南开大学

Nankai University

南　　开　　大　　学

计　网　学　院

大数据计算及应用实验报告

推荐系统编程作业

王隽毅、包烜濼、杨坤

年级：2019 级

专业：计算机科学与技术、信息安全

指导教师：杨征路

2022 年 6 月 10 日

目录

一、 概述

(一) 实验目的

本次实验目的在于采用大数据课程上学习到的三种不同算法——基于内容、奇异值分解和协同过滤。

数据集采用的是课程提供的 data-202205 数据集。

为了正确的实现推荐系统需要考虑 RMSE 值、训练时间、空间消耗的问题。

最终得到 test.txt 文件要求的对应用户对相应物品项的评分值。

(二) 选择语言

为了方便进行矩阵相关的操作，经过小组成员讨论后，选择 Python 作为开发语言。

二、 数据集说明

(一) 数据集说明

数据集为课程提供的 data-202205 数据集。

其中包括 train.txt、itemAttribute.txt 两个数据集用于训练模型以及需要给出预测得分项的 test.txt 文件，以及两个格式说明文件。

(二) 数据集基本统计

数据集的基本统计

用户总数：19835

train.txt 评分数：5001507

test.txt 评分数：119010

物品总数：624961

(三) 数据集处理

使用 train_split.py 脚本对 train.txt 进行处理。将 train.txt 中的数据按照 9: 1 的比例分为训练集 _train 和测试集 _test。

train.txt 对于每个用户拥有的评分条目，将其按照比例分配到训练集和测试集当中，其中训练集的数据占原本 train.txt 的 9/10，而测试集的数据占占原本 train.txt 的 1/10。

使用的处理数据集脚本实现如下:

划分训练集、测试集脚本

```
if __name__ == '__main__':
    train_file = open(r".\train.txt", encoding='utf-8', mode='r')
    for_train = open(r".\_train.txt", encoding='utf-8', mode='w')
    for_test = open( r".\_test.txt", encoding='utf-8', mode='w')
    while True:
        line=train_file.readline()
        if line=='':
            break
        item_num=int(line.split('|')[1])
        user_num=int(line.split('|')[0])
        test_num=item_num//10
        for_test.write("%d|%d\n" %(user_num,test_num))
        for _ in range(test_num):
            line=train_file.readline()
            for_test.write(line)
        train_num = item_num-test_num
        for_train.write("%d|%d\n" % (user_num, train_num))
        sort_list=[]
        for _ in range(train_num):
            line = train_file.readline()
            item=int(line.split('  ')[0])
            # score=int(line.split('  ')[1])
            sort_list.append((item,line))
        sort_list=sorted(sort_list,key=lambda x:x[0])
        for ele in sort_list:
            line=ele[1]
            for_train.write(line)
    train_file.close()
    for_train.close()
    for_test.close()
```

使用脚本后，生成训练集 _train.txt 和测试集文件 _test.txt，使得训练集 _train.txt 内包含评分总数达到 4509815 个，测试集文件 _test.txt 内评分总数到达 491692 个。

三、 算法细节

(一) 奇异值分解（SVD）

所谓 SVD 分解，即理论上，可以将任何一个矩阵分解为三个矩阵相乘，分别为左奇异矩阵、右奇异矩阵和奇异值矩阵，其中奇异值矩阵只有主对角线上的值非零，其值表示原矩阵的奇异值。

在本次实验中，根据训练数据集可以得到一个用户-物品的评分矩阵，成为效用矩阵 M。由于并不是每一个用户都对每一个物品进行了评分，故矩阵 M 为稀疏矩阵，含有值的矩阵元素只占较小部分。

本实验中将 SVD 方法应用于对矩阵 M 进行主成分分析。完整的 SVD 算法是对完整的稠密矩阵进行特征分解，而本次实验中得到的效用矩阵 M 为稀疏矩阵，大量矩阵元素的实际值未知，无法精准得到矩阵 M 的奇异值并分解，本实验页不需要用到效用矩阵的特征值，只需要对稀疏矩阵的未知值进行估计，故本次实验采用的方法为 SVD 分解的一个特例，UV 分解：只用到左右两个矩阵 m 行 d 列的 U 和 d 行 m 列的 V，使得两个矩阵的乘积 UV 和 M 在 M 的空白元素上较为接近，即使用 d 维特征近似地去刻画用户和物品之间的评分关系。

