推荐系统学习：协同过滤实现

发表于 2014-04-19 | 分类于 程序设计 | 7 条评论 | 阅读次数 4404

阿里大数据竞赛的第一季也即将告一段落，之前一直想发篇竞赛心得/攻略，但是看到有人发的博客居然被阿里查封了就不敢了...扯远了，这篇文章主要总结下自己看了项亮的《推荐系统实践》后的学习笔记。作为入门，这么书确实写的不错。

推荐系统的评测指标

为了评估推荐算法的好坏需要各方面的评估指标。

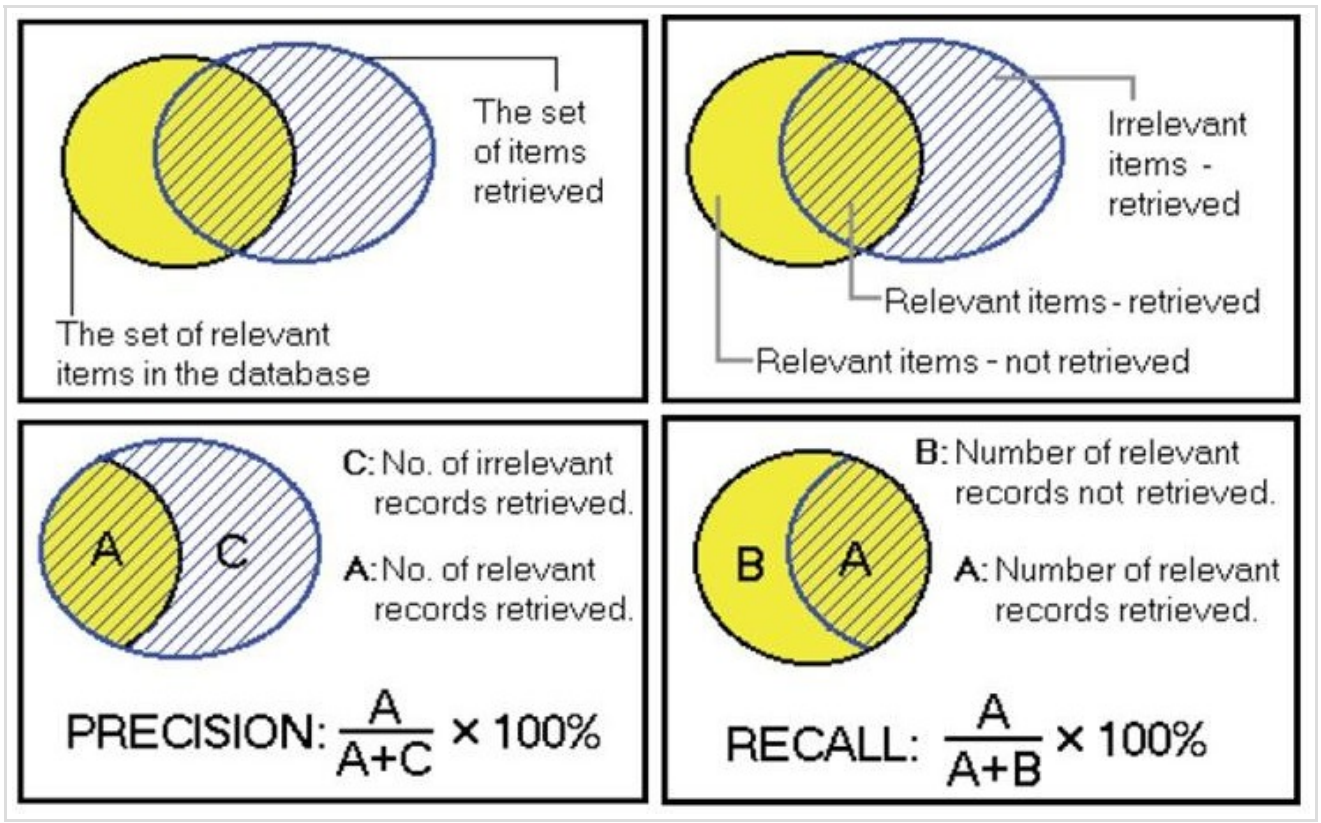
- 准确率

  准确率就是最终的推荐列表中有多少是推荐对了的。

- 召回率

  召回率就是推荐对了的占全集的多少。

  下图直观地描述了准确率和召回率的含义



- 覆盖率

  覆盖率表示推荐的物品占了物品全集空间的多大比例。

- 新颖度

  新颖度是为了推荐长尾区间的物品。用推荐列表中物品的平均流行度度量推荐结果的新颖度。如果推荐出的物品都很热门，说明推荐的新颖度较低，否则说明推荐结果比较新颖。

各个评测指标的代码实现如下：

```
#召回率和准确率
def RecallAndPrecision(self,train=None,test=None,K=3,N=10):
    train = train or self.train
    test = test or self.test
    hit = 0
    recall = 0
    precision = 0
    for user in train.keys():
        tu = test.get(user,{})
        rank = self.Recommend(user,K=K,N=N)
        for i,_ in rank.items():
            if i in tu:
                hit += 1
        recall += len(tu)
        precision += N
    recall = hit / (recall * 1.0)
    precision = hit / (precision * 1.0)
    return (recall,precision)

#覆盖率
def Coverage(self,train=None,test=None,K=3,N=10):
    train = train or self.train
    recommend_items = set()
    all_items = set()
    for user,items in train.items():
        for i in items.keys():
            all_items.add(i)
        rank = self.Recommend(user,K)
