Sicily 题目推荐系统

1. 项目简述。

本推荐系统采用基于物品过滤的方法构建，先根据 Sicily 的 MySQL 数据库中 status 表来构造“用户-题目”矩阵，或者称 U-P 矩阵（下面用 UP 来表示）。其中 UP(i,j) 表示 i 用户对 j 题目的喜好程度。Sicily 的 status 表记录了所有用户的提交记录以及结果，可以通过用户对某道题目的提交记录来评定其喜好程度，从而构造该矩阵。

然后基于 U-P 矩阵构造“题目-题目”的相似度矩阵，或者称 P-P 矩阵（用 PP 表示）。

PP(i,j)表示题目 i 和 j 之间的相似程度。

推荐系统基于 P-P 矩阵进行计算。主要有两个应用的地方，第一个是当用户查看一

道题目时，在该题目下方会有一个相关问题推荐，这个可以直接从 P-P 矩阵获得。另外一个应用之处是针对用户的题目推荐，根据用户的做题记录再根据 P-P 矩阵来进行计算。

使用基于物品的过滤方法的原因：速度比基于协作过滤快，题目数量变化不大，构

建 P-P 矩阵不需要太多时间，而且 U-P 矩阵是稀疏矩阵，基于物品过滤会更精准。

除此之外，在推荐系统之外给 Sicily 增加了简单的 SNS 功能，作为附加功能配合推荐系统使用。

下面将详细说明每个部分的内容。

2. 数据来源与数量级

Sicily 系统原始代码可以通过 svn co http://soj.me/svn/sicily 下载下来。

由于推荐算法的设计需要实际的数据，于是我们编写了一个爬虫程序，将比较重要的三个表抓取下来，并存放到本地的 MySQL 数据库中。这三个表分别是（status，user，problems 表）。

其中 user 表记录每个用户的用户 id，用户名，还有 ac 题数以及 submit 次数，可以用于衡量用户的能力水平。

problems 表记录每个题目的具体信息，包括题目描述，输入输出信息，还有该题目的 ac 次数和提交次数，可以衡量题目难度。

status 表记录所有用户的提交记录，包括提交时间，提交结果。是构造 U-P 矩阵的重要数据来源。

数量级（已抓取部分）：

用户总数：14615

题目总数：917

提交状态数：608907

3. U-P 矩阵权重计算方法，P-P 矩阵相似度计算。

（1） 如何根据 status 表评估 U-P 矩阵权重，权重计算公式以及原因。

由于 status 表反映了用户对于一道题目的提交记录，根据常理，对于一道题目的提交次数和 AC 次数能反映用户对于这道题目的喜爱程度，U-P 矩阵权重根据这个情况来设定。此外由于题目难度不同，计算权重时会把题目的 ac 率考虑进去。

假设某道题目，AC 次数为 totRight, 提交总次数为 totSubmit,
那么这道题目的 ac 率
假设某用户做这道题时，一共有 right 次 AC，wrong 次没有 AC，那么 U-P 权重 Wij 为：

取根据题目 AC 率来划分权重，能够根据题目难度来区分喜好程度，同时可以防止乱提交导致分值差异过大情况。最后结果取 log 是为了降低评分之间的差距，以免导致评分差值过大影响相似度计算。

（2）P-P 矩阵计算方法。求题目相似度方面我们尝试了四种算法（欧几里得，pearson 算法，加权余弦相似度，Tanimoto 系数）

假设对 U-P 矩阵进行翻转，得到 P-U 矩阵，然后对于两个题目向量 P1,P2，计算其相似度。

欧几里得距离：

设两个向量记做(p1,p2...pn), (q1,q2...qn)，则欧几里得距离为

$$\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+\ldots+(p_n-q_n)^2}=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$

Pearson 相关系数：

Pearson 相关系数是度量两个变量相关程度的方法，是介于-1 和 1 之间的值。1 表示变量完全正相关，0 表示无关，-1 表示完全负相关。公式如下：

$$r = \frac{\sum XY - \frac{\sum X \sum Y}{N}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{N}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{N}\right)}}$$

向量夹角余弦：

记两个向量为 P1(u1, u2, u3, … , un) P2(u1, u2, u3, … , un)

求两个向量夹角的余弦<p1, p2> =

Tanimoto 系数：

主要用于计算两个集合的相似度，一般情况下适合用于离散值的计算，但是也可以用于判断两个向量的相似度。

原始的公式如下：　　（Na 为集合 a 的元素个数，Nb 为集合 b 的元素个数，Nc 为集合 a，b 的交集的元素个数。

根据此公式，计算两个向量 P1，P2 的相似度如下：

令 P3 为向量，对于每一维 i, P3(i) = min(P1(i), P2(i))。

则

各个相似度算法比较：

算法	特点
欧几里得	求相似距离，但是对突变数据处理得不好，没有对难度进行识别
Pearson	能防止突变数据的影响，但是没有考虑难度相关性
余弦相似	求出向量相似度，考虑题目难度的加权后可以增加难度相关性
Tanimoto	计算简单，但忽略了各个维度上的特征，题目针对性不强。