设定 epoch 数目，每次训练遍历一次训练集文件，直接将文件作为稀疏矩阵，不再整个读入内存中，根据读取的每个用户实际做出的评分，对逐个 U 和 V 矩阵的元素进行调整，通过计算 RMSE 值衡量 UV 分解对效用矩阵的拟合程度的好坏，使用简易的梯度下降来调整矩阵的数值，多次重复轮询循环，使得最终 RMSE 的值小于阈值或两次轮询之间的减小量小于阈值即视矩阵分解收敛停止训练。

1. 生成转置训练数据集文本

为了简化计算，需要生成以物品编号为行，以对该物品打出过评分的用户编号及相应评分作为列的转置训练集文本，实验使用的方法是先使用脚本将原始的文本数据集遍历，获取不同的物品及相应的评分用户记录在内存，再从内存中逐行写入转置训练集文本文件中。其中为防止内存中需保存的物品为索引的评分记录占用空间过大，在转置过程中应用了分块思想，使用 7 次循环，每次循环记录 100000 个物品的评分记录，再写入文本文件，再进入下一个循环继续记录。

使用的脚本实现如下：

划分训练集、测试集脚本

```
if __name__ == '__main__':
    for_train = open(r".\train.txt", encoding='utf-8', mode='r')
    transpose = open(r".\train_trans.txt", encoding='utf-8', mode='w')
    for i in range(7):
        l = [[] for _ in range(100000)]
        for_train.seek(0,0)
        while True:
            line = for_train.readline()
            if line == '':
                break
            item_num = int(line.split('|')[1])
            user = int(line.split('|')[0])
            print("current user:{}".format(user))
            for _ in range(item_num):
                line=for_train.readline()
                item=int(line.split('  ')[0])
```

```
                score=int(line.split('  ')[1])
                if item in range(i*100000,(i+1)*100000):
                    l[item%100000].append((user,score))
        for index in range(len(l)):
            if i==6 and index>24960:
                break
            if len(l[index])!=0:
                transpose.write("{}|{}\n".format(index+i*100000,len(l[index]
                    )))
                for ele in l[index]:
                    transpose.write("{}  {}\n".format(ele[0],ele[1]))
            else:
                transpose.write("{}|{}\n".format(index+i*100000, 0))
    for_train.close()
    transpose.close()
```

使用脚本后，生成转置训练集 train_trans.txt。

2. 导入数据集并计算矩阵

这一部分在每个 epoch 当中重复进行，直至完成预先设定的 epoch 数。在对划分的训练集训练 100 次迭代进行数据分析，为生成最终的 test 中的评分，对整个 train.txt 进行 25 次迭代。

将本地的数据集和转置后的数据集文本读入，并初始化 UV 矩阵，令维度 d 为 1；实际在模型训练测试中尝试过 d 取值为 2 或 3 等更大值，最终发现当 d 为 1 时 RMSE 值最小，对其理论解释为由于稀疏矩阵中大量实际分数值未知，在试图用 d 作为各个物品的属性特征时试图将特征数量最小化即为将各个物品的共有属性数量最少化能增强 UV 矩阵拟合能力，降低拟合误差。

读取效用矩阵中关于特定用户的所有评分的数据集合，只有这些实际被用户赋予了评分的物品的分数项才参与 RMSE 误差值的计算，于是计算迭代过程为每次读取该行用户的一个评分值，分别在 U 和 V 矩阵中选取其对应的行和列的向量进行点乘得到平方误差，得到该行用户对应的所有评分过的物品在 V 中的对应列取出点乘后得到的误差平方和的导数，令其等于 0，计算得到此时 U 中对应行的各元素的调整后值存入 U 矩阵中，在读下一用户行继续更新 U。

