Web 聚类技术及其在搜索引擎中的应用

计算机软件与理论

研究生 李战胜　指导教师 杜亚军

摘　要

搜索引擎是当前研究的热门技术之一，用户通过输入查询词来获取搜索结果，从而能够在海量的互联网资源中获取有用信息。然而，当前搜索引擎返回的搜索结果数目非常庞大，要从这么多的结果中找到有用信息，有时显得很困难。如何更好地显示搜索结果，更好地帮助用户找到自己感兴趣的信息，是本文所要研究和解决的问题。

幸运地是，利用聚类技术对搜索结果按主题聚类，将很好地表现搜索结果。显然，传统的聚类技术一般都是针对数值数据进行的；而对于文本数据，尤其是 Web 文档类型的数据，需要新的能够处理高维、实时的聚类算法，并在文本特征提取和时间复杂度方面有更高的要求。

本文提出的新的聚类算法 MyCluster 是基于短语和潜在语义索引基础上的针对搜索结果的模糊 Web 聚类算法。在 MyCluster 聚类算法中采用由多个有序词构成的短语来建立文档特征向量，而不是传统方法上用单个词来建立，这样可以有效地避免数据噪音的干扰，而且还可以明显地降低特征矩阵的维度，缩减计算时间。Web 文档特征矩阵的建立将涉及 Web 文档的下载和解析、关键短语和非关键短语的识别和中文分词等关键技术。

Web 聚类结果由类标签和类内容构成。每个类标签都对应着许多类内容，即搜索结果。每个类标签代表着一个主题，显然类标签的可读性，也即主题鲜明，将直接影响用户查找信息的命中率。当然，类内容是用户获取最终信息的入口，它与类标签的相关性就很重要。我们采用线性代数中奇异值分解方法来发现类内容和归纳类标签，使类内很相似，类间不相似，而且类内容很好地关联了类标签。对形成的类标签和类内容，采用合并和排序策略，将很好地修正聚类结果。

为了更好地测试 MyCluster 算法的聚类效果，本文给出了基于 MyCluster

算法的聚类搜索引擎框架及聚类结果评价体系。实现聚类搜索引擎的过程中将涉及多种编程语言(例如 HTML、Javascript、CGI 和 C++等)和数学软件 MatLab，并实现了 C++在脱离 MatLab 环境情况下对 MatLab 生成的动态链接库的直接调用。我们的聚类结果评价体系通过类标签的可读性、类内容的相关性、类内容覆盖率和类重叠度等指标，来综合评价一个算法质量的好坏。

通过对比实验，我们发现 MyCluster 在类标签可读性和类内容相关性方面有很大地优势，但在类内容覆盖率方面有所欠缺，希望未来将能有所改进，并在增量聚类等方面作进一步的研究。

关键词：聚类搜索引擎，Web 搜索结果聚类算法，奇异值分解，MyCluster

Abstract

Currently, the technology of search engine is a hot in IR research. Only, a user can input a query and get some search results, so that they can realize the dream of getting the useful information from Internet. But, the total num of search results is very large, and it is difficult for users to find the useful information in those results. How to organize the search result and how to find the useful information? This thesis will ask those questions.

Fortunately, clustering according to the themes of the search results will be well to help user to find the information they are interested. Apparently, the conventional technologies only deal with the numerical data rather than the textual data. Especially, for dealing with the Web textual data, we propose a new clustering algorithm.

The new clustering algorithm, named MyCluster, is a fuzzy Web clustering algorithm based on the phrase and Latent Semantic Indexing. It is important to form a eigen vector of documents and we try to form it by using phrases constituted by some sequenced words rather than one word. The algorithm will effectively avoid the disturb of data noise, decrease the dimension of eigen matrix and save the time for computing. For forming the eigen matrix, it will include the key techonologies such as downloading and parsing Web documents, identifying key phrases and general phrases and Chinese words divided syncopation technology etc.

The result of MyCluster is composed of class labels and class contents. Each class label correspondings to some class contents, namely search results. Clearly, the readability of cluster labels will directly effect the rate of finding a useful information. Certainly, the class contents is a entry for users getting the information and the relation between the class contents and the class labels is very important. We use a method of Singular Value Decomposition to induce class labels and find class contents, so that the clusters have the characteristic that objects belonging to the same cluster are "similar" to each other, while objects from two different clusters are "dissimilar". Lastly, we incorporate and sort the clusters.

For testing the quality of clustering, we propose a clustering search engine based on our algorithm of MyCluster and a system of evaluation of search results clustering. In the process of realizing the clustering search engine, we will deal with HTML, Javascript, CGI, C++ and MatLab etc. In our system of evaluation of search results clustering, we use the readability of class labels, the relevance of class contents, the coverage of class contents and the overlap of class to evaluate the quality of algorithm.

By experiments, our MyCluster has some advantages of the readability of class labels and the relevance of class contents. But it has a default of the coverage of class contents. We will remedy it in future and further study in the increment clustering.

Keywords: Clustering Search Engine, The Clustering Algorithm of Web Search Results, Singular Value Decomposition, MyCluster

第1章　绪论

就像 Google 的创始人所说的，“我们若能更妥善地搜寻资料，实在已经改变世界”。在互联网高速发展的今天，人们发现查找信息变得越来越困难了。

如何才能更有效地查找我们感兴趣的信息，这一问题已经成为众多学者的研究对象。在[Selberg, 99]中提到：这个问题即是从 Web 中找出与用户给定查询词相关的文档集。为解决这个难题，涌现了众多的搜索引擎，如 Google[Google, 05], Yahoo[Yahoo, 05], 百度[Baidu, 05], 中搜[Zhongsou, 05]等。然而这些搜索引擎的搜索结果却并不尽如人意：使用者输入查询词，一般都会得到成千上万的搜索结果，然而其中大部分页面都是不需要的无关资料。虽然有一些技巧试图给那些有较多关键词或者罕见关键词的页面赋予更大的权重，却仍然不能保证与用户意图最相关的页面一定被排在最前面。因此用户别无选择，只能把检索到的页面一个个再筛选一遍。显然，这浪费人的时间和精力。

[Cutting et al., 92]等提出的称为“Scatter/Gather”的技术，试图更合理的组织搜索结果。这种技术对检索到的结果页面进行聚类操作，按照页面彼此之间的相似程度分为若干组，每组都有一个比较明确的主题，用户可以迅速地扫描每一组并选择那些和他的目标最相关的组。这是一个很好的尝试和开端，它将很好地解决人们查找信息难的问题。此后，陆续出现了 Grouper[Grouper, 99], Carrot2[Carrot2, 03], Vivísimo[Vivísimo, 05]等聚类搜索引擎。

1.1　研究动态

搜索引擎（Search Engine）的鼻祖是 1990 年由 McGill University 学生 Alan Emtage、Peter Deutsch、Bill Wheelan 发明的 Archie。虽然当时 World Wide Web 还未出现，但网络中文件传输还是相当频繁的，由于大量的文件散布在各个分散的 FTP 主机中，查询起来非常不便。因此，Alan Emtage 等想到了开发一个可以用文件名查找文件的系统，于是便有了 Archie。Archie 是第一个自动索引互联网上匿名 FTP 网站文件的程序，但它还不是真正的搜索引擎。

1994 年 4 月，Stanford University 的两名博士生，美籍华人 Jerry Yang（杨致远）和 David Filo 共同创办了 Yahoo。随着访问量和收录链接数的增长，Yahoo 目录开始支持简单的数据库搜索。因为 Yahoo!的数据是手工输入的，所以不能

真正被归为搜索引擎，事实上只是一个可搜索的目录。Yahoo!中收录的网站都附有简介信息，所以搜索效率明显提高。

1994 年初，Washington 大学 CS 学生 Brian Pinkerton 开始了他的小项目 WebCrawler[WebCrawler, 05]。1994 年 4 月 20 日，WebCrawler 正式亮相时仅包含来自 6000 个服务器的内容。WebCrawler 是互联网上第一个支持搜索文件全部文字的全文搜索引擎，在它之前，用户只能通过 URL 和摘要搜索，摘要一般来自人工评论或程序自动取正文的前 100 个字。WebCrawler 后来发展成为元搜索引擎。

DEC 的 AltaVista[AltaVista, 05]是一个迟到者，1995 年 12 月才登场亮相。但是，大量的创新功能使它迅速到达当时搜索引擎的顶峰。Altavista 最突出的优势是它的速度。AltaVista 是第一个支持自然语言搜索的搜索引擎，也是第一个实现高级搜索语法的搜索引擎（如 AND, OR, NOT 等语法)。

1998 年 9 月 27 日 Google 的发布，再一次改写了搜索引擎的发展史。它在 Pagerank[Page, Brin et al., 98]、动态摘要、网页快照、DailyRefresh、多文档格式支持、地图股票词典寻人等集成搜索、多语言支持、用户界面等功能上的革新，象 Altavista 一样，再一次永远改变了搜索引擎的定义。

给定一查询词来查找与之相关的文档集，需要考虑三方面的因素：（1）Web 是一动态的文档集合，它每时每刻都在更新；（2）互联网上存在海量的文档，而搜索引擎仅返回与查询词相关的很少结果；（3）如何将最相关的结果排在最前面，以利于用户的查找。为解决这些问题， 现有的搜索引擎（如 Google[Brin, Page, 98]）都采用 Crawler(或 Spider, 或 Bot)等技术获取互联网上的文档，经过索引，以所谓的查询词-结果列表方法把最相关的结果排在最前面。

由于搜索引擎仅凭用户的查询词来返回搜索结果，很低的查全率和查准率会导致用户需要翻很多搜索结果页才能找到自己感兴趣的信息，甚至根本找不到。Yahoo!采用目录的方式把互联网上的文档按类别进行分类，用户可以按照类别导航来查找资料。但是 Web 文档更新很快，用人工方式编辑形成的分类目录显然不适合当前信息极度膨胀的互联网，而利用人工智能方法—聚类将会很好地解决这些问题。

聚类（Clustering）是一个将数据集划分为若干类（class）或簇（cluster）的过程，并使得同一个组内的数据对象具有较高的相似度；而不同组中的数据对象是不相似的。如果把它应用到搜索结果上，将很好地把搜索结果分到相应地

类中。比如：查询“计算机”，将得到“计算机病毒”、“计算机考试”、“计算机科研网”等类。对于普通用户，也很容易通过先选定类，再查找对应的文档，将大大提高查找信息的命中率。这也即聚类搜索引擎（Clustering Search Engine）所要达到的目的。

聚类搜索引擎一般是元搜索引擎（Meta Search Engine ）与 Web 聚类技术的整合，通过元搜索引擎获取来自不同搜索引擎的搜索结果，然后对搜索结果（一般是文档的一个片段即 Snippet，而不是整篇文章）进行聚类，形成类标签（class label）和类内容（class content），每个类标签中均包含对应的类内容。

Scatter/Gather[Hearst and Pedersen, 96]较早对搜索结果进行聚类的系统，它是基于 *Buckshot* 和 *Fractionation* 两种聚类算法基础上，通过对标题（Title）的评价，把相似的文档聚合在一起，以区别不相关的文档。

Agglomerative Hierarchical Clustering (AHC)是基于 *K*-means，Single-pass 等基于距离的聚类算法，但精度很低，类标签很难表示。

Grouper[Grouper, 99]通过 HuskySearch 元搜索引擎获取搜索结果并利用基于短语（phrase）的后缀树聚类（Suffix Tree Clustering）算法对其动态聚类成带有类标签的簇集。它的特点是基于共享短语（shared phrases）而不是孤立的词（word），允许类重叠，是一种模糊聚类方法，可以增量聚类，时间为线性。

Carrot2[Carrot2, 03]和 Semantic Hierarchical Online Clustering [SHOC, 04]两者都用到了后缀数组（Suffix Array）来发现关键短语（key phrases），不同的是前者使用 SVD 算法来得到类标签，再利用向量空间模型（Vector Space Model ）形成簇集，而后者利用 SVD 来得到类标签和形成簇集。

Vivísimo[Vivísimo, 05], [Perez et al., 00]基于的原理是一种叫做准确描述所有配对（concise all pairs profiling）（简称为 CAPP）的方法。这种方法着眼于形成可描述的聚类。它的基本原理是将所有的类别成对的进行比较，找出能够将每一对类别区分开来的特征，然后对那些特征进行组织，形成最后的描述，保证每一对至少有一个特征能够将它和其他对区别出来。Vivísimo 自动聚类所依据的是搜索引擎返回的网址、标题和简单描述，而不是整个网页。

在导师的精心指导下，我们智能搜索引擎研究小组取得了较好的成绩。从搜索引擎的构架研究到个人 Web 搜索服务，再到聚类搜索引擎，我们积累了丰富的经验，并完成了智能搜索引擎和个性化平台的开发。在原有的个人 Web 搜索服务研究的基础上，我们把聚类技术应用到对搜索结果的处理上，是对搜索

结果表现方式的一次变革；提出了一个基于短语和潜在语义检索的 Web 搜索结果聚类算法 MyCluster，并在此基础上构造了聚类搜索引擎和评价体系。

1.2 研究的目的和范围

本文研究的主要目的是：提出一个高效、可靠的搜索结果聚类算法。研究工作主要包括：

1. 信息检索（Information Retrieval）的研究，及线性代数在 Web 挖掘中的应用；
2. 设计搜索结果聚类算法；
3. 设计执行其算法的应用框架；
4. 设计算法评价体系。

本文涉及的问题属于 Web 挖掘领域，是信息检索的子领域。Web 挖掘主要分为（1）Web 用法挖掘（Web Usage Mining）：用来自动发现用户获取 Web 服务的模式；（2）Web 结构挖掘（Web Structure Mining）：用来分析 Web 的超链结构；（3）Web 内容挖掘（Web Content Mining）：研究如何从互联网上获取信息的问题。本文属于第 3 种 Web 挖掘。

1.3 论文结构

本文剩下的章节将组织如下：

第二章：简单介绍 Web 搜索（即搜索引擎）及各种类型的搜索引擎，其中将涉及系统架构和搜索结果表现。

第三章：简单介绍 Web 聚类技术，包括中文分词技术和针对搜索结果的多种聚类算法。

第四章：详细介绍本文将提出的搜索结果聚类算法 MyCluster 所必备的概念和理论，包括关键短语和非关键短语的识别、奇异值分解方法等。

第五章：提出我们的搜索结果聚类算法 MyCluster。利用关键短语和非关键短语来构造特征矩阵，奇异值分解方法来发现类内容和归纳类标签，类合并和类排序完成聚类的最后过程；并给出执行 MyCluster 算法的系统框架。

第六章：提出我们的搜索结果聚类算法评价体系，并进行对比实验和分析。

第七章：总结。

第2章 Web 搜索

本章将重点介绍传统搜索引擎和其他各类搜索引擎，并分析它的结构包括爬行虫、文档索引、文档排序、文档缓存、查询处理和结果显示界面等；同时着重介绍了各种搜索结果表现方法，并进行对比分析；最后就如何获取搜索结果进行简要地阐述。

2.1 Web 搜索服务

Web 搜索服务即 Web 搜索引擎，目的是帮助用户更好地查找互联网上的信息和资源。现在的 Web 搜索引擎能搜索的文本类型非常广泛，包括 HTML 网页、PDF、PS、MS Word 和 MS PowerPoint 文档等，甚至还能搜索多媒体文件。用户使用 Web 搜索，只需登录搜索引擎站点（例如：Google：http://www.google.com），在搜索框中输入要查询的字符串，即查询词，很快就会返回搜索引擎提供的带有超链（hyperlink）的文档结果列表；搜索引擎往往把跟查询词最相关的结果排在最前面，它们很可能就包含用户想要的信息。带有超链的文档结果并不是源文，而是源文中的一小片段，这主要是基于检索速度方面的考虑。现代的搜索引擎不再局限于普通文档的搜索，范围扩大到对很多资源包括 FTP 资源，网上购物商场，地图，多媒体资源等，其代表有天网蚂蚁 FTP 文件搜索[KeepSo FTP, 05]，Google Groups[Google Groups, 05]，Froogle[Froogle, 05]等。现代搜索引擎可以分为传统搜索引擎、元搜索引擎、智能搜索引擎（包括个性化搜索引擎、聚类搜索引擎等）等三大类。

