基于搜索关键字的用户个性化推荐

复旦大学 *戴涵俊* daihanjun@gmail.com

复旦大学 *阿拉法特•居来提* arapat.mail@gmail.com

目录

前言

随着进入大数据时代，人们能够获取越来越多的信息。如何搜索到所需要的信息、如何主动给用户提供可能需要的信息，已经受到了越来越多的关注。

本文为“基于搜索关键字的用户个性化推荐”提出了一种新的模型。之后的内容将如下安排：

1. **问题分析**：包括问题描述、难点分析等内容；
2. **模型框架**：将从整体上介绍我们的模型，该模型包括关键字分析和个性化推荐两部分；同时将介绍我们主要用到的技术；
3. **数据预处理**：介绍如何针对本问题规范化带噪声的数据；
4. **关键字分析**：详细描述基于关键字匹配的算法；
5. **个性化推荐**：分析如何加入用户个性化元素；
6. **算法评测**：通过实验验证模型的可行性和有效性；
7. **未来扩展**：介绍未来可以加入的工作，以及当有更多资源时候，该模型如何扩展。

问题分析

问题描述

本题收集了从 2011 年 8 月 11 日到 2011 年 10 月 31 日近三个月的 Xbox 游戏搜索记录。每条记录包含了用户编号、产品编号、搜索和点击时间等信息。通过我们对数据的详细分析，其具体的格式和一些限制如下：

数据字段说明：

- **user:** 用户的 ID，用来唯一标识用户；
- **query:** 用户搜索时候输入的关键词。该关键词可能有噪声；
- **sku:** 用户所点击的产品 ID。该值也可能存在噪声，其出现的原因是，sku 记录的是用户输入搜索词之后点击的产品，但是并没有保证，该产品对应了该搜索词。换句话说，用户可能在输入搜索词之后，通过其它途径点击了产品；
- **category:** 用户点击产品所属类别。在本题中，所有产品都是 Xbox 上面的游戏，所以类别这个信息就没有用处了；
- **query_time:** 用户输入搜索关键词的时间，精确到秒；
- **click_time:** 用户点击产品的时间， 精确到秒；通过分析我们发现，数据均能够保证 click_time 处于 query_time 之后，并且两者之间的间隔不超过 5 分钟。

题目一共给出了两组数据——

训练数据：一共有 42365 条记录；

测试数据：一共有 28241 条测试记录。

其中测试数据中不包含 sku 字段，需要我们设计算法来预测该值。

难点分析

通过以上的问题描述，我们归纳本题的难点如下：

1. **重要信息缺失**：这里指的重要信息主要是每个产品的相关信息，如产品的名字、产品发布时间、产品所获得的用户评价，等等。其中最重要的是产品的名字，如果不知道产品名字，我们可以说对产品一无所知了，至于产品搜索和用户推荐就更无从说起。

2. **数据存在噪声**：在上一节中，我们提到了两点噪声，一是用户输入关键词带来的噪声，比如用户输入的关键词中有错误拼写；二是关键词与产品关联信息的噪声，即错误地将与搜索关键词不相关的产品联系起来。

3. **数据非常稀疏**：这里的稀疏性是指的交易记录量太少。根据我们对训练数据统计，相关的信息如下：

   不同产品数：413

   参与的用户数：38024

   总记录数：42365

   以上数据说明，平均每个用户点击的产品数量不到两个！这给发现和挖掘用户个性化信息带来了很大的困难。

模型框架

主算法

本题目是“基于搜索关键字的用户个性化推荐”，而要处理的任务是针对用户的搜索词推荐合适该用户的产品，所以一个好的算法应该同时考虑“关键字”和“个性化”这个两个方面。我们的算法描述如下：

