机器学习实验报告4

user-based CF

读入数据，记录每个物品的用户，每个用户的物品和评分，记录所有出现过的物品

```
def __init__(self, _k):
    self.similars = {}      #用户相似度矩阵
    self.item_users = {}    #物品包含的用户
    self.user_items = {}    #用户包含的物品
    self.user_ratings = {}  #用户的评分
    self.total_items = []   #总共的物品列表
    self.k = _k

def load_data(self, path):
    with open(path, 'r') as f:
        for i, line in enumerate(f, 0):
            if i != 0:
                line = line.strip('\n')
                user, item, rating, timestamp = line.split(',')
                self.item_users.setdefault(item, [])
                self.item_users[item].append(user)
                self.user_items.setdefault(user,[])
                self.user_items[user].append(item)
                self.user_ratings.setdefault(user,{})
                self.user_ratings[user].setdefault(item, 0.)
                self.user_ratings[user][item] = float(rating)
                self.total_items.append(item)
    self.total_items = list(set(self.total_items))  #去重
```

计算相似矩阵，遍历每一个物品，将所有评价过该物品的用户两两之间进行计算，计算用户之间的余弦相似度

```
#计算相似度矩阵
def similarity(self):
    U = {}
    V = {}
    #遍历所有物品
    for item, users in self.item_users.items():
        #双重循环，计算两个用户的余弦相似度
        for u in users:
            for v in users:
                if u != v:
                    self.similars.setdefault(u, {})
                    self.similars[u].setdefault(v, 0.)
                    U.setdefault(u, {})
                    U[u].setdefault(v, 0.)
                    V.setdefault(u, {})
                    V[u].setdefault(v, 0.)
                    #获取不同用户对相同物品的评分
                    x = self.user_ratings[u][item]
                    y = self.user_ratings[v][item]
```

```
                    self.similars[u][v] += x * y
                    U[u][v] += x * x
                    V[u][v] += y * y

    #双重循环，计算两个用户的余弦相似度
    for u, v_cnts in self.similars.items():
        for v, cnt in v_cnts.items():
            if ((U[u][v] ** 0.5) * (V[u][v] ** 0.5)) == 0:
                self.similars[u][v] = 0
            else:
                self.similars[u][v] = self.similars[u][v] / ((U[u][v] ** 0.5) *
(V[u][v] ** 0.5))
```

选择一个用户，计算他的推荐列表，遍历所有物品，选择未被用户评价过的物品，根据用户相似度从大到小遍历，如果相似用户评价了当前物品，则根据相似度进行计算

```
#计算当前用户的推荐列表
def recommendation(self, user):
    rank = {}
    #遍历所有物品
    for item in self.total_items:
        #如果当前物品未被用户评分
        if item not in self.user_items[user]:
            count = 0
            rank.setdefault(item, 0.)
            sum_similar = 0
            #按照相似度从大到小遍历与当前用户的相似用户
            for user_v, similar in sorted(self.similars[user].items(), key =
operator.itemgetter(1), reverse = True):
                if count == self.k:
                    break
                #如果相似用户评价了当前物品，根据相似度计算评分
                if item in self.user_items[user_v]:
                    count += 1
                    rank[item] += similar * self.user_ratings[user_v][item]
                    sum_similar += similar
            #没有相似用户评价当前物品
            if sum_similar != 0:
                rank[item] /= sum_similar
            else:
                rank[item] = 0

    return rank
```

item-based CF

读入数据，记录每个用户的物品和评分，记录所有出现过的物品

```
def __init__(self, _k):
    self.similars = {}      #物品相似度矩阵
    self.user_items = {}    #用户包含的物品
    self.user_ratings = {}  #用户的评分
    self.total_items = []   #总共的物品列表
    self.k = _k

def load_data(self, file_path):
```

```
    with open(file_path, "r") as f:
        for i, line in enumerate(f, 0):
            if i != 0:
                line = line.strip('\n')
                user, item, rating, timestamp = line.split(',')
                self.user_items.setdefault(user,[])
                self.user_items[user].append(item)
                self.user_ratings.setdefault(user,{})
                self.user_ratings[user].setdefault(item, 0.)
                self.user_ratings[user][item] = float(rating)
                self.total_items.append(item)
    self.total_items = list(set(self.total_items))  #去重
```

计算相似矩阵，遍历每一个用户，将该用户所评价过的所有物品两两之间进行计算，计算物品之间的余弦相似度

```
#计算相似度矩阵
def similarity(self):
    U = {}
    V = {}
    #遍历所有用户
    for user, items in self.user_items.items():
        #双重循环，计算两个物品的余弦相似度
        for i in items:
            for j in items:
                if i != j:
                    self.similars.setdefault(i, {})
                    self.similars[i].setdefault(j, 0.)
                    U.setdefault(i, {})
                    U[i].setdefault(j, 0.)
                    V.setdefault(i, {})
                    V[i].setdefault(j, 0.)
                    #获取相同用户对不同物品的评分
                    x = self.user_ratings[user][i]
                    y = self.user_ratings[user][j]
                    self.similars[i][j] += x * y
                    U[i][j] += x * x
                    V[i][j] += y * y

    #双重循环，计算两个物品的余弦相似度
    for i, j_cnt in self.similars.items():
        for j, cnt in j_cnt.items():
            if ((U[i][j] ** 0.5) * (V[i][j] ** 0.5)) == 0:
                self.similars[i][j] = 0
            else:
                self.similars[i][j] = self.similars[i][j] / ((U[i][j] ** 0.5) *
(V[i][j] ** 0.5))
```