2.1.1 传统搜索引擎

真正意义上的搜索引擎，通常指的是利用爬行虫收集互联网上几千万到几十亿个网页并对网页中的文字（即关键词）进行索引，建立索引数据库的全文搜索引擎。当用户查找某个关键词的时候，所有在页面内容中包含了该关键词的网页都将作为搜索结果。在经过复杂的算法排序后，这些结果将按照与搜索关键词的相关度高低，依次排列。现在的搜索引擎已普遍使用超链分析技术，除了分析索引网页本身的文字，还分析索引所有指向该网页的链接的 URL、

AnchorText、甚至链接周围的文字。传统搜索引擎基本上都具备一些共同的组件包括：爬行虫（Crawler, Spider, Robot），文档索引，文档排序，文档缓存，查询处理，结果显示界面等。传统搜索引擎的一般构架如图 2.1 所示。

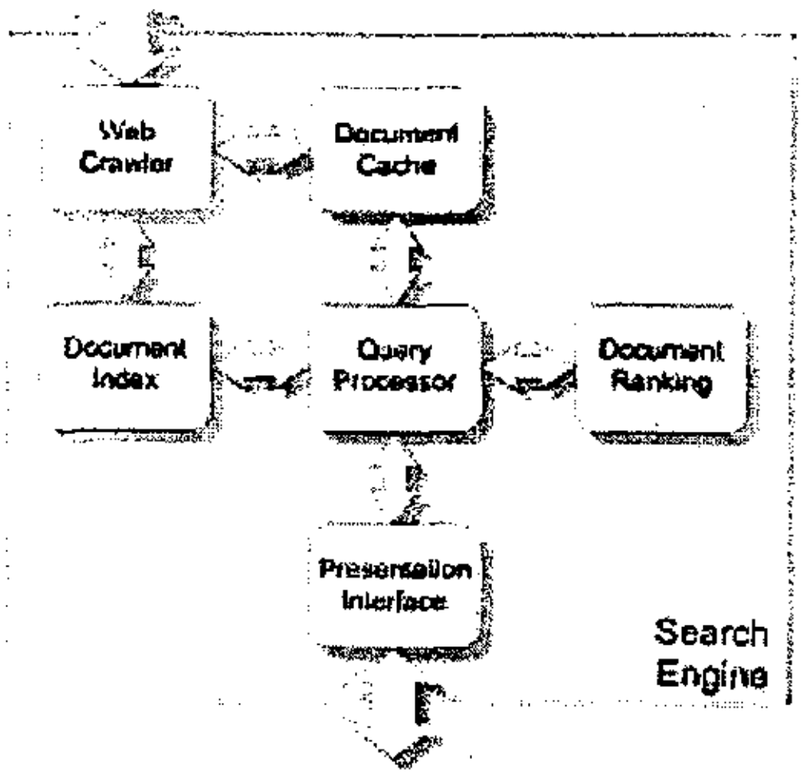

图 2.1：传统搜索引擎构架

2.1.1.1 爬行虫（Web Crawler）

爬行虫是能够从互联网上自动下载网页的计算机程序，一般采用并行下载方式。它与普通下载方式不同的是它能提取爬行过的网页中的 URL，并把它作为下一个自动爬行的目标网页，这样重复爬行，直到人为地终止程序或设置出口。爬行虫程序很复杂，涉及到网页排重、死链处理、动态网页下载、网页跳转等难题，还必须遵守 Robot Exclusion Protocol。

2005 年 4 月 2 日显示 Google 抓取互联网上的文档总数为 8,058,044,651 张，即便有这么多的文档也只占互联网的一小部分。Google 爬行虫的爬行周期从原来的一个月缩短为现在的一周，使用户得到的搜索结果有很高的新鲜度，提高了信息的查全率和查准率。

一般爬行虫程序的伪代码如下：

```
Crawler ( URLSet urlSet, DocSet docSet )
{
    While (URLSet 非空 )
```

```
{
    从 URLSet 中取一 URL;
    下载此 URL 对应的文档 Doc;
    从下载的文档 Doc 中解析出 URLs 保存到 NewURLSet 中;
    将下载的 Doc 添加到 docSet 中;
    把 URL 保存到 IndexedURLSet 中;
    For (NewURLSet 中的每个 URL)
    {
        URL 通过一算法映射为一固定长度的字符串;
        If (IndexedURLSet 中不存在此字符串)
            将 URL 加入到 URLSet 中;
    }
  }
}
```

爬行虫程序一般采用宽度优先和深度优先算法，也有其他一些智能爬行算法。

2.1.1.2 文档索引（Document Index）

文档索引就是对爬行虫下载的网页（或文档）建立索引结构，以提高检索速度。由于下载的网页数量很大，及网页本身包含的内容很丰富，要对网页中的每个词建立索引，数据量很大，也是不可能的。所以其中将涉及停用词处理、英文词根处理、中文切词分词、建立倒排表[Yates et al., 99]和数据压缩[Yates et al., 99]等，数据一般达到几百 G，通常以文件的形式存放在多台服务器上，而不是数据库。

2.1.1.3 文档评价（Document Ranking）

文档评价是对所有下载的网页进行重要性评价，其中涉及的关键因素较多，有词频、链接数、是否首页等。比较流行的评价算法有 PageRank[Page, Brin et al., 98]和 HITS[Chakrabarti, 98]，其中 PageRank 算法已经获得了很大的成功。

1．PageRank 算法：

假设：网页 A 存在网页 $T_1,...,T_n$ 指向它（即网页 A 的外链 citations），参数 d 为一衰减因子介于 0 到 1 之间。$C(A)$就指网页 A 的外链数。网页 A 的 PageRank 值计算如下：

$$PR(A)=(1-d)+d(PR(T_1)/C(T_1)+...+PR(T_n)/C(T_n))；$$

PageRank 值是用户浏览此网页的一个概率分布，它们总和等于 1。

2．HITS 算法：

HITS（Hyperlink－Induced Topic Search）算法是利用 Hub/Authority 方法的评价算法，其算法描述如下：将查询 q 提交给传统的基于关键字匹配的搜索引擎．从搜索引擎返回的网页中取前 n 个网页作为根集(root set)，用 S 表示。S 满足如下 3 个条件：

1）S 中网页数量相对较小；

2）S 中网页大多数是与查询 q 相关的网页；

3）S 中网页包含较多的权威网页。

通过向 S 中加入被 S 引用的网页和引用 S 的网页将 S 扩展成一个更大的集合 T.

以 T 中的 Hub 网页为顶点集 V_1，以权威网页为顶点集 V_2，V_1中的网页到 V_2 中的网页的超链接为边集 E，形成一个二分有向图 $SG=(V_1, V_2, E)$。对 V_1 中的任一个顶点 v，用 $h(v)$表示网页 v 的 *Hub* 值，对 V_2 中的顶点 u，用 $a(u)$表示网页的 *Authority* 值。开始时 $h(v)=a(u)=1$，对 u 执行 I 操作修改它的 $a(u)$，对 v 执行 O 操作修改它的 $h(v)$，然后规范化 $a(u)$，$h(v)$，如此不断的重复计算下面的操作 I，O，直到 $a(u)$，$h(v)$收敛。

I 操作：
$$a(u)=\sum_{v:(v,u)\in E} h(v) \qquad (1)$$

O 操作：
$$h(u)=\sum_{u:(v,u)\in E} a(v) \qquad (2)$$

每次迭代后需要对 $a(u)$，$h(v)$进行规范化处理：

$$a(u)=a(u)/\sqrt{\sum_{q\in V_2}[a(q)]^2} \qquad h(v)=h(v)/\sqrt{\sum_{q\in V_1}[h(q)]^2}$$

式(1)反映了若一个网页由很多好的 *Hub* 指向，则其权威值会相应增加(即权

威值增加为所有指向它的网页的现有 *Hub* 值之和)。式(2)反映了若一个网页指向许多好的权威页，则 *Hub* 值也会相应增加(即 *Hub* 值增加为该网页链接的所有网页的权威值之和)。

2.1.1.4 文档缓存（Document Cache）

文档缓存是指源文的存储，供检索时实时提取网页片段和做网页快照使用。

2.1.1.5 查询处理（Query Processor）

查询处理就是执行用户的查询需求，与文档索引、文档缓存和文档排序进行通信，把最终的结果返回给用户。

2.1.1.6 结果显示界面（Presentation Interface）

结果显示界面一般用来显示搜索结果总数、搜索时间和搜索结果等。传统搜索引擎大都采用列表形式来显示结果，用户通过翻页来完成查找。这也是与聚类搜索引擎在搜索结果显示方式上的最大区别。

2.1.2 元搜索引擎（Meta Search Engine）

元搜索引擎是一种调用其它独立搜索引擎的引擎,亦称“搜索引擎之母（The mother of searce engines)”。在这里，“元”（Meta）为“总的”、“超越”之意，元搜索引擎就是对多个独立搜索引擎的整合、调用、控制和优化利用。相对元搜索引擎，可被利用的独立搜索引擎称为“源搜索引擎”（source Engine），或“搜索资源”（searcing resources），整合、调用、控制和优化利用源搜索引擎的技术，称为“元搜索技术”（Meta-searching technique），元搜索技术是元搜索引擎的核心。

元搜索引擎分为并行处理式和串行处理式两大类。并行处理式元搜索引擎将用户的查询请求同时转送给它调用链接的多个独立型搜索引擎进行查询处理，串行处理式元搜索引擎将用户的查询请求依次转送给它调用链接的每一个独立型搜索引擎进行查询处理。

元搜索引擎是用户同时利用多引擎进行网络搜索的中介。检索时，元搜索引擎根据用户提交的检索请求，调用源搜索引擎进行搜索，对搜索结果进行汇集、筛选、删并等优化处理后，以统一的格式在同一界面集中显示。元搜索引擎虽没有网页搜寻机制，亦无独立的索引数据库，但在检索请求提交、检索接口代理和搜索结果显示等方面，均有自己研发的特色元搜索技术支持。对搜索结果的显示，不同的元搜索引擎有不同的处理技术，由于元搜索引擎设定的搜索结果排序依据、最大返回结果数量、相关度参数及优化机制等不同，调用相同的源搜索引擎的不同元搜索引擎显示搜索结果的数量多少、排序先后、结果信息描述选择亦有较大差异。

1995 年华盛顿大学硕士生 Eric Selberg 和 Oren Etzioni 推出第一个元搜索引擎—Metacrawler 以来，这一新型的网络检索工具异军突起，发展迅速，目前可用的元搜索引擎已近百种。尽管元搜索引擎存在着功能上的局限，但其以含盖较多的搜索资源，能够在尽可能短的时间内提供相对全面、准确的搜索结果等诸多优异功能受到用户的青睐，已渐成为一种不可或缺的极具潜力的网络检索工具。现在较流行的元搜索引擎有 Dogpile [Dogpile, 05]，Metacrawler [Metacrawler, 05]（见图 2.2）和 Mamma [Mamma, 05]。

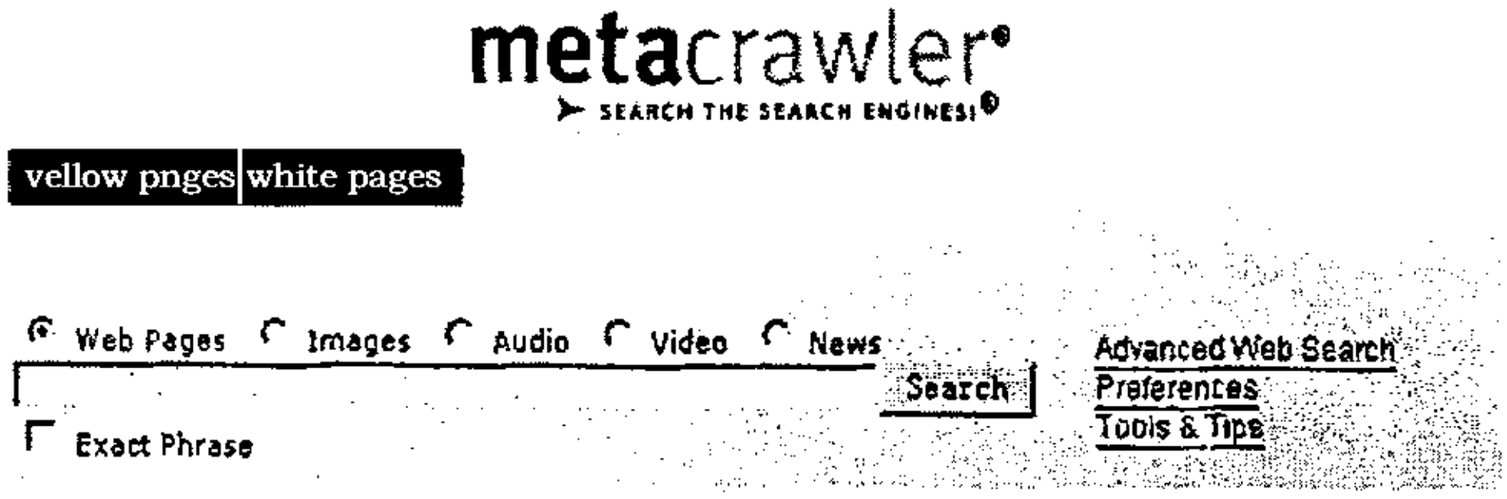

图 2.2：MetaCrawler 元搜索引擎

2.1.3 个性化搜索引擎（Personalized Search Engine）

个性化搜索引擎是接受用户输入的查询词，来查找用户的感兴趣信息。此系统在与用户的频繁交互中记下用户的个人兴趣爱好，并建立用户兴趣模型；当用户使用时，向用户提供某一方面的或全部的感兴趣信息。比如：一喜欢“猎豹”这种动物的用户搜索“猎豹”时，将得到有关“猎豹”这一生物的信息，而不是猎豹汽车或其它。

个性化搜索引擎系统必须满足以下三个要求：

1) 个性化的

个性化搜索引擎系统必须是为特定的用户服务的。

2) 高适应性的

由于用户的兴趣是在改变的，所以，系统必须能察觉用户的兴趣改变了；系统必须根据这些改变采取相应的措施。

3) 易扩展的

个性化搜索引擎系统应该能探测新的领域，以此来发现用户潜在的兴趣。

总之，个性化搜索引擎系统应该能及时地提供与用户的需求相匹配的信息，并在用户兴趣改变时作出相应的变化。所以，系统不但要了解用户当时的需求，还应能探索不同的领域来发现用户新的兴趣爱好。其代表有 Google Personalized[Google Personalized, 05], Zhongsou IPS[Zhongsou IPS, 05]（见图 2.3）。

