机器学习与数据挖掘 作业四

吴天 19334019

1 Abstract

本次实验实现了 User-based CF 和 Item-based CF 两个版本的协同滤波算法，并使用 MovieLens Latest Small Dataset、MovieLens 100K Dataset 数据集验证了协同滤波算法在上述数据集上的性能。同时，还考虑了不同近邻个数 k 的情况下的预测效果。但最终协同滤波算法的结果并不理想。

2 Assignment

(1). 实现协同滤波算法，分别实现 User-based CF 和 Item-based CF 两个版本。

(2). 从 GroupLens 网站 https://grouplens.org/datasets/movielens/ 找到自己认为合适的 2 个 MovieLens 数据集版本（或者找其他任意更感兴趣的数据集），进行如下分析:

(a) 推荐算法的评分预测效果用 RMSE 进行度量。

(b) 考虑在不同的近邻个数 k 的情况下，User-based CF 和 Item-based CF 的实验效果，并进行对比，分别找出 User-based CF 和 Item-based CF 的最佳的近邻个数 k。

3 Method

3.1 Dataset

本次实验采用 MovieLens Latest Small Dataset、MovieLens 100K Dataset 两个数据集。

其中 MovieLens Latest Small Dataset 共有 610 个用户，9742 部电影，100836 个评分。评分为 0.5 stars - 5.0 stars，间隔为 0.5 stars。

MovieLens 100K Dataset 共有 943 个用户，1682 部电影，100000 个评分。并且每个用户至少为 20 部电影进行评分。评分为 1 stars - 5.0 stars，间隔为 1 stars。

实验数据可以在https://grouplens.org/datasets/movielens/获取。

3.2 Divide the train set and the test set

采用 PPT 中的测试集和训练集的划分方法，将数据集中用户-电影的评分矩阵挖去一部分作为测试集，并且仅对测试集中原本就包含的数据进行预测，如图 1 所示：

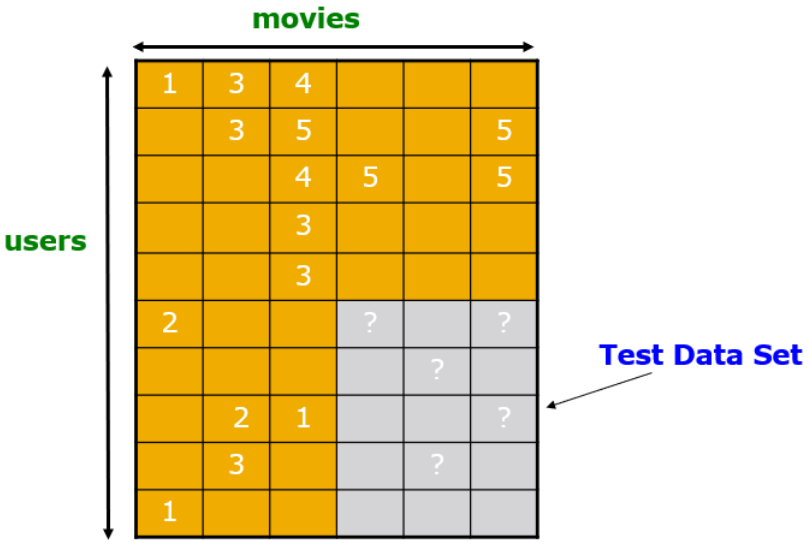


图 1: 训练集和测试集的划分

3.3 CollaboratIve Filtering

CollaboratIve Filtering 协同过滤是通过群体的行为来找到某种相似性 (用户之间的相似性或者标的物之间的相似性)，通过该相似性来为用户做决策和推荐。基于协同过滤的两种推荐算法，核心思想是很朴素的” 物以类聚、人以群分” 的思想。所谓物以类聚，就是计算出每个标的物最相似的标的物列表，我们就可以为用户推荐用户喜欢的标的物相似的标的物，这就是基于物品 (标的物) 的协同过滤。所谓人以群分，就是我们可以将与该用户相似的用户喜欢过的标的物的标的物推荐给该用户 (而该用户未曾操作过)，这就是基于用户的协同过滤，如图 2 所示：