对一个用户提出的搜索，我们返回一个产品，使得其具有最大的评分 Score，其中 Score 定义如下：

$$\mathbf{Score}(\mathbf{user}, \mathbf{query}, \mathbf{time}, \mathbf{sku}) =$$

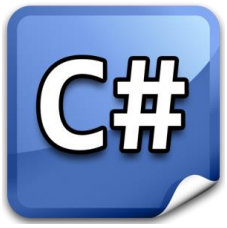
$$\boldsymbol{KeywordMatch}(\mathbf{sku}, \mathbf{user}, \mathbf{query}) + \boldsymbol{\lambda} * \boldsymbol{Recommendation}(\boldsymbol{sku}, \boldsymbol{user}, \mathbf{time})$$

对于用户的一次搜索请求，每个产品都会产生相应的分数。这个分数由两部分组成：

- **KeywordMatch:** 关键字匹配得分。该项根据用户输入的关键词和该产品具有的关键词匹配程度进行打分；
- **Recommendation:** 推荐程度得分。该项根据用户个人 ID 以及搜索的时间，给对应的产品推荐程度打分。这一项主要考虑到了用户的一些个性化信息；
- $\boldsymbol{\lambda}$**:** 两个打分函数之间的平衡系数。取决于更偏重关键词匹配还是推荐。

后文中将详细介绍 KeywordMatch 和 Recommendation 这两个函数的实现方法。而$\boldsymbol{\lambda}$可以通过经验值或者是校验数据集来设置。

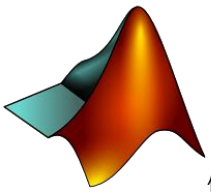
主要技术



主算法采用.Net C# 实现。我们的算法中涉及到许多的数据结构操作以及数据库相关操作。使用面向对象语言可以大大简化这些任务。同时 C#强大的调试跟踪功能也给开发带来了很大的方便。

使用 python 这样的脚本语言可以大大简化初期的一些数据分析工作。我们正是借助 python 进行前文中提到的一些数据分析，以及后文中所需要的数据挖掘。

训练数据和测试数据用关系数据模型存放在 MySQL 中。之所以用关系数据库，而不用诸如 Mongo DB 等 NoSQL 数据库的原因有二：1. 本项目所涉及的数据条目比较少，MySQL 足以应对；2. 在数据分析中需要涉及到表的 join 等操作。



使用 matlab 进行协同过滤中矩阵运算和数据挖掘后的绘图工作。

使用 github 版本控制系统。将在该次比赛结束后公开代码仓库。

数据预处理

查询词清洗

在进行数据噪声处理之前，我们首先要对用户输入的查询语句进行“清洗”，即去除一些无用信息，并规范化一些语言表达：

标点符号处理：

根据统计，用户输入的查询词中包含了“.”, “_”, “$”等非数字和英文字母的字符。这些字符需要区别对待。

需要删除的字符：包括”.”（常用在缩写后，如 Dr. vs.），”’”（单引号，常为所有格）等等。

需要替换为空格的字符：我们把空格作为单词与单词之间的分隔符。所以像“_”（下划线）等字符可以这么处理。

数字统一化处理：

在 Xbox 的游戏搜索中，数字常常作为游戏的版本号，如最终幻想 7，等等。然而不同人喜欢用的表达方式也不同，譬如最终幻想 7，有人喜欢用阿拉伯数字，有人喜欢用罗马数字。为了让它们表达同样的意思，这里都用阿拉伯数字规范化。

有些数字和字母之间没有空格隔开，这种情况也需要特殊处理（如”wwe12”，应该分解为”wwe”和”12”，分别代表游戏名称和版本号）。

大小写统一：

一般英文字母大小写不必区分处理。

通过以上清理手段，我们将每个 query 分解为若干个连续出现的单词。例如，对于查询“Gears_of war3”，我们通过处理后得到：

gears	of	war	3

关键词自动纠正

在使用 Google 等搜索引擎的时候，常常会拼写错某些关键词，而这时搜索引擎会非常人性化地猜测可能输错的词并进行修正。鉴于这次项目也是做的搜索引擎，所以一个合格的关键词自动纠正程序是非常必要的。