实现如下：

读入用户数索引数据集并迭代 U 矩阵

```
train = open(uv.train_path, mode='r')
train.seek(0, 0)
while True:
    line = train.readline()
    if line == '':
        break
    try:
        user = int(line.split('|')[0])
    except ValueError:
        print(line)
        print(user)
        exit(0)
    print("current u_row:{}".format(user))
    rate_num = int(line.split('|')[1])
```

```
    u_line = user
    cur_pos = train.tell()
    for u_col in range(uv.d):
        up = 0
        down = 0
        for _ in range(rate_num):
            line = train.readline()
            item = int(line.split('  ')[0])
            score = int(line.split('  ')[1])
            up += uv.v[u_col, item] * (score - (
                    np.dot(uv.u[u_line, :], uv.v[:, item]) - uv.u[u_line,
                        u_col] * uv.v[u_col, item]))
                    down += uv.v[u_col, item] ** 2
        if down != 0:
            z_point = up / down
            if z_point > uv.u[u_line, u_col]:
                uv.u[u_line, u_col] += abs(z_point - uv.u[u_line, u_col]) * 1
            else:
                uv.u[u_line, u_col] -= abs(z_point - uv.u[u_line, u_col]) * 1
        if u_col != uv.d - 1:
            train.seek(cur_pos, 0)
train.close()
```

对 V 进行类似上所述 U 的导数为 0 更新迭代计算，实现如下：

读入物品索引数据集并迭代 V 矩阵

```
train_trans = open(uv.train_trans_path, mode='r')
train_trans.seek(0, 0)
while True:
    line = train_trans.readline()
    if line == '':
        break
    try:
        v_col = int(line.split('|')[0])
    except ValueError:
        print(line)
        print(v_col)
        exit(0)
    print("current v_col:{}".format(v_col))
    user_num = int(line.split('|')[1])
    up = [0 for _ in range(uv.d)]
    down = [0 for _ in range(uv.d)]
    user = 0
    for _ in range(user_num):
        line = train_trans.readline()
        user = int(line.split('  ')[0])
        score = int(line.split('  ')[1])
        for v_line in range(uv.d):
            up[v_line] += uv.u[user, v_line] * (score - (
                    np.dot(uv.u[user, :], uv.v[:, v_col]) - uv.u[user, v_line
                        ] * uv.v[v_line, v_col]))
            down[v_line] += uv.u[user, v_line] ** 2
    for v_line in range(uv.d):
        if down[v_line] == 0:
            continue
        z_point = up[v_line] / down[v_line]
        if z_point > uv.v[v_line, v_col]:
            uv.v[v_line, v_col] += abs(z_point - uv.v[v_line, v_col]) * 1
        else:
            uv.v[v_line, v_col] -= abs(z_point - uv.v[v_line, v_col]) * 1
train_trans.close()
```

3. 计算 RMSE

该部分使用 calc_rmse 函数进行计算均方根误差。
导入之前划分的测试集文件，每读入测试集文件中的一行评分项，就进行误差的计算。
误差计算完毕之后，返回对应的误差。
calc_rmse 函数实现如下：

计算 RMSE

```
def calc_rmse(self):
    train = open(self.train_path, mode='r')
    se = 0
    count = 0
    while True:
        line = train.readline()
        if line == '':
            break
        user = int(line.split('|')[0])
        # print(user)
        rate_num = int(line.split('|')[1])
        for _ in range(rate_num):
            line = train.readline()
            item = int(line.split('  ')[0])
            score = int(line.split('  ')[1])
            error = (score - np.dot(self.u[user, :], self.v[:, item])) ** 2
            se += error
            # print("error:{}".format(error))
        count += rate_num
    rmse = (se / count) ** 0.5
    train.close()
    return rmse
```

