# 摘　要

许多协同过滤的算法已经被研究，但是很少有适用于大型（几百万的用户和数据条目）动态（基础项目集在不断地变化）环境的研究被发表。在这片文章中，我们将介绍我们用来给 Google 新闻用户做个性化推荐的协同过滤算法。我们主要用三个算法来生成推荐内容：使用 MinHash 聚类的协同过滤算法，概率潜在语义索引（PLSI），共同访问计数法（covisitation counts）。我们用一个线性模型来综合不同算法生成的推荐结果。我们的算法是内容无关的，因此领域独立，只要做很少的工作，就可以将该算法应用于其他语言和程序。这片文章将先描述算法和系统安装的细节，然后展示它在 Google 新闻的推荐引擎上工作的效果。

**分类和主题描述符**: H.4.m [Information Systems]: Miscellaneous

**总括**: Algorithms, Design

**关键字**：Scalable collaborative filtering, online recommendation system, MinHash, PLSI, Mapreduce, Google News, personalization

# 第1章 介绍

互联网永远不会缺乏内容。目前面临的挑战是如何为自己找到正确的内容：或是能够满足你当前信息需求的内容，或是你将喜爱去阅读、收听或观看的内容。搜索引擎能帮你解决前一个问题。特别是当你正在寻找的内容，可以被制定为一个关键字查询的时候。然而，在许多情况下，一个用户也许根本不知道她要找什么。这种场景常常发生在用户做一些诸如浏览新闻或观看视频等事情的时候，此时，用户常常会放弃搜索转而去浏览一些网站，如 news.google.com, www.netflix.com 等，同时抱着“来点有意思的”的心态去寻找一些可能会激起他们兴趣的内容。在这种情况下，我们将要做的是根据用户在相应网站上的历史活动记录所展现的兴趣来向她推荐内容。

协同过滤是一项基于用户和群体数据的旨在学习用户喜好并进行推荐的技术，是一项与基于内容的过滤（如基于关键词的搜索）相互补充的技术。协同过滤最广为人知的应用可能就是 Amazon.com 的推荐系统，它根据用户以前的购物记录来推荐新产品。已有多种协同过滤算法被提出（详见第 3 部分），我们的目的是建立一个可扩展的在线推荐引擎。它可以在一个具有大量网络资源的平台上做个性化推荐，如 Google 新闻。虽然推荐质量上没有特别的优势，但以下的特点足以使我们的系统从现存的大部分推荐系统中脱颖而出。

## 1.1 可扩展性

Google 新闻(*http://news.google.com*),在数天之内就会被几百万个不同的访问者访问。信息条目的数量，即由新闻集群确定的新闻故事的数量，也是几百万的数量级。

## 1.2 信息条目的混合

大多数系统都假设基础信息条目集是静态的或变动极少的，这就导致系统常常需要逼近地更新模型或重建模型来纳入新的信息条目。重建，通常是一个耗费资源的任务，所以不是很频繁（每个几小时一次）。然而，像 Google 新闻这样大的资源集，基础信息条目集在几分钟内就会经历一次变动（插入和删除），在任何给定的时间点，感兴趣的新闻总是那些在最近两小时内出现的信息条目。因此任何更新时间需要数小时的模型，它的信息也许不再激起用户的兴趣，实际上它的更新并没有起作用。

基于以上的原因，我们发现现存的推荐系统不能满足我们的需求，因此探索了一个新的可扩展的算法。我们相信，Amazon 在推荐上具有同样的可扩展性，但是在信息条目的混合上，我们的系统与他们有显著的不同。这篇文章描述了我们方法的基本算法及包含的系统组件。论文剩下部分的内容将如下组织：第 2 部分描述问题的设置。第 3 部分简要介绍了相关工作。第 4 部分介绍我们的算法；即，基于用户聚类（使用 MinHash 和 PLSI 算法）和信息条目间的共同访问的推荐算法。第 5 部分描述了如何实现这样一个系统。第 6 部分报告与其他协同过滤算法比较分析的结果以及对系统实际通信质量的评估。第七部分我们将以提出一些结论和开放性问题作结。

# 第2章 问题设置

Google 新闻是一个由计算机生成的新闻网站，它从世界上 4500 多个新闻站点收集新闻，经过分类、去重，根据读者的个人兴趣展现给用户。依据国家和语言的不同，我们的新闻具有多个版本。在 Google 新闻主页的左上角，有 Top Stories, World, U.S. Business 等主题分类。每个主题包含三个相关的头条新闻。Top Stories 的左边是一个导航栏，点击相关的分类链接，将会跳转到相应分类的完整页面。上面描述的是一个未登录用户的新闻主页。如果你登录了 Google 账户并参与了 Google 各种产品的 Search History 特性，你就可以享有以下两项额外的功能：

1. Google 将会记录你的搜索查询和新闻浏览，当你在线的时可以查看这些记录。这使你可以很方便地浏览以前读过的内容。
2. 在 Top Stories 部分的下面，你会看到标记着 “Recommended for *youremailaddress*” 的依据你的历史点击记录推送的三条新闻。

我们项目的目的是为已登陆的用户提供推荐，推荐的依据是她们的历史点击记录以及她们所属的集体的历史点击记录。在我们的设置中，用户对文章链接的一次点击被当作对该文章的一次正投票。这使得我们的问题设置与 Netflix, MovieLens 等向用户对电影征求一个 1-5 的等级评分的方式大不相同。主要的区别在于：

