

AN INTELLIGENT ADAPTIVE NEWS FILTERING SYSTEM

BY

YING HUANG

B.E., University of Science & Technology of China, 1996

M.E., University of Science & Technology of China, 1999

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

ACKNOWLEDGEMENTS

I wish to sincerely thank my research advisor, Dr. Daniel A. Reed, for his kind guidance and support along the development of this thesis. Thank him for providing me with this opportunity to benefit from his knowledge.

I would like to thank the members of the Pablo group who have helped and encouraged me in my work. I would like to thank Eric Shaffer, Don Schmidt and Jim Oly for giving me a hand whenever I met difficulties. Thanks are also due to Phoebe E. Lenear, Mary Pietrowicz, Stewart Wood, Mario R. Medina, and Mathew George for testing the recommendation system and providing helpful comments. I am also indebted to Deborah Israel for her valuable proofreading.

At last, I dedicate this thesis to my dear husband Yanning Liu for his encouragement and patience throughout my thesis writing. Without his help, it would be much difficult for me to reach this point.

This work was funded in part by National Science Foundation under grants NSF EIA-99-72884 and NFC PACI Computational Science Alliance Cooperative Agreement. I would like also to thank National Science Foundation for their support that helped me to finish this thesis.

TABLE OF CONTENTS

| | |
|---|-----------|
| Chapter 1 Introduction | 1 |
| 1.1 Motivation | 1 |
| 1.2 Thesis Organization | 2 |
| Chapter 2 Information Filtering | 4 |
| 2.1 Information Filtering and Information Retrieval | 4 |
| 2.2 Information Filtering Techniques | 6 |
| 2.2.1 Content-based Filtering Techniques | 7 |
| 2.2.2 Collaborative Filtering Techniques | 8 |
| 2.2.3 Hybrid Techniques | 10 |
| 2.3 Explicit and Implicit Feedback | 11 |
| 2.4 Test-bed — Smart Room | 11 |
| 2.4.1 RF/ID Badge | 13 |
| 2.4.2 Display Wall | 13 |
| 2.4.3 Elvin, Tick and Tickertape | 15 |
| 2.5 Filtering Approach | 17 |
| Chapter 3 Algorithms | 19 |
| 3.1 Vector-Space Models | 19 |
| 3.2 Latent Semantic Indexing (LSI) | 23 |
| 3.2.1 Term-by-Document Representation | 24 |
| 3.2.2 Weighting Schemas | 24 |
| 3.2.3 Singular-Value Decomposition (SVD) | 25 |
| 3.2.4 LSI Example | 28 |
| 3.2.5 News Filtering Using LSI | 32 |
| 3.3 Collaborative Filtering Algorithms | 34 |
| 3.3.1 User-by-Term Representation | 34 |
| 3.3.2 Weighting Schemas | 36 |
| 3.3.3 Neighborhood-Based Algorithms | 36 |
| 3.4 Combining LSI and Neighborhood-Based Algorithms | 38 |
| 3.5 Arbitration Algorithms | 40 |
| 3.5.1 Initialization | 41 |
| 3.5.2 Self Adaptation | 42 |
| Chapter 4 Implementation | 44 |
| 4.1 Overview | 44 |
| 4.2 Database Tables | 46 |
| 4.3 News Collector | 46 |
| 4.4 LSI Filter | 47 |
| 4.4.1 Pre-Processing Module | 49 |
| 4.4.2 SVD Processing | 50 |
| 4.4.3 Initial Prediction | 51 |
| 4.5 Similarity Comparison | 51 |
| 4.6 User Interface | 52 |

| | |
|--|-----------|
| 4.6.1 Registration Window | 52 |
| 4.6.2 TickerTape – The Display Window | 54 |
| 4.7 News Agent | 56 |
| 4.8 Feedback Module | 58 |
| 4.9 Collaborative Recommender | 58 |
| 4.10 ReLSI Module | 59 |
| Chapter 5 Experimental Results | 60 |
| 5.1 Statistical Information of News Source | 60 |
| 5.1.1 Article Number of Each News Section | 61 |
| 5.1.2 Similarity Within Each News Section | 63 |
| 5.1.3 Similarities Between News Sections | 65 |
| 5.2 User Experiments | 70 |
| 5.3 Simulated User Tests | 72 |
| 5.3.1 Given Conditions of Experiments | 73 |
| 5.3.2 Specializing to User Interests | 74 |
| 5.3.3 Adapting to User Interest Changes | 77 |
| 5.3.4 Explore New Domains | 80 |
| Chapter 6 Conclusions | 84 |
| Chapter 7 Future Work | 86 |
| 7.1 The Filtering Module | 86 |
| 7.2 Implicit Feedback | 87 |
| 7.3 User Interface | 87 |
| References | 89 |
| Appendix A Example of Filtering Using LSI | 92 |

LIST OF FIGURES

| | |
|---|----|
| 1.1 Illustration of a filtering system | 2 |
| 2.1 Model of information retrieval | 5 |
| 2.2 Model of information filtering | 5 |
| 2.3 Illustration of vector space models (3D)..... | 7 |
| 2.4 Illustration of collaborative filtering systems | 10 |
| 2.5 Photos of Smart Room | 12 |
| 2.6 Photo of a badge | 13 |
| 2.7 Photo of the display wall | 14 |
| 2.8 Photos of the PC cluster and projectors | 14 |
| 2.9 Architecture of an Elvin system | 15 |
| 2.10 TickerTape window | 16 |
| 2.11 Configuration of tickerTape properties | 17 |
| 3.1 Local weighting functions | 21 |
| 3.2 Global weighting functions | 21 |

| | | |
|------|---|----|
| 3.3 | Normalization methods | 22 |
| 3.4 | Similarity measures | 22 |
| 3.5 | Factorization of term-by-document matrix A_θ | 26 |
| 3.6 | Factorization of reduced-dimension matrix A | 26 |
| 3.7 | Simplified factorization of reduced-dimension matrix A | 26 |
| 3.8 | Sample dataset consisting of 9 documents | 29 |
| 3.9 | Two-dimension approximation A of A_θ | 31 |
| 3.10 | Results of the document vectors and similarites | 33 |
| 3.11 | User-by-item representation of a collaborative filtering model | 35 |
| 3.12 | Example of user-by-news matrix in our approach | 35 |
| 3.13 | Example of user-by-term matrix in our approach | 35 |
| 4.1 | Flowchart of the personalized/group news agent system | 44 |
| 4.2 | Flowchart of LSI Filter | 48 |
| 4.3 | Registration window | 53 |
| 4.4 | Editing the preference for user Ying Huang | 53 |
| 4.5 | Configuration of parameters | 55 |
| 4.6 | User preference dialog | 55 |
| 5.1 | Number of articles in section 1, 4, 5, 9, 17 | 61 |
| 5.2 | Number of articles in section 6, 7, 10, 13 | 62 |
| 5.3 | Number of articles in sport sections | 62 |
| 5.4 | Similarities within each news section | 64 |
| 5.5 | Average self-similarity of each news section | 65 |
| 5.6 | Temporal average similarity of each pair of sections (2D and 3D) | 66 |
| 5.7 | Similarities between any two sections with ID in $\{1,4,5,7\}$ | 67 |
| 5.8 | Similarity between section 6 and section 1,4,5,7 over time | 67 |
| 5.9 | Similarities between section 10 with section 1,4,5,7 over time | 68 |
| 5.10 | Similarities between section 13 with section 1,4,5,7 over time | 68 |
| 5.11 | Similarities between sport news sections over time | 69 |
| 5.12 | Real user experimental results | 71 |
| 5.13 | Breakdown of the database in our experiments | 73 |
| 5.14 | Specializing to user interests (experiment 1) | 75 |
| 5.15 | Specializing to user interests (experiment 2) | 77 |
| 5.16 | Adapting to user interest changes (experiment 1) | 78 |
| 5.17 | Adapting to user interest changes (experiment 2) | 80 |
| 5.18 | Average predictions for George and Wang | 82 |
| 5.19 | Predictions for Wang based on George's opinion | 82 |
| 5.20 | Predictions for Wang after he read 5 Football articles | 83 |
| A.1 | Term-by-document matrix of the new document and profile documents | 92 |
| A.2 | Two-dimension approximation A of A_θ | 94 |
| A.3 | Results of the document vectors and similarites | 95 |

CHAPTER 1

INTRODUCTION

1.1 Motivation

The Information Age is an era of exploding access to news sources; an age when the volume of information published by mailing lists, web sites, and other media is growing exponentially. Every day more than 2500 online newspapers provide access to tens of thousands of news articles covering a multitude of interests [6]. Countless newsgroups flourish in cyberspace. Each may have hundreds, sometimes thousands, of daily posts that are potentially significant to members of its user community. The airwaves are teeming, with TV and radio news broadcasts.

This abundance of news sources has created an environment in which people can spend more time sifting through potential information sources than they do absorbing the material that actually interests them. Filters are invented to relieve this information overload and enable people to spend their time on material of relevance.

Similar to information retrieval systems, information-filtering systems are designed to help users quickly find the needed information. Because information preferences can vary greatly across individuals, filtering systems must be tailored to individual interests. We believe an intelligent filtering system must satisfy the following three requirements:

- Customization: An intelligent information filtering system should be highly responsive to the specific interests of its users. Conventional filtering systems record user interests in profiles or in user models. However, extensive research is needed to determine how best to represent the user's interests so that the system can more precisely distinguish material that is relevant from material that is tangential.
- Adaptation: Repeated interactions with users put filtering systems in an excellent position to recognize changes, over time, in user needs. Building on the assumption that most user actions are consistent with long-term user interests, information filtering systems should exploit ongoing user feedback to identify changing individual preferences. Such systems should be able not only to adapt to changes in preferences that are manually indicated, but also to respond to such changes as detected through system usage.

- Exploration: An intelligent filtering system must be able to explore new domains for interesting information and then recommend new items based on what it knows about a user.

Figure 1.1 depicts a typical information filtering system with three basic modules: customization, adaptation, and exploration. The “Customization” module compares incoming documents with the user profile to recommend customized information for each user. Here user profiles are representations of user interests. Items from new domains are provided by the “Exploration” module. These two modules cooperate to satisfy the user’s information needs. Based on feedback from users, the “Adaptation” module attempts to adjust the system to changes in the user’s interests.