4. 计算 test.txt 中所需项的得分

该部分使用 display 函数进行计算得分。遍历 test.txt 测试文件，每遍历到对应用户对应物品项的一行，就进行一次估计值计算，写入结果文件中展示。

将 U 矩阵中所求用户对应的行与 V 矩阵对应所求物品项对应的列进行矩阵点乘，即可得到对应用户对应物品的预测得分，将该得分保存在最终文件当中。

display 函数实现如下：

计算预测得分

```
def display(self):
    valid = open(self.test_path, mode='r')
    res = open(self.res_path, mode='w')
    while True:
        line = valid.readline()
        if line == '':
            break
        user = int(line.split('|')[0])
        rate_num = int(line.split('|')[1])
        res.write(line)
        for _ in range(rate_num):
            line = valid.readline()
            item = int(line.split('  ')[0])
            score = np.dot(self.u[user, :], self.v[:, item])
            line = "{}  {}\n".format(item, score)
            res.write(line)
    valid.close()
    res.close()
```

5. 保存矩阵数据

为了保存矩阵训练后的结果，避免之后计算需要重新计算，就将矩阵保存在本地。

使用 save_mat 的函数进行矩阵数据保存。

save_mat 的函数实现如下：

保存矩阵信息

```
def save_mat(self):
    save = open(self.save_path, mode='w')
    for u_row in range(self.u_dim):
        for u_col in range(self.d):
            save.write("%f|" % self.u[u_row, u_col])
        save.write("\n")
    save.write("\n")
    for v_row in range(self.d):
        for v_col in range(self.v_dim):
            save.write("%f|" % self.v[v_row, v_col])
        save.write("\n")
    save.write("\n")
    save.close()
```

(二) 基于物品属性的线性回归

相比上述使用 epoch 循环训练，该方法使用线性回归来确定矩阵的数值。

有效利用了数据集中 itemAttribute.txt 的数据。

同样利用 _train 数据集进行回归训练，再利用 _test 数据集测试估计分数计算与正确分数的误差。

1. 导入 itemAttribute.txt 数据集并处理物品属性值矩阵

打开训练集 itemAttribute.txt 文件，遍历该文件的过程中设置物品属性值矩阵的值。

若 itemAttribute.txt 文件中的 Attribute 为 None，则视为该物品的该属性的数值为 0。

由于视为物品属性值的数值的绝对值大小变化幅度较大，故数据预处理过程中先对数据进行归一化处理，方便后续训练，并转置获得正确行列结构的矩阵。

导入 itemAttribute.txt 并处理物品属性矩阵，实现如下:

导入 itemAttribute.txt 数据集并生成物品属性矩阵

```
LR = LinearRegressionForUsers()
feature = open(LR.feature_path, mode='r')
feature_mat = np.zeros((624961,2))
for line in feature:
    item = int(line.split('|')[0])
    attr1 = line.split('|')[1]
    attr2 = line.split('|')[2].strip()
    attr1 =int(attr1) if attr1!='None' else 0
    attr2 =int(attr2) if attr2!='None' else 0
    feature_mat[item,0]=attr1
    feature_mat[item,1]=attr2
feature.close()
min = [np.min(feature_mat[..., i]) for i in range(feature_mat.shape[1])]
max = [np.max(feature_mat[..., i]) for i in range(feature_mat.shape[1])]
#print(feature_mat.shape[1])
feature_mat = np.array([[(ele - min[i]) / (max[i] - min[i]) for ele in
    feature_mat[..., i]] for i in range(feature_mat.shape[1])])
feature_mat=feature_mat.transpose()
#print(feature_mat)
```

2. 导入训练数据集并生成线性回归模型

处理完毕特征矩阵后，导入训练数据集。

开始进行线性回归计算，结合之前处理得到的物品属性矩阵，对每一个用户训练一个具有两个参数的线性回归模型，将生成好的模型保存入列表，用于后续对每个用户的未评分物品进行分数估计。