经过测试，最后考虑了向量的特征以及难度的相关度情况，决定采用余弦相似的方法，并且对向量夹角余弦算法进行改进，通过加入难度判断的权重，使得准确度比其他算法要好。改进如下：

一开始我们尝试直接求两向量的余弦，然后余弦值就当作两题相似度。这样做之后观察结果发现有些题目会出现一种情况：一道提交数较多题目与一道提交数比较少的题目很相似。

经分析后发现原因是 U-P 矩阵过于稀疏，有些向量的 0 分量过多，与其他题目的共同用户过少。考虑有三道题目 P，P1，P2，其中

P = (0, 0, 0, 2, 2, 2, 2)

P1 = (2, 2, 2, 2, 0, 0, 0)

P2 = (0, 3, 0, 2, 0, 0, 0)

这样算出来的<P, P2>会比<P, P1>大，从而认为 P 与 P2 更相似。但从直观来说 P 和 P1 应该更相似，因为 P1 和 P，P2 和 P 都是只有一个共同的用户完成了，但做 P1 的用户数明显比做 P2 的用户数更接近做 P 的用户数。

为了修正一道题跟其他所有题的共通用户都不多所导致的推荐不准确的情况，我们进行了如下的考虑：

一开始我们本来想通过矩阵分解的手段推测出这些 0 分量的值。但考虑到矩阵分解的时间复杂度较高，且根据 U-P 矩阵分解所预测出来的 0 分量数据不一定准确，所以我们放弃了这一方法。

最后考虑到如果两道题从用户的评分角度来看很相似，那么接下来就应该通过两道题目的通过人数和提交人数来判断相似。所以最终我们在相似度计算中加入了题目通过数和提交人数这一项来进行调整。

最终我们采用的公式如下:

令 P1、A1、S1 分别为题目 1 的用户评分向量、通过人数、提交人数
P2、A2、S2 分别为题目 2 的用户评分向量、通过人数、提交人数

sim_prob = 0.8 * <P1, P2> + 0.2 * factor(P1, P2)

factor(P1, P2) =

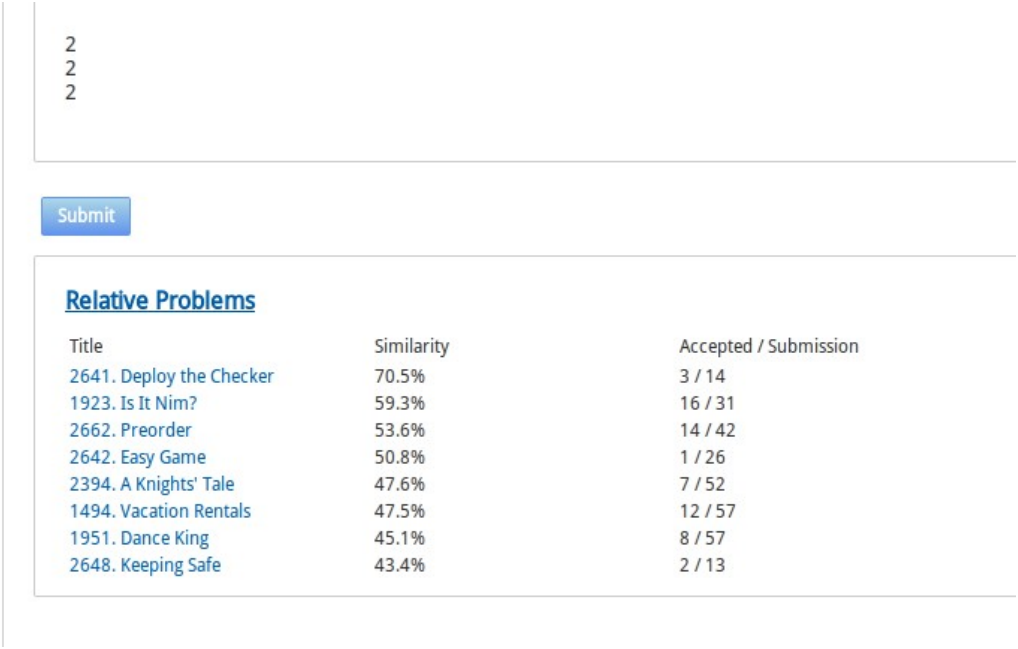
<P1, P2>就是根据用户评分计算向量余弦得到的相似度，factor 表达的就是两道题在通过人数和提交人数在数量级上的相似度。两者分别给 0.8 和 0.2 的权重得出最终两道题目的相似度。

4. 应用：针对题目的题目推荐，针对用户的题目推荐

（1） 针对题目推荐：

当用户点击查看一道题目时，在该题目下方会显示相关题目，该功能通过直接查询 P-P 矩阵获得，将与其相似度最大的 8 个显示出来。

譬如打开题目 2469，会在题目描述中显示相关题目（如下图）
而这道题目难度（12/59），属于比较难的题，由于使用了改进的相似度算法，在考虑相似的同时，加入了难度判断，使得推荐的题目在难度上类似。