譬如以下两条搜索：

user	sku	query
6783cc3d297e110a969ff33f7b5b1624a5b49848	9854804	Geara of war 3
0010a136a422578130886603598d52f326a8927b	9854804	Gears of war 3

显然，第一条记录中"Geara"是将"Gears"错误拼写后的结果。而它们都能够点击到"9854804"这个产品，说明关键词纠正起到了作用。

以下步骤描述了关键词等价网络构建的过程：

关键词等价网络构建程序流程

1. 对于每条查询，将其清洗后变成连续的单词。对于每一对连续的单词，我们记录下它们的前驱后继关系(譬如上面的例子中，Gears 是 of 的前驱，of 是 Gears 的后继）。这种做法即是自然语言处理中常用的 N-gram 模型。这种模型能够在一定程度上保留语言的结构信息，而不是简单地把语言看做零散的若干个单词的集合；
2. 统计单词出现的次数、前驱和后继的集合（因为有大量查询，所以一个单词可以有不止一个前驱或者后继）；
3. 对于任意两个单词，排查他们是否是潜在的可替换单词（这个过程将在下一段程序展示）；
4. 根据第三步得到的替换关系，建立单词等价网络，并得到每个等价网络中的代表单词（也将在后面的程序展示）。

有了等价网络，任意两个通过网络能够相连的单词均是等价的。譬如我们发现“modern"会错写为"moderm"，而"moderm"又可替换为"modem"，于是我们便会发现"modem"也是"modern"的一种可能的错误拼写。

下面的程序判断两个单词是否可替换：

关键词可替换性判断

1. 待判断的两个单词长度必须大于某个阈值；

2. 如果两个单词的编辑距离过大，那么不予考虑。所谓的编辑距离，就是Levenshtein，用来衡量两个字符串之间的相似度。具体的做法可以用动态规划处理，其转移方程为：

$$\mathrm{F[i][j]} = min\{F[i-1][j]+1, F[i][j-1]+1, F[i-1][j-1]+cost(i,j)\}$$

其中，F[i][j]代表第一个单词前 i 个字符和第二个单词前 j 个字符完全匹配所需要编辑的字符数量。Cost(i, j)代表两个字符之间编辑代价。

3. 如果两个单词具有共同的前驱或者后继，那么这两个单词很可能是等价的；

4. 如果两个单词的出现次数差距非常大，那么很可能其中一个是正确单词，而另外一个是这个单词的错误拼写。

我们可以看如下的一个例子——“Modern warfare 3”

如果把 modern 错写为 mondern，那么这个两个词会有共同的后缀 warfare，并且根据统计，针对"2670133"这个产品，modern 出现了 767 次，而 mondern 仅仅出现了 3 次。根据以上信息，我们可以发现这个两个单词是等价的。