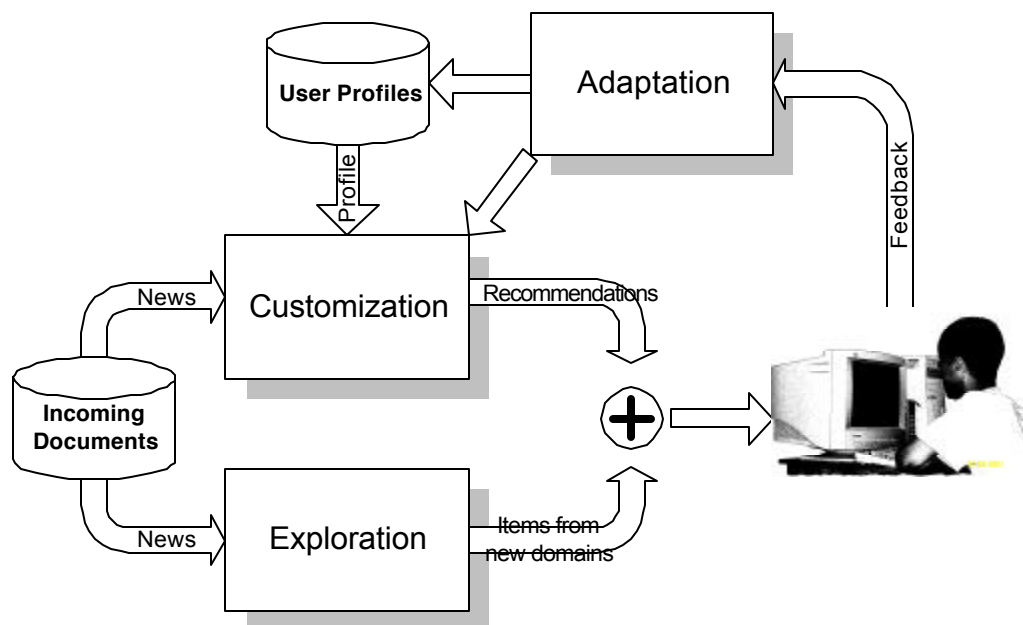


Figure 1.1 Illustration of a filtering system

1.2 Thesis Organization

Chapter 2 provides an overview on some related work in the area of information filtering. We discuss the relationship between information filtering and retrieval, two basic categories of information filtering techniques, and two feedback modes. Chapter 2 ends by briefly introducing our approach. Chapter 3 introduces the basic algorithms used to build the news agent and the

arbitrator. It details how these algorithms work. Chapter 4 describes the implementation – how we build our system using the algorithms described in Chapter 3. Chapter 5 presents the experiments conducted using our system as well as the experimental results analysis. Conclusions are explained in Chapter 6. Recommended areas for future work are given in Chapter 7. An example is given in Appendix A to help readers understand the Latent Semantic Indexing algorithm.

CHAPTER 2

INFORMATION FILTERING

Information filtering has been a major research field for many years. Because a tremendous amount of information is being created and delivered over networks, it is increasingly time consuming for Internet users to select items from the flow of information. Tools to regulate this flow are urgently needed to enable computer users to handle the flood of incoming information.

Thus, the concept of information filtering was proposed to help users eliminate irrelevant information. A practical information filtering system should be customized, which means it must remember all users and individualize its performance for each one. Furthermore, because a filtering system involves repeated user interactions, it should be able to follow their interests over time. Below, we discuss the design of a customized information filtering system. Before presenting our approach, we first examine some related/previous work done in this field.

2.1 Information Filtering and Information Retrieval

Conventional information retrieval (IR) is very closely related to information filtering (IF). They both have the goal of retrieving information relevant to user interests while minimizing the amount of irrelevant information [1]. In Belkin and Croft's paper [11], information retrieval and information filtering are considered as two sides of the same coin.

Information retrieval systems, such as library catalogs and WWW search engines, provide tools for system users to seek materials that match their interests by submitting queries. Figure 2.1 illustrates a general model of an IR system. The query, which must be expressed in a language understood by the IR system, is a representation of the information favored by the user. The query from the user is then compared with surrogates that represent the databases containing relevant texts. Once retrieved, these texts will be returned to the user for use and evaluation. The

evaluation will lead to the modification of the query, sometimes even the text surrogates. This process is generally known as relevance feedback. IR systems are usually optimized for ephemeral interest queries, such as searching for papers on a research topic in a library.

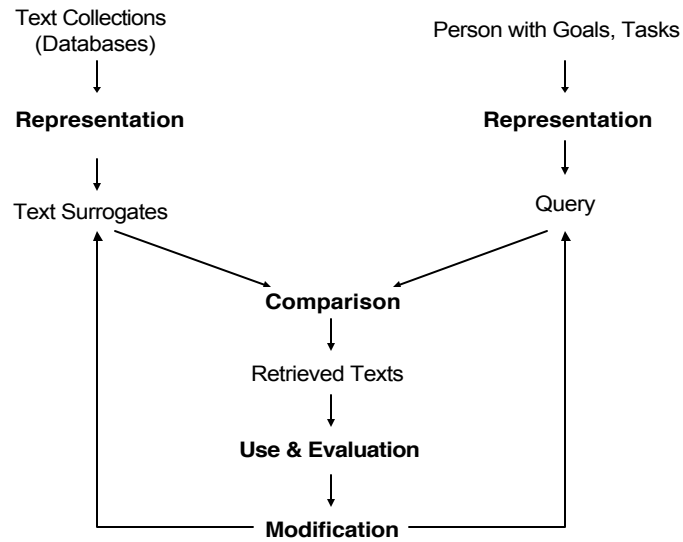


Figure 2.1 Model of information retrieval

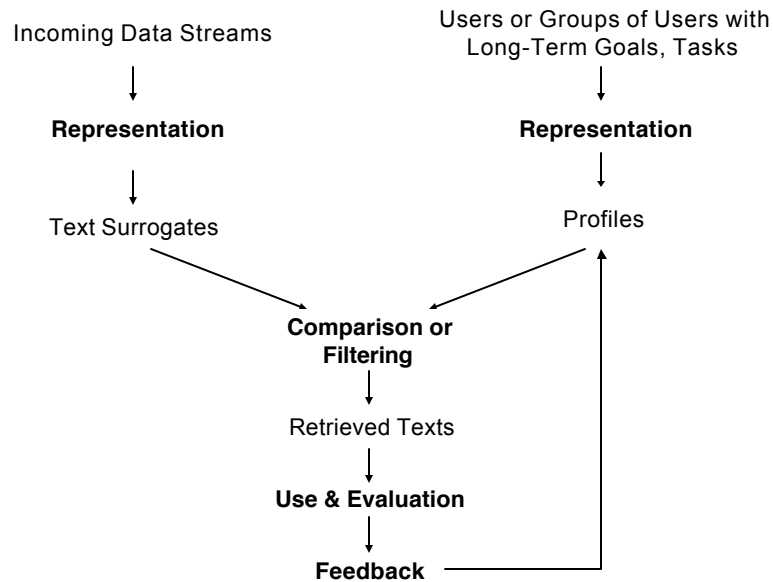


Figure 2.2 Model of information filtering

Figure 2.2 shows the process of IF, which is quite similar to IR. However, IF differs from IR in the following ways: 1) IF is typically applied to incoming data streams, such as the online news broadcast by newswire services, emails, or newsgroups; whereas in IR, the document collections are relatively fixed. Changes to databases usually do not occur often, and retrieval is not limited to new items; 2) user preferences in IF typically represent long-term interests, which are described in the user profiles, while queries in IR represent short-term interests that can be satisfied by the retrieval; 3) IF implies removing data from the incoming information stream to see what remains; while IR is a process of finding/extracting information from document collections.

Another way to see the relationship between IR and IF is to recognize that IF systems use many of the same techniques as IR systems. In both IR and content-based IF, a textual database or a user profile can be represented as a word-by-document matrix with each entry representing the frequency with which a word appears in a document. Thus, the retrieval (or filtering) projects the query (or incoming document) into the word-by-document vector space and retrieves (or filters in) the documents nearest to the query (or incoming document). We will discuss this in detail later. A number of IR techniques can be used to evaluate the effectiveness of most IF systems, measuring precision and recall.

2.2 Information Filtering Techniques

Information filtering has been studied extensively. As a result, two basic approaches have emerged for making recommendations: content-based filtering and collaborative filtering. Content-based filtering analyzes the content of incoming information and tries to pick items similar to those favored by the given user, whereas collaborative filtering identifies the users whose tastes are similar to those of the current user and recommends items these users found interesting. Recently, a significant amount of research has been done on combining these two methods to give better recommendations. We will refer to those as hybrid techniques and discuss some of the work done in this area.

2.2.1 Content-based Filtering Techniques

Content-based filtering is derived from the techniques used in the information retrieval community and is good at filtering text-based items. A content-based filtering system tries to

make recommendations by calculating the similarities between incoming items and user profiles. The most common and simplest method of content-based filtering is keyword matching. For example, the *kill* files and string search features provided in Usenet perform keyword-matching filtering. Based on this simple keyword matching approach, vector space models have been developed for better information retrieval/filtering.

In standard vector-space models [1], a user profile, a collection of documents, is represented by a term-by-document matrix with each entry representing the frequency with which a term appears in a profile document. Different weighting schemas can be applied to express the relative importance of different terms.

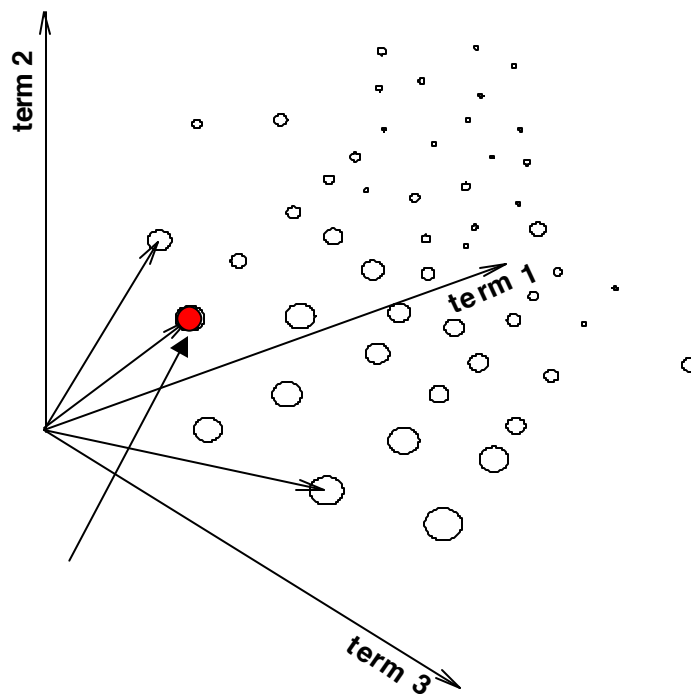


Figure 2.3 Illustration of vector space models (3D)

Here, the terms are considered to be independent. A vector space can be formed by all these terms with each one representing a dimension. Each profile document is a point in this vector space. Figure 2.3 shows an example with 3 terms and several documents. The similarity between two documents is computed as the inner product or cosine of the corresponding two vectors. We filter the incoming articles by comparing them with the profile documents and recommending the most

similar items. However, it is not reasonable to assume that all words are independent and that every word has only one meaning, which are known as synonymy and polysemy problems. New methods are needed to solve them.