1. 与接收一个 1-5 的确定的评分或像 amazon.com 那样将一次采购当作一个正投票来说，将点击当作一次正投票显得很粗糙。用户投票的真实性可以使用不同的机制进行跟踪，由于本文的目的是讨论协同过滤算法，而不是如何收集用户投票。为了专注于本文的目的，我们将假定点击确实代表了用户的兴趣[1]。

[1] 通常点击并不能代表用户对该新闻感兴趣，但在 Google 新闻中，这种情况并不常见，每条新闻都有摘要，用户在点击之前可以了解新闻的主要内容。

2. 相对于 Netflix, MovieLens 等向用户征求一个 1-5 的等级评分来说，当点击被算作用户的正投票时，却没有任何东西来表示用户的负投票。

## 2.1 我们操作的可伸缩性

Google 新闻网站是世界最流行的新闻网站之一，接收上百万的页面浏览量和上百万的用户点击。用户点击记录的大小有很大的变动范围，对于某个特定的用户来说，可以是零到几百，甚至是几千。我们用一个月的时间观测到的新闻数量是几百万的数量级[2]。此外，正如前面所说的那样，新闻集是不断变化的，每分钟都会有新的新闻被添加和旧的新闻被删除。

## 2.2 问题陈述

由前面的概述，我们的推荐系统可以被描述如下：有N用户(U$\{u_1, u_2, \dots, u_N\}$)，M条新闻S$\{s_1, s_2, \dots, s_M\}$，对于一个特定的用户u，她的点击记录为$C_u = \{S_{i_1}, S_{i_2}, \dots, S_{i_{|C_u|}}\}$，给u推荐K条她肯感兴趣的新闻。用户每次登陆进入主页时，我们计算一次。当用户点击 Top Stories 中的推荐的链接时，我们将展现一个推荐的完整页面。即使用一个不同的K值计算一次。此外，我们要求系统能即时地吸收新收集的点击记录，以提供一个即时的需求。

## 2.3 严格的时间需求

Google 新闻网站努力为每个页面维护一个严格的时间响应。实际上，主页面和任何一个主题的完整页面通常都在一秒内生成。除去服务器前端通过存取存储着新闻内容的各种索引来生成新闻集群以及生成作为回复 HTTP 请求的响应中的 HTML 内容的时间，就只剩下几百毫秒的时间用来给推荐引擎生产推荐内容。

已经描述了问题设置和潜在的挑战，接下来，在描述算法之前，我们还要对推荐系统的相关工作做一个简要的概述。

[2] 正如前文所提到的，我们从不同网站收集新闻再聚集起来展现给用户。为了专注于我们讨论的目的，当我们说一条新闻时是指被 Google 新闻标注为是同一条新闻的一组新闻文章。

# 第3章 相关工作

推荐系统大致可以分为两类：基于内容的推荐系统和使用协同过滤的推荐。在基于内容的推荐系统中，信息条目的相似度是依据它们的内容来定义的，那些与用户投以高分的信息条目的相似度被用来判定是否进行推荐。然而，在一些领域，如新闻领域，用户对某篇文章的兴趣并不总是能由文档所表现的主题来特征。此外，我们的目的是建立一个可以被应用到其他领域（如图像，音乐，视频等）的系统，其基础内容可能很难分析，因此我们决定开发一个内容无关的系统。单就 Google 新闻的推荐来说，基于内容的推荐可能会有同样的效果，我们计划在未来对它进行探索。协同过滤系统使用用户对信息条目的评分来计算推荐内容，通常是内容无关的。在 Google 新闻的系统中，信息条目的评分是一位 2 进制：被点击对应 1，没有被点击对应 0。协同过滤系统可以被细分为以下两类：基于记忆的协同过滤和基于模型的协同过滤。接下来，我们将简单地看一下这两类算法的相关工作，但更建议读者学习附文 [1]的内容。

## 3.1 基于记忆的算法

基于记忆的算法使用用户过去的评分来进行评分预测，通常某用户对一条信息的评分取值为其他用户对该信息的评分的加权平均数，权重正比于两个用户的相似度。通常用户相似度的测量包括皮尔逊相关系数和评分向量的余弦相似度。成对的用户相似度矩阵$\mathrm{w}(u_i, u_j)$通常是离线计算的。系统运行时，用户$u_a$对信息条目$s_k$的评分由以下公式给出：

$$r_{u_a, s_k} = \sum_{i \neq a} I_{(u_i, s_k)} w(u_a, u_i)$$

在我们的设置中，评分是一位二进制。如果用户$u_i$点击了$s_k$ ,指示变量$I_{(u_i, s_k)}$的值就是 1，否则为 0。因此预测评分可以用一个阈值进行二值化。

基于记忆的方法以其简单和相对简单的训练阶段而越来越流行，然而，正如附文 [2]所指出的那样，要想使基于内存的算法具有一个更好的可扩展性是一个很大的挑战。实际上，附文 [2]专注于通过实例的选取以减少训练集大小

的方法来达到这种可扩展性。然而他们的技术并不适用于我们的情况，因为我们的新闻集是不断地更新变化的。例如，在他们的方法中，有一个叫做 TURF1 的方法试图为每一个信息条目找到这样一个用户训练子集，对于任意给定的用户都足以预测出她在该信息条目上的评分。显然，这种方法无法在 Google 新闻上工作，因为一条离线状态下算出的旧新闻条目将因为其太旧而无法用于推荐。