导入后，成功生成基于物品属性的二元线性回归模型。

导入训练数据集，实现如下：

导入训练数据集

```
train = open(LR.train_path, mode='r')
LR_set=[]
while True:
    line = train.readline()
    if line == '':
        break
    user = int(line.split('|')[0])
    print("current user:{}".format(user))
    rate_num = int(line.split('|')[1])
    current_x=np.ones((rate_num,2))
    current_y=np.ones(rate_num)
    for i in range(rate_num):
        line = train.readline()
        item = int(line.split('  ')[0])
        score = int(line.split('  ')[1])
        current_y[i]=score
        current_x[i,0]=feature_mat[item,0]
        current_x[i, 1] = feature_mat[item, 1]
    LR.fit_gd(current_x, current_y, eta=0.005)
    LR_set.append(LR)
    LR=LinearRegressionForUsers()
    # if user==0:
    #     break
train.close()
```

线性回归方法的实现如下，关键部分为梯度下降的实现，每次通过导数公式计算好梯度，将参数 θ 参数减去梯度，重复迭代；当迭代次数超过预设次数或前后 MSE 的差值小于预设值时停止迭代，该用户的回归模型训练完成：

线性回归方法

```
def fit_normal(self, X_train, y_train):
    assert X_train.shape[0] == y_train.shape[0], \
        "the size of X_train must be equal to the size of y_train"
    X_b = np.hstack([np.ones((len(X_train), 1)), X_train])
    self._theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_train)
    self.intercept_ = self._theta[0]
    self.coef_ = self._theta[1:]
```

```
        return self

    def fit_gd(self, X_train, y_train, eta=0.01, n_iters=1e4):
        assert X_train.shape[0] == y_train.shape[0], \
            "the size of X_train must be equal to the size of y_train"

        def J(theta, X_b, y):
            try:
                return np.sum((y - X_b.dot(theta)) ** 2) / len(y)
            except:
                return float('inf')

        def dJ(theta, X_b, y):
            return X_b.T.dot(X_b.dot(theta) - y) * 2. / len(X_b)

        def gradient_descent(X_b, y, inital_theta, eta, n_iters=1e4, epsilon
            =1e-8):
            theta = inital_theta
            cur_iter = 0
            while cur_iter < n_iters:
                gradient = dJ(theta, X_b, y)
                last_theta = theta
                theta = theta - eta * gradient
                yPred = X_b.dot(theta)
                num = (yPred - y).dot(yPred - y)
                d = y.size
                self.MSE.append(num / d)
                self.theta_list.append((cur_iter + 1))
                if (abs(J(theta, X_b, y) - J(last_theta, X_b, y)) < epsilon):
                    break
                cur_iter += 1
            return theta

        X_b = np.hstack([np.ones((len(X_train), 1)), X_train])
        initial_theta = np.zeros(X_b.shape[1])
        self._theta = gradient_descent(X_b, y_train, initial_theta, eta,
            n_iters)
        self.intercept_ = self._theta[0]
        self.coef_ = self._theta[1:]
        return self
```

3. 计算 test.txt 中所需项的得分

该部分使用 predict 函数进行计算得分。

遍历 test.txt 文件，每遍历到对应用户对应物品项的一行，就进行一次对应的计算。

使用线性回归模型和特征矩阵计算，由于 itemAttribute 文件中存在不少物品的属性值条目缺失，故训练过程中相应的物品属性条目数值为 0 未对模型训练做出贡献，且在估分过程中直接自变量为 0 将参数影响消除，故在最终估分过程中若分数结果为模型参数的截距值则将分数值直接标记为-1，后续综合多方法时去除该分数条目采用其它算法；将得分保存在最终文件当中。