（2） 针对用户推荐。

在用户的 Feed 页面（在第 4 部分有说明），有一个根据用户过往提交记录来进行推荐的地方。这个方法基于已计算出来的 P-P 矩阵来计算。伪代码如下：

设 $solved 是用户已解决题目的集合, $unsolve 是未解决题目集合
Rate[u,j] 表示用户 u 对 j 题目的评价（U-P 矩阵）
Sij 是题目 i 和题目 j 的相似度（P-P 矩阵）

```
For i in $unsolve
{
    totSim = 0;
    Sim = 0;
    For j in $solved
    {
        Sim += Sij *  Rate[u,j];
        totSim += Sij;
    }
    R[i] = Sim / totSim; #归一化处理
}
Sort(R);  #排序并选择前 n 个输出
```

截图 2

你可能感兴趣的题目

题目	AC率	相关用户
2373	1/25	
2374	2/4	
1574	1/2	WXYZ
1331	1/2	WXYZ
2385	5/10	lyc
1746	289/852	ccd chyx xxq
1213	21/143	SHOIT
1297	54/424	SHOIT WXYZ
1058	76/189	lyc likit
1579	9/84	

5. SNS 功能介绍

在构造此推荐系统同时，给 Sicily 增加了类似于 Twitter、微博类的 SNS 功能，由于不是主要部分，于是目前界面比较简陋，功能相对简单，主要实现功能如下：
导航栏增加 Feed 页面，在 Feed 页面有几个功能：

（1） Follow 功能。

考虑到做题其实不应该是一个人的事情，增加用户之间的交流也是很重要的，于是增加类似于 Twitter 的 Follow 功能，可以关注用户喜欢的那些牛人们，然后可以通过 feed 页面查看他们最近发表的内容，主要是做某道题的思路，通过看 Feed 可以了解他们最近做题情况。
此外增加评论，转发功能，可以对别人的 feed 进行评论和转发。同时被评论或者转发会受到通知。

（2） 最近热题。

通过查询最近的 status，再根据用户的权重来计算出最近比较热门的题目，也即大家都在刷的题。其计算公式考虑到用户的水平给予不同的权重，水平高的给予更高的权重，使得结果更具有参考价值：

譬如用户 A 解决题目为 totSolve, 提交次数为 totUsub,
题目 B 的 AC 人数是 totAC, 提交次数是 totPsub
最近一段时间，用户 A 对于题目 B 提交了 sub 次，那么其权重为：

然后根据 Wb 排序输出。

（3） 用户推荐。

考虑到基于共同 follow 的人来对用户进行推荐往往比通过聚类效果会更好，这里的用户推荐是基于与当前用户有比较多的共同 follow 的人组成。通过划三角形的方式来进行推荐。

（4） Follow 的人最近做题

检索 status 表，把最近我关注的人的做题情况检索出来，提供给用户参考。

整个 Feed 页面：

6．效果评估

效果评估采取随机抽样的方法，抽取 2%的题目进行人工识别，并且进行相似度判定，相似度包括题目类型(DP,模拟，数学，数据结构等等)，以及难度（从简单到难分 ABCD 四个等级）。

统计结果发现，在针对题目的推荐的中，平均有 35.9%的题目类型和当前题目是一致的，在没有对题目打题目类型标签的情况下，只利用了用户的提交记录发现和当前题目类型相似的题，说明相似度算法还是有效的。

此外，针对题目难度的不同情况，其类型相似情况也有所不同：

（1）对于容易题来说，题目类型针对性不强，类型相似度仅为 29.1%。

（2）对于中难度题目来说，题目类型针对性较强，类型相似度为 44.6%。

难度的差异带来了题目类型相似度的不同，主要原因在于容易题 AC 的用户比较多，造成题目的特征不明显，相反，对于只有几十个人过的题目，向量特征比较明显，使得在进行向量相似判断时所占权重会更大。

对于推荐的题目难度而言，分成 ABCD 四个等级，并假设相邻难度距离为 1，经过统计，发现由于加入了难度控制因子，平均难度距离为 0.46，也就是说，对于 ABCD 四个等级难度的题目，所推荐的题目的难度和当前题目的难度之间的平均差距为 0.46，所以对于难度的控制是有效的。

7.代码

爬虫：

utils/user.py　　#抓取 user 表的程序
utils/grab.py　　#抓取题目的程序
utils/status.py　　#抓取 status 表的程序

SQL：

utils/sicily.sql　　#原 sicily 数据库的表都在此定义
utils/user_rate.sql　#记录 U-P 矩阵的表
utils/prob_sim.sql　　#记录 P-P 矩阵的表

外部程序：

utils/user_rate.py　　#计算 U-P 矩阵的程序，然后存放到 user_rate 表中。
utils/cal_prob_sim.py　#计算 P-P 矩阵的程序，存放到 prob_sim 表中

SNS 部分：

web/feed.php
web/feed_func.php　　#处理 feed 页面的主要程序

utils/comments.sql
utils/feeds.sql　　#feed 表和评论表的定义