另一种基于记忆算法的变形，尝试通过计算信息条目的相似度矩阵来代替计算用户的相似度矩阵。相似度的计算基于信息条目所获得的用户的评分，测量方法主要有皮尔逊相关系数和向量相似度。在测试阶段，与那些被用户评以高分的信息条目具有高度相似值的信息条目将会被推荐给用户。

## 3.2 基于模型的算法

相比于基于记忆的算法，基于模型的算法尝试通过用户以前的评分来对用户进行建模，并使用该模型来预测那些未知的信息条目的评分。这种方法的最早的例子之一（参见引文 [3]）提出了两种可供选择的概率模型：聚类模型和贝叶斯模型。这篇文章的缺点是它只将每个用户分到单个类别中，然而更直观的情况却是一个用户对不同的主题有不同的口味。与我们的方法相似，大多数在基于模型算法上的最新工作，都是通过将用户分到多个簇或类别中来捕获用户的多方面兴趣。基于模型的算法有：潜在语义索引（LSI） [4]，贝叶斯聚类 [3]，概率潜在语义索引（PLSI） [5]，多乘因子模型 [6]，马尔可夫决策过程 [7]和潜在狄氏分配 [8]。以上大多数算法的计算都非常耗费资源。我们更专注于开发一个新的，具有高度可扩展性的聚类模型。为此，我们重新设计了 PLSI 算法 [5]，用一个 MapReduce [9]计算模型来使它具有高度的可扩展性。

# 第4章 算法

我们用一种混合了基于记忆和基于模型的算法来生成推荐内容。对于基于模型的算法，我们使用了两种聚类算法：PLSI 和 MinHash；对于基于记忆的算法：我们使用了信息条目的共同访问的算法。对每条新闻都用每种算法分配一个数字评分（以使得更好的推荐获得更高的评分）。给定一个候选新闻集合，由聚类方法得到的分数（$r_{u_a,s_k}$）正比于如下公式，即

$$r_{u_a,s_k} \propto \sum_{c_i:\, u_a \in c_i} w(u_a, c_i) \sum_{u_j:\, u_j \in c_i} I(u_j, s_k)$$

$w(u_a, c_i)$正比于$u_a/c_i$的小数值。共同访问算法给每个候选新闻分配的评分正比于该新闻与其他新闻在该用户的点击记录里同时出现的次数。

每种算法所得的分数通过公式$\sum_a w_a r_s^a$结合（$w_a$是算法a的权重，$r_s^a$是新闻s使用算法a获得的分数），以用于所有新闻的排序。排序结果的前K条新闻将作为用户$u_a$的推荐内容。权重（$w_a$）通过以下学习获得：在一个预选的离散参数空间（可能的权重取值集合）中，对每个参数取值进行一次现场实验（详见6.5 节），以观察哪个参数获得最好的效果。未来，我们计划探索用 SVM [10]（使用线性内核）来学习这些权重。接下来，我们将详细地描述每一个算法。

## 4.1 MinHash

MinHash 是一个概率聚类算法，它用一个正比于两个用户的投票（点击）交集比的概率将用户分配到同一个群集中。对于每一个用户$u \in U$，她的历史点击记录为$C_u$，则用户$u_i, u_j$的相似度定义为她们点击记录交集比，即$S(u_i, u_j) = \frac{|C_{u_i} \cap C_{u_j}|}{|C_{u_i} \cup C_{u_j}|}$。该相似度也被称作雅格系数，取值在 0 到 1 之间，另一个与之对应的系数是距离函数，即$D(u_i, u_j) = 1 - S(u_i, u_j)$ [11]，只要测量出相似度$S(u_i, u_j)$，即可计算获得距离函数$D(u_i, u_j)$。在理想实验中，对于一个给定的用户$u_i$我们会计算该用户与其他所有用户$u_j$的相似度。然后用$S(u_i, u_j)$作为权重来向$u_i$推荐

$u_j$投票的新闻。然而，在实际中这样做显然是不可扩展的。也许有人会提出这样一个修剪措施，例如用一个哈希表来找出那些至少有一个共同投票的用户。但是即便这样，也不能将候选的用户对降低到一个可以管理的数量。因为有很多受欢迎的新闻会同时被很多用户投票。对于这样一个巨大的用户对，即使是离线计算也是不可行的。不必担心，局部敏感哈希算法 [12]可以拯救我们，它是一个可证明的次线性时间的近邻搜索技术。

### 4.1.1 LSH

LSH 技术最初是由迪克和莫特瓦尼为了有效地解决近邻搜索问题而引入的，从那以后，就可以在很多应用中发现它的身影 [13]。它的核心思想就是使用几个哈希函数对数据点进行散列，并确保对于每一个函数，那些邻近的对象发生碰撞的概率高于相距较远的对象。然后就可以通过散列查询点并检索包含这个点的桶中的元素来确定该点的近邻点。已知的 LSH 方案存在以下的距离或相似性的测量：海明规范 [14]，$L_p$规范 [15]，雅格系数 [16]，余弦距离和推土机距离（EMD） [11]。我们使用的是雅格系数。幸好存在一个 LSH 方案 Min-Hashing（即 Minwise Independent Permutation Hashing）可以用于测量雅格系数，它是由科恩 [17]首次引入用来计算传递闭包和可达集的（详见 Broder [18]）。