        for i,_ in rank.items():
            recommend_items.add(i)
    return len(recommend_items) / (len(all_items) * 1.0)

#新颖度
def Popularity(self,train=None,test=None,K=3,N=10):
    train = train or self.train
    item_popularity = dict()
    #计算物品流行度
    for user,items in train.items():
        for i in items.keys():
            item_popularity.setdefault(i,0)
            item_popularity[i] += 1

    ret = 0     #新颖度结果
    n = 0       #推荐的总个数
    for user in train.keys():
        rank = self.Recommend(user,K=K,N=N)    #获得推荐结果
        for item,_ in rank.items():
            ret += math.log(1 + item_popularity[item])
            n += 1
    ret /= n * 1.0
    return ret
```

基于用户的协同过滤

UserBasedCF的核心思想主要是找到和目标用户兴趣相似的用户集合，然后给目标用户推荐这个集合的用户喜欢的物品。关键在于计算用户与用户之间的兴趣相似度。这里主要使用余弦相似度来计算：

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

w_{uv} 代表用户 u 与 v 之间的兴趣相似度，$N(u)$表示用户 u 曾经喜欢过的物品集合, $N(v)$ 表示用户 v 曾经喜欢过的物品集合。

根据上述核心思想，可以有如下算法步骤：

1. 建立物品-用户的倒排表

文章目录 站点概览

2. 用户与用户之间的共现矩阵 C[u][v]，表示用户u与v喜欢相同物品的个数

3. 用户与用户之间的相似度矩阵 W[u][v]，根据上述相似度计算公式计算。

4. 用上面的相似度矩阵来给用户推荐和他兴趣相似的用户喜欢的物品。用户 u 对物品 i 的兴趣程度可以估

$$p(u,i) = \sum_{v \in S(u,K) \cap N(i)} w_{uv} r_{vi}$$

$S(u,K)$ 为和用户 u 兴趣最接近的 K 个用户，$N(i)$ 为对物品 i 有正反馈的用户集合，W[u][v] 为用户 u 和用户 v 的兴趣相似度，r_{vi} 为用户 v 对物品 i 的兴趣。