本次实验实现的协同过滤算法可分为以下 2 个步骤：

(1). 计算 similarity(用户间或项间相似度)：采用 pearson 相关系数。

(2). 计算 prediction(预测评分)：用户对未评分电影的预测评分。

3.3.1 Compute Similarity

在这次实现中，我采用了 Pearson correlation coefficient 来度量用户与用户之间，或是项与项之间的相关性。皮尔逊相关系数的大小位于 $[-1, 1]$ 区间内，其绝对值越大，表

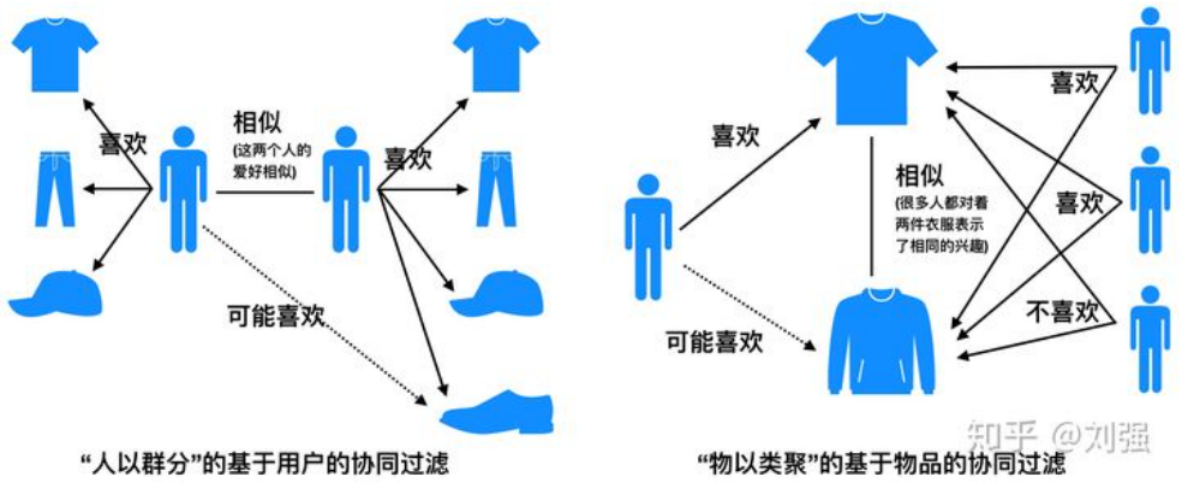


图 2: " 物以类聚、人以群分" 的朴素协同过滤推荐

明二者之间的相关性越高，皮尔逊相关系数为正值表示正相关，为负值表示负相关，当且仅当二者有严格线性关系时为绝对值等于 1。皮尔逊相关系数可以通过如下公式计算:

$$sim(x,y) = \frac{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})(r_{ys}-\overline{r_y})}{\sqrt{\sum_{s\in S_{xy}}(r_{xs}-\overline{r_x})^2}\sqrt{\sum_{s\in S_{xy}}(r_{ys}-\overline{r_y})^2}}$$

其中，S_{xy} 表示 xy 都有评分的集合，$\overline{r_x}$ 和 $\overline{r_y}$ 是 x，y 的平均评分。

3.3.2 Predict Score

预测评分采用加权平均的方法，Item-based CF 中公式如下:

$$r_{xi} = \frac{\sum_{y\in N(x;i)} S_{xy}\cdot r_{yi}}{\sum_{y\in N(x;i)} S_{xy}}$$

其中，S_{xy} 是用户 x 和用户 y 之间的相似度，r_{yi} 是用户 y 对项 i 的评分，$N(x;i)$ 是评分过项 i 的与用户 x 相似的用户集合。