Min-Hashing 方法的基本思想是先对信息条目集（S）进行随机排列，然后对每一个用户$u_i$，将她的哈希值$\mathrm{h}(u_i)$设置为该随机排列中第一个属于该用户的点击历史集$C_{u_i}$的信息条目的索引。显然对于一个随机排列，相当于从S的所有排列组成的集合中均匀地选择一个 [19]。于是，两个用户具有相同的哈希函数的概率等于她们的相似度，即雅格相关系数。因此，我们可以把 Min-Hashing 看作一个概率聚类算法，每一个哈希桶对应于一个集群，两个用户被放进同一个集群的概率等于她们的历史点击集合的交集比$\mathrm{S}(u_i, u_j)$。与附文 [12]类似，我们可以为每个用户串联p个哈希键（p），这样任意两个用户串联哈希键相同的概率就是$S(u_i, u_j)^p$。换句话说，通过串联哈希键，我们使集群更加细化，以便有更多这样的集群并且每个集群中用户之间的平均相似度更大。对于一个给

定的用户来找她的近邻来说，这种细化的集群具有很高的精确度和较低的回调次数。我们可以通过多次并行地重复这个步骤来提高回调次数。例如，我们可以将每个用户散列到q个集群中去，每个集群由p个串联的 MinHash 键进行标识。通常，我们将p和q的取值分别设定在$2-4$和$10-20$之间。

显然，在一个具有数百万的信息条目集上生成随机排列，然后再存储它们用于计算 MinHash 值是不可行的。为此，我们可以生成一组独立、随机的种子值，每个 MinHash 函数（即上面讨论的p×q）对应一个，用由每条新闻的Id和一个随机种子计算所得的值作为散列该新闻的哈希值。由此计算所得的哈希值当作随机排列索引的代理。通过选择哈希值的范围为 0 到$2^{64}-1$（64 位无符号整型），只要信息条目集的大小小于$2^{32}$，我们就可以确保我们不会遇到生日悖论，因此碰撞的概率很小。通过这种方法计算的 MinHash 值与理想的 MinHash 值具有相似的属性 [20]。接下来，我们描述我们是如何使用一个可扩展的方法在数以百万的用户和信息条目上计算 MinHash 值的，即使用 MapReduce 计算框架。

### 4.1.2 使用 MapReduce 的 MinHash 聚类

MapReduce [9]是一个非常简单的在大型计算机机群上进行计算的模型，它可以在相对较短的时间内处理大量的数据并且对机器的数量有很好的可伸缩性。几十或数百 TB 的数据用几千台机器的机群可以在数小时内处理完。计算工作被分为三个阶段。

将输入匹配成键值对：在 Map 阶段，我们在不同的机器上独立地、并行地读入输入记录，并且将每个输入匹配成具有 0 个或更多键值对的集合。在我们的情况下，每个输入记录（每个用户$u_i$有一个）是某个用户的历史点击集合$C_{u_i}$。我们遍历某用户的历史点击集，为该用户计算p×q个 MinHash 值。计算单个 MinHash 值非常简单：我们对历史点击集中的每个条目，用它的 Id 和对应的哈希函数种子进行散列[3]，然后取这些哈希值中最小的一个。最后，我们将该 MinHash 值分成 q 组，每组 p 个 MinHash 值。对每组，我们串联该组中的 MinHash 值来获得该组对应的集群 Id。输出的键值对（该用户从属于的每个集

[3] 每台 map 机器上都有一个相同的随机种子副本

群有一个）是集群 Id（键）和用户 Id（值）。

对键值对进行分割和重洗：在这个阶段，由上一阶段最后输出的键值对将被分为一些分区（碎片），通常依据键的值进行分割。每个碎片按照键进行排序以便具有相同键（在我们的情况中是集群 Id）的键值对出现在一起。

减少键值对：在 Reduce 阶段，我们获取每个集群的用户 Id 列表并修剪那些成员很少的集群。在一个单独的处理过程中，我们颠倒集群的成员关系，对每个用户维护一张她从属于的集群列表以及她的历史点击集。这些信息被存在一个大表中，以用户 Id 为键。（详见 5.2 部分的用户表 UT 的表述）。

## 4.2 PLSI

PLSI 在附文 [5]中被介绍，该文中哈夫曼开发了概率潜在语义模型用来进行协同过滤。它将用户（$u \in U$）和信息条目（$s \in S$）视为随机变量，分别取值于所有可能的用户和信息条目。用户和信息条目的关系是通过对用户和信息条目的联合分布进行建模，并作为一个混合分布来学习的。一个隐含变量Z（取值$z \in Z$，且$\|Z\| = L$）用来代表这个关系，变量 Z 可视作用户社区（具有相同爱好的用户群体）和信息条目主题（流派）。该模型可以写成下面这个公式给出的混合模型的形式：

$$p(s|u;\theta) = \sum_{z=1}^{L} p(z|u)p(s|z)$$

该模型是完全由参数θ指定的，代表着条件概率分布（CPD）$p(z|u)$和$p(s|z)$。该模型的关键点在于隐含变量 Z 的引入，它使得用户和信息条目条件独立。该模型也可以当作生成模型，即先选定随机变量 Z 中的一个状态 z，对任意用户 u 利用 CPD p(z|u)算出 z 的概率；然后，对于选定的 z 利用 CPD p(s|z)对信息条目 s 进行采样。