Latent Semantic Indexing (LSI) [2,3] is one of those methods. LSI does not consider words to be independent. Instead, LSI uses the singular-value decomposition (SVD) technique to reduce the dimensionality of the term-by-document space so that the major associations among terms and documents can be calculated and exploited while some noise caused by synonymy and polysemy can be ignored (see section 3.2 for details). Using this method, users can be recommended relevant documents even if these documents do not have any word in common with the user profiles.

A pure content-based system has several drawbacks. First, in some domains like music, it is difficult to extract the content of individual items because they have more features (e.g., melody) than what text can represent. Second, even for textual documents, the term analysis cannot give information regarding quality or style (e.g., humor and satire). Finally, a content-based system only recommends those items that score highest against the user profile, which could restrict the user to reading articles similar to those read before. (i.e., The system cannot evolve based on user interests changing.)

2.2.2 Collaborative Filtering Techniques

Rather than recommending items similar to those that proved satisfactory before, a collaborative filtering system tries to recommend items favored by other users with similar tastes [4,5]. In a collaborative filtering system, each user has a profile that records scores for all items he/she has rated. When the system tries to make recommendations for an active user, it will first identify similar users by comparing user profiles. Then, the system will select those users with the highest similarities to form a neighborhood and recommend the items those neighbors like. Figure 2.4 illustrates the idea of collaborative filtering.

Because humans determine the relevance, quality, and appeal of the items available, collaborative filtering systems need not attempt a qualitative analysis of item contents to recognize their potential applicability. Humans are much better at judging text qualitatively (e.g., acuity, authoritativeness, or writing style). What is more important is that collaborative filtering can

make serendipitous recommendations – recommending items that do not contain the expected content but are nonetheless valuable to the user.

Compared with content-based filtering, the concept of collaborative filtering is relatively new. Although it overcomes some shortcomings of content-based filtering, it introduces other problems:

- The first one is known as the “early rater problem.” The system can provide few useful recommendations to a new user because it is difficult to tell who his/her neighbors are. Also when an item appears for the first time, if nobody has ever tried it, this item will not get a chance to be recommended.
- The second problem is sparsity. The number of items available in many information domains often far exceeds what any individual is able to use and evaluate. For example, we can assume that few people would have read and formed an opinion on even 0.1% of 1,000,000 books in a large bookstore. In this case, the user-by-book matrix containing the bookstore’s ratings would be very large and sparse. Thus, it could be fairly difficult to form a neighborhood because different users often would be interested in using and rating entirely different items.
- Finally, in a small community, some users may not benefit from collaborative filtering because their opinions are totally different from others. This is generally called the “Gray sheep” problem.

Both content-based filtering and collaborative filtering have their own problems. Fortunately, if we combine the two methods together, the latter can address the problems of the former while the former will not be affected by the latter’s problems. So a new trend is to use their combination.

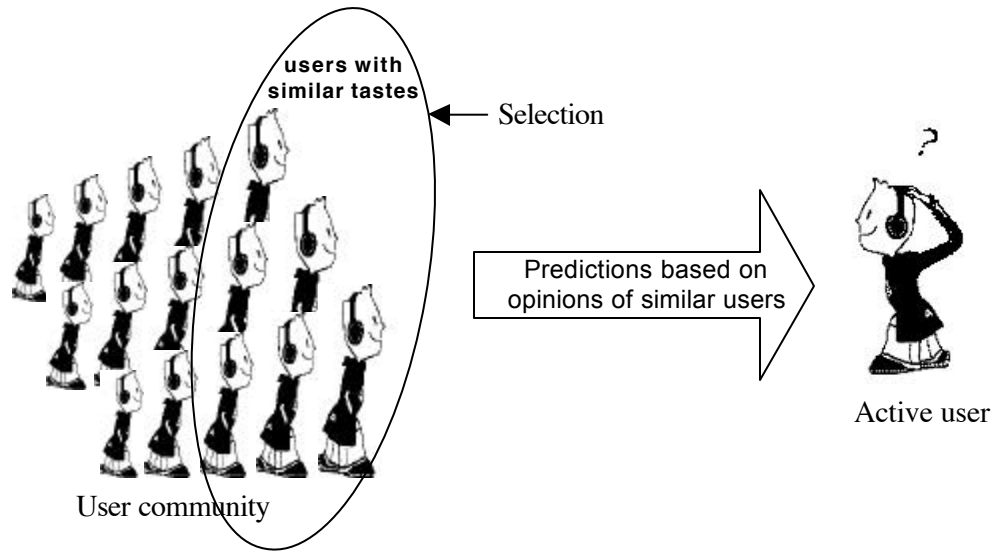


Figure 2.4 Illustration of collaborative filtering systems

2.2.3 Hybrid Techniques

Several research groups including *GroupLens* and *Fab* have combined both content-based and collaborative filtering. *GroupLens* implements a hybrid collaborative filtering system for Usenet news and uses content-based filters known as filterbots [13]. Filterbots evaluate new articles as soon as they are published and enter ratings for those documents (like a content-based filtering agent). The collaborative filtering component treats each filterbot as an ordinary user who enters ratings but does not request predictions. Filterbots can help solve the rating sparsity problem, but cannot help new users with the early rater start-up problem.

The *Fab* system is a hybrid, content-based collaborative system for recommending Web pages [7]. In *Fab*, profiles are used to show users' preferences. The profiles comprise the words of the web pages visited by users, and the weight assigned to those words via the users' ratings. The similarities between any two users can be computed by directly comparing their profiles. Items will then be recommended to a user either when they score high against the user's profile or when they are highly rated by the user's neighbors.

2.3 Explicit and Implicit Feedback

As illustrated in Figure 22, user feedback is needed to re-construct profiles. There exist two different feedback modes: explicit and implicit. Explicit feedback requires users to explicitly evaluate the recommended objects and register an opinion of the objects. *SIFT*, *Tapestry*, and *GroupLens* are examples of information filtering systems that use explicit feedback [18,19,12]. Although explicit feedback systems are easy to implement, they increase the cognitive load on their users.

On the other hand, systems using implicit feedback observe users' behavior, noting which articles are read or how long users spend on different objects. Implicit feedback is more convenient for users but harder to implement. *InfoScope*, which is a system for filtering Usenet news [20], uses three methods to implicitly measure users' interests in each message: whether the message is read or ignored, whether it is saved or deleted, and whether or not a follow up message has been posted. In his study, Stevens [20] observed that implicit feedback is effective in tracking long-term interests because it operates constantly without being intrusive.

Morita and Shinoda proposed a profile processing technique to collect users' preferences based on reading-time monitoring [21]. Their research showed a strong, positive correlation between the reading time and the explicit ratings provided by testers.

The Refer group has focuses on the correlations between users' Web browsing actions (implicit interests) and their explicit interests [22]. They built a Web browser called the Curious Browser to record scrolling, access time, mouse movements, and mouse clicks. According to their experiments, reading time and time spent scrolling are good implicit indicators of interests, while mouse movements and mouse clicks by themselves are not very useful.

2.4 Testbed — Smart Room

To evaluate a hybrid approach to information filtering, we build a filtering system for a “smart room.” This smart room [17] is “a testbed to explore and evaluate intelligent devices and augmented realities.” Figure 2.5 shows the smart room environment. Intelligent devices and control facilities include computers, PDAs, smart whiteboards, X10 automation sensors, tracking cameras, projectors, LCD displays, and a large display wall. These items are connected via multimodal networks: infrared, spread spectrum radio, Ethernet, and ATM.



Figure 2.5 Photos of Smart Room

Based on this infrastructure, we are implementing an intelligent adaptive recommendation system for news. Next, we will introduce the devices and control applications used to build our recommendation system.

2.4.1 RF/ID Badge

In the smart room, RF/ID technology is used to detect active users. RF/ID is based on bi-directional radio frequency communication between a base station and an ID badge worn by a user. When a badge (see Figure 2.6) is within the detectable range of the base station, a dialog will be set up in which the badge identifies itself by sending its ID information. Upon receiving this information, the base station analyzes the data and updates the database to indicate that this badge is active. Each user has a unique badge, and the mapping between them can be found in the database too. Badges are needed when a group of users share a display device to browse recommendations from the filtering system. To best accommodate the preferences of all active users in the room, the system must be able to identity each of them.

Figure 2.6



Photo of a badge

2.4.2 Display

Wall

Figure 2.7 shows the smart room's 24'x8' display wall. It consists of 4'x8' rear projection screens that displays graphics cooperatively by 8(3x6) video projectors. A PC cluster (18 Pentiums running Linux) drives these tiled screens. These PCs are connected by a high-speed network so that they can cooperate in partitioning, rendering, and assembling images. The 18 projectors are used to display the corresponding parts of images onto the wall (see Figure 2.8). This display wall is always used as a shared device to display recommendations for a group.



Figure 2.7 Photo of the display wall

2.4.3



Elvin, Tick and Tickertape

Elvin, Tick and Tickertape are technologies developed by the Distributed Systems Technology Center (DSTC) at the University of Queensland of Australia [28, 29]. Figure 2.9 shows the

architecture of a simplified Elvin system used in our approach. This system is a client-server model with Elvin as the server and all producer (e.g. news-agents) as well as consumer applications (e.g. tickertapes) as clients.

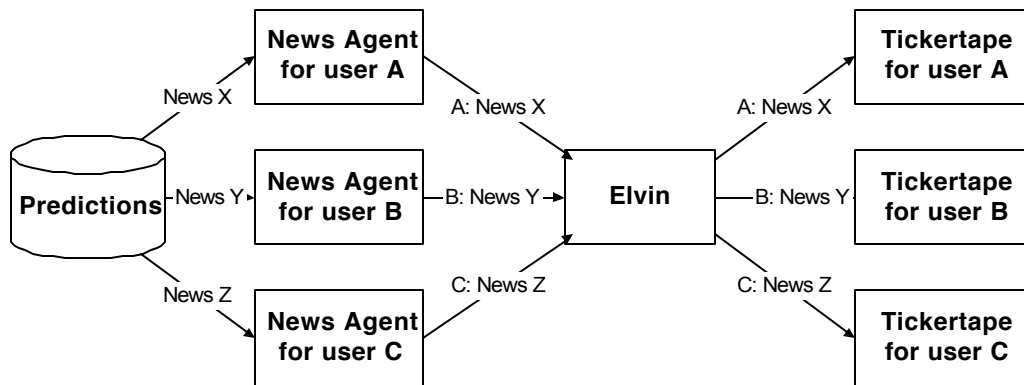


Figure 2.9 Architecture of an Elvin system

In brief, Elvin is a content-based notification/messaging server. All producer applications add their names to the beginning of each message they create before sending it to the Elvin server. Consumer applications may subscribe to any producer message. The Elvin server remembers these subscriptions, and routes each incoming message to one or more consumers based on its producer name.