关键词等价网络构建

1. 根据上面得到的两两关键词等价信息，我们可以构建出一个等价图，图中每个节点代表一个单词，这些单词均等价；

2. 通过对图的深度优先搜素，找到该图中出现次数最多的单词，作为这个图中的代表单词。也就是说，这个图中所有的单词都可以改为该代表单词。

对于用户输入的查询，我们可以这样进行纠正：

1) 如果查询单词已经被以前的用户查询过了，那么该单词已经被我们建立的等价网络所包含。因此，直接替换为该单词所处等价网络的代表单词。

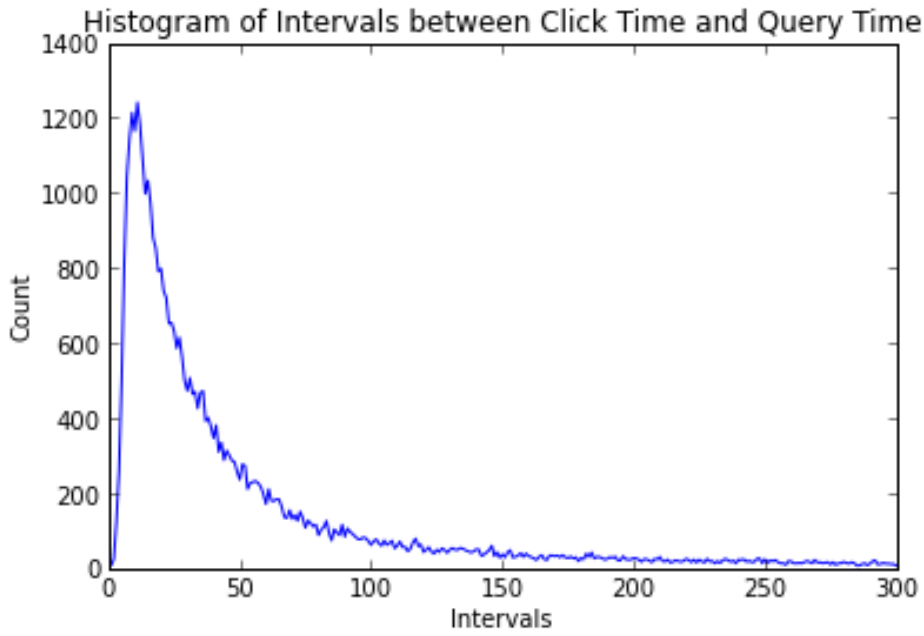
2) 如果该查询单词没有出现过，那么用该单词和其它所有单词运行“关键词可替换性判断”，找到可以替换的单词。

查询-点击相关性预处理

前文中提到了查询-点击噪声的问题，我们在这里予以解决。

本项目的训练数据和测试数据均提供了用户查询时间和用户点击时间这两个时间点。根据这个时间间隔，我们可以获取一定的关于查询-点击相关性的信息。譬如说，如果这两个时间非常接近，那么很可能是用户搜索了之后立刻点击到了一个想要的产品；如果两个时间非常远，那么很可能用户切换到了另外一个场景下，通过主页推荐或者其它方法点击了产品。

为了能够更直观感受时间间隔的分布，我们绘制了如下的图：



可以见得，上图中大多数用户会在搜索后大约 10 秒内点击产品。而随着时间间隔上升，点击率就急剧下降，同时在最后形成了“长尾”。

我们设定一个阈值为 156 秒，这个间隔是最大间隔值的平方根。间隔大于该值的查询训练数据就被忽略了。据统计，我们最终筛去了 7%的数据。然而去除这 7%的极有可能是噪声的数据，对于模型训练非常重要。

关键字分析

这一节，我们主要讲解前文中提到的 KeywordMatch 函数的实现。

该函数使用一个 sku 号码和用户查询作为输入，返回该产品与该查询的相关程度。注意，如果一个用户在训练数据中点击过该产品，那么测试的时候，该匹配返回值为 0（我们假设用户不会对同一个产品购买两次）。

算法流程如下：

关键字匹配分值

1. 将输入串清洗，之后用自动纠错函数进行拼写纠错；

2. 对于每个查询的单词，我们累加该单词所带来的收益 B，其计算公式如下：

$$\mathrm{B} = \mathrm{Log}(\mathrm{count} + 1) * \frac{count}{maxcount}$$

其中，count 为该单词在该产品所有历史查询中出现的次数。maxcount 为出现次数最多的单词出现的次数。Log 值中为 count 加一是为了防止 count 为 1 导致 Log 取 0。

上述公式表达了这样一种目的：如果一个查询单词在这个产品的查询历史记录中出现次数越多，那么该单词越有价值，用词频率代替频数以规范化；与此同时，如果这个词在该产品历史记录中出现次数多，可以说明该产品被搜索次数也较多（较受欢迎），所以乘上一个出现次数的对数，以奖励词频数。

3. 对于相邻两个单词，如果它们在该产品查询历史中同时出现过，那么就增加额外的奖励。

4. 最后函数返回的值为：

$$\mathrm{KeywordMatch} = \sum_{i=1}^{n} B(w_i) + \rho \sum_{i=1}^{n-1} Appear(w_i,\ w_{i+1})$$

