1、项目概述：

项目名称：

基于世纪家园的小型推荐系统

项目用途：

以世纪家园的数据为依托，设计推荐系统，通过对用户未打分的数据进行评分预测，来向用户推荐

2、需求分析：

业务需求

随着互联网的发展，信息过剩的问题逐渐显现出来，通过推荐系统，来提高网站的点击量、交叉销售能力、服务能力等，同时提高网站的吸引力以及顾客对于网站的忠诚度

用户需求

面对互联网信息过剩，用户很难从大量冗余信息中挑选出自己做需要的那部分，该系统基于用户相似度，能够为用户推荐较为适合自己的信息，大量节省用户检索信息的时间，提高效率

功能需求

该系统的主要功能为基于用户之间的相似度，来对用户未打分或评价的物品进行评分预测，并根据预测结果向用户进行推荐。

性能需求

由于网站含有大量的用户信息，因此，系统处理这些信息所需的时间就变得重中之重，本系统所用算法的时间复杂度控制在O(n3)， 能够满足实际需要

3、系统设计

3.1 系统流程图

开始

读取用户打分数据

将数据分割为

训练集和测试集

针对训练集计算

用户相似度矩阵

基于用户相似度矩阵来进行评分预测

比较预测评分和

测试集中相应评分

计算RMSE

爬取数据，存入文件

3.2主要程序说明

本程序使用python语言编写

阶段一：爬取数据。

为了提高爬取速度，采用多线程方式，开辟了10个线程来同时下载网页内容，并存入同一队列，另外由一个线程从队列中读取网页内容并抓取所需信息，如用户的年龄、身高、学历、收入、婚姻状况等。爬取结果会含有重复信息，利用数据库来清除掉重复信息。总体代码见附录。

阶段二：数据集分割。

将用户行为数据集按照均匀分布随机分成M份，挑选一份作为测试集，将剩下的M-1份作为训练集。然后在训练集上建立用户兴趣模型，并在测试集上对用户行为进行预测，统计出相应的评测指标。为了保证评测指标并不是过拟合的结果，需要进行M次实验，并且每次都使用不同的测试集。然后将M次实验测出的评测指标的平均值作为最终的评测指标。

下面的Python代码描述了将数据集随机分成训练集和测试集的过程：

def SplitData(data, M, k, seed):

test = {}

train = {}

random.seed(seed)

count = 0

for line in data:

if(line==""):

break

temp = line.split('::')

#print temp

count += 1

if ( count % M == k):

if(not test.has\_key(temp[0])):

test[temp[0]] = dict()

test[temp[0]][temp[1]]=string.atof(temp[2].strip('\n'))

else:

if(not train.has\_key(temp[0])):

train[temp[0]] = dict()

train[temp[0]][temp[1]]=string.atof(temp[2].strip('\n'))

return train, test

这里，每次实验选取不同的k（0≤k≤M1）和相同的随机数种子seed，进行M次实验就可以得到M个不同的训练集和测试集，然后分别进行实验，用M次实验的平均值作为最后的评测指标。这样做主要是防止某次实验的结果是过拟合的结果（over fitting），但如果数据集够大，模型够简单，为了快速通过离线实验初步地选择算法，也可以只进行一次实验。

阶段三：计算用户相似度矩阵

推荐系统中常用的相似度有：Pearson correlation，Cosine，Squared Distance，其中Pearson correlation的运用最为普遍，因此本系统采用Pearson correlation。Pearson correlation的取值范围为[-1,1]，当值为-1时，表示两组变量负相关，为0时则表示两组变量不相关，为1时表示两组变量正相关，其计算公式如下：



其中，u和v代表用户，集合表示用户u和用户v同时打过分的物品的集合，表示用户u给物品i所打分数，同理表示用户v给物品i所打分数，和分别表示用户u和用户v所打过分数的平均值。具体代码如下：

def Similarity(train):

w = dict()

sumrating = 0

sumdir = 0

userscore = dict()

for u,tt in train.items():

sumrating = 0

sumdir = 0

for i, rating in tt.items():

sumrating += rating

sumdir += 1

userscore[u] = sumrating / (sumdir \* 1.0)

sq = dict()

for u,tt in train.items():

for i,rui in tt.items():

if u not in sq:

sq[u] = 0

sq[u] += math.pow((rui - userscore[u]), 2)