选择一个用户，计算他的推荐列表，遍历所有物品，选择未被用户评价过的物品，根据物品相似度从大到小遍历，如果用户评价了相似物品，则根据相似度进行计算

```
#计算当前用户的推荐列表
def recommendation(self, user):
    rank = {}
    #遍历所有物品
    for item in self.total_items:
```

```
            #如果当前物品未被用户评分
            if item not in self.user_items[user]:
                count = 0
                rank.setdefault(item, 0.)
                sum_similar = 0
                #按照相似度从大到小遍历与当前物品的相似物品
                for item_j, similar in sorted(self.similars[item].items(), key = 
operator.itemgetter(1), reverse = True):
                    if count == self.k:
                        break
                    #如果当前用户评价了相似物品，根据相似度计算评分
                    if item_j in self.user_items[user]:
                        count += 1
                        rank[item] += similar * self.user_ratings[user][item_j]
                        sum_similar += similar
                #当前用户没有评价相似物品
                if sum_similar != 0:
                    rank[item] /= sum_similar
                #当前用户评价的相似物品数量不足k个
                if count < self.k:
                    rank[item] = 0

        return rank
```

数据处理

实验数据来源：https://grouplens.org/datasets/movielens/

选择一个用户，再从该用户中选择前n个物品，记录这n个物品的原始评分，再把这些物品从数据中删除。然后对数据进行CF，获取用户的推荐列表，记录这n个物品的新评分。根据新评分和原始评分，计算RMSE

```
#随机选择一个用户，从该用户中随机选择n个物品
user = '365'
#random_items = random.sample(itemBasedCF.user_items[user], 20)
random_items = itemBasedCF.user_items[user][0 : 20]
#记录n个物品的原始评分
old_ratings = {}
for item in random_items:
    old_ratings.setdefault(item, 0.)
    old_ratings[item] = itemBasedCF.user_ratings[user][item]
    #从数据中删除该n个物品
    itemBasedCF.user_items[user].remove(item)
#获取用户的推荐列表
itemBasedCF.similarity()
rank = itemBasedCF.recommendation(user)
#记录n个物品的新评分
new_ratings = {}
for item in random_items:
    new_ratings.setdefault(item, 0.)
    new_ratings[item] = rank[item]
#计算rmse
rmse = RMSE(old_ratings, new_ratings)
print('-------------item based CF-------------')
print('user:', user, '   k:', k)
print('rmse:', rmse)
```

实验结果

user based CF

```
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 10
rmse: 1.929272430172722
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 20
rmse: 1.8542258232703
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 30
rmse: 1.833913453432419
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 30
rmse: 1.833913453432419
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 40
rmse: 1.817846127332367
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 50
rmse: 1.8263333050604207
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 365    k: 60
rmse: 1.8208476555189521
```

```
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 610    k: 10
rmse: 0.7090850229268468
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
rmse: 0.689862675561343
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 610    k: 30
rmse: 0.6897447666225232
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 610    k: 40
rmse: 0.6767904543662046
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 610    k: 50
rmse: 0.6879431772375227
PS D:\vscode workspace\python\CF> python userBasedCF.py
-------------user based CF-------------
user: 610    k: 60
rmse: 0.7037919492017541
```

item based CF

```
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 10
rmse: 1.453412547174839
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 20
rmse: 1.4027976149209913
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 30
rmse: 1.3594603685756217
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 40
rmse: 1.3519621182432946
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 50
rmse: 1.357582186091826
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 365    k: 60
rmse: 1.3368814303849004
```

```
-------------item based CF-------------
user: 610    k: 10
rmse: 0.9164469433633351
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 610    k: 20
rmse: 0.940461987535913
PS D:\vscode workspace\python\CF> python itemBasedCF.py
Traceback (most recent call last):
  File "itemBasedCF.py", line 114, in <module>
    itemBasedCF.similarity()
  File "itemBasedCF.py", line 36, in similarity
    self.similars.setdefault(i, {})
KeyboardInterrupt
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 610    k: 30
rmse: 1.0232912586355851
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 610    k: 40
rmse: 1.1072735377945235
PS D:\vscode workspace\python\CF> python itemBasedCF.py
-------------item based CF-------------
user: 610    k: 50
rmse: 1.1436913919410254
```

实验分析

同一个用户，随着k值的增大，rmse会变小，但k增大到一定程度后，rmse会开始变大。

同一个用户，使用userCF和itemCF的效果也不同，有时候是userCF更好，有时候是itemCF更好。

对于大量的数据样本，物品的数量远超于用户的数量，userCF的运行速度远远快于itemCF；若用户的数量多于物品的数量，itemCF的运行速度更佳。

itemCF更能令用户信服，这是根据用户的历史行为所做出的推荐，如果用户有新的行为，一定会改变推荐的结果，但userCF不一定会改变推荐的结果。