其中 n 代表查询语句包含 n 个单词{w_i}。Appear()函数判断两个单词在训练数据中是否同时出现过。ρ为在两者之间取得平衡的系数。

个性化推荐

本文中使用经典的协同过滤算法来为用户进行个性化推荐。协同过滤的核心思想就是根据已有的一些历史记录，挖掘内在联系，从而对未来进行预测（推荐）。常用的协同过滤算法有如下几种，为了选择最恰当的方法，我们不妨先进行一些分析。

常用协同过滤算法比较

用户-用户相似度推荐：

该算法首先构建用户-产品矩阵。对于任意两个用户，根据他们的购买记录，选择合适的评价函数来评价这两个用户的相似度。推荐的时候，根据这些相似度对产品打分进行加权平均，以得到产品的推荐指数。

然而，在本项目中，用户个数接近 4 万个，求两两用户的相似度变得比较困难。于此同时，由于数据非常稀疏（如前文提到的），使得用户之间相似度也较难度量。

产品-产品相似度推荐：

这种方法的思想与上一个类似（把产品和用户的角色互换），只是先构建的是产品-用户矩阵，进而得到两两产品的相似度。由于本项目中只有 400 多个产品，所以这种方法在效率上会比较高。另外，基于产品的比基于用户的优的一点是，用户的口味千变万化，而产品较为简单，并且较为稳定。

隐变量模型：

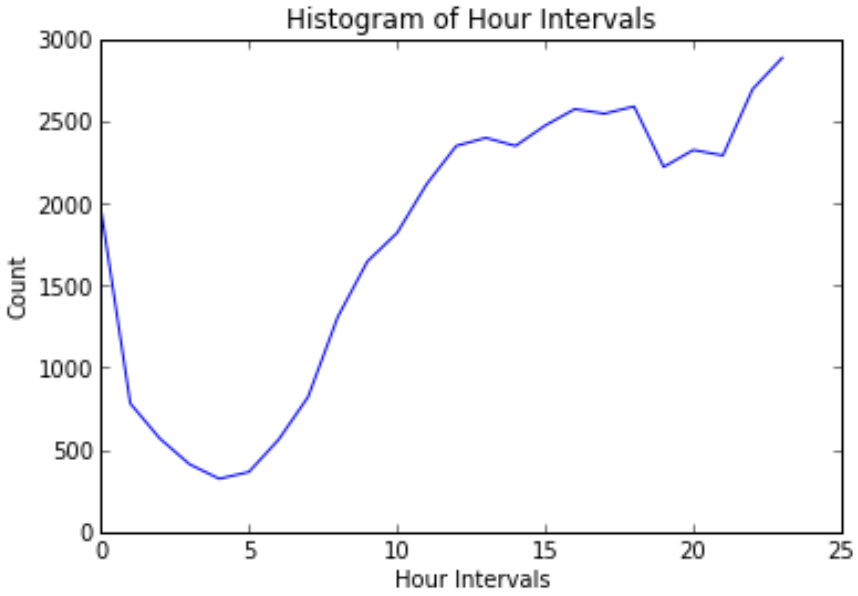
这种方法假设存在一些与产品选择相关的内在因素，比如轻松、娱乐、激烈等因素。该方法企图寻找这样一个隐变量所在的空间，将用户和产品均在该空间下进行表示，这样对产品的预测就转变为在该空间下进行内积了。为了训练这样一个模型，可以制定一个拟合标注（比如最小平方误差），并用梯度下降的方法训练。