导入 test.txt 文件并进行预测得分，实现如下：

导入 test.txt 文件并预测

```
    valid = open(LR.validate_path, mode='r')
    res=open(LR.res_path,mode='a')
    # se=0
    # count=0
    while True:
        line = valid.readline()
        if line == '':
            break
        user = int(line.split('|')[0])
        print("current test_user:{}".format(user))
        rate_num = int(line.split('|')[1])
        test_x =np.ones((rate_num,2))
        # test_y = np.ones(rate_num)
        LR=LR_set[user]
        items=[]
        for i in range(rate_num):
            line = valid.readline()
            item = int(line.split('  ')[0])
            # score = int(line.split('  ')[1])
            items.append(item)
            # test_y[i]=score
            test_x[i, 0] = feature_mat[item, 0]
            test_x[i, 1] = feature_mat[item, 1]
        y_predT = LR.predict(test_x)
        # se += (y_predT - test_y).dot(y_predT - test_y)
        # count+=rate_num
        res.write("{}|{}\n".format(user,rate_num))
        for ele0,ele1 in zip(y_predT,items):
            if ele0==LR.intercept_:
                ele0=-1
            res.write("{}  {}\n".format(ele1,ele0))
        # print(LR.intercept_)
        # if user==0:
        #     break
    valid.close()
    res.close()
```

predict 函数实现如下：

predict 函数

```
    def predict(self, X_predict):
        assert self.intercept_ is not None and self.coef_ is not None, \
            "must fit before predict!"
        assert X_predict.shape[1] == len(self.coef_), \
            "the feature number of X_predict must be equal to X_train"

        X_b = np.hstack([np.ones((len(X_predict), 1)), X_predict])

        return X_b.dot(self._theta)
```

(三) UV 分解与线性回归的结果合并

将 UV 分解的结果和基于物品属性线性回归的结果按照规则：若线性回归估计的分数值为-1，则该分值采用 UV 分解模型得到的分值，否则采用线性回归模型得到的分值；即可得到使用 UV 分解并结合使用 itemAttribute.txt 文件的结果。

合并结果的过程中，可以测算组合方案在验证集上的 RMSE 值以评估最终算法效果。

合并 UV 分解和线性回归的实现如下：

合并结果

```
    uv_path = r'.\res-valid-100run.txt'
    regre_path = r'.\res-valid-regre-0.txt'
    test_path = r'.\_test.txt'
    res_path =  r'.\res-combine.txt'
    res1 = open(uv_path, mode='r')
    res2 = open(regre_path, mode='r')
    res0=open(res_path,mode='w')
    res3=open(test_path,mode='r')
    se=0
    count=0
    while True:
        line1=res1.readline()
        line2=res2.readline()
        line3=res3.readline()
        if line1=='':
            break
        if line1.find('|')!=-1:
            res0.write(line1)
        else:
            item = int(line1.split('  ')[0])
            score1 = float(line1.split('  ')[1].strip())
            score2=float(line2.split('  ')[1].strip())
            if score2==-1.0:
```

```
                if score1<0:
                    score1=0.0
                if score1>100:
                    score1=100.0
                score=score1
            else:
                if score2<0:
                    score2=0.0
                if score2>100:
                    score2=100.0
                score=score2
            res0.write("{}  {}\n".format(item,score))
            real_score=int(line3.split('  ')[1])
            se+=(real_score-score)**2
            count+=1
    rmse=(se/count)**0.5
    print(rmse)
    res0.close()
    res1.close()
    res2.close()
    res3.close()
```

（四） 基于用户的协同过滤（UserCF）

由于用户数为 19835，而项目数为 624961。项目的数量级远大于用户的数量级，所以采取基于用户的协同过滤能有效减少计算量。

基于用户的协同过滤需要构造用户相似度矩阵。要预测某个用户对某项物品的打分，可以根据用户相似度矩阵找出数个与该用户相似的其他用户。根据相似的其他用户对该物品的打分来预测所求用户的打分。

