(UserCFPredict(userid, movieid)-1)
        if predscore < 0:
            predscore = 0
        elif predscore > 5:
            predscore = 5
        writeres.write(str(predscore) + "\n")
    writeres.close()
filename = "submission21.txt"
# PrintResult(filename)
```

7. 对于用户协同过滤的改进

用户协同过滤方法效果往往并不十分准确的主要原因在于评分矩阵过于稀疏，因此我们根据找到的论文中提到的方案，试图先通过协同过滤进行数据补足，以提升预测准确率。

```
def DataSupplement(DataMatrix):
    nDataMatrix = np.zeros((UserNum, MovieNum))
    for user in range(UserNum):
        for movie in range(MovieNum):
            if DataMatrix[user, movie] != 0:
                nDataMatrix[user, movie] = DataMatrix[user, movie]
            else:
                userid = Num2UserID[user]
                movieid = Num2MovieID[movie]
                nDataMatrix[user, movie] = UserCFPredict(userid, movieid)
    return nDataMatrix
# nDataMatrix = DataSupplement(DataMatrix)
```

不过可惜的是，由于用户和电影数量较多，该方案的计算量过大，在我们的电脑中跑不太动，因此我们放弃了这一思路。

SVD算法

8. 数据处理

将训练集和测试集转为numpy数组，同样，所有打分加上1。

```
def getData():   #获取训练集和测试集的函数
    train_data = []
    test_data = []
    for line in trainDF.itertuples():
        train_data.append([UserID2Num[line[1]], MovieID2Num[line[2]],
line[3]+1])
    for line in testDF.itertuples():
        test_data.append([UserID2Num[line[1]], MovieID2Num[line[2]]])
    train_data=np.array(train_data)
    test_data=np.array(test_data)
    print('load data finished')
    print('train data ',len(train_data))
    print('test data ',len(test_data))
    return train_data,test_data

train_data,test_data=getData()
```

9. SVD迭代计算预测两个向量

具体详解见注释：

```
class SVD:
    def __init__(self,mat,K=30):
        self.mat=np.array(mat)
        self.K=K
        self.bi={}
        self.bu={}
        self.qi={}
        self.pu={}
        self.avg=np.mean(self.mat[:,2])
        for i in range(self.mat.shape[0]):
            uid=self.mat[i,0]
            iid=self.mat[i,1]
            self.bi.setdefault(iid,0)
            self.bu.setdefault(uid,0)

 self.qi.setdefault(iid,np.random.random((self.K,1))/10*np.sqrt(self.K))

 self.pu.setdefault(uid,np.random.random((self.K,1))/10*np.sqrt(self.K))

    def predict(self,uid,iid):  #预测评分的函数
        #setdefault的作用是当该用户或者物品未出现过时，新建它的bi,bu,qi,pu，并设置初始值为0
        self.bi.setdefault(iid,0)
        self.bu.setdefault(uid,0)
        self.qi.setdefault(iid,np.zeros((self.K,1)))
        self.pu.setdefault(uid,np.zeros((self.K,1)))
```

```
 rating=self.avg+self.bi[iid]+self.bu[uid]+np.sum(self.qi[iid]*self.pu[uid]) #预测评分公式
        #由于评分范围在1到6，所以当分数大于6或小于1时，返回6,1.
        if rating>6:
            rating=6
        if rating<1:
            rating=1
        return rating

    def train(self,steps=200,gamma=0.035,Lambda=0.15):    #训练函数，step为迭代次数。
        print('train data size',self.mat.shape)
        for step in range(steps):
            print('step',step+1,'is running')
            KK=np.random.permutation(self.mat.shape[0]) #随机梯度下降算法，kk为对矩阵进行随机洗牌
            rmse=0.0
            for i in range(self.mat.shape[0]):
                j=KK[i]
                uid=self.mat[j,0]
                iid=self.mat[j,1]
                rating=self.mat[j,2]
                eui=rating-self.predict(uid, iid)
                rmse+=eui**2
                self.bu[uid]+=gamma*(eui-Lambda*self.bu[uid])
                self.bi[iid]+=gamma*(eui-Lambda*self.bi[iid])
                tmp=self.qi[iid]
                self.qi[iid]+=gamma*(eui*self.pu[uid]-Lambda*self.qi[iid])
                self.pu[uid]+=gamma*(eui*tmp-Lambda*self.pu[uid])
            gamma=0.95*gamma
            print('rmse is',np.sqrt(rmse/self.mat.shape[0]))

    def test(self,test_data):  #gamma以0.95的学习率递减
        print('test data size',test_data.shape)
        writeres = open(filename, "w")
        for i in range(test_data.shape[0]):
            uid=test_data[i,0]
            iid=test_data[i,1]
            predscore = round(self.predict(uid, iid)-1)
            writeres.write(str(predscore) + "\n")
        writeres.close()

a=SVD(train_data,40)
a.train()
a.test(test_data)
```

社交关系的思考

我们对社交关系的使用做了以下两种尝试：

冷启动，对于训练集中未参与评分的用户，预测其评分结果时我们引入社交关系，以其关注的账号的平均打分预测。

```
def ColdStart(userid, movieid):
    if userid in SocialRelation:    # userid 在 社会关系中
        count = 0
```

```
        ratingsum = 0
        for leaderid in SocialRelation[userid]:
            if DataMatrix[UserID2Num[leaderid], MovieID2Num[movieid]] != 0:
                ratingsum += DataMatrix[UserID2Num[leaderid],
MovieID2Num[movieid]]
                count += 1
        if count != 0:
            return ratingsum / count
        else:
            return MovieMeanScore[MovieID2Num[movieid]]
    else:
        return MovieMeanScore[MovieID2Num[movieid]]
```

以及用于**加权预测**，即不再遍历所有其他用户，而在所关注账号中使用相似度加权。不过该方案效果并不好，我们认为这是由于社交关系并不代表用户具有相同的观影偏好，因此不能默认其具有高相似性。

实验结果分析

用户协同过滤结果（RMSE）：

submission4.txt	1.4895220072226922
submission5.txt	1.5955736001816962
submission6.txt	1.5955736001816962
submission7.txt	1.5180811391737694
submission8.txt	1.497461122796132
submission9.txt	1.489788225339001
submission10.txt	1.4872986671105892
submission11.txt	1.4879806044832304
submission12.txt	1.5180811391737694

基于用户协同过滤的最佳结果未RMSE=1.4872，这一方案中主要通过改变近邻数量K的值来调试，最终取K=50。

SVD预测结果（RMSE）：

submission16.txt	2.1228008717843587
submission17.txt	1.4359913667046194
submission18.txt	1.4351962339423872
submission19.txt	1.4352410418639572
submission20.txt	1.4328575589011638
submission21.txt	1.4320135332906845

SVD中最佳效果RMSE=1.4320，主要通过改变特征向量长度、学习率、迭代次数等改进，最终我们尝试的取值为长度40，迭代200次。

注：每次的RMSE都不大相同，该结果是我们经过很多次训练以及调参以后出现的某次最优结果，因为我们采用的是**随机梯度下降**，且迭代次数由于运行时间问题并没有达到饱和。

所作优化

1. 只考虑相似度大于0的项，并采用K近邻计算相似度；
2. 采用两种方法预测（基于**内容**和基于**模型**的协同过滤推荐，即**用户协同过滤**和**SVD**算法）；
3. 调整**SVD**算法的参数，使其达到近似最优。