Tick is a command-line program used by producers to generate Elvin notifications in the format required by the tickertape application. The usage of the tick command is

tick -g group -f from -s subject [-p port] [-h host] [-m mime_type -a mime_args]

where: *group* is the name of the producer (the customized news agent) and it should be set as the username;

from is the name of the person who sends this message;

subject is the text of the message;

port & host is the port and IP address of Elvin server;

mime_type indicates the type of attachment so that Tickertape knows how to browse the attachment;

mime_args is the attachment, which is the URL of the news items in our project;

For example, the news messages in our system are of the following format:

<username, news_title, port, server_IP, elvin/url, news_URL>.

The tickertape is used to display recommended news messages. It is a scrolling, one-line window as shown in Figure 2.10. Each message scrolling in the tickertape window consists of a group name, the sender's name, and the text of the message. Like emails, these messages can also have attachments (mimes). Each message has a total timeout and its appearance changes over its life, fading from color to gray, finally vanishing altogether.



Figure 2.10 Tickertape window

Tickertape is a very convenient tool. Users can configure the appearance of messages, including color, font, and scrolling speed (see Figure 2.11). Users can also define mouse actions. For example, users can choose a comfortable way to view the embedded attachment of each individual message, such as “Double Click” or “Shift + Click.” What is more important is that users can specify subscriptions so that only such information can be filtered in.

In our system, we have one producer (news agent) and one consumer (tickertape) for each active user. Both of them are configured with the username. The news agent uses the tick tool to produce news messages for this active user with the username at the beginning. Elvin then delivers the messages to the corresponding user’s tickertape window (see Figure 2.9).

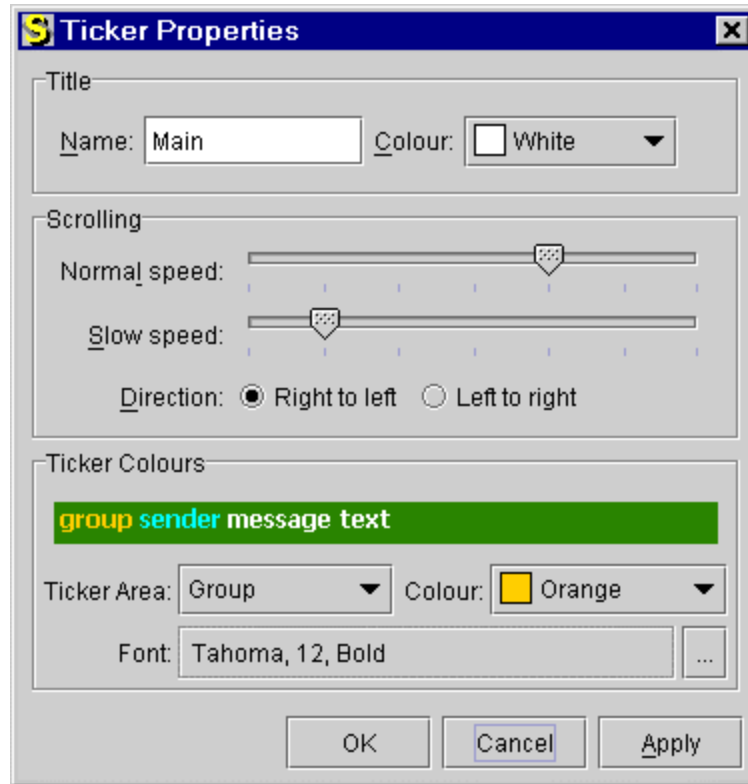


Figure 2.11 Configuration of tickertape properties

2.5 Filtering Approach

Our system is built on both content-based and collaborative filtering techniques. We focus on recommending news articles by having a news agent for each user, and moreover, we build an arbitration system to make recommendations for a group of users in a shared environment.

In summary, the main highlights of our work are:

- We integrate the content-based and collaborative filtering so that they complement to one another and help overcome each other's shortcomings.
- We use the LSI algorithm instead of standard vector-space methods in the content-based filtering so that the synonymy and polysemy problems can be solved. By using SVD, LSI can exploit underlying semantic structures to make better recommendations.
- We take advantage of the strength of human intelligence in understanding information content by using the collaborative filtering, to improve the accuracy of the recommendation system.

- We use implicit feedback (news being read or ignored) to update users' profiles so that no additional work on the part of users is necessary.
- During the process of making recommendations, we also provide some randomly selected news items. In doing so, we give users an opportunity to expand the interests reflected in their user profile. By taking advantage of these randomly recommended news from different news sections, users will expand their profiles to include some new fields. If they select the randomly recommended news from the new fields, they will be recommended more in those areas.
- We produce an arbitrator server that enables a group of users to share the same display device and accommodates the preferences of all active users in the room.

In next chapter, we will describe in details the LSI, collaborative filtering and arbitration algorithms. We will also talk about how to combine them together.

CHAPTER 3

ALGORITHMS

In this chapter, we introduce the core algorithms used in our system, including the Latent Semantic Indexing (LSI) algorithm, neighborhood-based collaborative filtering method, and adaptive arbitration algorithm. First, we describe the vector-space models that play such an important role in both LSI and collaborative filtering.

3.1 Vector-Space Models

Vector-space models [1] are a class of retrieval techniques that model documents as collections of terms from which one can derive the meaning of the documents inversely. Suppose we have a collection of M documents and a dictionary of N unique terms. Each document can be mathematically represented as a vector of term weights $D_i = (w_{i1}, w_{i2}, \dots, w_{iN})$, where D_i represents document i , and w_{ij} ($1 \leq j \leq N$) represents the weight of term T_j in document D_i . If term T_k ($1 \leq k \leq N$) does not occur in document D_i , the weight w_{ik} will be assigned zero.

Each unique term in this document collection corresponds to a unique dimension in the N -dimensional space and each document can be represented as a vector (or a point) within this term space. Similarly, a query can also be described as a vector. Therefore, the positions of the query vector and the document vector will tell their similarity. Then, the retrieval is to set a threshold and return all documents with a degree of similarity higher than the threshold.

Two factors are very important in designing vector-space models. The first is choosing a weighting schema. Weighting schemas can be categorized as: local, global and normalization weighting. Local weighting schemas focus on how to assign weights among terms T_j ($1 \leq j \leq N$) within any document D_i ($1 \leq i \leq M$) based on their frequencies of occurrence tf_{ij} . The common local weighting functions are listed in Figure 3.1. We can see from these formulae that the terms occurring more frequently will be assigned higher weights.

Global weighting schemas assign a weight for each term according to its usage across the whole document collection. The popular global weighting functions are listed in Figure 3.2. The “Normal” formula means that if a high-frequency term occurs in many documents, it should be assigned a low weight because this term is very possibly a common word. The “GfIdf” formula means that if a term occurs frequently in some documents but rarely in others, it should be set a high weight because it is a distinguishing feature of the former documents. Finally, the “Tfidf” formula is similar to “GfIdf” except that it uses the collection size instead of the global term frequency.

The weight of term T_j in document D_i (w_{ij}) may be its local weight, its global weight or the product of the two weights. Furthermore, we can use normalization schemas to normalize the resulting document vector $D_i = (w_{i1}, w_{i2}, \dots, w_{iN})$ (or \vec{v}). Common normalization methods are shown in Figure 3.3.

We can make different combinations of these weighting functions, choosing each one from a different group to achieve a specific purpose. For example, if we combine “Augment-Normalize,” “GfIdf” and “Cosine,” the formula to compute term weights would be:

$$w_{ij} = \frac{(0.5 + 0.5 * (tf_{ij} / \max_tf) * (gf_j / df_j))}{\sqrt{\sum_i [(0.5 + 0.5 * (tf_{ij} / \max_tf) * (gf_j / df_j))^2]}} \quad (3-1)$$

Binary:
$$L(i, j) = \begin{cases} 0 & \text{if } tf_{ij} = 0 \\ 1 & \text{if } tf_{ij} > 0 \end{cases}$$

Term-Frequency:
$$L(i, j) = tf_{ij}$$

Augment-Normalize:
$$L(i, j) = 0.5 + 0.5 * (tf_{ij} / \max_tf)$$

Log:
$$L(i, j) = \log(tf_{ij}) + 1$$

where:

$L(i, j)$ = the local weight of term T_j in document D_i

tf_{ij} = the frequency of occurrence of term T_j in document D_i

\max_tf = the maximum occurrence frequency

Figure 3.1 Local weighting functions

Normal:
$$G(j) = \sqrt{\frac{1}{\sum_i tf_{ij}^2}}$$

GfIdf:
$$G(j) = \frac{gf_j}{df_j}$$

Tfidf:
$$G(j) = \log\left(\frac{num_docs}{df_j}\right)$$

where:

$G(j)$ = the global weight of term T_j in document D_i

tf_{ij} = the frequency of occurrence of term T_j in document D_i

gf_j = the global frequency of term T_j ($1 \leq j \leq N$)

df_j = the number of documents in which term T_j ($1 \leq j \leq N$) appears

num_docs = the total number of documents in this collection

Figure 3.2 Global weighting functions

| | |
|---------|--|
| Sum: | $N(i, \vec{v}) = \vec{v} / (\sum_j w_{ij})$ |
| Cosine: | $N(i, \vec{v}) = \vec{v} / (\sqrt{\sum_j w_{ij}^2})$ |
| Max: | $N(i, \vec{v}) = \vec{v} / \max_w w$ |

where:

$N(i, \vec{v})$ = the normalized document vector

\vec{v} = the original document vector

$\max_w w$ = the biggest element in \vec{v}

w_{ij} = the weight of term T_j in document D_i in \vec{v}

Figure 3.3 Normalization methods

The second concern relevant to implementation of vector-space models is choosing the similarity measures. Different similarity measures have different emphases. By computing the inner product of two vectors, we actually give the product of the projected vector lengths. The cosine similarity measure, on the other hand, by computing the cosine of the angle between two vectors, de-emphasizes the lengths of the vectors (see Figure 3.4). (i.e., The cosine is the inner product of two vectors divided by their amplitudes.)

| | |
|----------------|---|
| Inner product: | $similarity = \sum_j w_{1j} * w_{2j}$ |
| Cosine: | $similarity = \frac{\sum_j w_{1j} * w_{2j}}{\sqrt{\sum_j w_{1j}^2} * \sqrt{\sum_j w_{2j}^2}}$ |

where:

w_{1j} = the weight of term T_j in document D_1

w_{2j} = the weight of term T_j in document D_2

Figure 3.4 Similarity measures

Vector space models assume that each term corresponds to a unique dimension, which implies that the terms are independent. However, this assumption is not realistic for two reasons: synonymy and polysemy.

Synonymy means that there are many ways to refer to the same object. Different users with different habits and knowledge backgrounds may use various terms to describe the same information. Even the same user may use different words to describe the same thing in different contexts. For example, when people refer to “notebook,” they may use “laptop,” “portable computer,” or “ThinkPad”. A query about “laptop” will fail to find articles about “portable computer” or “notebook” or “PDA” or “ThinkPad.” Furnas & Landauer & et.al. [30] showed that the probability of two users using the same main keyword for a single well-known object is less than 20%, which says that users often retrieve as little as 20% of the available relevant information.