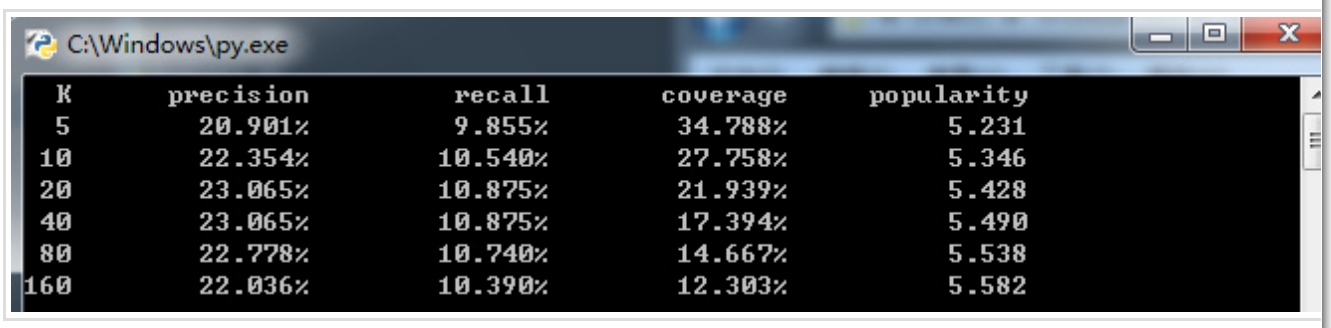
下面是UserBasedCF的代码实现：

```
class UserBasedCF:
    def __init__(self,train_file,test_file):
        self.train_file = train_file
        self.test_file = test_file
        self.readData()
    def readData(self):
        #读取文件，并生成用户-物品的评分表和测试集
        self.train = dict()     #用户-物品的评分表
        for line in open(self.train_file):
            # user,item,score = line.strip().split(",")
            user,item,score,_ = line.strip().split("\t")
            self.train.setdefault(user,{})
            self.train[user][item] = int(score)
        self.test = dict()      #测试集
        for line in open(self.test_file):
            # user,item,score = line.strip().split(",")
            user,item,score,_ = line.strip().split("\t")
            self.test.setdefault(user,{})
            self.test[user][item] = int(score)


    def UserSimilarity(self):
        #建立物品-用户的倒排表
        self.item_users = dict()
        for user,items in self.train.items():
            for i in items.keys():
                if i not in self.item_users:
                    self.item_users[i] = set()
                self.item_users[i].add(user)

        #计算用户-用户相关性矩阵
        C = dict()  #用户-用户共现矩阵
        N = dict()  #用户产生行为的物品个数
        for i,users in self.item_users.items():
            for u in users:
                N.setdefault(u,0)
                N[u] += 1
                C.setdefault(u,{})
                for v in users:
                    if u == v:
                        continue
                    C[u].setdefault(v,0)
                    C[u][v] += 1

        #计算用户-用户相似度，余弦相似度
        self.W = dict()     #相似度矩阵
        for u,related_users in C.items():
            self.W.setdefault(u,{})
            for v,cuv in related_users.items():
                self.W[u][v] = cuv / math.sqrt(N[u] * N[v])
        return self.W

    #给用户user推荐，前K个相关用户
    def Recommend(self,user,K=3,N=10):
        rank = dict()
        action_item = self.train[user].keys()     #用户user产生过行为的item
```

文章目录 站点概览

```
        for v,wuv in sorted(self.W[user].items(),key=lambda x:x[1],reverse=True)[0:K]:
            #遍历前K个与user最相关的用户
            for i,rvi in self.train[v].items():
                if i in action_item:
                    continue
                rank.setdefault(i,0)
                rank[i] += wuv * rvi
        return dict(sorted(rank.items(),key=lambda x:x[1],reverse=True)[0:N])   #推荐结果的取前N个
```

采用 MovieLens 数据集对 UserCF 算法测试之后各评测指标的结果如下



基于物品的协同过滤

ItemBasedCF 应该是业界的应用最广泛的推荐算法了。该算法的核心思想主要是：给目标用户推荐与他喜欢
品相似度较高高的物品。我们经常在京东、天猫上看到「购买了该商品的用户也经常购买的其他商品」，就是
基于 ItemBasedCF。一般我们先计算物品之间的相似度，然后根据物品的相似度和用户的历史行为给用户生
荐列表。

物品 i 和 j 之间的相似度可以使用如下公式计算：

$$w_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

从上面的定义可以看到，在协同过滤中两个物品产生相似度是因为它们共同被很多用户喜欢，也就是说每个用
可以通过他们的历史兴趣列表给物品“贡献”相似度。

根据上述核心思想，可以有如下算法步骤：

1. 建立用户-物品的倒排表
2. 物品与物品之间的共现矩阵 C[i][j]，表示物品 i 与 j 共同被多少用户所喜欢。
3. 用户与用户之间的相似度矩阵 W[i][j] ， 根据上述相似度计算公式计算。
4. 用上面的相似度矩阵来给用户推荐与他所喜欢的物品相似的其他物品。用户 u 对物品 j 的兴趣程度可以
   为