### 4.2.1 使用 MapReduce 的 EM 算法

可以通过数据训练来学习上节中提到的共生模型：用 T 大小的数据来评估 CPDs $p(z|u)$和$p(s|z)$，确保所有数据点发生的概率最大化（原理参照最大似然估计），即等价于最小化下面的经验对数损失公式：

$$\mathrm{L}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \log\big(p(s_t|u_t;\theta)\big)$$

我们用期望最大化（EM）算法来学习该模型的最大似然参数。EM 算法的详细内容以及它的推导可以参照附文 [5]。它是一个迭代算法，每次迭代分为两个步骤：E 步骤，包含 Q 变量的计算（即，A-验隐含类概率），公式如下：

$$q^*\big(z;u,s;\hat{\theta}\big) \coloneqq p\big(z|u,s;\hat{\theta}\big) = \frac{\hat{p}(s|z)\hat{p}(z|u)}{\sum_{z\epsilon Z}\hat{p}(s|z)\hat{p}(z|u)}$$

；M 步骤使用上面计算的 Q 函数来计算下面的分布：

$$p(s|z) = \frac{\sum_u q^*\big(z;u,s;\hat{\theta}\big)}{\sum_s\sum_u q^*\big(z;u,s;\hat{\theta}\big)},$$

$$p(z|u) = \frac{\sum_s q^*\big(z;u,s;\hat{\theta}\big)}{\sum_z\sum_s q^*\big(z;u,s;\hat{\theta}\big)}$$

注意，在上面的公式中，$\hat{p}$代表 EM 算法上一次迭代的参数估计[4]。要具有可扩展性，在单个机器上执行 EM 算法是不可行的，为了估算需要的主存容量，我们设 M=N=10,000,000，L=1000。而计算 CPDs 需要的内存将是$(M+N)\times L\times 4\sim 80GB$（使用 double 类型存储，单个 double 类型 4 字节）。接下来，我们演示如何使用 MapReduece 框架并行地用 EM 算法来计算 PLSI 参数，从而使得它具有可扩展性。为使用 MapReduece 框架，我们需要先改写公式：

$$q^*\big(z;u,s;\hat{\theta}\big) \coloneqq p\big(z|u,s;\hat{\theta}\big) = \frac{\frac{N(z,s)}{N(z)}\hat{p}(z|u)}{\sum_{z\epsilon Z}\frac{N(z,s)}{N(z)}\hat{p}(z|u)}$$

$$N(z,s) = \sum_u q^*\big(z;u,s;\hat{\theta}\big)$$

$$N(z) = \sum_s\sum_u q^*\big(z;u,s;\hat{\theta}\big)$$

$$\hat{p}(z|u) = \frac{\sum_s q^*\big(z;u,s;\hat{\theta}\big)}{\sum_z\sum_s q^*\big(z;u,s;\hat{\theta}\big)}$$

