基于隐变量的推荐模型评分预测报告

作者：黄宇辰　　学号：10185102253

1-方法介绍

1.1 基于隐变量的推荐模型的优点

协同过滤的中心思想就是通过人与事务的评分矩阵，去对人或者物品进行聚类，从而找到相似的人和相似的物品。但是这样的模型有一个重要问题，就是由于评分矩阵的稀疏性，在计算过程中，每一个维度的增减都对结果造成很大影响。

基于隐变量的方法就可以很好的解决这个问题。

1.2 模型中的矩阵分解方法

在代码中，实现了梯度下降法与交替最小二乘法。在训练集上测试模型后，发现交替最小二乘法效果较好。

2-核心代码介绍

环境： python3.7

使用库如下：

```
import pandas as pd
import numpy as np
import math
from math import sqrt
import matplotlib.pyplot as plt
from collections import defaultdict
```

2.1 读取数据

读取数据的类，分别用于读取训练数据与测试数据

```
class clearinfo:
    def __init__(self,train_filepath,test_filepath):
        self.train_filepath = train_filepath
        self.test_filepath = test_filepath
    def getinfo_train(self):
        train_data = pd.read_csv(self.train_filepath,header=0)
        return train_data
    def getinfo_test(self):
        test_data = pd.read_csv(self.test_filepath,header=0)
        return test_data
```

2.2 预处理数据

由于在进行后续隐变量模型的矩阵分解算法时，需要构建评分矩阵。所以先对所有用户与事务进行标号，并在字典中存储对应关系。

```
def grade(df):
    global U_id,id_U,B_id,id_B
    for i in range(0,len(df)):
        if df['user_id'][i] not in U_id:
            U_id[df['user_id'][i]]=i
            id_U[i]=df['user_id'][i]
            df['user_id'][i]=i
        else:
            df['user_id'][i] = U_id[df['user_id'][i]]
        if df['business_id'][i] not in B_id:
            B_id[df['business_id'][i]]=i
            id_B[i]=df['business_id'][i]
            df['business_id'][i]=i
        else:
            df['business_id'][i]=B_id[df['business_id'][i]]
    return df
tr_grade = grade(df_train)
```

```
def grade_te(df):
    for i in range(0,len(df)):
        df['user_id'][i]=U_id[df['user_id'][i]]
        df['business_id'][i]=B_id[df['business_id'][i]]
    return df
te_grade = grade_te(df_test)
```