不过这种方法同样需要较为完整的训练数据。对于我们所用的稀疏数据，效果并不是很好。

采用的方法：基于产品相似度的动态时间协同过滤

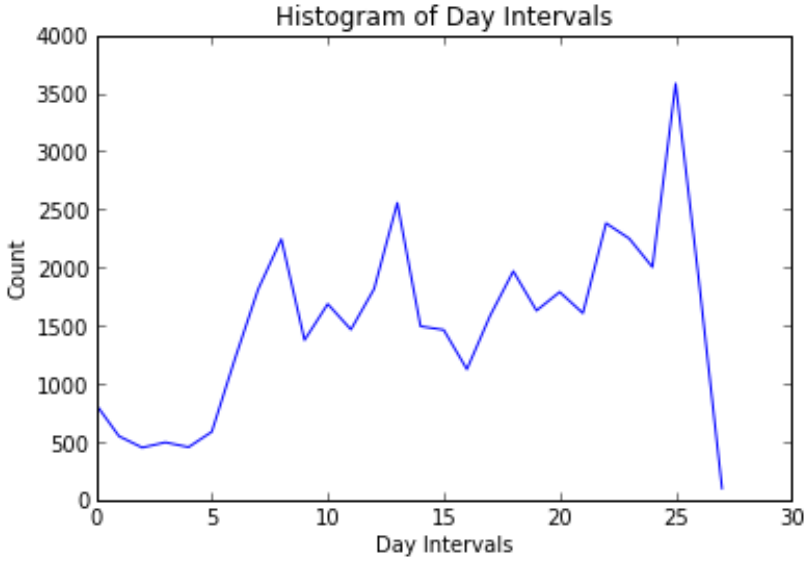
通过以上的分析，我们发现基于产品相似度的推荐方法会更适合于我们的需求。

需要额外说明的一点是，用户的需求是随着时间变化的。以下几个图将会说明这一点：



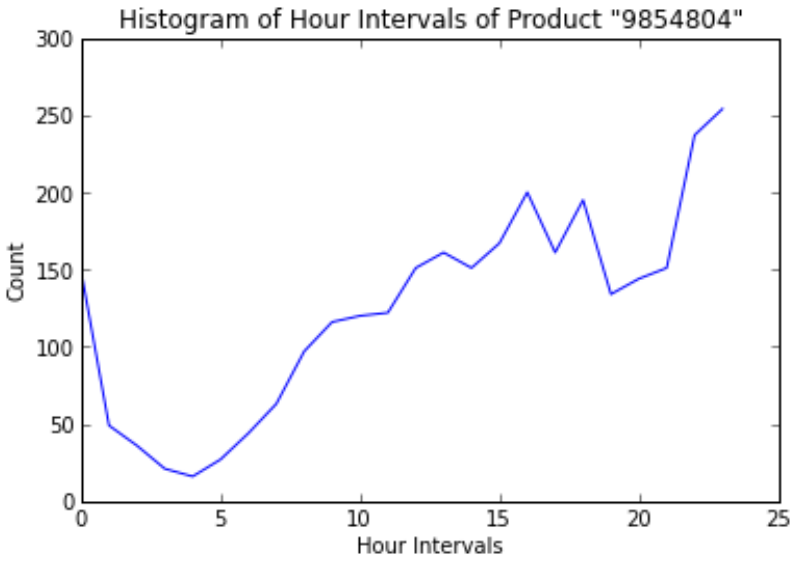
上图是所有点击情况关于小时的分布。可以看到，下午以及半夜的时候点击率最高，而清晨以及上午时候点击率就明显偏低。由于是游戏搜索，其原因很可能是因为清晨和上午是工作时间，而傍晚以及半夜时候是用户自由支配的娱乐时间。