$$p(u,j) = \sum_{i \in N(u) \cap S(j,K)} w_{ji} r_{ui}$$

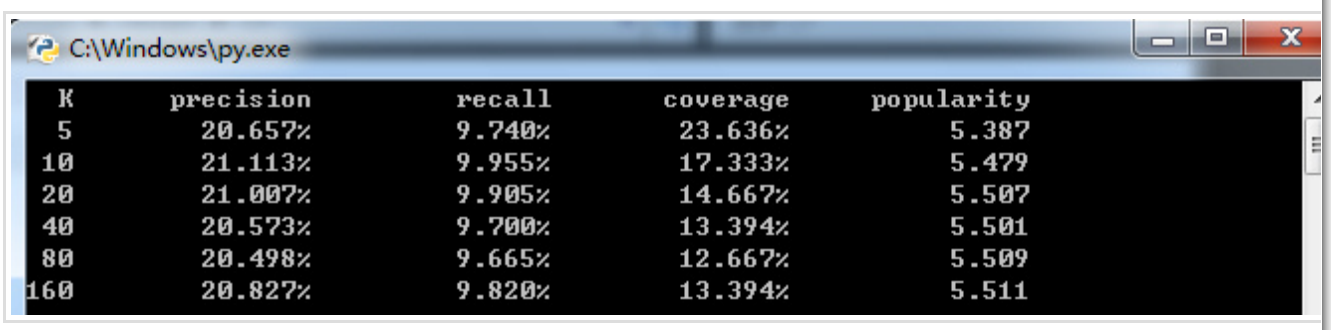
$S(j,K)$ 为和物品 j 最相似的前 K 个物品， $N(u)$ 为对用户 u 所喜欢的物品集合， W[j][i] 为物品 j 和物品 i 之间
似度， r_{ui} 为用户 u 对物品 i 的兴趣。

下面是ItemBasedCF 的代码实现：

```
class ItemBasedCF:
    def __init__(self,train_file,test_file):
        self.train_file = train_file
        self.test_file = test_file
        self.readData()
    def readData(self):
        #读取文件，并生成用户-物品的评分表和测试集
        self.train = dict()     #用户-物品的评分表
        for line in open(self.train_file):
            # user,item,score = line.strip().split(",")
```

```
            user,item,score,_ = line.strip().split("\t")
            self.train.setdefault(user,{})
            self.train[user][item] = int(score)
        self.test = dict()      #测试集
        for line in open(self.test_file):
            # user,item,score = line.strip().split(",")
            user,item,score,_ = line.strip().split("\t")
            self.test.setdefault(user,{})
            self.test[user][item] = int(score)

    def ItemSimilarity(self):
        #建立物品-物品的共现矩阵
        C = dict()  #物品-物品的共现矩阵
        N = dict()  #物品被多少个不同用户购买
        for user,items in self.train.items():
            for i in items.keys():
                N.setdefault(i,0)
                N[i] += 1
                C.setdefault(i,{})
                for j in items.keys():
                    if i == j : continue
                    C[i].setdefault(j,0)
                    C[i][j] += 1
        #计算相似度矩阵
        self.W = dict()
        for i,related_items in C.items():
            self.W.setdefault(i,{})
            for j,cij in related_items.items():
                self.W[i][j] = cij / (math.sqrt(N[i] * N[j]))
        return self.W

    #给用户user推荐，前K个相关用户
    def Recommend(self,user,K=3,N=10):
        rank = dict()
        action_item = self.train[user]     #用户user产生过行为的item和评分
        for item,score in action_item.items():
            for j,wj in sorted(self.W[item].items(),key=lambda x:x[1],reverse=True)[0:K]:
                if j in action_item.keys():
                    continue
                rank.setdefault(j,0)
                rank[j] += score * wj
        return dict(sorted(rank.items(),key=lambda x:x[1],reverse=True)[0:N])
```

采用 MovieLens 数据集对 ItemCF 算法测试之后各评测指标的结果如下



UserCF 和 ItemCF 的区别和应用

UserCF 算法的特点是：

- 用户较少的场合，否则用户相似度矩阵计算代价很大
- 适合时效性较强，用户个性化兴趣不太明显的领域
- 对新用户不友好，对新物品友好，因为用户相似度矩阵不能实时计算
- 很难提供令用户信服的推荐解释

对应地，ItemCF 算法的特点：

- 适用于物品数明显小于用户数的场合，否则物品相似度矩阵计算代价很大
- 适合长尾物品丰富，用户个性化需求强的领域

文章目录 站点概览

- 对新用户友好，对新物品不友好，因为物品相似度矩阵不需要很强的实时性
- 利用用户历史行为做推荐解释，比较令用户信服

因此，可以看出 UserCF 适用于物品增长很快，实时性较高的场合，比如新闻推荐。而在图书、电子商务和电
域，比如京东、天猫、优酷中，ItemCF 则能极大地发挥优势。在这些网站中，用户的兴趣是比较固定和持久
而且这些网站的物品更新速度不会特别快，一天一更新是在忍受范围内的。

参考文献

- 项亮《推荐系统实践》
- 基于用户的协同过滤算法

#协同过滤

面试精选：链表问题集锦 使用 Python 轻松抓取网