Item-based CF 中公式如下:

$$r_{xi} = \frac{\sum_{j\in N(i;x)} S_{ij}\cdot r_{xj}}{\sum_{j\in N(i;x)} S_{ij}}$$

其中，S_{ij} 是项 i 和项 j 之间的相似度，r_{xj} 是用户 x 对项 j 的评分，$N(i;x)$ 是用户 x 评分过的与 i 相似的项集合。

3.3.3 User-based CF

基于用户的协同过滤算法旨在寻找相似的用户，然后在相似的用户间推荐物品。

(1). similarity: 计算用户间的相似度。每个用户都有一个已评价过的项 list，那么该 list 就是用户的一个属性向量，用户的相似度就是该向量间的相似度。

(2). prediction：假设用户 A 和 B、C 是相似用户。假设 Item 是 B、C 评价过但 A 未评价过的物品。那么我们就可以预测 A 的评分。以 B、C 和 A 的相似度为权重，计算 B、C 对物品的评分均值即可。

3.3.4 Item-based CF

基于物品的协同过滤算法旨在寻找相似的物品，然后向目标用户推荐其已购买的物品的相似物品。

(1). similarity: 提取所有用户对 Item1, Item2, 两个物品的评分，每个物品对应一条评分向量，向量间的相似度就是物品间的相似度。(注意计算向量间相似度时，必须元素对应，即某个用户必须同时对两个物品进行了评分)

(2). prediction: 假设目标用户评价过 Item1, Item2，未评分 Item3。以 Item1 和 Item2 与 Item3 的相似度为权重，用户对 Item1 和 Item2 的评分均值即为目标用户对 Item3 的预测评分。

3.4 Evalution Metric

我们使用 RMSE（Root Mean Squared Error）即均方根误差来衡量算法预测评分结果的效果，在整个评估过程中，原始数据集提供的评分作为基本事实。均方根误差是预测值与真实值偏差的平方与观测次数比值的平方根，衡量的是预测值与真实值之间的偏差，并且对数据中的异常值较为敏感，采用如下公式计算:

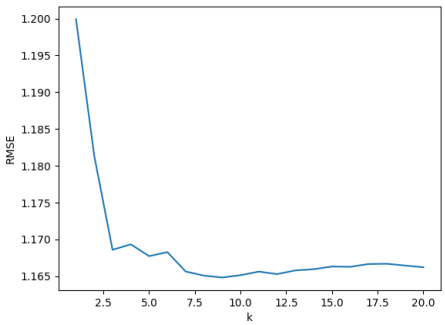
$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \widehat{y_i})^2}$$

其中，m 是预测点的个数，y_i 是预测值，$\widehat{y_i}$ 是实际值。RMSE 值越小，代表预测的效果越好。

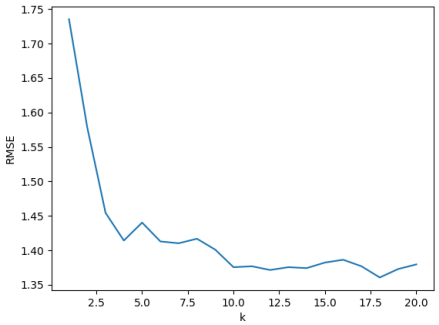
4 Result

4.1 Nonlinear Classfier and Regularize

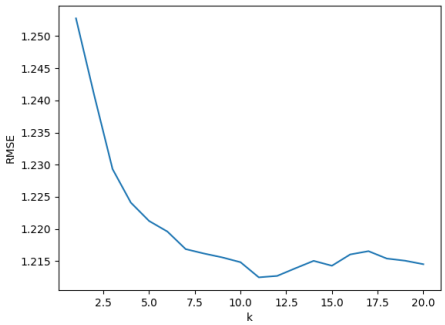
本次实验采用的数据集是 MovieLens Latest Small Dataset 和 MovieLens 100K Dataset 两个数据集。在这 2 个数据集上，分别应用基于用户的协同滤波算法，和基于物品的协同滤波算法，绘制预测结果的 RMSE 随近邻个数 K(120) 变化的图，如图 3 所示：