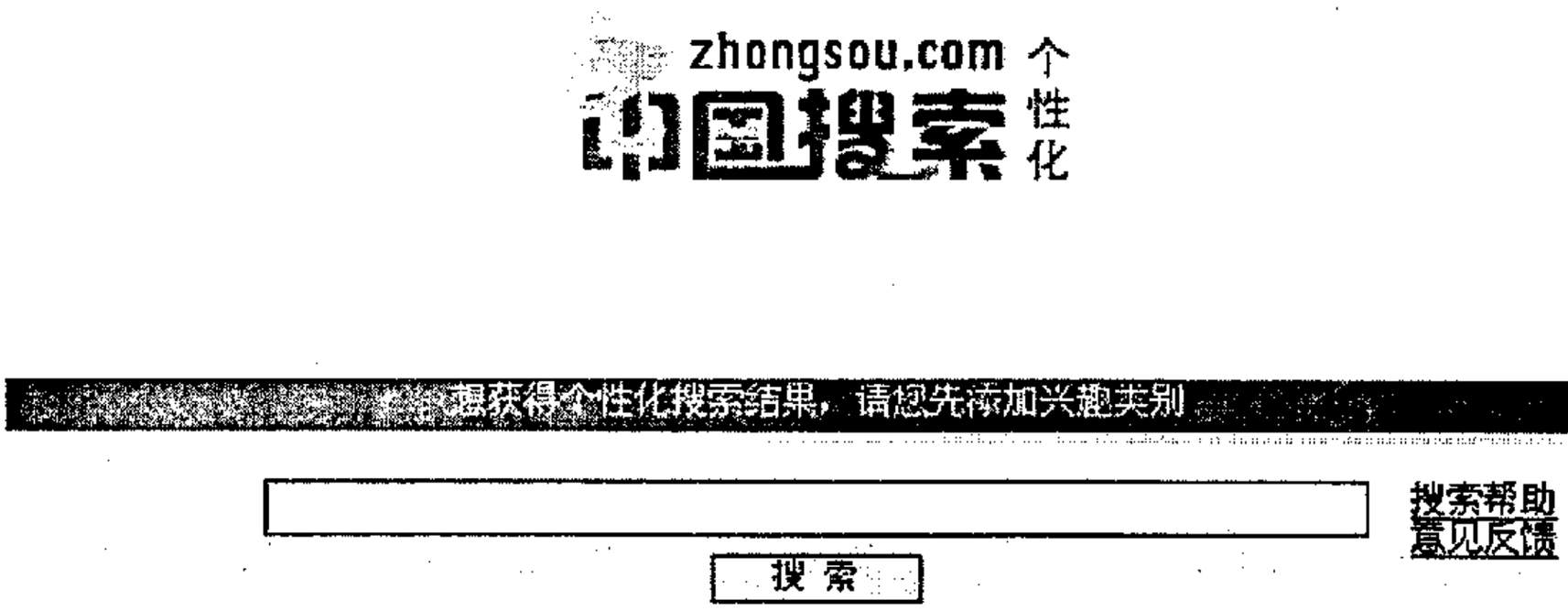

图 2.3：中搜个性化搜索引擎

2.1.4 聚类搜索引擎（Clustering Search Engine）

聚类搜索引擎是元搜索引擎与 Web 聚类技术的整合，它通过元搜索引擎获取来自不同传统搜索引擎的搜索结果，然后对搜索结果（一般是文档的一些片段，而不是整篇文章）进行聚类，形成簇集，每个簇中均包含对应的文档集。

聚类搜索引擎与传统搜索引擎的最大区别就在于：搜索结果表现形式的不同。传统搜索引擎的搜索结果是一按相关性排序的结果列表，而聚类搜索引擎则是将搜索结果进行再聚类，形成类标签和对应的类内容的过程。使用聚类搜索引擎时，用户首先也是输入查询词，然后选择感兴趣的类标签，得到相应的类内容（即搜索结果）。实质上它起到“信息导航”的作用，也是对搜索结果表现方式的一次变革，将大大提高用户浏览信息的命中率和效率。其代表有 Vivísimo[Vivísimo, 05]（见图 2.4），Carrot2[Carrot2, 05]，Mooter[Mooter, 05]等。

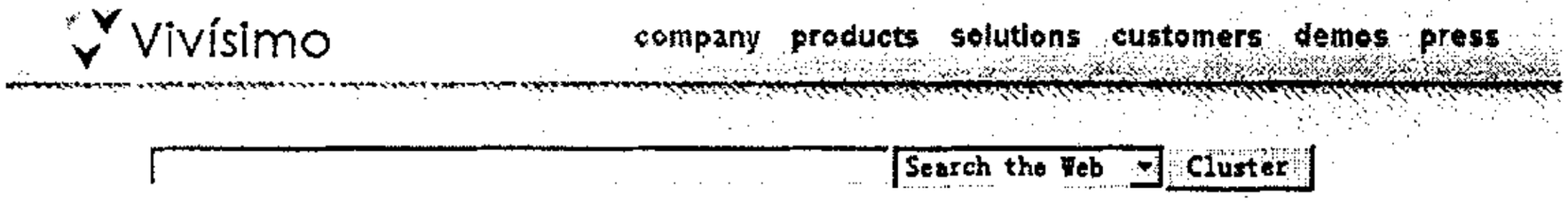

图 2.4：Vivísimo 聚类搜索引擎

2.2 搜索结果表现

搜索结果是反映一个搜索引擎质量好坏的重要指标，也是用户获取自己感兴趣信息的入口。返回与查询词最相关的搜索结果一直是所有搜索引擎提供商追求的目标，在搜索结果一定的情况下，如何更好地组织和表现搜索结果，将直接影响用户查找信息的效率。搜索结果表现形式目前主要有：（1）评价列表（2）人工目录（3）聚类。

2.2.1 评价列表（Ranked List）

评价列表（见图 2.5）是目前搜索引擎采用的比较常见的方式，它按文档与查询词的相关程度来排序，把最相关的搜索结果显示在前面。评价列表中的每个结果通常由标题（title）、URL 和来自文档的片段等构成。用户通过浏览评价列表来确定信息所在的位置，然后点击位置上的 URL 打开对应的文档，即完成用户的一次查找过程。但它也存在着很多不足之处[Grouper, 99] [Weiss, 01]：

1）用户为了找到所需的文档，往往得浏览很长的结果评价列表，甚至通过翻页才能找到；

2）用户无法知道查询词与文档的相关程度，也无法知道文档间的相关性；

3）结果没有经过分类，用户很难快速找到需要的文档。

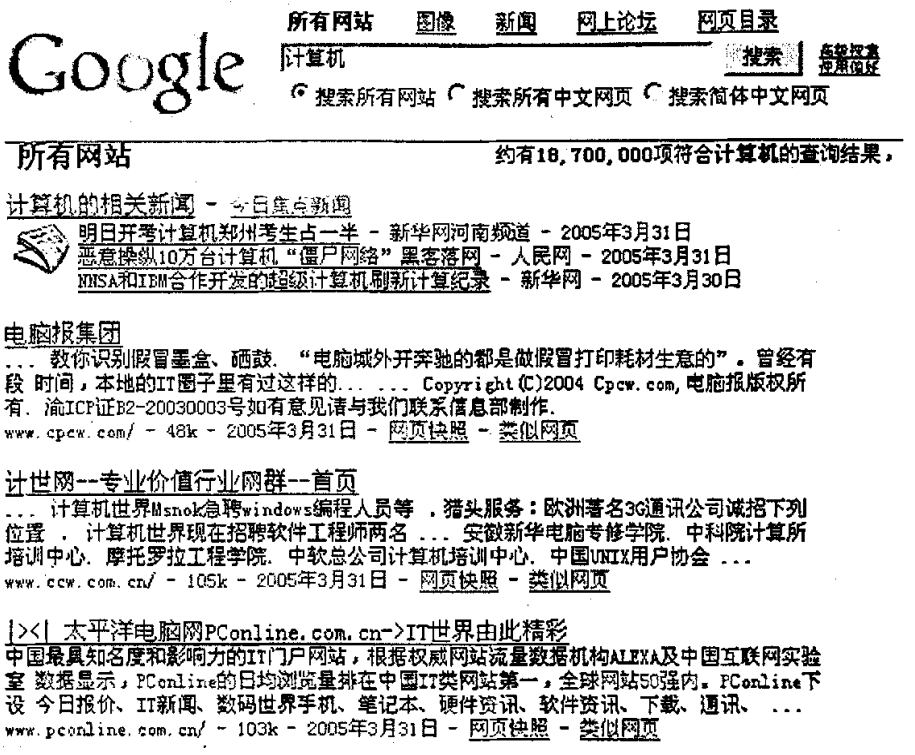

图 2.5：Google 搜索结果列表

2.2.2 人工目录（Human-made Web Directories）

人工目录方式类似与图书馆书目编排，它是完全基于人工分类和编辑，将互联网上的文档按类别存放和显示。由于人工劳动的局限性，人工目录方式的搜索引擎往往只收录网站和很少的网页。很显然，它的更新速度很慢，而且文档数量有限。但是，文档与类别的相关性比较好，用户查询的准确率比较高。目前比较流行的人工目录有 Yahoo[Yahoo, 05]，LookSmart[LookSmart,05]和 Open Directory Project[ODP, 05]（见图 2.6）。

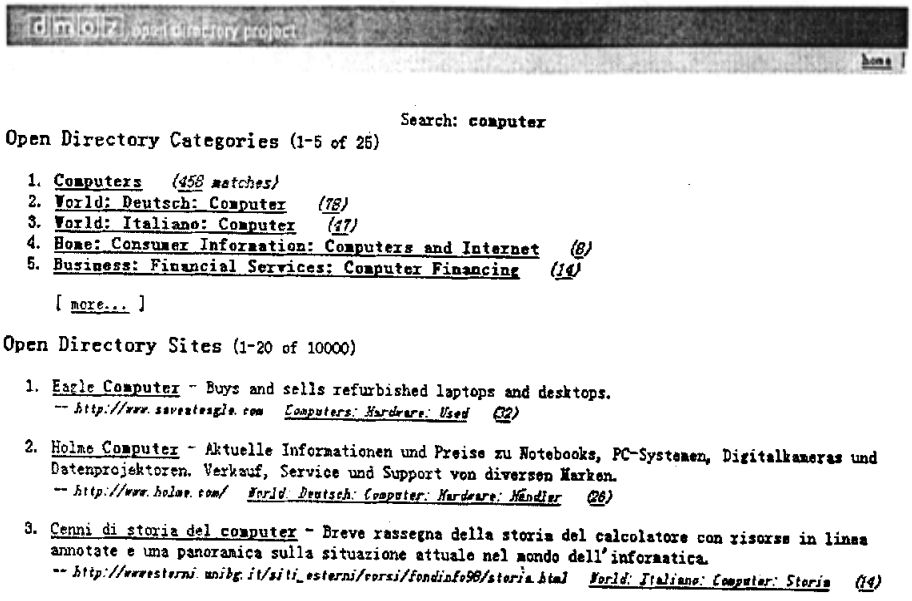

图 2.6：ODP 人工目录

2.2.3 Web 搜索结果聚类（Web Search Results Clustering）

Web 搜索结果聚类（见图 2.7）就是将搜索引擎返回的结果运用 Web 聚类技术自动归类，最后按类标签（即类名）和类内容（即类包含的文档）的形式呈现给用户。它与人工目录形式的本质区别是：前者运用的是聚类技术，而后者是分类技术，且是人工分类。聚类结果一般在结果页面左侧显示类标签和其包含的类内容数目；当用户点击其中一个类标签时，右侧将显示此类标签对应的类内容。

它具有以下特点：

1）动态地将文档归到具体的类中；

2）每个类标签无需人工标出；

3）仅仅对 title 和 snippet 进行聚类，而不是整篇文档。

Web 搜索结果聚类也被称为后-检索文档聚类算法，它应该满足以下条件[Zamir and Etzioni, 98]：

A．相关性（Relevance）

算法产生的类应该是类内相似，类间不相似。

B．类标签（Class Label）

用户能够通过快速浏览类标签就能找到感兴趣的类。类标签应该具有可读性。

C．类重叠（Overlapping Class）

由于一些文档具有多个主题，采用模糊聚类方法避免它们只分到单个类中。

D．速度

对于在线聚类，速度是非常重要的，线性时间复杂度是最理想的。

E．增量处理

为了节省时间，算法要能够在搜索结果到达时就进行处理，进行增量聚类。

现有的较成熟的系统也即聚类搜索引擎，主要有 Vivisimo 和 Carrot2 等。

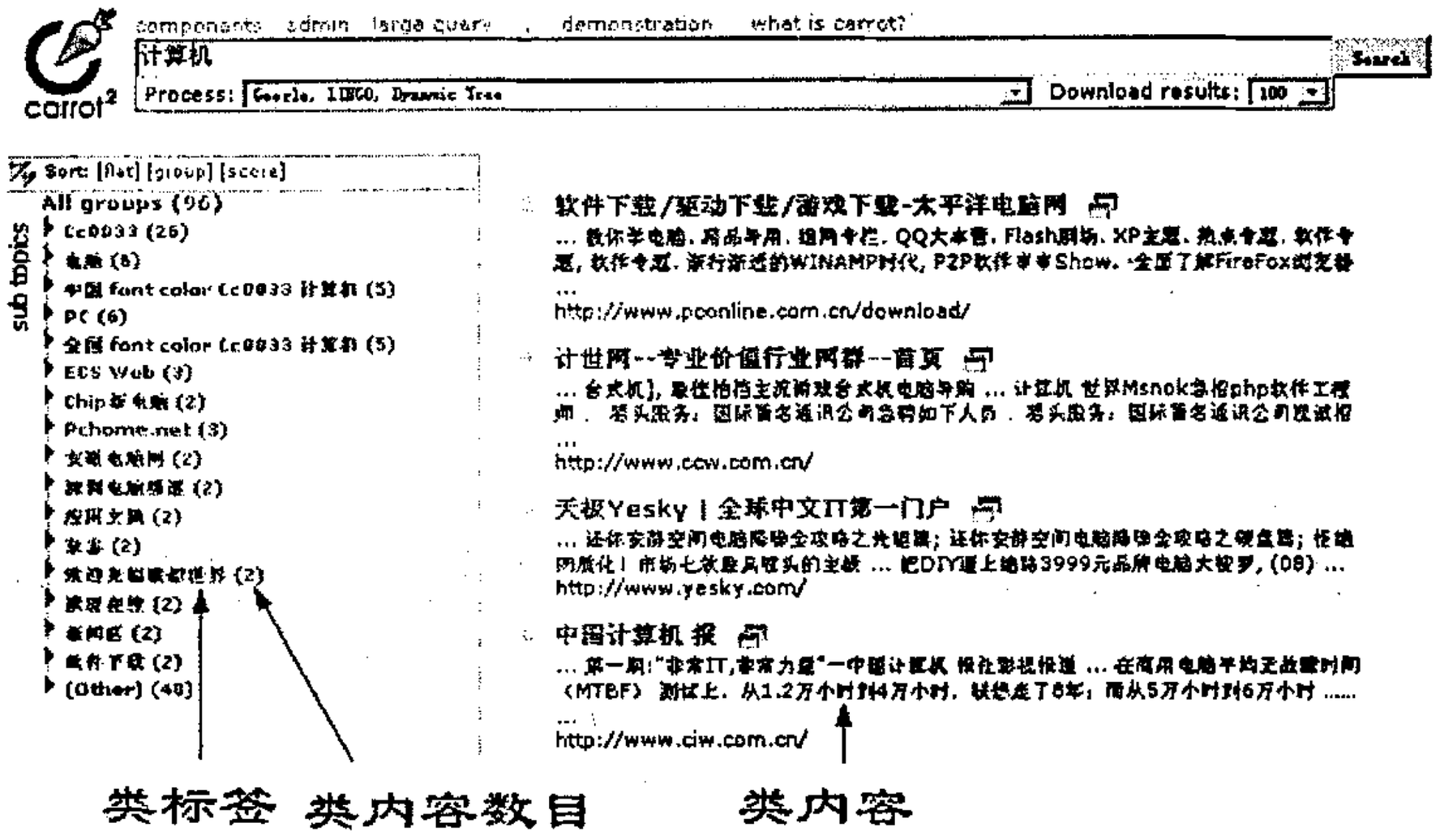

图 2.7：Carrot2 搜索结果

2.3 搜索结果获取

搜索结果的获取方法包括：（1）利用搜索引擎提供商提供 Web Services 接口（2）利用 HTTP 协议和 Socket 协议来下载 html 源代码，并解析。

在众多搜索引擎提供商中，目前只有 Google 提供了 Web Services 接口，而且是免费的。我们可以利用它提供的 APIs[Google APIs, 05]来直接获得搜索结果，包括 snippet、title、URL 等。

显然也可以通过 HTTP 协议和 Socket 协议来下载 HTML 源代码，然后通过解析 HTML 源代码，来得到搜索结果中的 URL，标题和片段等。显然它很依赖于搜索结果的 HTML 源代码，具有不稳定性，而且速度比直接用 Web Services 接口慢。

第3章　Web 文本聚类算法

本章将介绍聚类算法理论基础和数值、文本类型的聚类算法；并着重介绍其中将涉及的中文分词、Web 文本聚类等关键技术；同时详细介绍了 *k-means*、STC 等较流行的 Web 文档聚类算法。

3.1　中文分词技术（Chinese Words Divided Syncopation Technology）

在中文信息处理中，分词算法扮演着非常重要的角色。目前中文分词的准确率一般在 96%以上。

3.1.1　中文电子词表

由于中文信息没有特定的词组边界，因此从特定文本里提取有效的关键词就较英文资料困难许多。假如简单地以单个汉字作为信息处理的基本单元，既缺乏必要的语义表达，又会带来大量的冗余信息。