[4] 第一次迭代前，我们将$\hat{p}$设置为规格化的随机值，以形成一个概率分布。

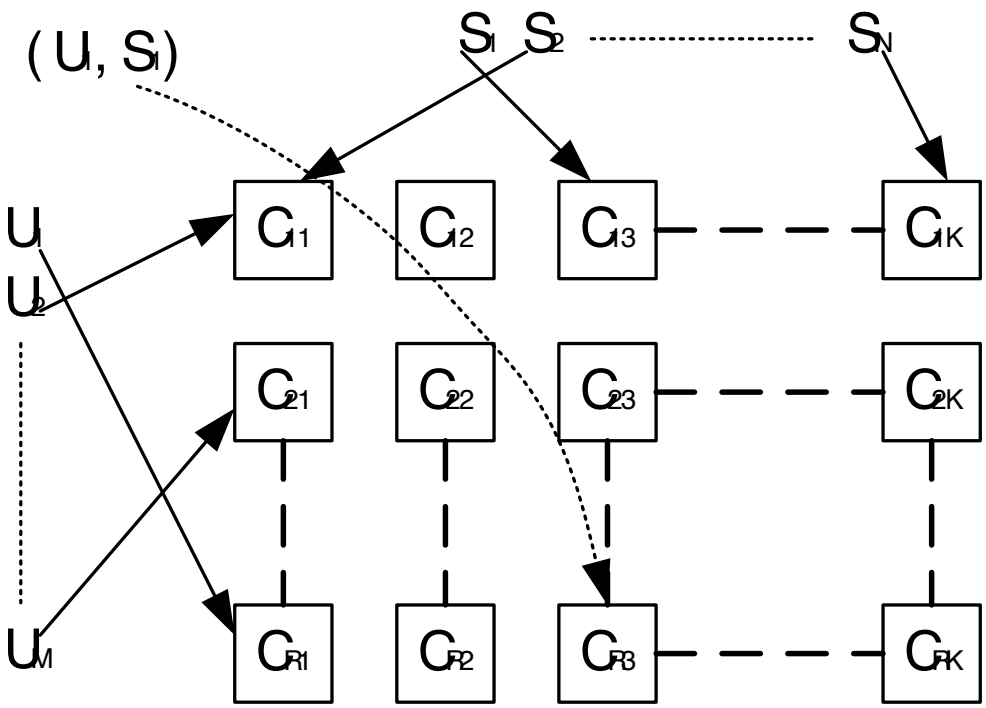


图 1：应用 MapReduece 的 EM 算法中用户和信息条目的分片

给定一个用户新闻对$(u,s)$,为计算$q^*(z;u,s;\hat{\theta})$，应该从上一次迭代中获得以下充分统计量：$\hat{p}(z|u)$，$N(z,s)$和$N(z)$。我们假设每次迭代前，对于每个用户和每条新闻这些统计量都是可用的。观察上面公式可得：只要有以上的充分统计量，点击日志中的每个$(u,s)$对的$q^*(z;u,s;\hat{\theta})$值都可以独立并行地计算。我们将描述 MapReduce 单次迭代的执行过程。考虑有一个大小为R×K的计算机网格（如图 1） 。用户和信息条目被各自分成 R，K 组（使用一个作用于它们 Id 的函数进行分组），$(u,s)$对应的点击数据被发送给网格中的$(i,j)$号机器，其中 i 是用户 u 从属于的分片，j 是信息条目 s 从属于的分片[5]。注意到第$(i,j)$号机器只需要各自加载第 i 分片中的用户和第 j 分片中的信息条目所对应的 CPDs 和充分统计量即可。这样，每台机器的内存需求变为只需加载原来用户 CPD 的$\frac{1}{R}$、信息条目 CPD 的$\frac{1}{K}$。计算完各自的$q^*(z;u,s;\hat{\theta})$后，输出三组键值对$(u,q^*)$, $(s,q^*)$和$(z,q^*)$。

[5] 点击数据在迭代过程中不变，只需要在开始时进行一次分片即可。

接收的键值对对应于信息条目 s 的 reduce 分片计算下次迭代所需的$N(z,s)$（为所有的 z 值）；接收的键值对对应于用户 u 的 reduce 分片计算$p(z|u)$，而$N(z)$由接收的键值对对应于 z 的 reduce 分片进行计算[6]。注意，所有 reduce 分片上的计算都是简单的加法。

### 4.2.2 使用动态数据集合 PLSI

尽管过去的研究显示 PLSI 算法要好过其他的算法，但是它有一个根本问题，即每次新的用户或信息条目被添加时，整个模型就需要重新训练。通过并行化模型，我们可以快速地学习到概率分布。但是该模型仍然不是实时的。对于只有少量的新增用户和信息条目的情况下，论文中给出一些模型更新的启发式的算法。尽管每天新增的用户数量不多，但是新闻集的变动很大，使得那些启发式的算法都不能被应用。为了解决这个问题，我们使用从上面的模型学习所得的$\mathrm{p}(z|u)$值来开发一个近似的 PLSI 版本。其中 z 值被视作集群，而分布函数表示了组成的集群的分数值。我们追踪从每个集群中的每条新闻所观察到的活动。当一个用户点击一条新闻时，我们更新该用户所从属于的每个集群中与该条新闻相关的计数（权重计数的更新基于该用户集群成员的置信度，即$\mathrm{p}(z|u)$）。规格化的权重矩阵就是$\mathrm{p}(s|z)$的分布。注意，这样计算的$\mathrm{p}(s|z)$可以实时更新。但是该模型仍不能解决用户添加的实时性问题。尽管有一些启发性的算法可以用来解决这个问题，但我们将它放到以后的工作中。对于新增的用户，我们依靠新闻到新闻的共同访问的算法，接下来我们将描述该算法。它是基于新增用户有限的历史来生成有用的推荐的。

## 4.3 使用用户聚类进行推荐

一旦我们使用用户聚类，在服务时就要为每个集群维护以下统计数据：不同的新闻收到的由该集群中所有用户成员触发的点击次数（随时间衰减）。在 PLSI 中，点击计数是用集群的分数值来进一步加权计算的。注意到由于集群是细化的，每个集群包含的用户数是有限的，因此，被集群成员点击的无重复

[6] 接收的键值对对应于 z 的 reduce 分片接受到很多数据（每个$(u,s)$对对应一条记录），因此将它们直接聚集到一个分片可能会成为一个瓶颈，我们可以在 MapReduce 的洗牌阶段做一些预处理。

的新闻的数量通常不大（最多几千）。当评估一个候选新闻s作为用户u的可能推荐时，我们先按照以下步骤利用聚类来计算一个未规格化的新闻得分：获取该用户从属于的所有集群；对每个集群，查找该集群中的所有用户成员在新闻s上的点击次数（按时间折算，并按该集群中所有用户的点击数之和来规格化）；最后，将从每个集群获得的次数相加来计算新闻得分。然后将由此方法得到的分数归一化以使它们的值都在 0 和 1 之间。我们独立地使用 MinHash 和 PLSI 聚类来计算这些规格化得分。

## 4.4 共同访问

我们基于信息条目的推荐生成技术使用了共同访问实例，其中共同访问是指有两条新闻在指定的时间（通常是几小时）内被同一用户点击的事件。设想这样一个图，它的节点代表信息条目（新闻），带权边代表共同访问实例的时间折现数值。边可以有方向，用来表示一条新闻在另一条之后被点击，也可以不需要如果我们不关心点击顺序。我们用邻接表来表示这个图，并用一个以信息条目的 Id 为键的大表进行存储（详见 5.2 部分有关新闻表 ST 的描述）。每当我们收到一个点击动作时（设用户为$u_i$，新闻为$s_k$），我们检索该用户最近的历史点击集$C_{u_i}$并遍历它的所有相。对所有的信息条目$s_i \in C_{u_i}$，我们修改$s_j$和$s_k$对应的邻接表子链，添加一条与当前点击对应的记录。如果该记录已存在，我们就更新其年龄折现计数。给定一条信息条目，它的近邻就是那些与它被共同访问的信息条目集，权值是年龄折算值，用以表示它们被共同访问的频率。这给人以下简单的直觉“那些看过这条信息的用户也看过它的近邻信息”。