2.3 隐变量模型的实现

```
class LF:
    def __init__(self,df,k,norm):
        #先生成一个User_Business评分矩阵，这里先构建一个全0矩阵，稍后填充
        self.UB =
np.mat(np.zeros((int(df[['user_id']].max())+1,int(df[['business_id']].max())+1))
)
        #找到对B评论的所有U，和U评论的所有B
        self.B_U = defaultdict(set)
        self.U_B = defaultdict(set)
        for i in range(0,len(df)):
            user,business,stars =int(df['user_id'][i]),int(df['business_id']
[i]),df['stars'][i]
            self.UB[user,business]=stars
            self.B_U[business].add(user)
            self.U_B[user].add(business)
        self.k= k  #选取的k
        self.norm = norm
        #构建预测评分矩阵
        self.User =  np.mat(np.random.uniform(sqrt(1/k),sqrt(5/k),
(self.UB.shape[0],k)))
        self.Business =  np.mat(np.random.uniform(sqrt(1/k),sqrt(5/k),
(self.UB.shape[1],k)))
```

```
    #定义损失函数
    def loss(self):
        ret = self.norm * (np.sum(np.square(self.User)) +
np.sum(np.square(self.Business)))
        #User * Business 的转置
        pred = self.User * self.Business.T
        for i in range(self.UB.shape[0]):
            for j in range(self.UB.shape[1]):
                if self.UB[i,j] != 0:
                    ret += (self.UB[i,j] - pred[i,j]) ** 2
        return ret

    #梯度下降
    #lr学习率，maxd最大迭代深度，th阈值
    def grad_fit(self,lr = 0.01,maxd = 15,th = 100):
        d = 0
        x = []
        loss_val = []
        train_score = []
        val_score = []
        while d < maxd and self.loss() > th:
            for uid in range(1,self.UB.shape[0]):
                grad = 2 * self.norm * self.User[uid]
                for bid in self.U_B[uid]:
                    grad = grad - 2 * (self.UB[uid,bid] - self.User[uid] *
self.Business[bid].T) * self.Business[bid]
                self.User[uid] = self.User[uid] - lr * grad
            for bid in range(1,self.UB.shape[1]):
                grad = 2 * self.norm * self.Business[bid]
                for uid in self.B_U[bid]:
                    grad = grad - 2 * (self.UB[uid,bid] - self.User[uid] *
self.Business[bid].T) * self.User[uid]
                self.Business[bid] = self.Business[bid] - lr * grad
            x.append(d)
            loss_val.append(self.loss())
            train_score.append(self.RMSE_score(tr_grade))
            val_score.append(self.RMSE_score(tr_grade))
            d += 1
        return x,loss_val,train_score,val_score

    #交替最小二乘法
    #maxd最大迭代深度，th阈值
    def als_fit(self,maxd = 25,th = 100):
        d = 0
        x = []
        loss_val = []
        train_score = []
        val_score = []
        while d < maxd and self.loss() > th:
            for uid in range(1,self.UB.shape[0]):
                left = np.mat(np.zeros((1,self.k)))
                right = np.mat(np.zeros((self.k,self.k)))
                for bid in self.U_B[uid]:
                    right += self.Business[bid].T * self.Business[bid]
                    left += self.UB[uid,bid] * self.Business[bid]
                right += self.norm * np.identity(self.k)
                if abs(np.linalg.det(right)) < 1e-6:
                    self.User[uid] = left * np.linalg.pinv(right + self.norm * 
np.identity(self.k))
                else:
                    self.User[uid] = left * np.linalg.inv(right + self.norm * 
np.identity(self.k))
                #采用moore-penrose伪逆
            for bid in range(1,self.UB.shape[1]):
                left = np.mat(np.zeros((1,self.k)))
                right = np.mat(np.zeros((self.k,self.k)))
                for uid in self.B_U[bid]:
                    right += self.User[uid].T * self.User[uid]
                    left += self.UB[uid,bid] * self.User[uid]
                right += self.norm * np.identity(self.k)
                if abs(np.linalg.det(right)) < 1e-6:
                    self.Business[bid] = left * np.linalg.pinv(right + self.norm 
* np.identity(self.k))
                else:
                    self.Business[bid] = left * np.linalg.inv(right + self.norm 
* np.identity(self.k))
                #同上，采用moore-penrose伪逆
            x.append(d)
            loss_val.append(self.loss())
            train_score.append(self.RMSE_score(tr_grade))
            val_score.append(self.RMSE_score(tr_grade))
            d += 1
        return x,loss_val,train_score,val_score

    #计算评价指标RMSE
    def RMSE_score(self,df):
        r = 0
        n = 0
        pred = self.User * self.Business.T
        for i in range(0,len(df)):
            uid,bid,stars =int(df['user_id'][i]),int(df['business_id']
[i]),df['stars'][i]
            if uid < pred.shape[0] and bid < pred.shape[1]:
                r += (pred[uid,bid] - stars) ** 2
                n += 1
        return sqrt(r/n)

    #预测结果
    def pred(self,df_test):
        ans = []
        pred = self.User* self.Business.T
        for idx,row in df_test.iterrows():
            uid,bid = int(row['user_id']),int(row['business_id'])
            if uid < pred.shape[0] and bid < pred.shape[1]:
                ans.append(pred[uid,bid])
            else:
                ans.append(3)
        return ans
```

2.4 得到预测结果

这里提交在交替最小二乘法下的结果。

```
ans = model.pred(df_test)
info = clearinfo(train_filepath,test_filepath)
df_test  = info.getinfo_test()
df_test.drop('date',axis=1,inplace=True)
df_test['stars'] = ans
```