中文电子词表的研究是中文信息处理中基础性的工作，它在中文文本的自动分词、文本检索等诸多领域都有重要的应用价值。国内自 80 年代中后期就开展了中文电子词表的研制，现有的词表有采用 B^+ 树（或其变种）作为词表索引数据结构，也有利用现成的关系数据技术的，还有些系统由于词汇量不大，也有利用纯文本的，之所以采用 B^+ 树（或其变种）作为索引的数据结构，是因为早期计算机的主存容量的不足和操作系统存储管理的缺陷，采用这种结构可以有效地降低 I/O 次数，提供词表的访问性能，但随着硬件技术的不断提高，采用 B^+ 树（或其变种）的优势已基本丧失；另外，目前主流的 DBMS 大多少采用 B^+ 树（或其变种）作为文件组织形式，利用关系数据库技术构建的词表在访问时也会涉及到大量的 I/O 操作，势必影响效率；采用纯文本方式构建词表，由于数据没有经过有效的组织，内部查找时的计算复杂度为 $O(n)$(n 为词表中的词条数)，这种访问效率必须大大改进。

由于社会的发展变化，中文词的数目在不断的增加，中文词是一个开放的集。不同字开头的词数目变化很大，多的可达数百个，少的可能只有一个或者没有；词长度的变化也很大，有单字成词的，也有六、七字成词的，这就要求

在设计词表的数据结构时，除了考虑访问效率外，还必须充分考虑存储利用率。

我们采用的 Hash 方法实质上是一一映射，首字不同，地址不同，避免了模式冲突，利用 CC_i 可经过公式的运算而直接得到首字索引项 I_i，这一过程不进行任何匹配，找到索引项后，如待查项为单字，还要看 CC_i 是否能够独立成词，如果不能够独立成词，则根据待切分项的第 2 个字开始进行词条的匹配。同时，由于查找过程是标准的二分查找，若记 $\bar{n}$ 为所有汉字作首字时的平均词数，假设每个词被查询的机会均等，则查找一个词的计算复杂度为 $O(\log \bar{n})$。

3.1.2 分词匹配算法

目前采用较多的机械分词方法——最大匹配法在增字过程中，每增一字都有许多重复匹配的操作，大大降低了分词效率。利用词表中同一首字下的词条按升序排列这一条件，提出了近邻匹配算法。

考虑当前待切分的中文字串 $S=CC_0CC_1...CC_{L-1}$ (L 为汉字个数)，根据 CC_0 可以计算出相应的索引项 Ii 的地址，从而可得到以 CC_0 为词首字的词数 n 及指向所有词条：$W_{i,0}W_{i,1}...W_{i,n-1}$ 的指针 P_i，目标是找到长度最长且完全匹配的词条。在实际匹配过程中，现在词表中查询子串 CC_0CC_1，得到索引 $index$（如果 CC_0CC_1 不在词表中，则取最接近 CC_0CC_1 且排在其前的索引号），然后在 $index$ 之后寻找最长且完全匹配的词条。终止这一寻找过程的条件有两个：①当前匹配长度小于最大匹配长度；②词表中的词条比字串大。然后用同样方法切分下一个词条。

3.1.3 本分词软件的特点

本系统中采用分词软件分为初始化模块，分词模块，词转 ID 号模块。在分词模块中它把待分词的文档分词后保存到一个结构体中，其结构体如下所示：

```
typedef struct
{
    long wordID;              //词 ID 号
    char part[4];             //词性
    float weight;             //词的概率
}Word;
```

```
typedef struct
{
    long wordNUM;          //词个数
    Word *word;            //词
}Vector;                   //词向量
```

本分词软件的特色是：

1）有一个保持原始文章中词顺序的结构体，它使得后面提到的“关键子句的获取”成为可能，其结构体如下：

```
typedef struct{
    long wordID;           //词 ID 号
    char part[4];          //词性
    long tokenSequence;    //词序号
}Word2;
typedef struct
{
    long wordNUM;          //词个数
    Word2 *word;           //词
}Vector2;
```

2）在 ID 号转词模块中可以将一特定的 ID 号（数字）转换成对应的词，在字典中词和 ID 号是一一对应的。用 ID 号来代替字符串（词），即节省空间有加快检索速度。

3）分词结果中的每个词都标出词性，共有名词、动词等 40 种。

3.2 Web 聚类算法

本节将介绍聚类算法基础理论、常见聚类算法和 Web 聚类算法。通过了解聚类基础理论，认识数值聚类算法和文本算法的本质区别，以及 Web 文本聚类算法中应该注意的因素。

3.2.1 聚类基础理论

聚类（Clustering）是一个将数据集划分为若干簇（Cluster）或类（Class）

的过程，并使得同一个类内的数据对象具有较高的相似度；而不同类中的数据对象是不相似的。相似或不相似的描述是基于数据描述属性的取值来确定的。通常就是利用（各对象间）距离来进行表示的。许多领域，包括数据挖掘、统计学和机器学习都有聚类研究和应用。

聚类过程通常分为三步：

第一步是特征抽取。它的输入是原始样本，由领域专家决定使用哪些特征来深刻地刻画样本的本质性质和结构。特征抽取的结果是输出一个矩阵，每一行是一个样本，每一列是一个特征指标变量。

选取特征的优劣将直接影响以后的分析和决策。如果第一步就选择了和聚类意图根本无关的特征变量，企图得到良好的聚类结果则无异于缘木求鱼。因为无论后续步骤采用多么优良的聚类算法和阈值选择方案，都不可能计算出执行者的意图。合理的特征选取方案应当使得同类样本在特征空间中相距较近，异类样本则相距较远。

在有些应用中还需要将得到的样本矩阵进行一些后处理工作。比如为了统一量纲就对变量进行标准化处理，这样采用不同量纲的变量才具有可比性；在有些场合可能选择的特征变量太多，不利于以后的分析和决策，这时可以先进行一下降维处理；仅凭经验和领域知识选择的特征变量有可能是相关的，进行主成分分析就可以消除变量间的相关性，从而得到一些相互独立的特征变量。

第二步是执行聚类算法，获得聚类谱系图。聚类的输入是一个样本矩阵，它把一个样本想象成特征变量空间中的一个点。聚类算法的目的就是获得能够反映 N 维空间中这些样本点之间的最本质的“抱团”性质。这一步没有领域专家的参与，它除了几何知识外不考虑任何的领域知识，不考虑特征变量在其领域中的特定含义，仅仅认为它是特征空间中一维而已。聚类算法的输出一般是一个聚类谱系图，由粗到细地反映了所有的分类情况；或者直接给出具体的分类方案，包括总共分成几类，每类具体包含那些样本点等等。

第三步是选取合适的分类阈值。在得到了聚类谱系图之后，领域专家凭借经验和领域知识，根据具体的应用场合，决定阈值的选取。选定阈值之后，就能够从聚类谱系图上直接看出分类方案。没有领域专家的参与，不考虑具体的应用背景，而仅仅依赖于从聚类谱系图出发寻找聚类指数突变点，或者求最小生成树的长边等等，往往不会得到满意的结果。

3.2.2 传统聚类算法

作为统计学的一个分支，传统聚类技术已有多年的研究历史。聚类方法大致可分为：基于距离的聚类、基于密度的聚类和基于网格的聚类等。其中基于距离的聚类方法有 *k-means*、*k-medoids*、CURE 和 BIRCH 等；基于密度的聚类方法有 STING、DENCLUE、DBCLASD、DBSCAN 和 OPTICS 等；基于网格的聚类方法有 ROCK、CHAMELEON、ARHP、BGG+、PDDP 和 CLIQUE 等。当然，还有一些基于模型的方法和基于模糊理论或粗糙集理论的聚类方法等。

3.2.2.1 基于划分方法

给定一个包含 n 个对象或数据行，划分方法将数据集划分为 k 个子集（划分）。其中每个子集均代表一个聚类（ $k \leqslant n$ ）。也就是说将数据分为 k 组，这些组满足以下要求：（a）每组至少应包含一个对象；且（b）每个对象必须只能属于某一组。需要注意的是后一个要求在一些模糊划分方法中可以放宽。

给定需要划分的个数 k，一个划分方法创建一个初始划分；然后利用循环再定位技术，即通过移动不同划分（组）中的对象来改变划分内容。一个好的划分衡量标准通常就是同一个组中的对象“相近”或彼此相关；而不同组中的对象“较远”或彼此不同。当然还有许多其它判断划分质量的衡量标准。

为获得基于划分聚类分析的全局最优结果就需要穷举所有可能的对象划分。为此大多数应用采用一至二种常用启发方法：（a）*k-means* 算法，该算法中的每一个聚类均用相应聚类中对象的均值来表示；和（b）*k-medoids* 算法，该算法中的每一个聚类均用相应聚类中离聚类中心最近的对象来表示。这些启发聚类方法在分析中小规模数据集以发现圆形或球状聚类时工作的很好。但为了使划分算法能够分析处理大规模数据集或复杂数据类型，就需要对其进行扩展。

3.2.2.1.1 *k-means* 算法

类比于数值计算中的迭代法，*k-means* 算法首先给出一个粗糙的初步分类，然后按照某种原则动态修改聚类结果，直到得到合理的分类结果。*k-means* 算法一般需要人为给定类数 k，或者一些阈值。动态分类法执行步骤如下：

STEP 1：选择 k 个初始凝聚点，作为类中心的估计

STEP 2：对每一个样本，按照某种原则划归某个类中

STEP 3：重新计算各类的重心

STEP 4：跳转到 *STEP 2* 直到各类重心稳定。

初始凝聚点是一批有代表性的点，是希望形成类的中心，可以采用以下几种方法来选择：

1．前 k 元法：直接取前 k 个样本作为初始凝聚点。

2．经验法：凭经验确定样本点集合分为几类，对每一类指定一个代表点。

3．随机法：首先确定将样本集合分为几类，然后将每个样本点随机地分配给每一类，计算每一类的均值作为初始凝聚点；或者直接随机指定几个样本点作为初始凝聚点。

4．密度法：先人为给定两个数 d_1，$d_2(d_2> d_1)$，对每一个样本点统计和它相距不超过 d_1 的样本点数目，并称为它的密度。将样本点按照密度由大到小排列，首先将密度最大的样本点加入到凝聚点集中，然后考虑密度次大的样本点，如果该点到现有的任一凝聚点的距离都大于 d_2，则将该样本点加入到凝聚点集中，否则放弃该点。如此按照密度序进行下去，就会得到一个凝聚点集合。

得到初始类中心之后，一般采用最近邻法将其他样本划归不同的类。对于每个样本点，计算它和每个凝聚点之间的距离，并且将它归入和它最近的凝聚点所有的类。

k-means 算法的目标是将样本集合分为 k 类，每一类都聚集在离凝聚点周围的一个小区域，而且类与类间是可以比较明显地区分。*k-means* 算法运算速度快，内存开销小，比较适合于大样本量的情况，但是聚类结果受初始凝聚点的影响很大，不同的初始点选择会导致截然不同的结果；并且当按最近邻归类时如果遇到两个凝聚点距离相等的情况，不同的选择也会造成不同的结果。因此 *k-means* 算法具有很大的不确定性，并且初始类数的确定需要领域专家参与。

k-means 算法是一种迭代算法，而迭代算法的一个重要性质就是是否收敛。MacQueen 等人使用随机过程理论证明了 *k-means* 算法是收敛的，后来 Diday 给出了一个初等的证明。

但是 *k-means* 算法只适用于聚类均值有意义的情况。因此在某些应用中，诸如：数据集包含符号属性时，直接应用就有困难了。*k-means* 算法一个缺点就是用户还必须事先指定聚类个数 k，还不适合用于发现非凸形状的聚类，或具有

各种不同大小的聚类。此外还对噪声和异常数据也很敏感，因为这类数据可能会影响到各聚类的均值（计算结果）。

本系统中实现的 *k-means* 算法采用前 *k* 元法来选择初始凝聚点。从聚类效果上可看出，类标签很难表示，聚出的类太少，质量很差，而且抗干扰性弱。

3.2.2.2 基于层次方法

层次方法就是通过分解所给定的数据对象集来创建一个层次。根据层次分解形成的方式，可以将层次方法分为自下而上和自上而下两种类型。自下而上的层次方法从每个对象均为一个（单独的）组开始；逐步将这些（对象）组进行合并，直到组合并在层次顶端或满足终止条件为止。自上而下层次方法从所有均属于一个组开始；每一次循环将其（组）分解为更小的组；直到每个对象构成一组或满足终止条件为止。

层次聚类方法尽管简单，但经常会遇到如何选择合并或分解点的问题。这种决策非常关键，因为在对一组对象进行合并或分解之后，聚类进程将在此基础上继续进行合并或分解，这样就既无法回到先前的（聚类）状态；也不能进行聚类间的对象交换。因此如果所做出的合并或分解决策（在某一点上）不合适，就会导致聚类结果质量较差。此外由于在作出合并或分解决策前需要对许多对象或聚类进行分析评估，因此使得该类方法的可扩展性也较差。将循环再定位与层次方法结合起来使用常常是有效的，即首先通过利用自下而上层次方法；然后再利用循环再定位技术对结果进行调整。一些具有可扩展性的聚类算法，如：BIRCH 和 CURE，就是基于这种组合方法设计的。

3.2.2.3 基于密度方法

大多数划分方法是基于对象间距离进行聚类的。这类方法仅能发现圆形或球状的聚类而在较难发现具有任何形状的聚类。而基于密度概念的聚类方法实际上就是不断增长所获得的聚类直到“邻近”（数据对象或点）密度超过一定阈值（如：一个聚类中的点数，或一个给定半径内必须包含至少的点数）为止。这种方法可以用于消除数据中的噪声（异常数据），以及帮助发现任意形状的聚类。

DBSCAN 就是一个典型的基于密度方法，该方法根据密度阈值不断增长聚类。OPTIICS 也是一个基于密度方法，该方法提供聚类增长顺序以便进行自动或交互式数据分析。

3.2.2.4 基于网格方法

基于网格方法将对象空间划分为有限数目的单元以形成网格结构。所有聚类操作均是在这一网格结构上进行的。这种方法主要优点就是处理时间由于与数据对象个数无关而仅与划分对象空间的网格数相关，从而显得相对较快。

STING 就是一个典型的基于网格的方法。CLIQUE 和 Wave-Cluster 是两个基于网格和基于密度的聚类方法。

3.2.2.5 基于模型方法

基于模型方法就是为每个聚类假设一个模型，然后再去发现符合相应模型的数据对象。一个基于模型的算法可以通过构造一个描述数据点空间分布的密度函数来确定具体聚类。它根据标准统计方法并考虑到“噪声”或异常数据，可以自动确定聚类个数；因而它可以产生很鲁棒的聚类方法。基于模型聚类方法主要有两种：统计方法和神经网络方法。

3.2.3 Web 聚类算法

聚类起源于统计学，主要应用在数值数据上。随着计算机科学和数据挖掘的发展，聚类数据逐渐扩展到文本、多媒体等其他数据上。显然数据类型不同，聚类算法也会很大差异。传统方法是采用一些好的数值数据聚类算法(如：HAC 和 *k-means*）来处理文本类型数据。这些算法都要求预先定义测量两物体相似度的方法。相似度也指两物体的“距离”，“距离”近的将被分到同一类中。“距离”的测量方法比较多，其中基于向量空间模型（Space Vector Model）的余弦法被普遍采用。显然，两文档的相似度用余弦法计算比较简单。近年也出现了区别于传统方法的一些新的发展方向，如采用后缀树的聚类算法 STC 等，但它们较适合非中文环境。

3.2.3.1 后缀树聚类（Suffix Tree Clustering）

后缀树聚类（STC）算法通过识别文档集中的共同短语，并以此为基础建立类。相比传统算法把一个文档看成词的集合，STC 则把文档看成词的有序集。显然，传统方法损失了一些有用的信息。STC 的优点是利用短语不仅可以用来发现类，还可以用来描述类。

3.2.3.1.1 短语（phrase）

短语就是指几个有序的词。在信息检索（IR）中，识别短语的方法通常有 *n-grams* 法。*n-grams* 法通过在文本上使用 *n* 个词大小的滑动窗口来获取短语。然而，实验表明当 *n*>2 时，*n-grams* 对提高结果的质量贡献不大。