On the other hand, polysemy means that words have more than one meaning. Therefore, the fact that a document contains a certain term does not necessarily mean that the query and the document are relevant. For instance, physicians use the word “shock” in a totally different way from mechanical engineers (In the field of medicine, “shock” means the loss of consciousness, while in mechanical engineering, “shock” is a strong wave across which there are jumps of pressure and temperature). Thus, when both use “shock” as the keyword in indexing, some of the information retrieved could be irrelevant to them. So, it is possible that some terms may share a dimension, while some terms may need more than one dimension. Such complex associations among terms cannot be reflected by vector-space models, and other methods have been developed to uncover the latent semantic structures. Latent semantic indexing (LSI) is one successful example.

3.2 Latent Semantic Indexing (LSI)

Latent Semantic Indexing [2,3] is based on standard vector retrieval methods and is designed to solve the synonymy and polysemy problems described in section 3.1. LSI assumes that there are some “latent” structures in the pattern of co-occurrences of words across document collections and tries to model dependencies among terms and documents. It uses singular-value decomposition (SVD) to reduce the dimension of the term-by-document space (the N -dimensional space will be reduced to a k -dimensional one).

All N terms and M documents can be represented as vectors within the dimension-reduced space. Therefore, the terms are no longer orthogonal, and the synonyms will correspond to the same dimensions or have similar directions. The documents with similar term usage patterns will be near one another even if they do not share common terms, which means that LSI can reveal the latent semantic relationships among documents. Thus, for a given query, which probably does not contain all synonymy keywords, LSI can still obtain all relevant information. As in the previous example, LSI will learn that “laptop” and “portable” occur in many of the same contexts and have similar vectors, so queries about one will probably retrieve documents about the other.

3.2.1 Term-by-Document Representation

Like vector-space models, a document collection in a LSI model can also be represented as a term-by-document matrix in which each row stands for a unique term, each column stands for a document, and each entry represents the weight of a term in a document. However, in vector-space models all term vectors are orthogonal, and documents are represented as a linear combination of such independent term vectors ($D_i = (w_{i1}, w_{i2}, \dots, w_{iN})$). In the LSI model, the term-by-document matrix is reduced to an approximation with rank k ($k < N$). The rows and columns of the new matrix will be used to represent term and document vectors respectively. In section 3.2.3, we can see that term vectors and document vectors can also be formed simply by the “singular values” and “singular vectors” of the term-by-document matrix.

3.2.2 Weighting Schemas

LSI uses all of the three weighting schemas introduced earlier to increase or decrease the relative importance of terms within each document and across the entire collection, respectively. The weighting schema used in our LSI model is called “tfidf” (term frequency times inverse document frequency), which is the combination of the local weighting function “Log,” the global function “Tfidf,” and the normalization function “Cosine.” “tfidf” can be expressed as

$$w_{ij} = \frac{(\log(tf_{ij}) + 1.0) * \log(num_doc / df_j)}{\sqrt{\sum_j [(\log(tf_{ij}) + 1.0) * \log(num_doc / df_j)]^2}} \quad (3-2)$$

where:

tf_{ij} = the frequency of occurrence of term T_j in document D_i .

df_j = the number of documents in which term T_j ($1 \leq j \leq N$) appears

num_docs = the collection size

Here, the factor $\log(num_doc / df_j)$ is the inverse document frequency, which is used to normalize the term weights so that 1) a term occurring in many documents will be assigned a smaller weight; and 2) longer documents are not unfairly given more weight.

3.2.3 Singular-Value Decomposition (SVD)

Singular-value decomposition [2,10] is a technique usually used in eigenvector decomposition and factor analysis. By using SVD, a large term-by-document matrix will be decomposed into a dimension-reduced matrix that only keeps the k largest orthogonal factors from which the original matrix can be best approximated. This k orthogonal factor space reflects the major associative patterns in the words while ignoring some noise that may be due to differences in word usage or low-frequency terms. In this dimension-reduced space, each term and each document can be represented as a linear combination of these k orthogonal indexing dimensions (a vector). Their positions reflect the correlations among them.

SVD factorizes a weighted term-by-document matrix A_0 such that $A_0 = U_0 S_0 V_0^T$, where U_0 and V_0 are orthogonal, and $S_0 = diag(\sigma_1, \dots, \sigma_r)$, $r = \min(t, d)$, with $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0$. These σ_i are called the singular values, the r columns of V_0 are called the right singular vectors and the r columns of U_0 are called the left singular vectors. Figure 3.5 describes this factorization processing.

If we only keep the k largest singular values of S_0 and set the remaining smaller ones to zero ($S_0 \Rightarrow S$), the product $U_0 S V_0^T$ will be an approximation to the matrix A_0 . We call this approximated product A . Figure 3.6 shows the factorization of A . We can further simplify the factorization of A by deleting the zero rows and zero columns in S and the corresponding rows and columns in matrices U_0 and V_0 (see Figure 3.7). Now we can easily see that the dimension of the term-by-document matrix is reduced to rank k .

$$\begin{array}{c}
 \begin{array}{ccccc}
 & \text{documents} & & & \\
 \text{terms} & \boxed{A_0} & = & \boxed{U_0} & \times & \boxed{S_0} & \times & \boxed{V_0^T} \\
 & t \times d & & t \times r & & r \times r & & r \times d \\
 & A_0 & = & U_0 & * & S_0 & * & V_0^T
 \end{array}
 \end{array}$$

Figure 3.5 Factorization of a term-by-document matrix A_0 (t-by-d)

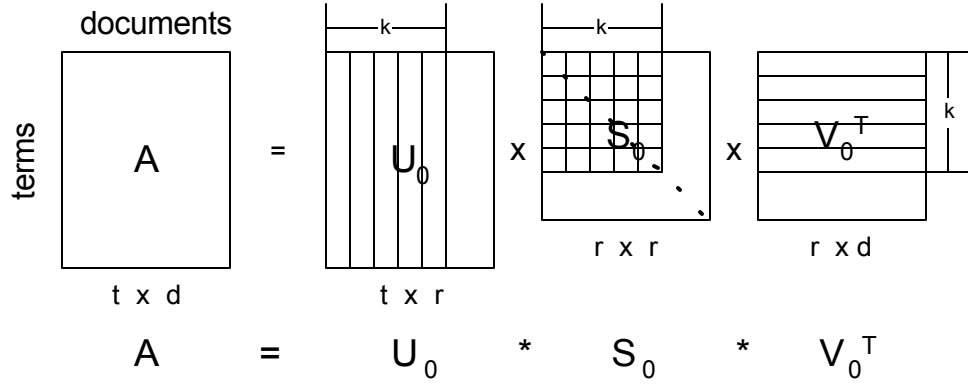


Figure 3.6 Factorization of a dimension-reduced matrix A (t-by-d)

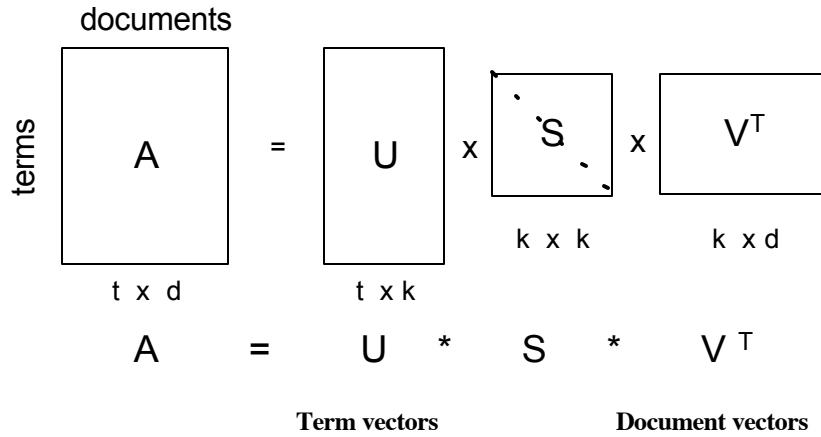


Figure 3.7 Simplified factorization of a dimension-reduced matrix A (t-by-d)

After we get the dimension-reduced representation of a document collection, we can use its rows and columns as the term and document vectors respectively. Thus, we can easily compute the similarities among terms, documents and queries.

1) Similarity between two terms

We can use the inner product (or cosine) of any two term vectors to estimate the similarity between them. By computing

$$AA^T = (USV^T)(USV^T)^T = US^2U^T = (US)(US)^T, \quad (3-3)$$

element $AA^T[i][j]$ tells how similar the usage of term i and term j are across the entire document collection. It is important to notice that each element $AA^T[i][j]$ is indeed the inner product of row i and row j of the matrix US . Hence we can

consider each row of US as a term vector, and the similarity between two terms can be computed as the inner product of the corresponding two vectors. It will be the same if we use rows of the matrix U or $US^{\frac{1}{2}}$ as term vectors. Because S is diagonal, right-multiplying such a matrix is only a space stretching or shrinking — the positions of the points are the same, except the scale of each axis has been changed by a factor corresponding to the diagonal elements of S .

2) Similarity between two documents

In the same way, the similarity between any two documents i and j can be represented as the entry (i, j) of the matrix $A^T A$.

$$A^T A = (USV^T)^T (USV^T) = VS^2V^T = (VS)(VS)^T \quad (3-4)$$

From formula 3-4, we can see that each element $A^T A[i][j]$ is the inner product of row i and row j of the matrix VS . Thus, we can use rows of either VS , $VS^{\frac{1}{2}}$ or V as coordinates for the corresponding documents, and the inner product (or cosine) of two document vectors will show the similarity between these two documents.

3) Similarity between queries and documents

To compare a query with all database documents, first we need to project it into a pseudo-document vector Q using the steps below.

1. Given a query, we can extract its keywords to form the term vector q_0 .
2. Taking q_0 , we can weigh it using the same weighting schema used for the whole document collection, and get the new vector q_1 .
3. Then we can derive the query vector Q using the following algebra:

$$Q = q_1^T US^{-1} \quad (3-5)$$

i.e. the query vector consists of the sum of the term vectors $(q_1^T U)$ scaled by the inverse of S^{-1} .

Thus, the similarities between the query and other documents can be computed as the distances (or angles) between the query vector and other document vectors.

3.2.4 LSI Example

Figure 3.6 is a simple dataset containing 9 documents. These documents are classified into two clusters, one is the field of combustion on stretched/strained flames (labeled c1-c5) and the other is the research of viscous flows, an active area in fluid dynamics (labeled m1-m4). Only the words that occur in more than one document will be considered as terms (which are in bold font in the documents).