下面的图展示的是点击情况在 8 月 11 号到 10 月 31 号这段时间的分布情况。图中每 3 天一个间隔进行选取。



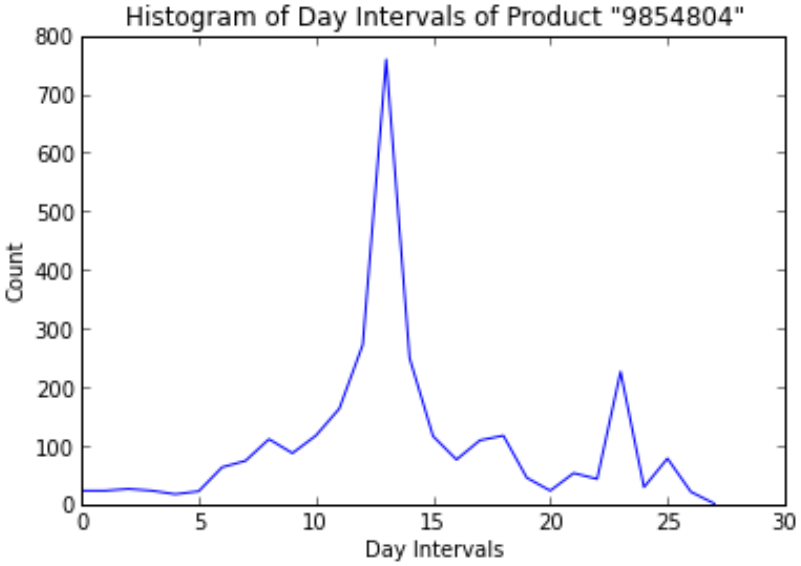
可以看到，产品交易在不同月份也有着波动。

以上说明的是总体点击率关于两个时间维度的分布。下面我们集中精力观察产品“9854804”的点击分布。



上图是关于 24 小时的分布情况。其分布基本与所有产品的总体分布一致。因此，这种分布往往与个人的习惯最相关。

下图是该产品在这 81 天里面的点击分布情况。



可见，这个产品在某段日子里面爆发了一阵，如果一个用户在这段时间进行搜索，很有可能他需要这个产品。反之，如果用户在 8 月 11 日去搜索，那么这个用户搜索该产品的可能性就比较小了。以上的这些信息，如果不根据时间进行分析，而只分析点击总量的话，是得不到的。因此，我们有必要建立关于时间的动态推荐模型。

1. 在查询之前需要做如下的准备工作：

   a) 建立产品-用户矩阵。矩阵为 01 矩阵，1 代表用户购买了某产品。

   b) 建立产品-产品相似度矩阵。相似度用 Jaccard Similarity 方法衡量，即

$$\text{similarity} = \frac{|A \cap B|}{|A \cup B|}$$

   其中 A 和 B 为点击过对应产品的用户集合。

   c) 建立每个产品关于时间的点击率分布。

2. 对于用户 user 在时刻 t 的查询，其和产品 i 的推荐值计算方法如下：

$$\text{Recommendation} = \text{dayVal(i,t)} + hourVal(i,t) + \eta * \frac{\sum_{k \in used(user)} Sim(i,k)}{\sum_{j \in C(i,d)} Sim(i,j)}$$

   其中 dayVal 和 hourVal 分别计算产品在对应时间上所占的点击率。used(user)计算了用户点击过的产品，d 为用户点击过的产品中，与 i 最低的相似度。C(i, d)表示与 i 相似度大于等于 d 的产品集合。Sim(i, j)衡量了两个产品的相似度。

   上面的公式中，前面两项是基准线，最后一项是用户的个性化数据。为了在这两者之前平衡，需要调整η。

算法评测

本项目是在笔记本上完成的。笔记本的配置为：Windows 7 64 bit; Intel Core i5 2.3GHz; 4.00 GB RAM。整个程序（从最初的模型训练到完成测试数据的评测）跑完只需要不到 10 分钟。

由于测试数据不存在标准结果，为了量化最后的结果，我们分离出训练数据的 10%进行测试。

为了更准确地观察算法表现，对于每个查询，我们让程序预测 5 个可能的点击，并按照可能性排序，如果排名第一的正好是被用户点击的，那么得到 1 分，点击的是排名靠后的则得分会少一些；如果点击的没有被预测出则得 0 分。

我们取平均的得分作为最后的得分，在 10%的测试数据上，我们可以得到 0.71 左右的分数。需要指出的是，该程序并没有精细地调整参数。也就是说，如果有足够的数据可以用来交叉验证的话，我们可以用网格搜索的办法调整一个好的参数，获得更优的结果。