而 STC 算法是用后缀树（Suffix Tree）来识别短语，实验表明短语越长，越能描述类的内容。短语（phrase）的定义如下：

一个短语在上下文中是单个词或多词的一个序列，但不穿越短语边界。短语边界是指在词之间插入了的标点符号或 HTML 标签。把两短语边界之间的词序列记为句子。显然，一个短语可以任意长。

禁止短语穿越短语边界的原因是:(1)短语边界通常标记着主题漂移(topical shift)；(2) 可以减少构造后缀树的时间。

3.2.3.1.2 后缀树（Suffix Tree）

后缀树是字符串匹配和查询的有效数据结构。后缀树在发现最长重复子串[Weiner, 73]、相似字符串匹配[Landau and Vishkin, 89]、字符串比较[Ehrenfeucht and Haussler, 88]和文本压缩[Rodeh et al., 81]等基本字符串问题上得到广泛的研究和应用。

后缀树（Suffix Tree）的定义如下：

一个含有 *m* 个词的字符串 *S* 的后缀树 *T* 是具有 *1..m* 个叶子的根有向树。每个内结点，除了根结点，至少有两个孩子且每条边都标上 *S* 的一个非空子串。一个结点不存在两条边有开始于同一词的边标签（edge label）。对于任何叶子 *i*，沿根到叶子 *i* 的边标签的串联就表示从位置 *i* 开始的 *S* 的后缀，记为 *S*[*i*..*m*]。

图 3.1 显示了字符串"I know you know I know"的后缀树结构。圆表示内结点，四方块表示叶子。建立一长度为 m 的字符串 S 的后缀树时间复杂度为 $O(m^2)$。

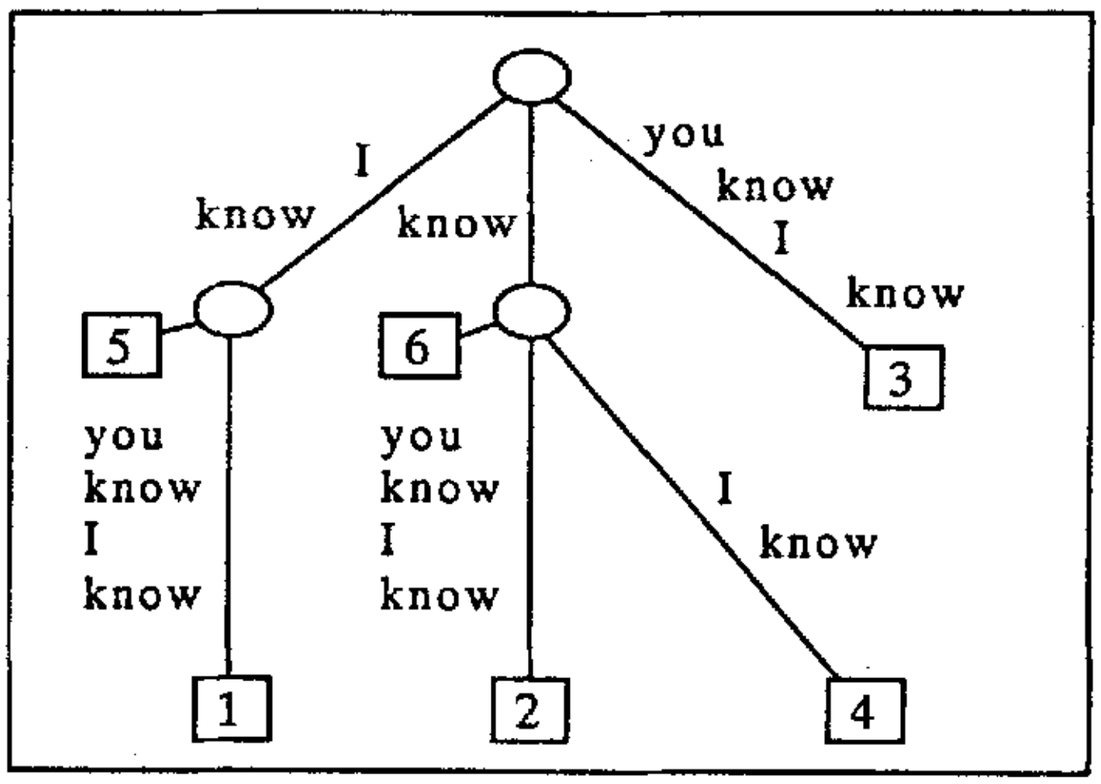

图 3.1：字符串"I know you know I know"的后缀树结构

3.2.3.1.3 后缀树聚类 STC

STC 算法是利用文档中的共同短语来计算文档间的相似度，从而形成短语簇（phrase cluster）。短语簇（phrase cluster）定义如下：

一个短语簇就是指一短语至少在两个文档中出现，而且文档集必须包含这个短语。一个最大短语簇（maximal phrase clusters）是一个短语簇，在不改变包含此短语的文档集的情况下，它的短语不能被任何词所扩展。

STC 是一种线性时间聚类算法，根据待聚类网页中的相似短语进行聚类。此算法可以分为三个步骤。

（1）网页“清洗”。这一步骤可以看作是网页特征的抽取。它对代表网页特征的字符串进行过滤，标明各句之间的间隔，去掉不是文字的标记符号（如 HTML 标记、大部分的标点）。

（2）确定短语簇。短语簇是一些具有共同短语的文档的集合。它是在对文档特征进行抽取的同时使用 STC 算法进行计算后得到的。对于每一个基本簇，根据它包含的文档特征的数量以及组成短语的词的个数赋予一定的权值。但是，在停用词表中出现的词或者过于高频词或者低频词对基本簇的权值没有贡献。

利用 STC 算法来识别所有的最大短语簇，利用后缀树识别短语簇的过程可

以看成是建立所有文档中短语的倒排。可以利用下面的公式对短语簇进行打分。

最大短语簇 m 包含短语 m_p 的分数 $s(m)$计算如下：

$s(m) = |m| * f(|m_p|) * \sum tfidf(w_i)$；

其中，$|m|$指短语簇 m 中文档数，w_i 指短语 m_p 中的词个数，$tfidf(w_i)$是对 m_p 中词的打分，$|m_p|$指短语 m_p 中非停用词的个数。

TFIDF（Term Frequency–Inverse Document Frequency）是信息检索中使用很普遍的技术用来评价单个词[Salton and Buckley, 88]。其基本原理：在文档中越频繁的词，在表达文档含义时越重要。类似，在所有文档中出现的越少的词，越能将文档区分开。STC 算法中 $tfidf(w_i,d)$的计算方法如下：

$tfidf(w_i,d) = (1+\log(tf(w_i,d))) * \log(1+N/df(w_i))$；

其中，$tf(w_i,d)$指词 w_i 在文档 d 中出现的次数，N 是整个文档集中的文档数，$df(w_i)$指有多少个文档中出现了词 w_i。

(3)合并短语簇为最后的结果。其主要的依据是短语簇的相似度。

两短语簇 m_i 和 m_j 的相似度 $sim(m_i,m_j)$计算如下：

如果$|m_i \cap m_j| / |m_i| > \alpha$与 $|m_i \cap m_j| / |m_j| > \alpha$，则 $sim(m_i,m_j) = 1$；否则，$sim(m_i,m_j) = 0$。

其中$|m_i|$和$|m_j|$分别表示两短语簇 m_i，m_j 包含的文档数，$|m_i \cap m_j|$表示具有共同短语簇的文档数目，一般α=0.6。

合并短语簇的方法是：合并相似度为 1 的两短语簇，直到不能合并为止。

STC 算法的主要特点有：（1）它是一种模糊聚类方法，允许交叉聚类。（2）使用短语而不是词去判断文档的相似性，同时也考虑这些短语出现的位置和顺序。它用共同短语来揭示聚类的内容，对用户而言这个也是一个有丰富信息量的摘要。（3）速度快，它是对元搜索引擎的结果进行聚类，在元搜索引擎返回结果的同时就开始工作，通常情况下在接收到最后一篇网页之后就可以显示出结果，不会产生明显的迟滞现象。

但 STC 算法不适合中文环境，当文档数变大时，将耗费大量时间。

第4章　MyCluster——概念预备

MyCluster 是本文提出的 Web 搜索结果聚类算法，其中涉及的概念将在本章作详细介绍。首先，是介绍我们提出的新概念—共现短语、关键短语和非关键短语及其构造算法；接着介绍比较流行的信息检索技术—向量空间模型（Vector Space Model）；最后介绍信息检索技术中的潜在语义分析（Latent Semantic Indexing）概念和奇异值分解（Latent Semantic Indexing）方法。

4.1　关键短语（Key Phrase）和非关键短语（General Phrase）

通过大量的实验分析，我们发现关键词周围的短语最能表达片段的主题；在多个文档中共同出现的短语也是文档之间相似的重要特征。基于以上的结论，我们提出了关键短语和非关键短语等新概念。关于短语、句子、短语边界等概念参考 3.2.3.1.1 中的介绍。

4.1.1　定义

我们把用户在查询框中输入的字符串，称为查询词（Query）。一般查询词 Q 由一个或多个关键词 K 构成。把搜索引擎返回的搜索结果中的一个结果记作一个文档 D，故所有的搜索结果也就称为文档集 DS。显然文档 D 由一条或多条句子 S 和短语边界 PB（定义参考 3.2.3.1.1）构成的，而句子 S 由一个或多个短语 P 构成，它们被短语边界 PB 分隔开。短语 P 可由一个或多个有序的词 W 构成。短语 P 的长度 PL 是指短语 P 中包含的词 W 的数目。若在一句子 S 中两短语 P_1 和 P_2 是相连的，我们称 P_1 为 P_2 的左邻居 LP，P_2 则为 P_1 的右邻居 RP。词 W 一般指文档经分词后的生成的最小单元。在中文环境，字 C 是构成词 W 的最小单元。而这里词 W 是整个文档 D 中的最小单元。

定义 1：共现短语（Co-Occurrence Phrase）是指在两短语 P_1 和 P_2 中共现的连续的词构成的短语 CP，且 P_1 中 CP 添加它的左邻居 LP 或右邻居 RP 组成的新短语在 P_2 中均不存在，反之亦然。

例如：短语“计算机软件及应用研究室”和短语“计算机应用研究室”的共现短语是“计算机”和“应用研究室”。

定义 2：关键文档（Key Document）是文档集 *DS* 中的一个文档 *D*，且 *D* 中必须包含查询词 *Q* 中的一个或多个关键词 *K*。

定义 3：非关键文档（General Document）是文档集 *DS* 中的一个文档 *D*，且 *D* 中不包含查询词 *Q* 中的任意关键词 K。

定义 4：关键句（Key Sentence）是关键文档 *KD* 中的一条句子 *S*，且 *S* 中必须包含查询词 *Q* 中的一个或多个关键词 *K*。

显然关键词代表了用户查询意图，若句子中出现了关键词，则表明此句子是关键的，比较重要的。

定义 5：非关键句（General Sentence）是非关键文档 *GD* 中的一条句子 *S*。

定义 6：关键短语（Key Phrase）是关键句 *KS* 中的一个短语 *P*，并且是在两条或多条关键句 *KS* 中的共现短语 *CP*，且这些 *KS* 必须不全属于同一关键文档 *KD*。

定义 7：非关键短语（General Phrase）是指非关键句 *GS* 中的一个短语 *P*，并且是在两条或多条非关键句 *GS* 中的共现短语 *CP*，且这些 *GS* 必须不全属于同一非关键文档 *GD*。

举例：用户输入查询词“计算机软件”，搜索引擎返回的 5 条搜索结果，见下表 4.1。

表 4.1：查询词为“计算机软件”对应的搜索结果

序号	片段
0	...计算机软件 综合站点...
1	...海环软件 物流软件 物流信息...
2	...计算机软件新技术国家重点实验室...
3	...上海市计算机软件评测重点实验室...
4	...湖南电子信息应用教育中心...

根据 4.1.1 中的定义得到的结果如下表 4.2 所示。（短语边界 PB 一般为标点符合，空格等。）

表 4.2：分析结果

名称	值
Q	“计算机软件”
K	{“计算机”，“软件”}
D_0	“...计算机软件 综合站点...”
D_1	“...海环软件 物流软件 物流信息...”
D_2	“...计算机软件新技术国家重点实验室...”
D_3	“...上海市计算机软件评测重点实验室...”
D_4	“...湖南电子信息应用教育中心...”
DS	{D_0，D_1，D_2，D_3，D_4}
S_{00}	“计算机软件”
S_{01}	“综合站点”
S_{10}	“海环软件”
S_{11}	“物流软件”
S_{12}	“物流信息”
S_{20}	“计算机软件新技术国家重点实验室”
S_{30}	“上海市计算机软件评测重点实验室”
S_{40}	“湖南电子信息应用教育中心”
CP_0	“计算机软件”
CP_1	“软件”
CP_2	“重点实验室”
CP_3	“信息”
KD	{D_0，D_1，D_2，D_3}
GD	{D_4}
KS	{S_{00}，S_{10}，S_{11}，S_{20}，S_{30}}
GS	{S_{40}}
KP	{CP_0，CP_1，CP_2}
GP	{}

4.1.2 共现短语（Co-Occurrence Phrase）构造算法

一般地，构造共现短语就是查找两个字符串所有的相对最长公共子串问题，可以利用 LCS—最大子串匹配算法来实现。

LCS 问题就是求两个字符串最长公共子串的问题。解法就是用一个矩阵来记录两个字符串中所有位置的两个字符之间的匹配情况，若是匹配则为 1；否则为 0。然后求出对角线最长的 1 序列，其对应的位置就是最长匹配子串的位置。

下表 4.3 是短语“计算机软件新技术国家重点实验室”和短语“上海市计算机软件评测重点实验室”的匹配矩阵。利用 LCS 算法，不难找到，对角线上连续为“1”的组成相对最长的匹配子串，即共现短语。这样就可以得到所有的共现短语为：“计算机软件”和“重点实验室”。

表 4.3：短语匹配矩阵

词	计算机	软件	新技术	国家	重点	实验室
上海市	0	0	0	0	0	0
计算机	1	0	0	0	0	0
软件	0	1	0	0	0	0
评测	0	0	0	0	0	0
重点	0	0	0	0	1	0
实验室	0	0	0	0	0	1

设两短语的长度分别是 m 和 n，则共现短语构造算法的时间复杂度为 $O(m*n)$，空间复杂度为 $O(m*n)$。

4.2 向量空间模型（Vector Space Model）

目前，在信息处理方面，文本的表示主要采用向量空间模型 (VSM)。向量空间模型的基本思想是以向量来表示文本：$(W_1, W_2, W_3 \ldots\ldots W_n)$，其中 W_i 为第 i 个特征项的权重，特征项可以选择字、词或词组，形成一个 *term-document* 矩阵。传统 VSM 一般选择词作为特征项。因此，要将文本表示为向量空间中的一个

向量，就首先要将文本分词，由这些词作为向量的维来表示文本，最初的向量表示完全是 0、1 形式，即如果文本中出现了该词，那么文本向量的该维为 1，否则为 0。这种方法无法体现这个词在文本中的作用程度。所以二值逐渐被更精确的词频代替，词频分为绝对词频和相对词频，绝对词频，即使用词在文本中出现的频率表示文本，相对词频为归一化的词频，其计算方法主要运用 TF-IDF 公式。

目前存在多种 TF-IDF 公式，如下所示：

$$W(t,\vec{d}) = \frac{tf(t,d)\times\log(N/n_t+0.01)}{\sqrt{\sum_{t\in\vec{d}}\left[tf(t,\vec{d})\times\log(N/n_t+0.01)\right]^2}}$$

其中，$W(t,\vec{d})$ 为词 t 在文本 $\vec{d}$ 中的权重，而 $tf(t,\vec{d})$ 为词 t 在文本 $\vec{d}$ 中的词频，N 为文本集中文本的总数，n_t 为文本集中出现 t 的文本数，分母为归一化因子。

另外还存在其他的 TF-IDF 公式，例如：

$$W(t,\vec{d}) = \frac{(1+\log_2 tf(t,\vec{d}))\times\log_2(N/n_t)}{\sqrt{\sum_{t\in\vec{d}}\left[(1+\log_2 tf(t,\vec{d}))\times\log_2(N/n_t)\right]^2}}$$