Documents:

C1: Numerical model of flame stretching

C2: Numerical simulation of premixed deflagrations under variation of the rate of stretch

C3: Extinction of strained premixed deflagrations of hydrogen/air/steam mixture

C4: Effects of straining on the local structure of freely propagating premixed flames

C5: Flame extinction in stretched PMMA diffusion flames

M1: Numerical investigation of high Re-number internal flow

M2: 1DFLOW – a model for viscous incompressible flow

M3: Study of the **viscous effects** on the vortex/wall interaction

M4: Viscous cavity flow under different Re-number

| Terms | Documents | | | | | | | | |
|--------------|-----------|----|----|----|----|----|----|----|----|
| | C1 | C2 | C3 | C4 | C5 | M1 | M2 | M3 | M4 |
| numerical | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| model | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| flame | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| stretch | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| premix | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| deflagration | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Extinction | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| strain | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| effect | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| re-number | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| flow | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| viscous | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

[illegible]

0 0 0 0 0 0 0 0 0.3795

$V_0 =$

0.4610 -0.1080 -0.3885 -0.2756 0.3778 -0.0046 -0.3042 0.2726 0.4892
0.3876 -0.0044 0.3120 -0.5903 0.2483 -0.0183 0.4164 -0.3701 -0.1778
0.3126 0.1739 0.6346 0.0129 -0.3057 -0.4126 -0.1672 0.3863 0.1679
0.3943 0.1495 0.3038 0.4628 0.2118 0.5196 -0.3249 -0.3078 -0.0258
0.5881 0.1900 -0.4782 0.2567 -0.4080 -0.1458 0.2765 -0.0390 -0.2401
0.1380 -0.4836 0.0524 -0.2743 -0.3640 0.4986 -0.1534 0.3648 -0.3626
0.1015 -0.5253 -0.0065 0.1845 0.2466 -0.5346 -0.3480 -0.1842 -0.4220
0.0636 -0.2030 0.1243 0.3896 0.4465 0.0608 0.5621 0.5137 -0.0675
0.0688 -0.5905 0.0988 0.1849 -0.3134 0.0103 0.2520 -0.3352 0.5731

If we only keep the first two largest singular values of S_0 , we can get A_0 's two-dimensional approximation $A = USV^T =$

$\begin{bmatrix} 0.30 & 0.17 & 0.62 & 0.44 & 0.33 & 0.21 & 0.27 & 0.21 & 0.14 & 0.06 & 0.09 & 0.07 \\ -0.23 & -0.24 & 0.16 & 0.03 & 0.12 & 0.07 & 0.14 & 0.12 & -0.02 & -0.41 & -0.62 & -0.51 \end{bmatrix}^T$

$* \begin{bmatrix} 3.2989 & 0 \\ 0 & 2.6001 \end{bmatrix} *$

$\begin{bmatrix} 0.46 & 0.39 & 0.31 & 0.39 & 0.59 & 0.14 & 0.10 & 0.06 & 0.07 \\ -0.11 & -0.01 & 0.17 & 0.15 & 0.19 & -0.48 & -0.53 & -0.20 & -0.59 \end{bmatrix}$

We can write the resulting A in the format of a term-by-document matrix shown in Figure 3.9.

Terms

Documents

C1 C2 C3 C4 C5 M1 M2 M3 M4

| | | | | | | | | | |
|--------------|------|------|-------|-------|-------|-------|-------|-------|-------|
| numerical | 0.52 | 0.39 | 0.20 | 0.30 | 0.47 | 0.42 | 0.41 | 0.18 | 0.42 |
| model | 0.33 | 0.22 | 0.07 | 0.13 | 0.21 | 0.38 | 0.39 | 0.16 | 0.41 |
| flame | 0.89 | 0.79 | 0.71 | 0.86 | 1.28 | 0.08 | -0.02 | 0.04 | -0.11 |
| stretch | 0.65 | 0.56 | 0.46 | 0.58 | 0.86 | 0.16 | 0.11 | 0.08 | 0.05 |
| premix | 0.47 | 0.42 | 0.40 | 0.48 | 0.70 | -0.00 | -0.06 | 0.00 | -0.11 |
| deflagration | 0.30 | 0.27 | 0.25 | 0.30 | 0.44 | 0.01 | -0.02 | 0.01 | -0.05 |
| extinction | 0.38 | 0.35 | 0.34 | 0.41 | 0.60 | -0.05 | -0.10 | -0.02 | -0.15 |
| strain | 0.29 | 0.27 | 0.28 | 0.33 | 0.48 | -0.06 | -0.10 | -0.02 | -0.14 |
| effect | 0.22 | 0.18 | 0.13 | 0.17 | 0.26 | 0.09 | 0.07 | 0.04 | 0.06 |
| re-number | 0.21 | 0.08 | -0.12 | -0.08 | -0.08 | 0.55 | 0.59 | 0.23 | 0.65 |
| flow | 0.31 | 0.13 | -0.18 | -0.12 | -0.12 | 0.82 | 0.87 | 0.34 | 0.97 |
| viscous | 0.25 | 0.10 | -0.16 | -0.10 | -0.11 | 0.67 | 0.72 | 0.28 | 0.79 |

Figure 3.9 Two-dimensional approximation A of the original term-by-document matrix A_0

Now let us take a look at what the dimension reduction has done to the relations between documents (including queries). Suppose we have a query asking for information about “flame.” If we use the keyword matching method, only documents C1, C4 and C5 will be returned because they all contain the term “flame.” Although documents C2 and C3 are also relevant to this query, they cannot be retrieved because $\langle \text{flame}, C2 \rangle = \langle \text{flame}, C3 \rangle = 0$ (see Figure 3.8).

Fortunately if we use the LSI method, we can retrieve all of them. From Figure 3.9, we can see that the entries of $\langle \text{flame}, C2 \rangle$ and $\langle \text{flame}, C3 \rangle$ are 0.79 and 0.71, respectively. Hence, the possibility of C2/C3 containing the term “flame” is 0.79/0.71. Thus, we can guess that all documents C1~C5 could be retrieved. To make it clear from the mathematical point of view, we will give the calculation following the steps described in section 3.2.3.

First, we can project the query into a vector:

$$Q = q_1^T U S^{-1} = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$\begin{aligned}
& * \\
& \begin{bmatrix} 0.30 & 0.17 & 0.62 & 0.44 & 0.33 & 0.21 & 0.27 & 0.21 & 0.14 & 0.06 & 0.09 & 0.07 \\ -0.23 & -0.24 & 0.16 & 0.03 & 0.12 & 0.07 & 0.14 & 0.12 & -0.02 & -0.41 & -0.62 & -0.51 \end{bmatrix}^T \\
& * \\
& \begin{bmatrix} 1/3.2989 & 0 \\ 0 & 1/2.6001 \end{bmatrix} \\
& = [0.1879 \quad 0.0615]
\end{aligned}$$

Second, we can compute the document vector $(VS)^T$ for each article:

| C1 | C2 | C3 | C4 | C5 | M1 | M2 | M3 | M4 |
|-------|-------|------|------|------|-------|-------|-------|-------|
| 1.52 | 1.29 | 1.02 | 1.29 | 1.95 | 0.46 | 0.33 | 0.20 | 0.23 |
| -0.29 | -0.03 | 0.44 | 0.39 | 0.50 | -1.25 | -1.38 | -0.52 | -1.53 |

Finally, we can compute the cosine value between the query vector and each document vector:

| Q~C1 | Q~C2 | Q~C3 | Q~C4 | Q~C5 | Q~M1 | Q~M2 | Q~M3 | Q~M4 |
|--------|--------|--------|--------|--------|--------|---------|--------|---------|
| 0.8763 | 0.9438 | 0.9959 | 0.9998 | 0.9977 | 0.0380 | -0.0814 | 0.0473 | -0.1663 |

We see that all documents C1~C5 can achieve high similarity against this query. Therefore, using SVD, the query can retrieve all related documents (C1~C5).

3.2.5 News Filtering Using LSI

LSI is widely used in information retrieval systems and is regarded as one of the best content-based filtering techniques. In our system, we record the text of the news articles read by each user and use it as the user profile. Therefore a user profile is a collection of documents just like a database. On the other hand, we regard each incoming news item as a query. By computing the similarities between the query and the profile documents of a user, we can predict how well this item fits with this user's preferences (see 4.3.2 for implementation details).

We change the example described in section 3.2.4 to illustrate how to use LSI in our system. A user, Ben, is interested in both “combustions/flames” and “viscous fluid dynamics.” He has read the documents c1~c5 and m1~m4. Now an incoming document is:

C6: Dynamics of edge flames

Computing the similarities between C6 and all profile documents (C1~C5 and M1~M4) in the same way yields the same results:

| C6~C1 | C6~C2 | C6~C3 | C6~C4 | C6~C5 | C6~M1 | C6~M2 | C6~M3 | C6~M4 |
|--------|--------|--------|--------|--------|--------|---------|--------|---------|
| 0.8763 | 0.9438 | 0.9959 | 0.9998 | 0.9977 | 0.0380 | -0.0814 | 0.0473 | -0.1663 |

Document c6 fits Ben's first preference well and should be included. Actually, the prediction for c6 is 0.9627 if the average of the largest 5 similarities is used.

Another way to use LSI in filtering is to combine the new document with all documents in the user profile to form a new term-by-document matrix. By applying SVD to it, we can use the columns of VS as the document's vectors and the cosine value of any two vectors as their similarity. Therefore, in the previous example, we can combine c6 and the documents in Ben's profile to form a 12*10 matrix. Figure 3.10 shows the results of the document vectors and the similarities (see Appendix A for the computation process).

Figure 3.10 shows that the two methods obtain the "same" results. The second approach is used in our system.

| c6 | c1 | C2 | c3 | c4 | c5 | m1 | m2 | m3 | m4 |
|-------|------|------|-------|-------|-------|------|------|------|------|
| 0.67 | 1.53 | 1.20 | 0.96 | 1.31 | 2.02 | 0.41 | 0.31 | 0.20 | 0.20 |
| -0.19 | 0.29 | 0.12 | -0.35 | -0.34 | -0.51 | 1.28 | 1.37 | 0.54 | 1.54 |



| c6~c1 | c6~c2 | c6~c3 | c6~c4 | c6~c5 | c6~m1 | c6~m2 | c6~m3 | c6~m4 |
|--------|--------|--------|--------|--------|--------|---------|--------|---------|
| 0.8964 | 0.9337 | 0.9973 | 0.9999 | 0.9998 | 0.0420 | -0.0504 | 0.0795 | -0.1422 |

Figure 3.10 Results of the document vectors and the similarities

3.3 Collaborative Filtering Algorithms

Rather than recommending items similar to those that proved satisfactory before, collaborative filtering algorithms [4,5] tries to recommend items favored by other users with similar tastes. In other words, collaborative filtering uses the historical preferences of a community to predict how well a user will favor an item that he/she has not rated.

There are two ways for users to express their opinions: explicitly or implicitly. Explicit measures are usually a set of numerical ratings. For example, the *GroupLens* system [12,13] uses numbers from 1 to 5 to rate Usenet news, and users can click on one of the five rating buttons on the browser to express opinions after they read a news item.