回顾我们的评分函数：

$$\mathbf{Score(user, query, time, sku)}$$

$$= \boldsymbol{KeywordMatch}\mathbf{(sku, user, query)} + \boldsymbol{\lambda} * \boldsymbol{Recommendation}\boldsymbol{(sku, user}\mathbf{, time)}$$

$$= \sum_{i=1}^{n} \boldsymbol{B(w_i)} + \boldsymbol{\rho} \sum_{i=1}^{n-1} \boldsymbol{Appear(w_i,\ w_{i+1})} +$$

$$\mathbf{dayVal(i, t)} + \boldsymbol{hourVal(i, t)} + \boldsymbol{\eta} * \frac{\sum_{k \in used(user)} \boldsymbol{Sim(i, k)}}{\sum_{j \in C(i,d)} \boldsymbol{Sim(i, j)}}$$

可以发现它有如下的特性：

1. 如果用户没有输入搜索词，那么系统将直接给他推荐可能喜欢的产品；
2. 如果用户没有搜索记录，那么系统会用默认的基准线估计其兴趣；
3. 如果用户既没有输入搜索词，也没有任何搜索记录，那么系统会提供给他当前时间的最受欢迎的产品。

以上三点来看都是非常符合我们的直觉的。

由于测试数据没有标准值供我们参考，我们可以人工观察系统的部分输出结果，判断其输出的合理性。通过观察我们发现：

1. 如果用户指定的关键字为特定游戏名称，则相应的游戏将作为最有可能的推荐。以关键字“Gears of War 3”为例，在训练数据中该关键字出现 2422 次，其中 1399 次用户点击的游戏 ID 为 "9854804"，被选择率达到 57.8%，推断这即是游戏“Gears of War 3”的产品 ID。观察系统对测试数据中 1779 次关键字“Gears of War 3”搜索请求的回应，其中 ID “9854804”被 1674 次判断为最可能的点击，占到总数的 94.10%。

2. 系统会自动忽略没有太多实际意义的关键字，例如“xbox”等。举例来说，第 86 组测试数据关键字为 "Gears of War"，第 90 组测试数据关键字为 "Xbox 360 gears of war"。由于 xbox 360 是运行平台的名称，因此这个关键字不应该对搜索结果有太大影响。实际运行结果证实系统对两组测试前四位最可能游戏的判断是完全一致的。

3. 对于指定游戏类型但是与特定游戏无关的搜索请求，例如仅含关键字"kinect"和其他无意义关键字如“xbox"、"sensor”等等，但是不包含任何特定游戏名称的搜索请求，系统用返回这一类型游戏中最受欢迎的产品，与这一类**“模糊关键字”**搜索中用户通常的期待一致。例如训练数据中仅指定"kinect"的搜索请求有 483 次，其中有 97 次用户点击的产品 ID 为 “2897055"，占到总数的五分之一。可以判断这款产品为最受欢迎的 Kinect 游戏。在实际测试数据测试中，同样的搜索请求出现了 292 次，其中有 170 次系统返回了这款游戏为最推荐的产品，选中率为 58.22%。

   以上三点观察证实了这个并不复杂的推荐算法实际运行效果非常的人性化。

未来扩展

虽然这个项目花费了我们许多的时间和精力，不过它仍然有很多可以扩展的空间。即，如果能够得到更多的数据，那么算法会有更好的表现：

产品名称数据：

如果能够得到每个产品的名称，那么就不用根据用户查询-点击历史来猜测产品的名字了。这样会获得更精确的关键词匹配结果。

产品发布时间数据：

有了产品的发布时间，我们就不会给在发布之前的搜索推荐该产品。

产品打分：

用户对产品的打分也非常重要。在本项目中，我们只能知道用户是否点击的历史记录，也就是“有”或者“没有”这样的二值打分。相对于二值分数，一个 5 分满分的实数值分数会更加精确，从而获得更加有效的协同过滤模型。