数据集:

一般来说,不同的数据集包含不同的行为,目前比较有代表性的数据集有下面几个

无上下文信息的隐性反馈数据集:

每一条行为记录仅仅包含用户ID和物品ID

Book-Crossing 1 就是这种类型的数据集。

无上下文信息的显性反馈数据集

每一条记录包含用户ID、物品ID和用户对物品的评分。

有上下文信息的隐性反馈数据集

每一条记录包含用户ID、物品ID和用户对物品产生行为的时间戳

Lastfm数据集 1 就是这种类型的数据集

有上下文信息的显性反馈数据集:

每一条记录包含用户ID、物品ID、用户对物品的评分和评分行为

发生的时间戳。Netflix Prize 2 提供的就是这种类型的数据集

本章使用的数据集基本都是第一种数据集,即无上下文信息的隐性反馈数据集

仅仅基于用户行为数据设计的推荐算法一般称为协同过滤算法:

基于邻域的方法( neighborhood-based )、

基于用户的协同过滤算法

这种算法给用户推荐和他兴趣相似的其他用户喜欢的物品。

基于物品的协同过滤算法

这种算法给用户推荐和他之前喜欢的物品相似的物品。

隐语义模型 ( latent factor model )、

基于图的随机游走算法( random walk on graph )

协同过滤算法:

首先,将用户行为数据集按照均匀分布随机分成 M 份(本章取 M =8 ),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。

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)

for user, item in data:

if random.randint(0,M) == k:

test.append([user,item])

else:

train.append([user,item])

return train, test

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

评测指标:

过准确率 / 召回率评测推荐算法的精度

召回率:有多少比例的用户—物品评分记录包含在最终的推荐列表中

准确率:推荐列表中有多少比例是发生过的用户—物品评分记录

def Recall(train, test, N):

hit = 0

all = 0

for user in train.keys():

tu = test[user]

rank = GetRecommendation(user, N)

for item, pui in rank:

if item in tu:

hit += 1

all += len(tu)

return hit / (all \* 1.0)

def Precision(train, test, N):

hit = 0

all = 0

for user in train.keys():

tu = test[user]

rank = GetRecommendation(user, N)

for item, pui in rank:

if item in tu:

hit += 1

all += N

return hit / (all \* 1.0)

除了评测推荐算法的精度,本章还计算了算法的覆盖率:

def Coverage(train, test, N):

recommend\_items = set()

all\_items = set()

for user in train.keys():

for item in train[user].keys():

all\_items.add(item)

rank = GetRecommendation(user, N)

for item, pui in rank:

recommend\_items.add(item)

return len(recommend\_items) / (len(all\_items) \* 1.0)

我们还需要评测推荐的新颖度:

用推荐列表中物品的平均流行度度量推荐结果的 新颖度

def Popularity(train, test, N):

item\_popularity = dict()

for user, items in train.items():

for item in items.keys()

if item not in item\_popularity:

item\_popularity[item] = 0

item\_popularity[item] += 1

ret = 0

n = 0

for user in train.keys():

rank = GetRecommendation(user, N)

for item, pui in rank:

ret += math.log(1 + item\_popularity[item])

n += 1

ret /= n \* 1.0

return ret

在计算平均流行度时对每个物品的流行度取对数,这是因为物品的

流行度分布满足长尾分布,在取对数后,流行度的平均值更加稳定。

基于用户的协同过滤算法:

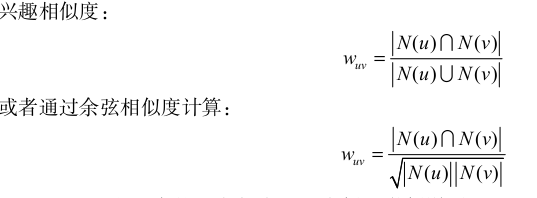
(1) 找到和目标用户兴趣相似的用户集合

(2) 找到这个集合中的用户喜欢的,且目标用户没有听说过的物品推荐给

目标用户。

计算相似度:

Jaccard 公式简单地计算u和v 的 兴趣相似度



以余弦相似度为例,实现该相似度可以利用如下的伪码:

def UserSimilarity(train):

W = dict()

for u in train.keys():

for v in train.keys():

if u == v:

continue

W[u][v] = len(train[u] & train[v])

W[u][v] /= math.sqrt(len(train[u]) \* len(train[v]) \* 1.0)

return W