Implicit ratings are commonly generated by analyzing user behavior (e.g., reading or ignoring, reading time, and mouse actions). Two approaches are usually used in text filtering. One is to map “reading or ignoring” into explicit ratings with rate “1” representing “reading” and “0” representing “ignoring,” the other is to map the time spent on reading into explicit ratings [21, 22]. In our system, we use the implicit ratings with “reading” as 1 and “ignoring” as 0.

The only difference between the filtering systems using explicit and implicit feedback is the way they collect ratings. Otherwise, they can use the same collaborative technologies to provide predictions. Because of this, common collaborative filtering techniques are introduced before a description of how we use implicit ratings in this algorithm.

3.3.1 User-by-Term Representation

A collaborative filtering system collects ratings from the whole community and uses them to make predictions. Generally the rating collection can be represented as a user-by-item matrix in which each row stands for a user, each column stands for an item and each entry represents the rating of an item given by a user. Figure 3.11 shows an example. In this case, we have 3 users and 4 items. The first two users have already rated all items, while Ben has not rated item D. Here, the problem is how to use the existing ratings to predict Ben’s opinion for item D.

| | Item A | Item B | Item C | Item D |
|------|--------|--------|--------|--------|
| Eric | 5 | 2 | 4 | 3 |
| Jim | 5 | 1 | 4 | 3 |
| Ben | 5 | 2 | 5 | ? |

Figure 3.11 User-by-item representation of a simple collaborative filtering model

In our approach, the items are news articles, and the ratings are either “1” or “0.” An example of user-by-news matrices in our case is shown in Figure 3.12.

| | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | ... |
|--|----|----|----|----|----|----|----|----|-----|
|--|----|----|----|----|----|----|----|----|-----|

| | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|-----|
| User1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | ... |
| User2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| User3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | ... |

Figure 3.12 Example of a user-by-news matrix

Because the IDs of news articles that have ever been rated can vary from 0 to 2,147,483,647 (in our database, we use a variable of INT type to represent the news IDs), the sparsity problem can be very serious. Because it is very likely that different users read different news items, sometimes it could be difficult to form a neighborhood for a user. Furthermore, if user A and user B have read different news articles, they will never be considered to be neighbors even if what they have read is very similar.

To solve such problems, we use the words instead of articles as different items and the occurrence frequency of each term in the articles read by each user as the rating. This idea is borrowed from the Fab system [7]. Because the number of common words is about 30000, which is much less than the max of INT (2,147,483,647), the user-by-term matrix is then much smaller and less sparse (see Figure 3.13).

| | T1 | T 2 | T 3 | T4 | T5 | T6 | T7 | T8 | ... |
|-------|----|-----|-----|----|----|----|----|----|-----|
| User1 | 3 | 1 | 1 | 0 | 7 | 23 | 12 | 1 | ... |
| User2 | 5 | 0 | 2 | 4 | 2 | 1 | 11 | 1 | ... |
| User3 | 4 | 3 | 1 | 5 | 3 | 0 | 0 | 1 | ... |

Figure 3.13 Example of a user-by-term matrix in our approach

3.3.2 Weighting Schemas

As mentioned in section 3.3.1, the rating collection can be represented as a user-by-term matrix. Thus, we can also use the same term-weighting schemas introduced earlier to weigh terms in user profiles. The weighting schema used here is a combination of the local weighting function “Augment-Normalize,” the global function “Tfidf,” and the normalization function “Cosine.” Formula 3-6 shows its definition.

$$w_{ij} = \frac{(0.5 + 0.5 * (tf_{ij} / \max_tf)) * \log(num_doc / df_j)}{\sqrt{\sum_j [(0.5 + 0.5 * (tf_{ij} / \max_tf)) * \log(num_doc / df_j)]^2}} \quad (3-6)$$

where:

tf_{ij} = the frequency of term T_j in document D_i .

df_j = the number of documents in which term T_j ($1 \leq j \leq N$) appears

max_tf = the maximum tf in document D_i .

$num_docs = 2$ (two user profiles)

Here, the “Augment-Normalize” local function is chosen so that users who read more articles are not unfairly given more weight. The factor $\log(num_doc / df_j)$ is the inverse document frequency, which is used to normalize the term weights so that a term occurring in many user profiles will be assigned a smaller weight.

3.3.3 Neighborhood-Based Algorithms

Given the weighted rating collection of a user community, we can use collaborative filtering to make recommendations. The most popular method used in collaborative filtering is the neighborhood-based algorithm [4]. In this method, a neighborhood is a subset of users whose tastes are similar to those of the active user. The prediction for any item is computed as the weighted combination of the neighbors’ ratings for this item. To be specific, the neighborhood-based method involves three steps:

1. compute the similarities between the active user and the others;
2. select a subset of users as the neighborhood for this active user;
3. generate the prediction based on neighbors’ opinions.

Similarity weighting

The simplest method in similarity weighting is to compute the Pearson correlation coefficient. Different variations are developed based on this method. The formula 3-7 is the definition for calculating the correlation coefficient, where $w_{a,u}$ is the similarity between the active user A and any neighbor U , $r_{u,i} / r_{a,i}$ represents user U ’s/ A ’s rating for item i , and \bar{r}_u / \bar{r}_a and σ_u / σ_a are the mean and variance of all ratings rated by user U/A respectively.

$$w_{a,u} = \frac{\sum_{i=1}^m (r_{a,i} - \bar{r}_a) * (r_{u,i} - \bar{r}_u)}{\sigma_a * \sigma_u} \quad (3-7)$$

Another important method is adopted from information retrieval techniques – vector similarity: the preferences of a user can be considered as a rating vector, and the similarity between any two users can be computed as the cosine of their vectors. The definition is

$$w_{a,u} = \sum_i \frac{r_{a,i}}{\sqrt{\sum_{k \in I_a} r_{a,k}^2}} \frac{r_{u,i}}{\sqrt{\sum_{k \in I_u} r_{u,k}^2}} \quad (3-8)$$

where the squared terms in the denominator serve to normalize ratings so that users who rated more items will not be more similar than others.

In our approach, because the rating history of each user is a weighted term-frequency vector, it is easy to use the vector similarity method to evaluate similarities.

Selecting Neighborhoods

Correlation-thresholding and best-n-neighbors are two methods usually used to select neighbors. Correlation-thresholding selects only the neighbors whose similarities to the active user are higher than a given threshold. Setting a threshold can be a difficult issue. If we set a high threshold, then only the most similar neighbors can be selected, which is favorable for prediction accuracy. The obvious downside is that high threshold will lead to a small neighborhood, which could result in less or no prediction coverage for some items. On the other hand, a low threshold will bring in more noise, decreasing the prediction veracity.

The best-n-neighbors method selects the best n neighbors with the highest similarities. The advantage of this technique is that the users' similarities will not affect the prediction coverage. However, picking a larger n will also bring in more noise for those users who have high correlates, while picking a smaller n will cause the loss of prediction coverage.

We use the correlation-thresholding method in our approach with the threshold equaling 0.5. That is because we use LSI as the primary algorithm to generate predictions, and only use the collaborative algorithm as a complement. We hope to improve accuracy by combining this collaborative algorithm. The prediction coverage is less important.

Computing Predictions

The prediction for the active user can be calculated as the weighted combination of the ratings of all neighbors. Usually similarities are used as the weights. The simplest method is to compute a weighted average of these ratings, as used in *Ringo* [23]. This method assumes that all ratings from different users have the same distribution, and its definition is in formula 3-9. This method is also used in our approach but in an accumulative way (see section 3.4 for details).

$$P_{a,i} = \frac{\sum_{u=1}^n r_{u,i} * w_{a,u}}{\sum_{u=1}^n w_{a,u}} \quad (3-9)$$

3.4 Combining LSI and Neighborhood-based Algorithms

To make better recommendations, we integrate the content-based and collaborative filtering so that they complement to one another and help overcome each other's shortcomings. Here, we use LSI as the primary algorithm to generate predictions, and only use collaborative algorithm as the complement. The prediction for any new item consists of two parts:

$$prediction = P_{LSI} + P_{Collaborative} \quad (3-10)$$

The LSI algorithm is employed to compute the prediction (P_{LSI}) for each incoming news item and for each user. When a user selects a news item to read, which implies that this user likes it, the collaborative algorithm will be activated to add the collaborative filtering prediction for all other users with similar tastes.

For example, suppose user X selected article D . Therefore, for each user U whose similarity to X is larger than 0.5, the collaborative algorithm will increase D 's prediction for U . The increment is $w_{x,u} * C$ where $w_{x,u}$ is the similarity between X and U , and C is described in formula 3-11.

$$C = \delta * \frac{1}{\sum_i w_{u,i}} \quad (3-11)$$

where $\sum_i w_{u,i}$ represents the sum of similarities between U and U 's neighbors, and δ is the weighting factor that controls the contribution of the collaborative filter. By multiplying the factor δ , we limit the increment to being no larger than δ .

On the other hand, suppose before user A read document D , user B , C and D have already read it. Thus, the prediction of D for user A is:

$$\begin{aligned} prediction &= P_{LSI} + P_{B \rightarrow A} + P_{C \rightarrow A} + P_{D \rightarrow A} \\ &= P_{LSI} + \delta * \frac{w_{a,b}}{\sum_i w_{a,i}} + \delta * \frac{w_{a,c}}{\sum_i w_{a,i}} + \delta * \frac{w_{a,d}}{\sum_i w_{a,i}} \\ &= P_{LSI} + \delta * \frac{\sum_{k \in \{b,c,d\}} w_{a,k}}{\sum_i w_{a,i}} \end{aligned}$$

where $\sum_i w_{a,i}$ represents the sum of similarities between A and all A 's neighbors;

$\sum_{k \in \{b, c, d\}} w_{a, k}$ represents the sum of similarities between A and B, C, D .

In conclusion, the prediction for document i for any active user A is:

$$P_{a,i} = P_{LSI} + \delta * \frac{\sum_{u=1}^n (r_{u,i} * w_{a,u})}{\sum_{u=1}^n w_{a,u}} \quad (3-12)$$

where P_{LSI} is the prediction generated by the LSI algorithm;

n is the number of A 's neighbors;

$$r_{u,i} = \begin{cases} 1 & \text{if neighbor } U \text{ has read document } i; \\ 0 & \text{if neighbor } U \text{ has not read it;} \end{cases}$$

and $w_{a,u}$ is the similarity between A and U .

3.5 Arbitration Algorithms

Both the LSI content-based filtering algorithm and the collaborative algorithm focus on filtering information for a single user. However, in shared environments such as the smart room, more than one person could use the same display wall. In such cases, an arbitrator is needed to adjust the recommendation of news items to best accommodate the preferences of all active users in the room.