该公式中参数的含义与上式相同。

本系统 MyCluster 所采用的 VSM 与上面的模型不同的是选择关键短语和非关键短语作为特征项，形成一个 *phrase-document* 矩阵。特征项的权重不再是 TFIDF，而是短语中包含的所有词的 TFIDF 值之和。每个词的 TFIDF 计算公式如下：

$$W(t,\vec{d}) = tf(t,\vec{d})\times\log_{10}(N/df(t)\times\omega_t)$$

其中，$W(t,\vec{d})$ 为词 t 在文本 $\vec{d}$ 中的权重，而 $tf(t,\vec{d})$ 为词 t 在文本 $\vec{d}$ 中的词频，N 为文本集中文本的总数，$df(t)$ 为文本集中出现 t 的文本数，ω_t 为词 t 的长度，即包含字节数。

故，关键短语和非关键短语的特征项权重计算方法如下：

$$W(p,\vec{d}) = \sum_{i=1}^{n} W(t_i,\vec{d})$$

其中，$W(p,\vec{d})$ 指关键短语或非关键短语的特征项权重；n 为短语中包含的词数目。

举例：采用 4.1.1 节中的数据，文档集中的文档经分词（参考 3.1）后形成的词、词 ID 及其 TFIDF 值如表 4.4 所示，特征项的权重计算结果如表 4.5 所示，最终形成的 *phrase-document* 矩阵如表 4.6 所示。

表 4.4：文档分词结果

文档	词	词 ID	TFIDF	文档	词	词 ID	TFIDF
D_0	计算机	74017	1.000000	D_3	上海市	11432	1.477121
D_0	软件	27017	0.698970	D_3	计算机	74017	1.000000
D_0	综合	64190	1.301029	D_3	软件	27017	0.698970
D_0	站点	70208	1.301029	D_3	评测	50401	1.301030
D_1	海	35689	1.000000	D_3	重点	45260	1.000000
D_1	环	94563	1.000000	D_3	实验室	7922	1.176091
D_1	软件	27017	1.397940	D_4	湖南	72027	1.301030
D_1	物流	89876	2.602060	D_4	电子	18555	1.301030
D_1	信息	18746	1.000000	D_4	信息	18746	1.000000
D_2	计算机	74017	1.000000	D_4	应用	97086	1.301030
D_2	软件	27017	0.698970	D_4	教育	24970	1.301030
D_2	新技术	70696	1.477121	D_4	中心	83088	1.301030
D_2	国家	41654	1.301030				
D_2	重点	45260	1.000000				
D_2	实验室	7922	1.176091				

表 4.5：特征项权重

特征项代号	特征项	权重	文档
CP_0	计算机软件	1.698970	D_0
CP_0	计算机软件	1.698970	D_2
CP_0	计算机软件	1.698970	D_3
CP_1	软件	0.698970	D_0
CP_1	软件	1.397940	D_1
CP_1	软件	0.698970	D_2

CP_1	软件	0.698970	D_3
CP_2	重点实验室	2.176091	D_2
CP_2	重点实验室	2.176091	D_3

表 4.6：*phrase-document* 矩阵

文档	D_0	D_1	D_2	D_3	D_4
CP_0	1.698970	0	1.698970	1.698970	0
CP_1	0.698970	1.397940	0.698970	0.698970	0
CP_2	0	0	2.176091	2.176091	0

4.3 潜在语义索引（Latent Semantic Indexing）

LSI[Berry, 95][Deerwester, 90]是一种新的信息检索技术，通过统计手段，LSI可以把虽然不含查询字串但却相关的文档提取出来，经过转换后，相关的词汇会经由文件所包含的内容而产生关联，和“概念检索”有相同之处。使用 LSI 技术就意味着搜索引擎在检索网页时，试图把某些查询术语和其潜在概念联系起来。例如，把 iMac 和苹果公司的电脑联系起来。

作为一种 IR 向量空间技术，LSI 被证实比在 Salton 的 SMART 系统中使用的传统向量空间技术性能更好。其工作原理是利用矩阵理论中的“奇异值分解”技术，将词频矩阵转化为奇异矩阵：首先从全部的文档集中生成一个标引项—文档矩阵，该矩阵的每个分量为整数值，代表某个特定的标引项出现 在某个特定文档中次数。然后将该矩阵进行奇异值分解，较小的奇异值被剔除。结果奇异向量以及奇异值矩阵用于将文档向量和查询向量映射到一个子空间中，在该空间中，来自标引项—文档矩阵的语义关系被保留，同时标引项用法的变异被抑制。最后，可以通过标准化的内积计算来计算向量之间的夹角余弦相似度，再将文档按与查询的相似度降序排列。

按照潜在语义索引的思想，我们通过分析特征项（term）和文档（document）的联系来发现文档潜在的语义关系。将 term-document 矩阵看作一个双向图，来表示特征项和文档之间的联系。若许多特征项链接到同一文档，则说明这些特征项在语义空间上很接近；反之，若许多文档链接到同一特征项，则说明这些文档在语义空间上也是很接近的。也就是说，在语义空间上比较接近的特征项

和文档就应该被归到同一个类中。我们用奇异值分解(Singular Value Decomposition，SVD) [Golub, 89]方法来计算特征项和文档在语义空间上的接近程度。

向量空间模型中文本的相似性只是涉及两个文档本身的语法定义（在TFIDF加权的向量空间中，其他文档也可通过IDF对某文档发挥作用）。然而，间接的证据促使我们在两个没有共同特征项的文档中建立语义联系。

潜在语义索引就是用线性代数的概念把这种直觉形式化。对于*term-document*矩阵，其中的特征项i在文档j中不出现时$a_{ij}=0$，反之$a_{ij}=1$(当然也可以用词频或者TFIDF作为权重，a_{ij}为*term-document*矩阵中(i,j)的值)。设A为一$t \times d$矩阵，也即*term-document*矩阵，它的奇异值(即矩阵AA^T的特征值)是$\sigma_1, ..., \sigma_r$，且$|\sigma_1| \geq ... \geq |\sigma_r|$。根据奇异值分解法，将$A$分解为3个矩阵$U$，$\Sigma$，$V^T$，即$A=U\Sigma V^T$；$\Sigma=diag(\sigma_1,..., \sigma_r)$为对角矩阵，$U$和$V$是列正交矩阵，如图4.1所示。

如果LSI只保存最大的k(可调参数)个奇异值及U、V中对应的k列，使得$A_k=U_k \Sigma_k V_k^T \approx A$。$U_k$的每一行将一个特征项表示成为一个$k$维向量；同理$V_k$的每一行也将一个文档表示成一个$k$维向量，如图4.2所示。

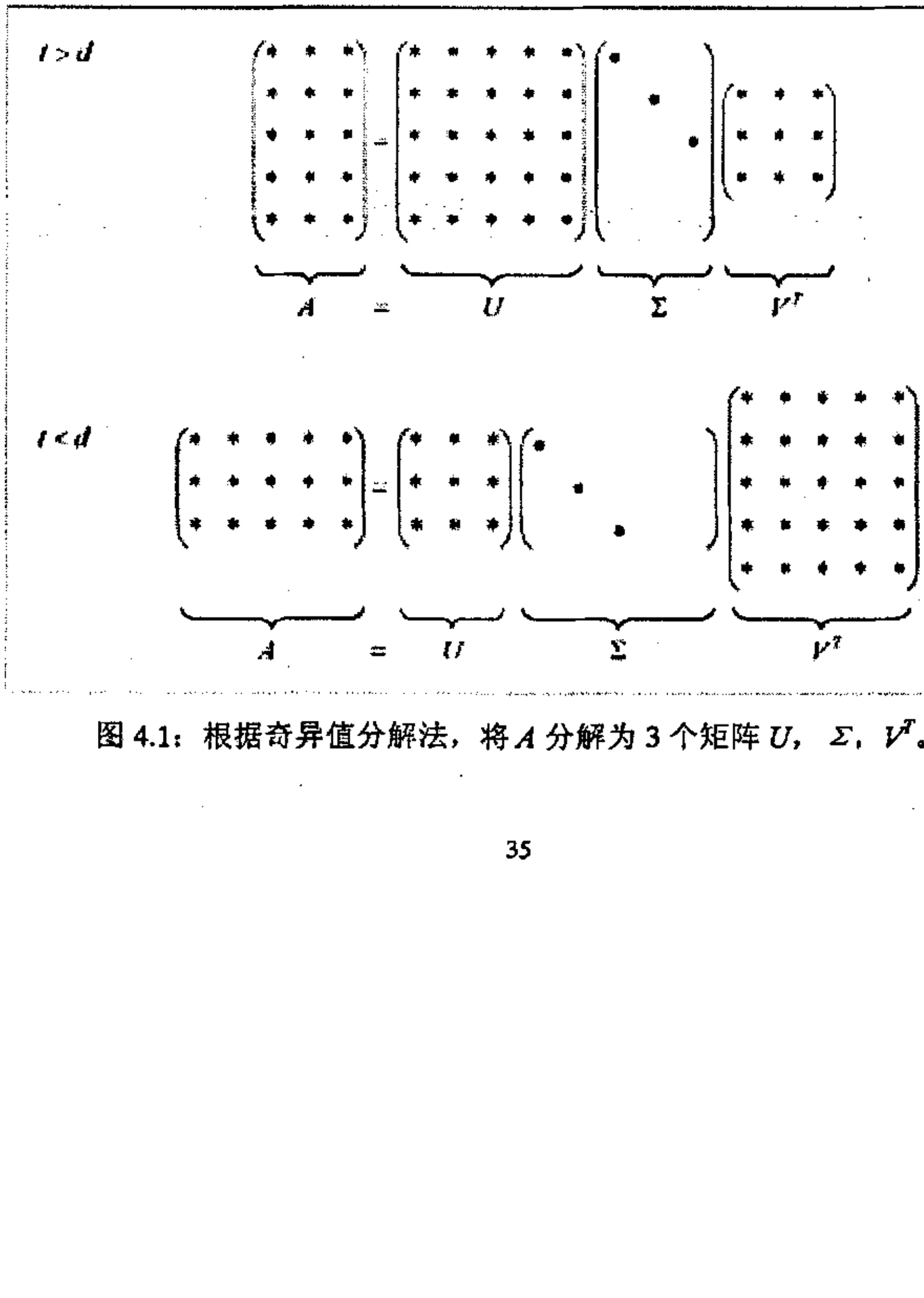

图4.1：根据奇异值分解法，将A分解为3个矩阵U，Σ，V^T。

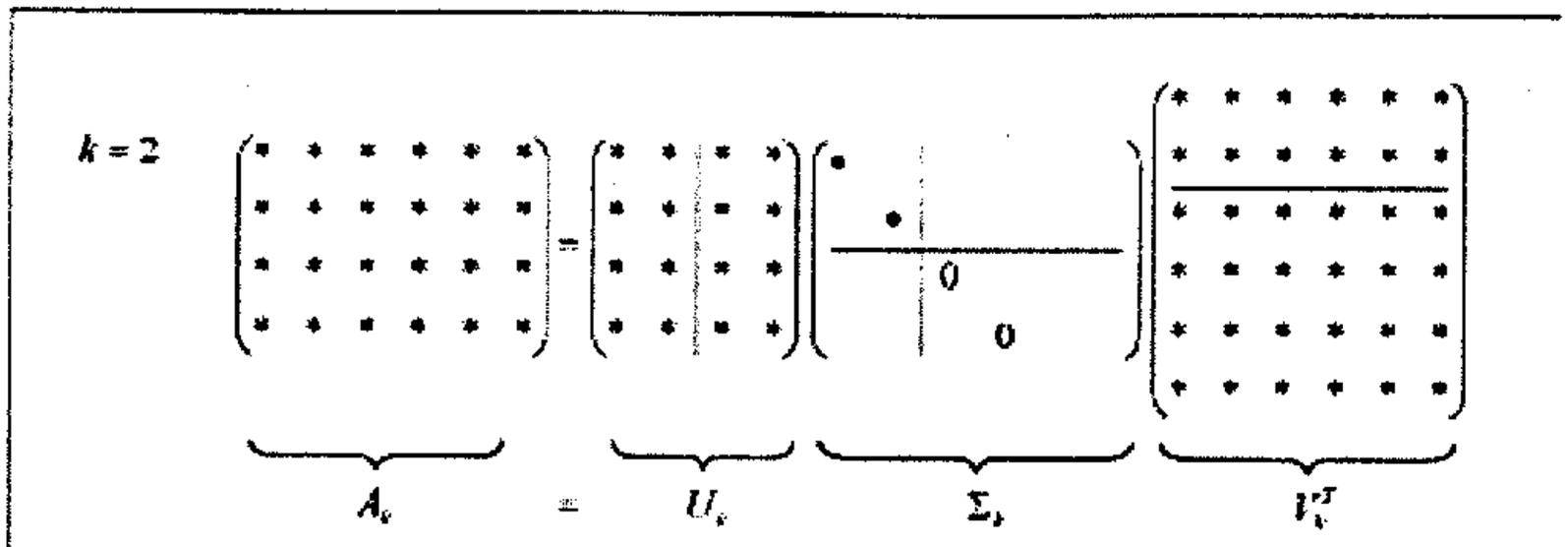

图 4.2：当 K=2 时，$A_2=U_2\,\Sigma_2 V_2^T \approx A$

举例：本系统中矩阵 A 采用的是 *phrase-document* 矩阵，而不是 *term-document* 矩阵。根据 4.2 节中的数据，将矩阵 A 分解为三个矩阵 U，Σ，V^T，如下图 4.3 所示。显然，有 $\sigma_1 = 4.2384$，$\sigma_2 = 1.5113$，$\sigma_3 = 1.1410, r = 3$（矩阵 A 的秩）。

A =

1.6990	0	1.6990	1.6990	0
0.6990	1.3979	0.6990	0.6990	0
0	0	2.1761	2.1761	0

U =

-0.6614	0.2653	0.7012
-0.3053	0.7583	-0.5759
-0.6851	-0.5950	-0.4202

Σ =

4.2385	0	0	0	0
0	1.5113	0	0	0
0	0	1.1410	0	0

V^T =

-0.3155	-0.1007	-0.6672	-0.6672	0
0.6501	0.7014	-0.2066	-0.2066	0
0.6913	-0.7056	-0.1102	-0.1102	0
0.0000	0	-0.7071	0.7071	0
0	0	0	0	1.0000

图 4.3：矩阵 A，U，Σ，V^T

第5章　MyCluster——算法

本章将详细介绍我们提出的 Web 搜索结果聚类算法—MyCluster。前面提到的概念包括关键短语、非关键短语、 空间向量模型和潜在语义索引是本算法的基础。

5.1 算法

MyCluster 算法主要包括两个阶段：（1）预处理；（2）聚类。在预处理阶段主要包括搜索结果的获取和解析等；在聚类阶段将涉及到特征抽取、类标签（即类名）归纳、类内容（类包含的文档）发现、类合并和类排序等。整个算法流程如下：

1. 输入：查询词
2. 获取搜索结果；
3. 解析搜索结果页；
4. 提取关键短语和非关键短语；
5. 建立 *phrase-document matrix*；
6. SVD 算法；
7. 计算类数目；
8. 发现类内容；
9. 归纳类标签；
10. 合并类；
11. 排序类；
12. 输出：聚类结果

5.1.1 预处理

5.1.1.1 获取搜索结果

根据用户输入的查询词 Q 构建查询 URL，利用 HTTP 和 SOCKET 协议来获

取某一搜索引擎返回的结果。

5.1.1.2 解析搜索结果页

下载程序下载的搜索结果是一 HTML 代码，要获取 title 和 snippet 等，就必须解析 HTML 源代码，其伪代码如下：

```
ParseWebPage(  )
{
    根据搜索结果页中的代码特征，获取每个搜索结果的 title 和 snippet 等；
    去除搜索结果中的所有 HTML 标记；
    将解析的结果保存到 SearchResultSet 结构体中。
}
```