对于一个给定用户$u_i$，我们按一下步骤给一个候选信息条目 s 生成基于共同访问的推荐分数：我们先取得该用户的最近历史点击集$C_{u_i}$，限于过去的几小时或几天[7]。对所有$s_i \in C_{u_i}$，我们先从邻接表中存储$s_i$的子链中查找$s_i$,s 对所对应的记录，然后将该记录中的值经过归一化（依据$s_i$子链中所有记录值之和进行归一化）后加到$s_i$的推荐得分中。最后，对所有信息条目的得分进行一个

[7] 我们将基于共同访问的推荐视作用户短期行为的函数（过去几小时内的点击历史），而将基于用户的推荐（MinHash 和 PLSI）视作用户长期行为的函数

线性伸缩的归一化，以使得所有的得分值在 0 到 1 之间。

## 4.5 候选信息生成

到目前为止，我们都假设在生成推荐之前有一个候选信息条目集。这些候选信息可以由两种方式生成：新闻前端（NFE）可以基于以下因素生成一组候选信息，例如新闻版次，用户的语言偏好，新闻的新鲜度，以及用户选择的可制定的内容等。确切的用于生成候选集的评分方法是独立于我们的推荐系统的。对于一个特定用户，可以交替地使用以下方法生成候选集：即取所有与该用户从属于同一集群的用户点击的所有新闻和所有与该用户历史点击集中的新闻被共同访问的所有新闻的并集。按照我们的算法，只有以上集合中的新闻才会获得非零得分，因此，它是一个充分候选集。

# 第5章 系统安装

将以上算法应用到一个实时的推荐系统需要以下三个主要组件：一个离线组件负责定期地根据用户的点击历史对她们进行聚类；一组在线服务负责两类主要任务：（a）当有用户点击一条新闻时，更新用户和新闻的相关统计量。（b）当有用户请求推荐时生成相应的推荐内容；以及两类数据表：（a）用户表 UT，由用户 Id 进行索引，用于存储用户的点击历史和聚类信息。（b）新闻表 ST，由新闻 Id 进行索引，用于存储每个 story-story 和 story-cluster 对的实时点击计数。下面，我们将来描述每个组件的细节内容。

## 5.1 离线处理

系统日志将以 MapReduce 的方式定期检查 UT 表中用户的点击记录。在这一步骤中，我们查看用户在一定时间内（通常是几个月）的点击情况。并且使用 4.1 和 4.2 中描述的 MinHash 和 PLSI 算法对用户进行聚类。计算出的聚类信息将会被存到 UT 表中作为该用户信息的一部分用于生成推荐内容。

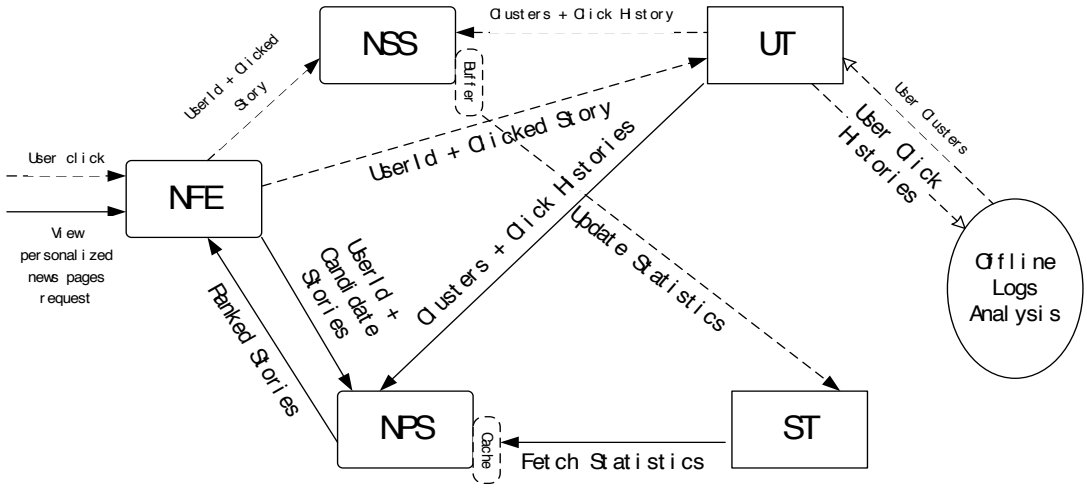


图 2 ：系统组件

## 5.2 数据表

用户表 UT 和数据表 ST 都是二维表，用于存储每个用户和每条新闻的各种统计量。用户表的行是由用户 Id 进行索引的，对于每个用户，两类数据将

被存进表中：（a）聚类信息：该用户所从属于的 MinHash 集群和 PLIS 集群的 Id 列表；（b）点击历史：用户点击的新闻 Id 列表。

这两类信息共同组成用于生成推荐内容的特定用户的信息。

新闻表由新闻 Id 进行索引的，每行对应一条新闻，对每条新闻（S），我们主要需要维护两类统计量：（a）聚类统计量：对于一个集群C（这里可以是 MinHash 或 PLSI 集群），新闻S被该集群C中所有用户点击的次数之和（PLSI 中按照$p(z|u)$进行加权）。（b）共同访问统计量：新闻S和新闻$S'$被共同访问的次数。这里的共同访问信息可表示 4.4 部分用于描述共同访问图的邻接表。