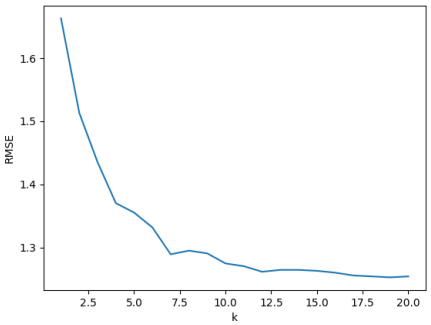
(a) Latest Small Dataset 中使用 User-based CF，RMSE 随近邻个数 K 变化图



(b) 100K Dataset 中使用 User-based CF，RMSE 随近邻个数 K 变化图



(c) Latest Small Dataset 中使用 Item-based CF，RMSE 随近邻个数 K 变化图



(d) 100K Dataset 中使用 Item-based CF，RMSE 随近邻个数 K 变化图

图 3: User-based CF 和 Item-based CF 两种协同滤波算法在 MovieLens Latest Small Dataset 和 MovieLens 100K Dataset 上 RMSE 随近邻个数 K 变化图

从运行结果中得到当近邻个数 $k \in [1, 20]$ 时，最小的 RMSE 值以及对应的 k 值，如表格所示：

	Latest Small(User-based)	100K(User-based)	Latest Small(Item-based)	100K(Item-based)
k	9	18	11	19
min RMSE	1.164834506	1.360625266	1.212479581	1.25250678

从实验结果中，我们可以看到，一开始 K 值较小的时候 RMSE 的值是比较大的，

随着 K 值的不断增大，RMSE 的值会趋向于稳定。我在实现协同滤波算法计算相似度后，加权平均求预测分数的时候，只计算相似度为正值的用户或项的加权平均，即只考虑与代求分数正相关的分数。为什么要这样实现呢？因为当 K 值过大，在求加权平均的时候有可能会考虑进很多负相关的数据，导致加权平均的分母 (相似度之和) 为零或负数，进而导致预测的结果出错。并且，只考虑正相关的数据，也符合算法中找到相似用户，或者相似项的思想。因此，实验结果中，一开始 RMSE 值较大，是因为 K 值较小，参考的相似数据较少，存在较多偶然性，预测的分数不那么精确，而随着 K 值的不断增大，RMSE 的值会趋向于稳定，是由于此时大部分相似数据已经被选取完了，随着 K 值增大，新加入参与加权平均的数据在相似度上已经非常小了，对分数预测的结果已经影响不大。

此外，由于划分测试集和训练集的时候，将一些数据删除了，并且删除不存在随机性，待测用户和待测电影的数据被删了太多，有的分数变得无法预测。一方面，在 User-based CF 中不存在评分过该电影的用户或者评分过该电影的用户中，没有与待预测用户看过相同电影；另一方面，在 Item-based CF 中，该用户没有评分过其他电影，或者与待测电影类似的电影，但该用户没有评分过。由于每个用户看过的电影数目较少，总电影数较多，这些情况出现非常常见（特别是 User-based CF 存在的问题），就算数据没被删除也容易出现这种情况，但我的数据集的划分方法更加加剧了这些情况，更严格的做法是应该要在划分测试集和训练集时加入随机性。如果出现了无法预测的问题，会设置预测该点评分为 3 分。

如果把所有的 predict 分数都设置为 3，得到结果是在 MovieLens Latest Small Dataset 和 MovieLens 100K Dataset 这两个数据集上 RMSE 分别是 1.143749651499891 和 1.124251093148722，全部蒙 3 分比两种算法预测的最好的 K 值下的结果还要好，说明这次实验的效果非常差，还不如直接蒙……此外，还尝试在得到结果后，根据实际评分是离散量，对结果进行微调，但提升效果也不理想.