SearchResultSet 结构体如下：

```
typedef struct{
    int   resultCount;                    //结果数
    long* docIndex;                       //文档 Index,从 0 开始
    char** url;                           //URL
    char** title;                         //标题
    char** snippet;                       //片段
    int keywordCount;                     //关键词数目
    char** keyword;                       //关键词
}SearchResultSet;                         //搜索结果集
```

5.1.2 聚类

5.1.2.1 特征抽取

通常把文档中的词作为一个文档的特征，利用词在文档中的词频来建立 *term-document* 矩阵。根据大量实验表明，短语比词更能表达文章的主题。所以，本系统中采用关键短语和非关键短语来建立特征矩阵 *phrase-document matrix*。

根据前面的介绍，可以利用共现短语（Co-Occurrence Phrase）构造算法来

获得关键短语和非关键短语。其伪代码如下：

```
GetKeyPhares(  )
{
```

1．对 SearchResultSet 中的片段和标题进行中文分词，结果保存在 ResultInfoSet 结构体中；

2．利用查询词来识别关键文档、关键句、非关键文档和非关键句；

3．利用共现短语（Co-Occurrence Phrase）构造算法来提取关键短语和非关键短语。

4．将关键短语和非关键短语保存在 KeyPhraseSet 中。

```
}
```

ResultInfoSet 结构体：

```
typedef struct
{
    long termID;                          //词 ID
    int termLen;                          //词长度
    float tf;                             //TF
    float idf;                            //IDF
    float weight;                         //词权重
}ContentTermArr;                          //词集
typedef struct
{
    int contentTermNum;                   //词集中词的个数
    ContentTermArr* contentTermArr;       //词集
}ResultInfo;                              //单一搜索结果
typedef struct
{
    int resultInfoCount;                  //搜索结果项个数
    ResultInfo* resultInfo;               //单一搜索结果集
}ResultInfoSet;                           //搜索结果集
```

KeyPhraseSet 结构体：

```
typedef struct
```

```
{
    int conceptCount;                    //关键短语包含的词个数
    long *concept;                       //关键短语包含的词 ID
    int docCount;                        //关键短语分布在文档的数目
    long *docID;                         //文档 ID
    float* weight;                       //关键短语权重
}KeyPhraseArray;
typedef struct
{
    int keyPhraseCount;                  //关键短语数目
    KeyPhraseArray *keyPhraseArray;      //关键短语数组
}KeyPhraseSet;
```

接着，我们以关键短语和非关键短语序号为数组纵向下标，以文档序号为横向下标，关键短语和非关键短语的权重为数组元素的值建立 *phrase-document* 矩阵，也即文档的特征矩阵。

```
VSM( )
{
    根据 KeyPhraseSet，来计算关键短语和非关键短语的权重；
    建立 phrase-document 矩阵 A。
}
```

5.1.2.2 SVD 算法

根据 SVD 算法，将 A 分成三个矩阵分别是 U，V，Σ。即 $A=U\Sigma V^T$，其中 U 的行表示关键短语或非关键短语，列表示类，矩阵 U 中的元素值即为类标签值；V^T 的行表示类，列表示文档，矩阵 V^T 中的元素值即为类内容值。我们利用 MatLab C++ Library 来实现对矩阵 A 的 SVD 运算。其具体步骤如下：

1．利用 MatLab 生成.m 文件，其内容如下：

```
function [U,S,V,r] = MySVD( X )
[U,S,V] = svd(X);
tol = max(size(X))*S(1)*eps;
```

```
t = sum(S > tol);
r=sum(t);
```

2. 用 MatLab 自带的 mcc 命令将.m 文件编译成 C++可调用执行的动态链接库文件 libSVD.dll 和 libSVC.lib；

3. C++程序直接调用动态链接库 libSVD.dll，获取结果。如果要实现脱离 MatLab 环境下运行，则必需有 MatLab 其他 C++库的支持。

伪代码如下：

```
SVD(  )
{
    获取A;
    调用 libSVD.dll，生成 U，V，Σ;
    由 V 得到 V^T。
}
```

5.1.2.3 类数目

本系统实现的 MyCluster 算法无需指定类的数目。设 k 为类的数目，取 $k(0<k<r)$满足下式（1）的最小值作为类的数目：

$$\frac{\| A_k \|_F}{\| A \|_F} = \frac{\sqrt{\sum_{i=1}^{k}\left(\sigma_i^2\right)}}{\sqrt{\sum_{i=1}^{r_A}\left(\sigma_i^2\right)}} \geq q \qquad (1)$$

其中，q 是一类数目阈值（一般取 q=0.9），A 是一 *phrase-document* 矩阵，A_k 是一 k 阶近似，r_A 是矩阵 A 的秩，σ_i 是 A 的第 i 个奇异值，$\|A\|_F$ 是矩阵 A 的 Frobenius 范数。

举例：根据 4.3 中提供的数据，若取 q=0.95，则可得到 k=2。显然，有：

U_2=

-0.6614	0.2669
-0.3053	0.7583
-0.6851	-0.5950

Σ_2=

4.2385	0
0	1.5113

V^T_2=

-0.3155	-0.1007	-0.6872	-0.6872	0
0.6501	0.7014	-0.2066	-0.2066	0

计算类数目的伪代码如下：

```
GetClassCount()
```

```
{
    获取 q，r_A，σ_i；
    For i=1 to r
    {
        If( 公式（1）成立 )
        {
            k = i;
            Break;
        }
    }
}
```

5.1.2.4 类内容发现

类内容发现也即发现属于某一类的文档集。设 t 为类内容阈值（一般取 t=0.1），取 V^T_2 的第 i 行中所有 $V^T_2[i,j]$值大于 t 的文档归为一个类 i。其伪代码如下：

```
GetClassContents()
{
    1. For i=1 to k
    {
        对于所有文档 j，如果 V^T_2[i,j]>t，则将文档 j 归为类 i;
    }
    2. 如果存在还没归类的类，则归为“其它”类，类中文档权重均为 0。
}
```

根据 4.3 中提供的数据，取 t=0.1，则得到如下类：

第 1 类：　　{ D_0, D_1 }

“其它”类：　{ D_2, D_3, D_4 }

5.1.2.5 类标签归纳

类标签归纳也即给类命名。取 U_2 的第 i 列中的 $U_2[i,j]$的最大值对应的 j 作为类 i 的标签。

```
GetClassLabels()
{
    For i=1 to k
    {
Max = U2[i,0];
s = 0;
For 对于所有短语 j
{
    If(U2[i,j] >=Max )
    {
        If(U2[i,j] ==Max)
            Tmp[++s] = j;//Tmp 作临时保存
        Else
        {
            Tmp[0] = j;
                Max = U2[i,j];
                s = 0;
            }
        }
    }
    将 Tmp[]值保存到 Label[i]中；
}
 “其它”类的类标签为“其它”，类标签权重为 0；
}
```

根据 4.3 中提供的数据，可得到类标签如下：

第 1 类：软件

第 2 类：其它

5.1.2.6 类合并

用 ClusterSet 结构体来保存类数目、类内容和类标签。结构体如下所示：

```
Typedef struct{
    long docIndex;                  //类中文档序号
    float docScore;                 //类中文档得分
}Doc;                               //类内容
typedef struct{
    char* label;                    //类标签
    float labelScore;               //类标签得分
    int docCount;                   //类中文档数
    Doc *doc;                       //类内容
}Cluster;                           //类
typedef struct{
    int count;                      //类数目
    Cluster* cluster;               //类
}ClusterSet;                        //类集
```

1．根据类标签合并两个类：

若类 x 与 y 的标签完全相同，则合并两个类：

（1）若类内容相同，合并后类内容权重值取两者的平均值；

（2）若类内容不同，则直接合并两者。

2．根据类内容合并两个类：

对于类 x，y：

（1）若 $x \in y$，则删除 x，保留 y；

（2）若 $y \in x$，则删除 y，保留 x；

（3）若 $|x \cap y| / |x \cup y| > \alpha$ 时，将类 x 和 y 的标签和内容均合并。

其中 α 为类合并阈值（一般取 α =0.75）

根据 4.3 中提供的数据，类合并结果如下：

第 1 类：类标签：软件；类内容：$\{D_0, D_1\}$

第 2 类：类标签：其它；类内容：$\{D_2, D_3, D_4\}$

5.1.2.7 类排序

类排序方法：类标签按 U 中类标签的值从大到小排序；类内容按 V^T 中类内容的值从大到小排序。

根据 4.3 中提供的数据，类排序结果：

第 1 类：类标签：软件；类内容：{ D_0, D_1 }

第 2 类：类标签：其它；类内容：{ D_2, D_3, D_4 }

也即：

第 1 类：

软件：

1. ...计算机软件 综合站点...
2. ...海环软件 物流软件 物流信息...

第 2 类：

其它：

1. ...计算机软件新技术国家重点实验室...
2. ...上海市计算机软件评测重点实验室...
3. ...湖南电子信息应用教育中心...

5.2 算法参数

算法中的许多参数决定了类的数目、类内容等，最终将影响聚类效果。参数如下表 5.1 所示：

表 5.1：算法参数

算法参数	建议值	描述	影响
类数目阈值 q	0.7—1（默认：0.9）	决定类的数目	q 越大，类的数目也越大
类内容阈值 t	0—0.5（默认：0.1）	决定类内容	t 越小，类包含的文档越多
类合并阈值 a	0.5—0.9（默认：0.75）	决定类的合并	a 越小，两个类合并可能性越大

我们通过大量的实验分析得到：当取 q=0.9，t=0.1，α=0.75 时，聚类效果最佳。

5.3 算法讨论

MyCluster 算法具有以下的优点：

1. 类标签的可读性

类标签来自关键短语和非关键短语，是句子间的共现短语，具有一定的代表性，而且是经过 LSI 算法提炼后形成的，可读性强。

2. 类区分度高

经过 LSI 算法计算后形成类，它们之间区分度高，而且类内容与类标签的相关性比较大。

3. 类重叠

搜索结果中的片段可能有多个主题，所有有些片段被分到多个类中，是合情理。

4. 模块化设计

模块化设计有利于系统的扩展和完善。

同样本算法也还存在一些问题：

1. 不支持增量聚类；
2. 有时类标签数目过多；
3. 计算代价过大；
4. 类标签中的短语可能不出现在类内容；
5. 未被归类的文档数比重过大。

5.4 MyCluster 框架

我们提出的基于 MyCluster 算法的聚类搜索引擎是元搜索和 Web 聚类技术的整合，共分为以下几个模块：

1. 查询处理模块

查询处理模块：主要接受用户的输入，将用户输入的查询词形成查询 URL，

然后提交给获取搜索结果模块。

2. 搜索结果获取解析模块

获取搜索结果模块根据 URL，实时下载搜索结果页，并解析。

3. 搜索结果聚类模块

搜索结果聚类模块对解析的结果运用 MyCluster 聚类算法进行聚类。

4. 聚类结果表现模块

聚类结果表现模块根据聚类结果，在表示页的左边显示类标签，当点击类标签时，在右边显示对应的类内容。

本系统运行流程如图 5.1 所示：

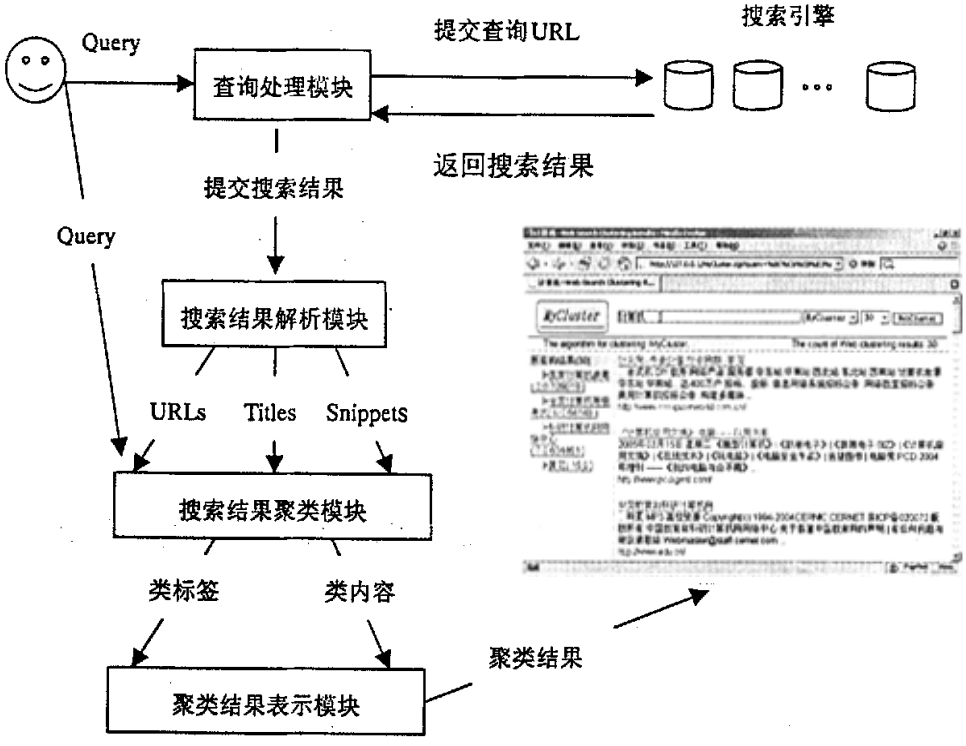

图 5.1：MyCluster 框架

第6章 MyCluster ——评价

6.1 搜索结果聚类的评价体系

如何评价一个聚类结果的好坏，建立一个可行的聚类算法评价体系是非常重要的。下面将介绍传统搜索引擎评价方法和我们自己的聚类搜索引擎评价体系。

6.1.1 传统评价方法

传统搜索引擎评价方法包括精确度（precision）和召回率（recall）。假设：D 为一文档集；文档集 A 为根据用户查询词从 D 中返回的文档集合；文档集 R 为 D 中与查询词相关的文档集合；RA 为 R 和 A 的交集，即 $RA=R \cap A$。

定义 1：精确度：是指搜索到的相关的文档的比例，有 $precision = |RA|/|A|$。

定义 2：召回率：是指相关的文档被返回的比例，故有 $recall = |RA|/|R|$。

聚类结果评价方法有：标准 IR 法[Zamir, 99]、Merge-then-cluster 法[Zamir, 99]和用户评价法。我们采用用户评价法来评价 MyCluster 算法。

6.1.2 用户评价法

受标准 IR 法中精确度和召回率评价标准的启发，我们提出了针对聚类搜索引擎的评价体系。我们的评价体系包含类标签可读性、类内容相关性、类内容覆盖率和类重叠度等指标。

评价体系所涉及的参数定义如下：

l – 类标签数目（不包括“其它”类）；

u –可读性好的类标签数（不包括“其它”类）；

c – 所有类标签包含的类内容总数（由于类重叠的存在，可能大于搜索结果总数）；

r – 与类标签相关的文档数（不包括“其它”类）；

n – 与类标签不相关的文档数（不包括“其它”类）；

s－ 搜索结果总数；

o－ “其它”类中包含的文档数。

定义 1：类标签的可读性（Cluster Label Readability）指可读性好的类标签所占总标签的比例。故有：

CLR = u/l。

定义 2：类内容的相关性（Cluster Content Relevance）指一类标签对应的类内容中与类标签相关的文档所占的比例。故有：

CCR = r/(r+n)。

定义 3：类内容覆盖率（Cluster Content Coverage）指未被分到“其它”类的文档所占搜索结果总数的比例。故有：

CCC = (s-o)/s。

定义 4：类重叠度（Cluster Overlap）指文档被重复分到类中的比例。故有：

CO = c/s-1。

6.2 评价结果

6.2.1 算法参数对聚类结果的影响

我们选择三个查询词（包括“计算机”、“聚类”和“计算机 科学”），分别对不同的类数目阈值 q、类内容阈值 t 和类合并阈值 α 进行测试。取三查询词对应的评价指标值的平均值作为评价标准，结果见图 6.1，6.2，6.3。搜索结果数均取 40。