由于通过交替最小二乘法得到的结果中，存在负数评分和大于5的评分结果，所以将负数评分调整为0，大于5的评分结果调整为5。

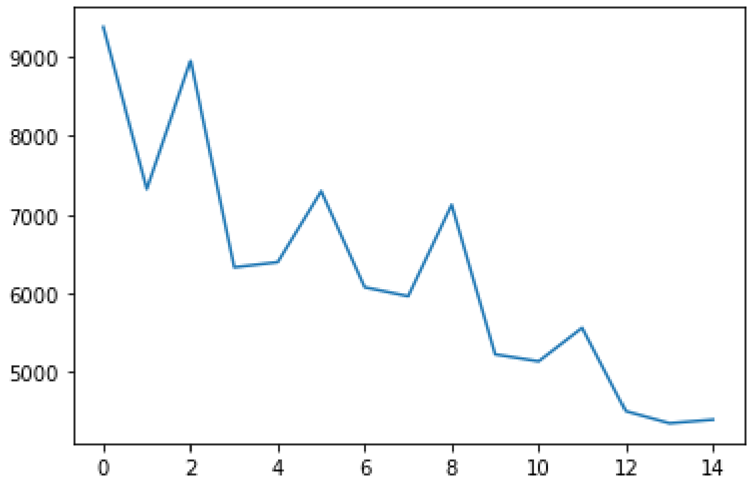
```
for i in range(0,len(df_test)):
    if df_test['stars'][i]>5:
        df_test['stars'][i]=5
    if df_test['stars'][i]<0:
        df_test['stars'][i]=0
```

3-实验结果分析

3.1 梯度下降矩阵分解法

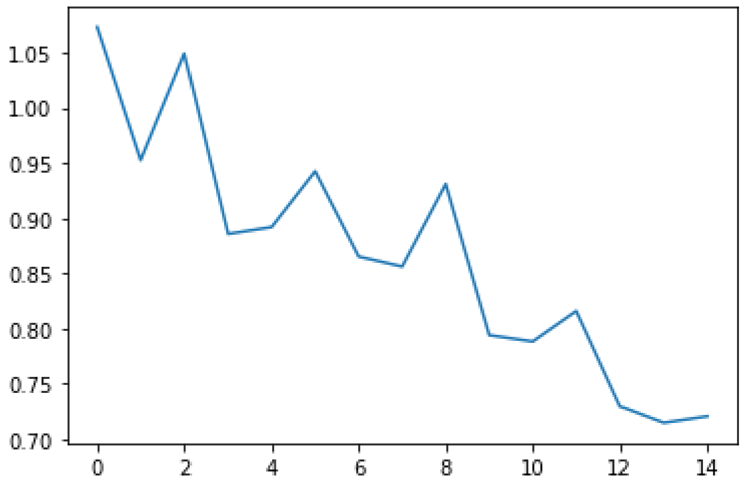
```
plt.plot(x,loss_val)
```

```
[<matplotlib.lines.Line2D at 0x1c3600a1970>]
```



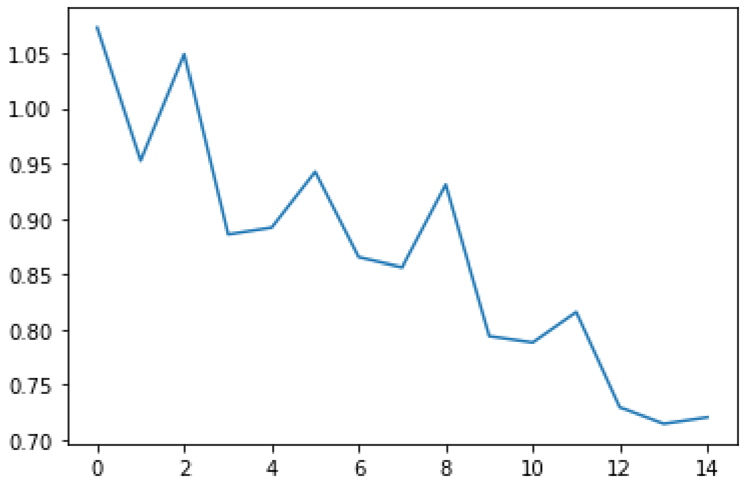
```
plt.plot(x,train_score)
```

```
[<matplotlib.lines.Line2D at 0x1c3440c0ee0>]
```

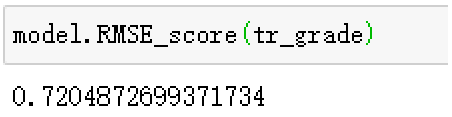


```
plt.plot(x,val_score)
```

```
[<matplotlib.lines.Line2D at 0x1c34410b3d0>]
```