使用余弦相似度来评估项之间的相似性。

1. 计算每个用户评分均值

由于用户打分的习惯问题，打分可能存在统一偏高或偏低的问题，先计算平均值，再将评分减去平均值，更能反映用户真实的喜好情况。

将本地的训练数据集导入，计算每个用户的评分均值。

该过程由 calcMean 函数实现。

calcMean 函数，实现如下：

计算每个用户评分均值

```
    def calcMean(self):
        '计算所有打分的均值'
        dataFile = open(self.dataFilePath,'r')
        lines = dataFile.readlines()
        lineNum = 0
        while lineNum<len(lines):
            self.userNum+=1
            #print(lines[lineNum].strip().split())
            uid,count = lines[lineNum].strip().split('|')
            temp =dict()
            for i in range(int(count)):
                item = lines[i+lineNum+1].strip().split()
                temp[item[0]]=int(item[1])
            lineNum+=(int(count)+1)
            self.rateNum+=len(temp)
            self.rateSum+=sum(list(temp.values()))
            self.userRateVectors.append(temp)
        self.rateMean = self.rateSum/self.rateNum
        dataFile.close()
        #print(self.userNum)
```

2. 预测与构建相似度矩阵

如果需要预测某个用户（预测用户）对某个物品，首先先寻找拥有对该物品打分过的用户（参考用户），若预测用户与参考用户之间没有计算过相似度，则使用余弦相似度公式进行计算，并保存到相似度矩阵中。如果已经计算过相似度则不重复计算。

由于用户与用户之间的相似度可以用对角矩阵保存，从而实现进一步节省空间消耗。

寻找出设定的 N 个最相似用户，然后根据这些参考用户的评分进行预测。

预测过程中，遍历 test.txt 文件，每遍历到对应用户对应物品项的一行，就进行一次对应的预测计算。

该过程由 predict 函数实现。

predict 函数，实现如下：

预测用户评分

```
    def predict(self):
        '对_test.txt中item进行打分预测'
        testFile = open(self.testFilePath,'r')
        resFile = open(self.resultFilePath,'w+')
        data = testFile.readline()
        time = 0
        while(data!=''):
                time+=1
                resFile.write(data)
                uid,count = data.strip().split('|')
                userVector = self.getUserVector(uid)
                for i in range(int(count)):
                    item = testFile.readline().strip().split(' ')[0]
                    #output itemid to resuslt file
                    print('to predict:',uid,item)
                    simList = self.getSimUsers(uid,userVector,item)
                    if(self.allSame):
                        rate = self.sameRate
                        self.allSame=False
                    else:
                        weightsum ,Sum=0,0
                        for i in range(len(simList)):
                            Sum+=simList[i][0]*simList[i][1]
                            weightsum+=simList[i][0]
                            #rate =self.rateMean+self.userDeviation+self.
                                itemDeviation+ Sum/weightsum
                            rate = Sum/weightsum
                    if rate>100:
                        rate = 100
                    if rate<=0:
                        rate = 0
                    resFile.write(item+'  '+str(rate)+'\n')
                data = testFile.readline()
        testFile.close()
        resFile.close()
```

获取与预测用户最为相似的参考用户集合的过程由 getSimUsers 函数实现。
getSimUsers 函数，实现如下：

获取参考用户集合

```
    def getSimUsers(self,userid,userVector,itemID):
        '获取与uID最为相似的user集合N'
        res = list()
        Sum,num=0,0
        for i in range(len(self.userRateVectors)):
            if itemID in self.userRateVectors[i].keys():
                Sum+=self.userRateVectors[i][itemID]
                num+=1
                itemTemp = self.userRateVectors[i].copy()
                #归一化
                m = sum(list(itemTemp.values()))/len(itemTemp)
                for j in itemTemp.keys():
                    itemTemp[j]-=m
                #print(temp)
                userTemp = userVector.copy()
                #归一化
                m=sum(list(userTemp.values()))/len(userTemp)
                self.userDeviation = m-self.rateMean
                for j in userTemp.keys():
                    userTemp[j]-=m
                if sum(itemTemp.values())!= 0 and sum(userTemp.values())!=0:
                    s = self.sim(userTemp,itemTemp,int(userid),int(i))
                    res.append((s,self.userRateVectors[i][itemID]))
                    res.sort(key=takeFirst,reverse=True)
                    if len(res)>self.N:
                        res.pop()
                elif sum(userTemp.values())==0:
                    self.allSame=True
                    self.sameRate = sum(userVector.values())/len(userVector)
        try:
            self.itemDeviation = Sum/num - self.rateMean
        except ZeroDivisionError :
            print('Sum:',Sum)
            print('Num',num)

        return res
```