5 code

代码实现中，将 CollaboratIve Filtering 算法作为一个类，初始的参数有近邻个数 K、训练数据、测试数据，训练数据和测试数据都为 $users \times items$ 的矩阵，如图 4 所示：

```
class cf:
    def __init__(self, k, data, test):
        self.k = k
        self.data = data
        self.test = test
        self.pridicts = copy.deepcopy(test)
```

图 4: CF 类

5.1 CollaboratIve Filtering

方法包含了 CollaboratIve Filtering 的两个步骤：计算 similarity 和计算预测评分。对于每个待求的评分，计算相似度并进行预测，如果无法预测，则设置它的预测评分为 3 ，如图 5 所示：

```python
def user_base(self):
    r,c = self.pridicts.shape
    for i in range(r):
        for j in range(c):
            if(self.pridicts[i][j]):
                sims = self.count_sim(i,j)
                kSimIndex = heapq.nlargest(self.k, range(len(sims)), sims.take)
                fenzi = 0
                fenmu = 0
                for similar in kSimIndex:
                    if(sims[similar] > 0):
                        fenzi += self.data[similar][j] * sims[similar]
                        fenmu += sims[similar]
                if(fenmu == 0):
                    self.pridicts[i][j] = 3
                else:
                    self.pridicts[i][j] = fenzi/fenmu
```

(a) user-based

```python
def item_base(self):
    r,c = self.pridicts.shape
    for j in range(c):
        for i in range(r):
            if(self.pridicts[i][j]):
                sims = self.count_sim(i,j)
                kSimIndex = heapq.nlargest(self.k, range(len(sims)), sims.take)
                fenzi = 0
                fenmu = 0
                for similar in kSimIndex:
                    if(sims[similar] > 0):
                        fenzi += self.data[i][similar] * sims[similar]
                        fenmu += sims[similar]
                if(fenmu == 0):
                    self.pridicts[i][j] = 3
                else:
                    self.pridicts[i][j] = fenzi/fenmu
```

(b) item-base

图 5: User-based 或 Item-base 方法

5.2 Similarity

方法采用皮尔逊相关系数计算相似度，user-based 和 item-based 中两个函数计算相似度的区别仅在于行列坐标的不同，，如图 6 所示：

```python
def count_sim(self, i_, j_):
    r,c = self.data.shape
    sims = np.zeros(r)
    #print(np.count_nonzero(self.data, axis=0)[j_])
    for i in range(r):
        if i == i_ or self.data[i][j_]==0:
            sims[i] = -1
            continue
        #皮尔逊相关系数
        fenzi = 0
        fenmu1 = 0
        fenmu2 = 0
        count1 = np.count_nonzero(self.data[i_])
        count2 = np.count_nonzero(self.data[i])
        if(count2 == 0 or count1 == 0):
            sims[i] = -1
            continue
        tempi_ = self.data[i_]-np.mean(self.data[i_])*c/count1
        tempi = self.data[i]-np.mean(self.data[i])*c/count2
        for j in range(c):
            if(self.data[i][j] and self.data[i_][j]):
                fenzi += tempi_[j]*tempi[j]
                fenmu1 += np.square(tempi_[j])
                fenmu2 += np.square(tempi[j])
        if(fenmu1==0 or fenmu2==0):
            sims[i] = -1
            continue
        sims[i] = fenzi/((np.sqrt(fenmu1))*(np.sqrt(fenmu2)))
    return sims
```