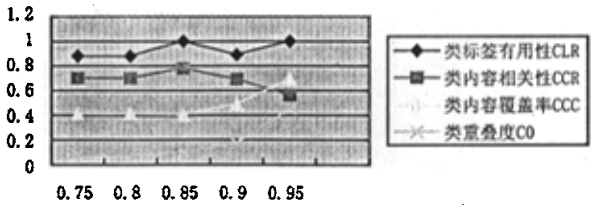

图 6.1：评价指标值（横坐标为 q，纵坐标为指标值，t=0.1，α=0.75）

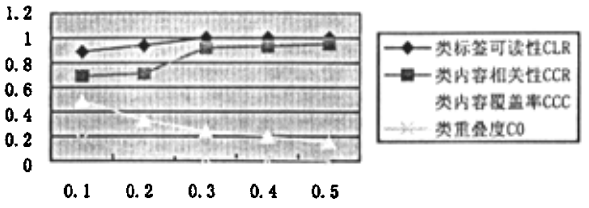

图 6.2：评价指标值（横坐标为 t，纵坐标为指标值，q=0.9，α=0.75）

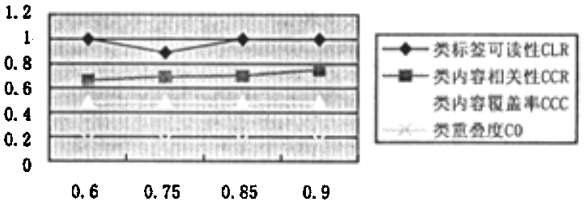

图 6.3：评价指标值（横坐标为 α，纵坐标为指标值，q=0.9，t=0.1）

综合考虑体系的指标值，我们认为：取类数目阈值 q=0.9，类内容阈值 t=0.1，类合并阈值 α=0.75 效果最好。

6.2.2 搜索结果数对聚类结果的影响

我们选择三个查询词（分别为“计算机”、“聚类”和“计算机 科学”），分别对搜索结果数为 20、40、100 的进行测试。取三查询词对应的评价指标值的平均值作为评价标准，结果见图 6.4。MyCluster 算法参数选择如下：类数目阈值 q=0.9；类内容阈值 t=0.1；类合并阈值 α=0.75。

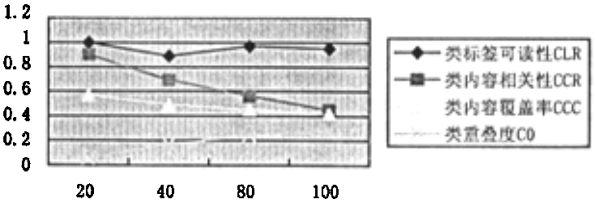

图 6.4：评价指标值（横坐标为搜索结果数，纵坐标为指标值）

从以上的测试中可以发现类标签 CLR 的有用性比较高，但随着搜索结果的增多，类内容 CCR 的相关性在下降，覆盖率 CCC 变化不大，类重叠度 CO 在上升。整体来说，聚类质量是不错的。

6.2.3 Carrot2 与 MyCluster 的比较

Carrot2 是波兰一大学的研究成果，是一个供学者、研究人员学习测试的好平台，其地址为 http://carrot.cs.put.poznan.pl/carrot2-remote-controller/index.jsp。我们选择三个查询词（分别“计算机”、“聚类”和“计算机 科学”）来分别测试 Carrot2 与 MyCluster 的评价指标值。MyCluster 中的算法参数采用 6.1.1.2.1 中的结论。测试的搜索结果数均为 50。

分别使用 STC 和 MyCluster 算法得到三个查询词的搜索结果评价如下表 6.1 所示；但仅给出搜索结果数和类标签等。每个类标签的右边括号中第一个数字为类标签包含的类内容数目，第二个数字为此类标签中相关类内容的比重，“其他”类不计算此比重。评价指标值见图 6.5。

表 6.1：搜索结果标签

查询词	算法	
	STC	MyCluster
计算机	搜索结果数：45 Cc0033(13，0%) Computer(5，60%) 电脑(5，50%) Chip 新电脑(2，100%) PC(3，100%) 应用文摘(2，50%) MP3(2，50%) 软件下载(2，50%) other(22)	搜索结果数：50 计算机考试(3，100%) 科研计算机网网络中心(11，100%) 招标公告(5，40%) 其他(34)

聚类	搜索结果数：43 分析(14，57%) Cc0033(3，0%) Search(3，67%) 判别分析(3，67%) 算法(3，100%) 系统(2，50%) 文本自动(2，100%) China Data Mining Research(2，100%) 基于模糊(2，100%) other(21)	搜索结果数：50 灰色聚类分析法(5，40%) 指纹图谱(9，33%) 中的应用(10，80%)
计算机 科学	搜索结果数：59 Cc0033(26，0%) 与技术学院(4，100%) 学院(5，100%) 与技术系（5，80%） 与工程系（3，100%） Computer（3，67%） 简介（2，100%） 清华大学（2，50%） 自由的百科全书（2，100%） Diyinside（2，100%） other	搜索结果数：50 中国计算机（3，67%） 大学计算机（8，100%） 大学计算机科学（4，100%） 计算机科学系（6，67%） 计算机工程（3，33%） 其它（31）

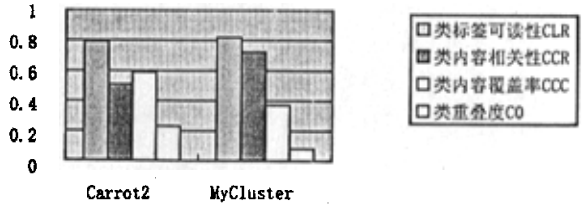

图 6.5：评价指标值（横坐标为不同的聚类搜索引擎，纵坐标为指标值）

从评价的指标值可得出：我们的 MyCluster 比 Carrot2 在类内容相关性有优势，但在类内容覆盖率上明显不及。Carrot2 的类标签可读性稍差，且不大适合中文检索。

第7章　结论

作为计算机科学中一个相当年轻的研究方向，Web 搜索结果聚类技术将有非常广阔的发展空间。一方面是由于进一步的研究将使聚类算法朝着更快速和精确的方向发展；另一方面是由于 Vivisimo 已经在商业领域的成功运用。

本文分析了各种搜索引擎和聚类算法，以及 Web 搜索结果聚类算法。我们发现在中文领域中聚类搜索引擎还是一片空白，而且一些 Web 搜索结果聚类算法往往将一篇文章看成是许多词的组合。受到 STC 算法的启发，我们把文章看成是短语的组合，提出了以短语和潜在语义索引为基础的模糊 Web 搜索结果聚类算法。在建立特征矩阵时，我们不采用传统的以词和文档来建立特征矩阵，而是以短语和文档来建立，此方法可以有效地避免数据噪音的干扰和提高类标签的可读性，而且可以明显地降低特征矩阵的维度，缩减计算时间。在归纳类标签和发现类内容过程中，我们采用潜在语义索引方法从语义层次上来对文档聚类，而不采用传统基于 VSM 的余弦法。

为了更好地验证 MyCluster 算法的聚类质量，本文给出了基于 MyCluster 算法的聚类搜索引擎框架及聚类结果评价体系。在实际编程实现 MyCluster 中，涉及到 HTML、Javascript、CGI 和 C++等 Web 编程和数学软件 MatLab，并实现了 C++在完全脱离 MatLab 环境下对 MatLab 生成的动态链接库的直接调用。我们的提出聚类结果评价体系包括类标签的有用性、类内容的相关性、类内容覆盖率和类重叠度等指标，来综合评价一个算法的质量。

通过对比实验，我们发现 MyCluster 在类标签可读性和类内容相关性方面有很大地优势，但在类内容覆盖率方面有所欠缺。

本文贡献：

1．提出了以短语和潜在语义索引为基础的模糊 Web 搜索结果聚类算法 MyCluster；

2．编程实现了基于 MyCluster 算法的聚类搜索引擎；

3．给出了 Web 搜索结果聚类算法的评价体系；

4．通过实验对比分析，总结出了 MyCluster 算法优势和不足之处。

我们希望以后在层次聚类算法、增量聚类算法、算法质量和实验分析方面有更深入的研究，逐步使我们的研究成果应用到商业领域中。

参 考 文 献

[KeepSo FTP, 05] FTP Search Engine. Online at *http://ants.keepso.com/*:2005-04-05|2005-04-06.

[AltaVista, 05] Web Search Engine. Online at *http://www.altavista.com/*:2005-04-02|2005-04-03.

[Baidu, 05] Web Search Engine. Online at *http://www.baidu.com*: 2005-04-02|2005-04-03.

[Berry, 95] M. W. Berry, S. T. Dumais, and G. W. O'Brien. *Using Linear Algebra for Intelligent Information Retrieval.* SIAM Review, 37 (4). pp573-595. 1995.

[Brin, Page, 98] S. Brin and L. Page. *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Computer Networks ISDN System. Vol. 30. pp107-117. 1998.

[Carrot2, 03] J. Stefanowski and D. Weiss. *Carrot2 and Language Properties in Web Search Results Clustering.* Proceedings of the First International Atlantic Web Intelligence Conference, Madrit, Spain. 2003.

[Carrot2, 05] Clustering Search Engine.

Online at *http://www.cs.put.poznan.pl/dweiss/carrot/*:2005-04-15|2005-04-16.

[Chakrabarti, 98] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. *Automatic resource compilation by analyzing hyperlink structure and associated text.* Proceedings of the Seventh International World Wide Web Conference. pp65-74. 1998.

[Cutting et al., 92] D. Cutting, D. Karger, J. Pedersen, and J. W. Tukey. *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections.* Proceedings of the 15th Annual International ACM/SIGIR Conference, Copenhagen. 1992. pp318-329.

[Deerwester, 90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. *Indexing by Latent Semantic Analysis.* Journal of the American Society for Information Science. Vol. 41. pp. 391-407. 1990.

[Dogpile, 05] Dogpile Meta Search Engine.

Online at *http://www.dogpile.com* :2005-04-10|2005-04-11.

[Ehrenfeucht and Haussler, 88] Ehrenfeucht, A. and Haussler, D. *A new distance metric on strings computable in linear time.* Discrete Applied Math. Vol. 20. pp191-203. 1988.

[Froogle, 05] Web Shopping Search Engine.

Online at *http://www.froogle.com*: 2005-04-10|2005-04-11.

[Golub, 89] G. H. Golub and C. F. van Loan. *Matrix Computations.* Johns Hopkins University

Press, London. 1989.

[Google, 05] Search Engine. Online at *http://www.google.com* : 2005-04-02|2005-04-03.

[Google APIs, 05] Google Web Services.

Online at *http://www.google.com/apis/*:2005-04-10|2005-04-11.

[Google Groups, 05] Usenet Search Engine.

Online at *http://groups.google.com/*:2005-04-10|2005-04-11.

[Google Personalized, 05] Personalized Search Engine.

Online at *http://labs.google.com/personalized/*:2005-04-10|2005-04-11.

[Grouper, 99] O. Zamir and O. Etzioni. *Grouper: a dynamic clustering interface to web search results.* In Proceedings of the Eighth International World Wide Web Conference, Toronto, Canada. pp283-296. May 1999.

[Hearst and Pedersen, 96] M. A. Hearst and J. O. Pedersen. *Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results.* Proceedings of the Nineteenth Annual International ACM SIGIR Conference, Zurich. June. pp76-84. 1996.

[Landau and Vishkin, 89] Landau, G. M. and Vishkin, U. *Fast Parallel and serial approximate string matching*. Journal of Algorithms, Vol. 10. pp157-169. 1989.

[LookSmart, 05] A Human-Made Web Directory.

Online at *http://search.looksmart.com/*:2005-04-02|2005-04-03.

[Mamma, 05] Mamma Meta Search Engine.

Online at *http://www.mamma.com/*:2005-04-10|2005-04-11.

[Metacrawler, 05] Metacrawler Meta Search Engine.

Online at *http://www.metacrawler.com/*:2005-04-10|2005-04-11.

[Mooter, 05] Cluster Search Engine.

Online at *http://www.mooter.com/moot* /:2005-04-15|2005-04-16.

[ODP, 05] Open Directory Project – a human-made web directory.

Online at *http://dmoz.org* /:2005-04-05|2005-04-06.

[Page, Brin et al., 98] L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank citation ranking: Bringing order to the web*. Technical report, Stanford University, Stanford, CA. 1998.

[Perez et al., 00] R. Valdes-Perez, V. Pericliev, and F. Pereira. *Concise, Intelligible, and Approximate Profiling of Multiple Classes.* International Journal of Human Computer Systems. Vol. 53, No.3. pp411-436. 2000.

[Rodeh et al., 81] Rodeh, M., Pratt, V. R. and Even, S. *Linear algorithm for data compression via string matching.* Journal of the ACM, 28(1). pp16-24. 1981.

[Selberg, 99] E. W. Selberg. *Towards Comprehensive Web Search.* Doctoral Dissertation, University of Washington, 1999.

[SHOC, 04] D. Zhang, Y. Dong. *Semantic, Hierarchical, Online Clustering of Web Search Results.* In Proceedings of the 6th Asia Pacific Web Conference (APWEB), Hangzhou, China. Apr 2004.

[Yates et al., 99] R. B. Yates and B. R. Neto. *Modern Information Retrieval.* ACM Press. 0-201-39829-X.

[Vivísimo, 05] Clustering Search Engine.

Online at *http://vivisimo.com/*:2005-04-15|2005-04-16.

[WebCrawler, 05] Meta Search Engine.

Online at *http://www.webcrawler.com/*:2005-04-10|2005-04-11.

[Weiss, 01] D. Weiss. *A Clustering Interface for Web Search Results in Polish and English.* Master Thesis, Poznan University of Technology. 2001.

[Wiener, 73] Weiner, P. *Linear pattern matching algorithms*. In Proceedings of the 14th Annual Symposium on Foundations of Computer Science (FOCS'73). pp1-11. 1973.

[Yahoo, 05] Web Search Engine.

Online at *http://www.yahoo.com/*:2005-04-02|2005-04-03.

[Zamir and Etzioni, 98] O. Zamir and O. Etzioni. *Document Clustering: A Feasibility Demonstration*. Proceedings of the 19th International ACM SIGIR Conference on Research and Development of Information Retrieval, pp 46-54. 1998.

[Zamir, 99] O. Zamir. *Clustering Web Documents: A Phrase-Based Method for Grouping Search Engine Results*. Doctoral Dissertation, University of Washington. 1999.

[Zhongsou, 05] Web Search Engine.

Online at *http://www.zhongsou.com/*:2005-04-02|2005-04-03.

[Zhongsou IPS, 05] Personalized Search Engine.

Online at *http://ips.zhongsou.com/*:2005-04-10|2005-04-11.

科 研 成 果

一、攻读硕士学位期间发表的论文

1. Zhansheng Li, Yajun Du, Yang Xu, Yuekun Wang, and Dongmei Qi. The Personalized PageRank Based on User Behaviors. The 6th International FLINS Conference on Applied Computational Intelligence. 2004.

2. 李战胜，杜亚军，齐冬梅. 个人 Web 搜索服务系统的研究. 第 21 届全国数据库会议. 2004.

二、攻读硕士学位期间完成的项目

1. 2003 年 1 月至 2004 年 6 月，在学校完成“个性化智能搜索引擎”和“个人搜索服务系统”项目。

2. 2004 年 6 月至 2005 年 1 月，在北京中搜公司完成“个性化搜索引擎”项目。

3. 2005 年 1 月至 2005 年 4 月，在北京中搜公司完成“智能导航”项目。

申 明

本学位论文是在导师的指导下完成的研究工作和取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包括为西华大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。论文成果归西华大学所有，特此申明。

作者签名 [illegible] 2015 年 6 月 10 日

导师签名 [illegible] 2015 年 6 月 1 日

致 谢

这三年，可以说是进步的三年、成功的三年。我无怨无悔，因为每天我都在成长。在此，我首先感谢我的导师杜亚军副教授。在他的精心而耐心地指导下，我学会了很多，学习上有了质的飞跃。同时也很感谢彭宏副教授在学术讨论上的指导和建议。也特别感谢王月昆同学的帮助，在与她交流的过程中，我吸取了很多思想和启示。也一同感谢我们智能化研究小组的成员和2002级所有同学及朋友给我生活上、学习上的帮助。同时，也非常感谢北京中搜公司给我提供的实习机会。最后，感谢我的家人给我生活上、物质精神上的帮助。我也一定不会辜负大家的期望，我会努力的。

作者

2005 年 4 月