对于上述的每类统计量，我们需要维护一些归一化统计量：对于每个集群C，我们需要维护该集群中所有用户的点击之和；对于每条新闻S，我们需要维护S的共同访问对之和（只要S是共同访问对中的两条新闻之一即可，无论访问顺序）。

为了加强那些被用户表以感兴趣的最新新闻的作用，并削弱老新闻可能依靠其年龄来获得高点击率的事实，对于上面所说的各种计数并不是以简单地计数的方式进行维护。而是使用随时间衰变的计数方式，使得最近的点击拥有更多的权重。

为了使得该系统是个在线的推荐系统，数据表 UT 和 ST 需要提供快速的实时的数据响应机制，使得无论是按行号（如 UT 表中用户 Id），还是按行列对（如 ST 中按用户 Id 和集群 Id）对相应统计量进行更新和检索，都能在几毫米内完成。适合存储这些数据表的可选之一是使用 Bigtable [21]，它是一个分布式存储系统，可以在数以千计的商业服务器上存储大规模结构化数据。

## 5.3 实时服务

我们需要在线服务器来提供两项主要功能：每当用户点击一条新闻时，更新数据表中的各种统计量和信息；当有一个给定用户 Id 的用户发出推荐请求时，生成一个经过排序的新闻推荐列表。

图 2 给出一个可能的实现：其中，新闻统计服务（NSS）负责每当新闻前端服务器（NFE）发来一个用户点击时更新 ST 表中的数据。新闻个性化服务（NPS）负责每当新闻前端服务器（NFE）发来一个推荐请求时生成新闻推荐列表。NFE 充当一个连接个性化服务与用户的代理服务。

## 5.4 组装各个组件

上述的新闻推荐系统中的各种组件主要通过处理由 NFE 发出的两类用户请求作为响应用户动作的整个工作流来进行相互交流：一类请求用来推荐新闻（由 NFE 转发到一个 NPS）；另一类请求在用户点击一条新闻时产生，用来更新统计量（由 NFE 转发到一个 NSS）。下面将介绍处理每类请求的具体步骤：

### 5.4.1 推荐请求（图 2 中的实箭头）：

当一个用户请求个性化新闻推荐时，NFE 连接一个 NPS，传送用户 Id 和一组候选新闻以计算得分（请参照 4.5 部分生成候选新闻）。当接收到这样一个请求时，NPS 需要先从 UT 表获取用户信息（集群 Id 和最近点击历史），其次是该用户所从属的集群（包括 MinHash 和 PLSI 两类集群）对应的点击计数以及她的点击历史中所有新闻的共同访问计数。这些计数统计量从 ST 表中获取，为了提高效率，NPS 可以使用一个合适的过期窗口来本地缓存这些统计量。基于这些统计量，正如第四部分描述的那样，NPS 计算出三类推荐的分（基于 MinHash 和 PLSI 的集群-新闻的分和基于共同访问的新闻-新闻的分），并通过一个线性联合来获得每个候选新闻的最终得分。最后，NPS 将最终得分返回给 NFE。

### 5.4.2 更新统计量请求（图 2 中的虚箭头）：

当一个用户点击一条新闻时，这个信息就会被记录到 UT 表中她的点击历史中。NFE 也会用一个请求来连接一个 NSS 以更新所有由该点击引起的任何可能的统计量的变动。为了更新统计量，NSS 需要从 UT 表中获取用户信息（集群 Id 和点击历史）。对于用户从属的每个集群（包括 MinHash 和 PLSI），需要更新该集群中该新闻对应的点击计数，在 PLSI 集群中以$\mathrm{p}(z|u)$为权重。另外，我们需要更新该用户最近的点击历史中的每条新闻同最近一次点击对应的那条新闻的共同访问计数。这些计数同 ST 表中适当规格化的计数一起被 NSS 更新。当然，为了提高效率，NSS 可以先缓存这些更新，然后定期地将它们写回 ST 表。

将上述的两项功能分配到各自独立的服务器有以下优势：即使统计服务器

NSS 停止运行，系统无法更新统计量，个性化服务器 NPS 仍可以继续正常工作，甚至可以使用 ST 表中陈旧的统计量来生成推荐内容。这可以很好地减小服务器停机期间的推荐质量的降低。

由于点击记录和统计量需要被实时地更新和检索（如通过前面提到的 Bigtable 基础设施），上面描述的系统是一个在线的，为用户提供即时需求的系统。因此，一个用户的每次点击都会影响到分配给不同候选新闻的得分，并有可能改变用户实际看到的推荐新闻集合。系统的在线特性使得我们必须处理新闻的高度混合过程，从而确保我们可以向用户推荐有意义的新闻（即新闻取自那些在各大新闻网站出现不久的新闻）。为了减少那些偏激用户的点击行为产生的有可能影响其他用户的新闻推荐质量的不正常统计量。当我们在更新统计量或使用用户点击历史进行用户聚类时，那些具有反常点击率的用户点击可以被忽略。

# 说明

论文之后的两部分是系统性能评估和总结，由于前五部分的翻译已符合毕业设计的要求且内容足以提供毕业设计需要的相关内容，故后面内容未做翻译，详见原文 Google News Personalization: Scalable Online Collaborative Filtering.