(a) user-based 中计算相似度

```python
def count_sim(self, i_, j_):
    r,c = self.data.shape
    sims = np.zeros(c)
    #print(np.count_nonzero(self.data, axis=0)[j_])
    for j in range(c):
        if j == j_ or self.data[i_][j]==0:
            sims[j] = -1
            continue
        #皮尔逊相关系数
        fenzi = 0
        fenmu1 = 0
        fenmu2 = 0
        count1 = np.count_nonzero(self.data[:,j_])
        count2 = np.count_nonzero(self.data[:,j])
        if(count2 == 0 or count1 == 0):
            sims[j] = -1
            continue
        tempj_ = self.data[:,j_]-np.mean(self.data[:,j_])*r/count1
        tempj = self.data[:,j]-np.mean(self.data[:,j])*r/count2
        tempj_ = tempj_.reshape(-1,1)
        tempj = tempj.reshape(-1,1)
        #print(tempj.shape)
        for i in range(r):
            if(self.data[i][j] and self.data[i][j_]):
                fenzi += tempj_[i][0]*tempj[i][0]
                fenmu1 += np.square(tempj_[i][0])
                fenmu2 += np.square(tempj[i][0])
        if(fenmu1==0 or fenmu2==0):
            sims[j] = -1
            continue
        sims[j] = fenzi/((np.sqrt(fenmu1))*(np.sqrt(fenmu2)))
    return sims
```

(b) item-base 中计算相似度

图 6: User-based 或 Item-base 方法

5.3 计算 RMSE

计算 RMSE 的实现代码见图 7:

```python
def rmse(self):
    r,c = self.pridicts.shape
    count = np.count_nonzero(self.pridicts)
    return np.sqrt(r*c*mean_squared_error(self.pridicts, self.test)/count)
```

图 7: 计算 RMSE

5.4 读取数据

参数为数据集的路径以及训练数据占总数据的比重。数据集均被保存成.csv 文件格式, 读取数据，将数据表示为 $users \times items$ 的矩阵，划分并返回训练数据矩阵和测试数据矩阵，没有数据的位置矩阵的值为 0，读取数据的实现代码见图 8:

```python
def readData(file, proportion=0.8):
    df=pd.read_csv(file)
    userNum = df['userId'].max()
    movieNum = df['movieId'].max()
    matrix = [[0]*movieNum for _ in range(userNum)]
    dfRow = df.shape[0]
    for i in range(dfRow):
        matrix[df['userId'][i]-1][df['movieId'][i]-1] = df['rating'][i]
    trainUserNum = int(userNum * proportion)
    trianMovieNum = int(movieNum * proportion)
    matrix = np.array(matrix)
    testMatrix = matrix[trainUserNum:,trianMovieNum:]
    testMatrix = copy.deepcopy(testMatrix)
    for i in range(trainUserNum, userNum):
        for j in range(trianMovieNum, movieNum):
            matrix[i][j] = 0
    #print(np.sum(matrix))
    return np.array(matrix), np.array(testMatrix)
```

图 8: 读取数据

5.5 main 函数

main 函数的主要作用是使用实现的方法对评分进行预测，并使用不同的近邻个数 K, 获取预测结果，绘制预测结果的 RMSE 随近邻个数 K 的变化图，以及打印最好的 RMSE 和 K 值，main 函数实现代码见图 9:

6 Summary

通过这次实验，我对老师课堂上讲到推荐系统最基本的协同滤波算法知识的掌握更加深入了，并且自己动手将基于用户和基于物品的协同滤波算法实现，感觉收获很大。

但是这次实验的结果并不理想，预测的分数不那么精确，如果要在实际上使用该算法，可能需要更多的数据，使数据矩阵的密度更大，或者采用别的算法。

```python
data, test = readData("/home/aistudio/ratings.csv")
print("read data already")
ks = range(1,21)
rmses = []
min_rmse = 9999
min_k = 0
for k in ks:
    user_base_cf = cf(k, data, test)
    user_base_cf.user_base()
    rmse = user_base_cf.rmse()
    rmses.append(rmse)
    if(rmse < min_rmse):
        min_rmse = rmse
        min_k = k

plt.plot(ks, rmses)
plt.xlabel("k")
plt.ylabel("RMSE")
plt.savefig("res.png", format='png')
plt.close()
print("the best k value is ",min_k)
print("the minimum rmse is ",min_rmse)
print("finished...")
```

图 9: main 函数

7 Supplementary material

本实验的代码已上传至 GitHub，可以在我的 Github 获得。
GitHub 仓库是https://github.com/PointerA/ML_DM_course