#print u,sq[u]

demo = 0

for u,tt in train.items():

w[u] = dict()

for v, ff in train.items():

demo = 0

fenmu1 = 0

fenmu2 = 0

for aa in ff.keys():

if aa in tt.keys():

fenmu1 += (train[u][aa]-userscore[u])\*\*2

fenmu2 += (train[v][aa]-userscore[u])\*\*2

demo += (tt[aa] - userscore[u])\*(ff[aa] - userscore[v])

#w[u][v] = demo / math.sqrt(sq[u] \* sq[v])

if(fenmu2 == 0 or fenmu1 == 0):

w[u][v] = 0

else:

w[u][v] = demo / math.sqrt(fenmu1 \* fenmu2)

return w

阶段四：基于用户相似度矩阵来进行评分预测，并计算推荐结果的RMSE

基于用户的邻域算法认为预测一个用户对一个物品的评分，需要参考和这个用户兴趣相似的用户对该物品的评分，即:



这里，S(u, K)是和用户u兴趣最相似的K个用户的集合，N(i)是对物品i评过分的用户集合，是用户v对物品i的评分， 是用户v对他评过分的所有物品评分的平均值。由于用户之间的相似度有正有负，而无论正负，只要其绝对值足够大，就具有很大的相关性，故在此当对其他用户和某个用户相似度排序时使用的是相似度的绝对值，而非真实值，这样能使推荐更加准确。具体实现代码如下：

def Recomend(train, w, K):

# build inverse table for item\_users

item\_users = dict()

for u, items in train.items():

for i in items.keys():

if i not in item\_users:

item\_users[i] = set()

item\_users[i].add(u)

predict = dict()

userscore = dict()

userscore = Mean(train)

#此处建立相似度矩阵的副本，该副本的wuv值为原值的绝对值，在下面用来排序

tew = w

for u,tt in tew.items():

for v,wuv in tt.items():

tew[u][v] = math.fabs(wuv)

temp1 = 0

temp2 = 0

for u, tt in train.items():

predict[u] = dict()

for i, user in item\_users.items():

if i in tt.keys():

continue

cc = 0

temp1 = 0

temp2 = 0

for v, wuv in sorted(tew[u].iteritems(), key=itemgetter(1), reverse=True):

if v not in user:

continue

cc += 1

if cc > K:

continue

temp1 += w[u][v] \* (train[v][i] - userscore[v])

temp2 += math.fabs(w[u][v])

if(temp2 == 0):

continue

predict[u][i] = userscore[u] + temp1 / temp2

sigtem = 0

count = 0

for u, tt in test.items():

for i, rating in tt.items():

if i not in predict[u].keys():

continue

sigtem += (rating - predict[u][i])\*\*2

count += 1

#print count

#print sigtem

#print u,i,predict[u][i]

RMSE = math.sqrt(sigtem / (count \* 1.0))

print RMSE

return predict

4 推荐结果分析

最终计算得到的RMSE为1.09，相对于NMF等优化算法的结果，此值略大，但由于该算法不涉及到迭代等问题，效率明显较高，尤其是当要处理的数据集较大的时候，该算法的优越性便体现出来了。另外，在预测结果中，会发生少数结果超过评分上限（本系统中为5分），对于这种情况，处理方法为直接取5分。若数据集中某项物品无人打分，则本算法无法对其进行评分预测。可采用的措施为将其相对于某用户的预测评分设置为该用户所打过的所有分数的平均分，以此来消除此类特殊情况。但由于本系统所用数据集中未出现该情况，故为采用特殊处理。

5 参考文献

《推荐系统实践》 项亮 著 人民邮电出版社 2012.6

附录：