3. 计算 RMSE

该部分使用 calcRMSE 函数进行计算均方根误差。
导入之前划分的测试集文件，每读入测试集文件中的一行评分项，就进行误差的计算。
误差计算完毕之后，返回对应的误差。
calcRMSE 函数实现如下:

计算 RMSE

```
def calcRMSE(self):
    Sum =0
    num =0
    resFile = open(self.resultFilePath,'r')
    testFile = open(self.testFilePath,'r')
    data1 = testFile.readline()
    data2 = resFile.readline()
    while(data1!='' and data2!=''):
        count = data1.strip().split('|')[1]
        num+=int(count)
        for i in range(int(count)):
            item1=testFile.readline().strip().split()[1]
            item2=resFile.readline().strip().split()[1]
            Sum+=(float(item2)-float(item1))**2
        data1=testFile.readline()
        data2 = resFile.readline()
    rmse = math.sqrt(Sum/num)
    resFile.close()
    testFile.close()
    return rmse
```

四、 实验结果

(一) RMSE

算法	RMSE(验证集上)
奇异值分解	31.28
线性回归	29.89
协同过滤	43.02
奇异值分解 + 线性回归	28.97

表 1: 不同算法的 RMSE 对比表

(二) 训练时间

算法	运行时间
奇异值分解（epoch=100）	2.5
线性回归	1.3
协同过滤	7.2
奇异值分解 + 线性回归	3.8

表 2: 不同算法在训练集上的训练时间对比表 (单位: 小时)

(三) 空间消耗

算法	消耗空间
奇异值分解	$0(N^2+M^2)$
线性回归	$0(2N)$
协同过滤	$O(N^2)$
奇异值分解 + 线性回归	$0(N^2+M^2)$

表 3: 不同算法的空间消耗对比表

五、 算法分析

(一) 理论分析

设用户数为 N，物品数为 M。

使用 SVD 算法，需要保存 U、V 两个矩阵，两个矩阵的空间复杂度分别为 $0(N^2)$ 和 $0(M^2)$。所以总的空间复杂度为 $0(N^2+M^2)$。

而使用基于物品属性的线性回归算法，需要保存每个用户的两个属性，空间复杂度为 $0(2N)$。

使用下三角矩阵保存协同过滤的相似度矩阵，若是基于用户，则需要保存的矩阵的空间复杂度为 $0(N^2)$；若是基于物品，则需要保存的矩阵的空间复杂度为 $0(M^2)$。显然，由于课程提供的数据集用户数 N 远小于物品数 M，使用基于用户的协同过滤，空间复杂度为为 $0(N^2)$ 远小于基于物品的协同过滤的空间复杂度 $0(M^2)$。

基于用户的协同过滤算法计算用户相似度矩阵需要 $0((N-1)!M)$ 的时间复杂度。而其他两个算法涉及机器学习，训练所需要的时间与参数的设置以及收敛的速度有关。

(二) 实验结果分析

由于协同过滤在验证集上的 RMSE 误差值较大，本次实验最终采取的是奇异值分解与线性回归相结合的推荐系统算法。

消耗时间为 25 次迭代的 UV 分解模型的 45mins，加上学习率为 0.005 的线性回归模型 1h 20mins 共为 2h 5mins。

空间消耗为 $0(N^2+M^2)$。

表现较好。将其对 test.txt 中预测评分项的预测结果作为最终答案。

六、 总结

通过本次实验，小组成员深刻了解了奇异值分解、线性回归、协同过滤应用于推荐系统的原理，学会了计算余弦相似度。

尝试使用了 itemAttribute.txt 中的 Attribute 提高算法的性能。

小组成员学习了基于用户和基于物品的推荐系统两种推荐方式的特点和适用场景。小组成员学会了实现基于物品属性的线性回归和基于用户的协同过滤的方法。

本次实验为小组成员将来的学习奠定了坚实的基础。