梯度下降法下的RMSE为：



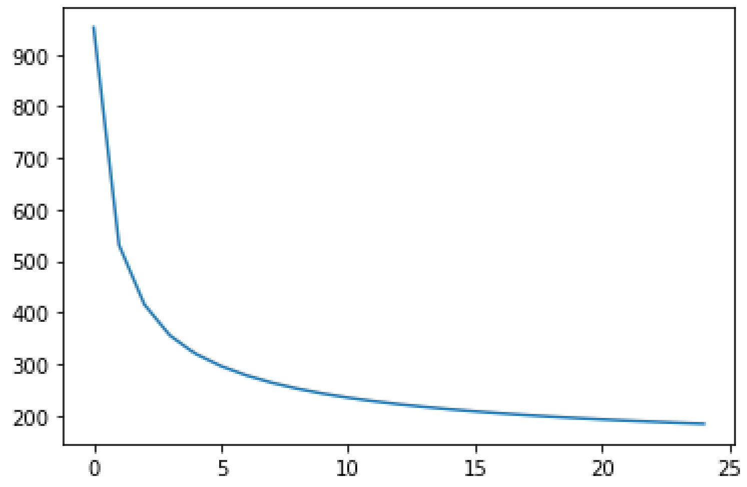
```
model.RMSE_score(tr_grade)
```

```
0.7204872699371734
```

3.2 交替最小二乘法

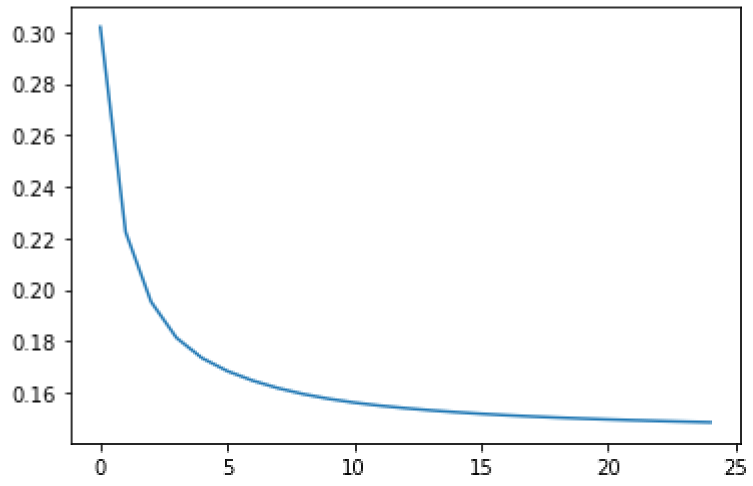
```
: plt.plot(x1,loss_val1)
```

```
: [<matplotlib.lines.Line2D at 0x1c343951790>]
```



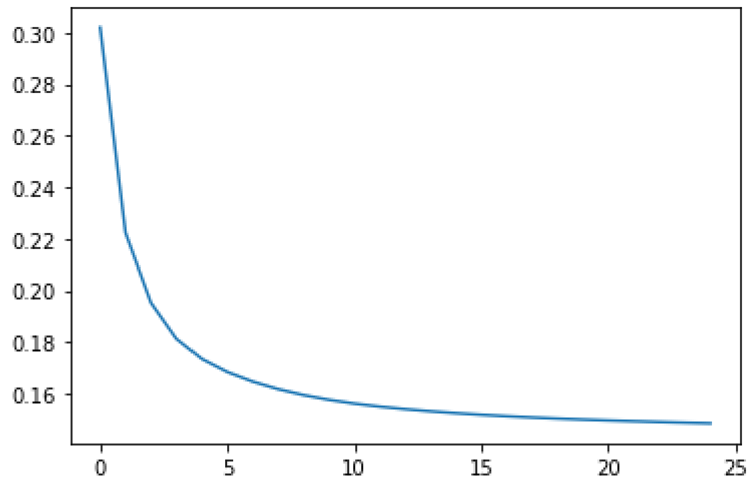
```
: plt.plot(x1,train_score1)
```

```
: [<matplotlib.lines.Line2D at 0x1c343989850>]
```

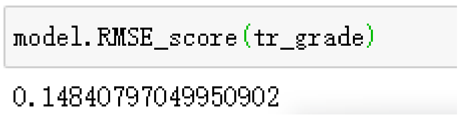


```
: plt.plot(x1,val_score1)
```

```
: [<matplotlib.lines.Line2D at 0x1c3439c19d0>]
```



交替最小二乘法下的RMSE为:



```
model.RMSE_score(tr_grade)
```

```
0.14840797049950902
```

3.3 选取结果

由上述结果，可以看出，交替最小二乘法下的结果效果更好。

文件为： answer_ALS.csv