To make the best decision, the arbitrator must have the following information about the current group: 1) current active users; 2) the active users using their own display device to access their news agents; and 3) their individual interests. To obtain the above information, the arbitrator does the following:

- Use RF/ID technology to detect the active users wearing badges.
- If a user is using a laptop or PC to access his/her news agent, the agent will inform the database that this user is currently online. Thus, the arbitrator can check the database to identify those users who are using their own private display devices.
- A prediction table is maintained in the database to provide estimations about how much each user will like each news item. Thus, the arbitrator can access the database for the users' preferences.

Finally, the arbitrator can use the arbitration algorithm to generate recommendations shown on the display wall shared by a group of users.

The arbitration algorithm takes as input the $M*N$ prediction table where M is the number of the unread news in the database, and N is the number of current users in the smart room. At the beginning of each time slot, the arbitrator decides how to recommend both shared news for the whole group and private news for each individual who does not have a private device. To select shared news, the arbitrator will compute the sum of the predictions for all N users as the group preference for each news item. The news with the highest preferences will be chosen.

Similarly, when selecting private news for each individual, the arbitrator will pick news with the highest predictions for this user if the news item has not already been selected as a shared one. Now, the question is how many news items should be recommended for the whole group and how many news items should be selected for each individual. To solve this problem, we use an adaptive algorithm.

To easily describe this algorithm, we use the parameters in Table 3.1. All these parameters except P_s , $Shared_News_Items$, P_i , $Private_News_Items[i]$ can be configured by users before or during use of the arbitrator. The last four parameters are used by the arbitrator to maintain the adaptive information.

Table 3.1 Parameters used in the arbitration algorithm

| Parameter Name | Description |
|----------------------------|--|
| Time_Slot | the time interval during which the new items selected at the beginning of this slot can be displayed |
| <i>User_Num</i> | the number of the active users inside the smart room |
| <i>User_Num_wo</i> | the number of the users without private display device |
| <i>Max_Items</i> | the number of news items that can be displayed in one time slot |
| <i>Serendipitous_Items</i> | the number of items that are serendipities |
| <i>Init_Shared_Portion</i> | the initial share of news items that are shared ones |
| <i>Min_Shared_Portion</i> | the minimum share of news items that are shared ones |
| <i>Max_Shared_Portion</i> | the maximum share of news items that are shared ones |
| α, β, δ | the relaxation coefficients |
| P_s | the adaptive share of the shared news |

| | |
|-------------------------------|---|
| Shared_News_Items | the number of the shared news recommended in this slot |
| P_i | the adaptive share of the private news for user I |
| Private_News_Items [i] | the number of the private news recommended for user I |

3.5.1 Initialization

When the arbitrator is invoked by a group, it will initialize the following parameters:

$$P_s = \text{Init_Shared_Portion}, \quad \text{Shared_News_Items} = \text{Max_Items} * P_s$$

$$P_i = (1 - P_s) / \text{User_Num_wo}, \quad \text{Private_News_Items}[i] = \text{Max_Items} * P_i$$

Here, the proportion of the shared news is initialized as *Init_Shared_Portion*, and the remaining is equally shared by users without private display devices.

3.5.2 Self Adaptation

The arbitrator adapts itself with a period of *Time_Slot*. We call this period a time slot. At the beginning of each time slot, the arbitrator checks the database to find how many users are present (*User_Num*), and how many users are not using private devices (*User_Num_wo*). After that, it will adjust itself in the following way:

1. If *User_Num_wo* equals to zero, there is no need to recommend private news. Thus, $P_s = 1$ and $\text{Shared_News_Num} = \text{Max_Items}$. Go to step 4.
2. Otherwise, suppose the number of shared news items selected to read in the previous slot is N_s , and the number of private news read by user i during the previous slot is N_i where $1 \leq i \leq \text{User_Num_wo}$. If nothing has been selected during last time slot (i.e. $N_s + \sum N_i = 0$), we do not need to change the shares. Go to step 4.
3. Otherwise:

If more shared news items are selected (i.e. $\frac{N_s}{N_s + \sum N_i} > P_s$), the share for this part will be increased as $P_s = \min(P_s + \alpha * (\frac{N_s}{N_s + \sum N_i} - P_s), \text{Max_Shared_Portion})$; or if fewer shared news are selected (i.e. $\frac{N_s}{N_s + \sum N_i} < P_s$), the share for this part will be reduced as $P_s = \max(P_s - \beta * (P_s - \frac{\sum N_i}{N_s + \sum N_i}), \text{Min_Shared_Portion})$; where $0 < \alpha, \beta < 1$ are the relaxation coefficients. We can easily see that P_s is between $\text{Min_Shared_Portion}$ and $\text{Max_Shared_Portion}$.

We can use a similar method to compute the private share for each user. Here, all users without private devices are divided into three groups with N_1, N_2 and N_3 representing the group size respectively ($N_1 + N_2 + N_3 = \text{User_Num_wo}$). If this user is a new comer,

$$P_i = \frac{(1 - P_s)}{\text{User_Num_wo}} ; \text{ or if } \frac{N_i}{N_s + \sum N_i} < P_i , P_i = \delta * \frac{(1 - P_s)}{\text{User_Num_wo}} ; \text{ or if } \frac{N_i}{N_s + \sum N_i} \geq P_i , P_i = \frac{(1 - P_s)}{\text{User_Num_wo}} * (1 + N_1 * (1 - \delta) / N_3) \text{ where } 0 < \delta < 1 \text{ is a}$$

decay coefficient. Go to step 4.

4. Pick up *Shared_News_Items*s news items with the highest group preference for the whole group and *Private_News_Items*[i] best news articles for each individual without private device.

This arbitrator keeps looping until it is stopped.

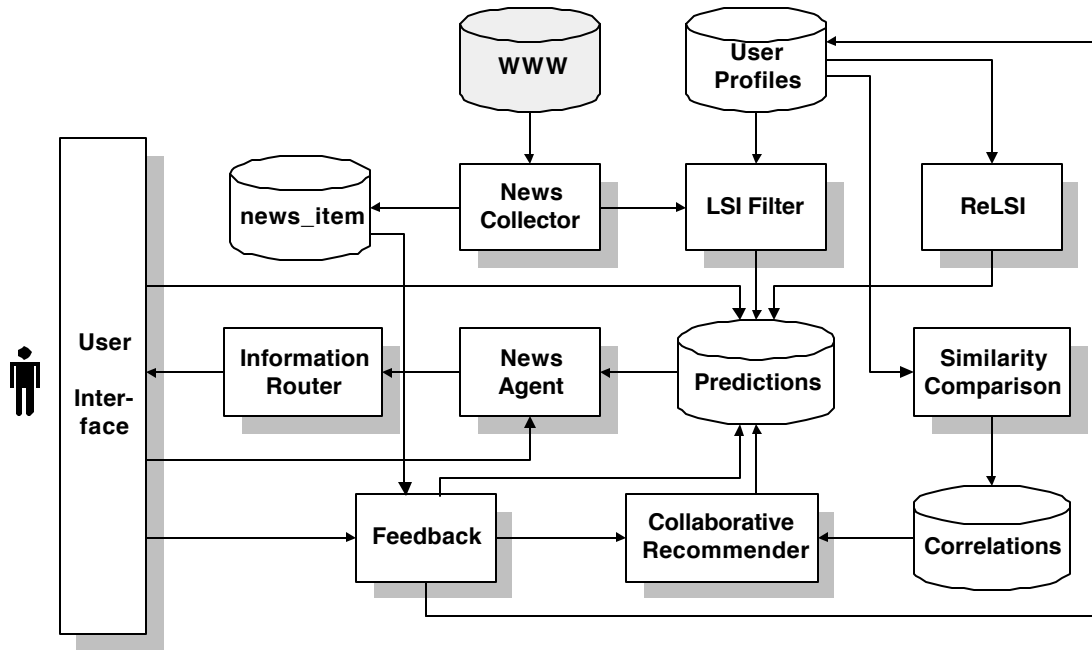
In Chapter 4, we start to detail the implementation of our system based on the algorithms described in this chapter.

CHAPTER 4

IMPLEMENTATION

4.1 Overview

We now discuss our personalized/group news filtering system based on the algorithms described in Chapter 3. This system is composed of the following modules (see Figure 4.1) that encapsulate distinct functions.



- “News Collector” is the module collecting the latest news from on-line newspapers and inserting it into the table “news_item” in the database. It also obtains the text body of each news article and passes it to “LSI Filter.”
- “LSI Filter” uses the LSI algorithm (section 3.2) to generate predictions for each registered user by comparing incoming news stories with user profiles. All resulting predictions will be inserted into the table “prediction” in the database.
- Module “Similarity Comparison” compares the similarities between any two users using the vector similarity method (section 3.3.2). The results are inserted into the table “correlations” in the database.

- The previous three modules work in the background, while the following modules only work once users activate them.
- The user interface module includes several parts: a registration window, an initial preference configuration page, and a tickertape window (see section 2.4.3). Once a user has registered and configured his/her initial preferences, he/she can use the tickertape to display the recommended news.
- “News Agent” is the module in charge of recommending news to different users. Whenever a registered user opens a tickertape window, the tickertape will tell this module personal information about this user. Thus, a customized agent will be forked to recommend news for this user.
- The “Information Router” module uses Elvin (see section 2.4.3) to route news messages generated by the customized news agents to the corresponding tickertapes (or the users).
- The “Feedback” Module waits for users’ implicit feedback: if a user chooses to read a news article, the article will be attached to the user profile and the corresponding entry in the “prediction” table will be deleted. Finally, “Feedback” will invoke module “Collaborative Recommender” to recommend this article to the user’s neighbors.
- Module “Collaborative Filtering” uses the neighborhood-based algorithm (see section 3.3.3) to adjust the predictions generated by “LSI Filter” based on the associations among all registered users.
- The “ReLSI” module will redo the LSI filtering for all unread news for the user once he/she finishes using this recommendation system and changes the user profile.

The above highlights will be explained in detail in the following sections.

4.2 Database Tables

Here, we list the database tables used in our filtering system in Table 4.1.

Table 4.1 Tables used in our filtering system

| Name | Entry Format |
|--------------|---|
| news_item | (news_id, storyid, section, title, url, received) |
| news_section | (section_id, name) |
| user | (user_id, name, username, sections, keywords, online, redo) |

| | |
|--------------|--|
| user_favor | (user_id, section_id, favor) |
| prediction | (user_id, news_id, section_id, prediction, others) |
| User_user_SI | (user1_id, user2_id, similarity) |

- To keep attributes for each news item, the “news_item” table is maintained in the database. It stores information of story IDs, news sections, news titles, news URLs, and the received time. The “news ID” field is the primary key for this table, which is increased automatically every time a new entry is added.
- The table “news_section” simply records the mapping between the section IDs and section names.
- The table “user” keeps the mapping between user IDs and usernames as well as initial user preferences (sections and keywords) and user status (e.g., “online” or “redo”).
- The table “user_favor” records the current feeling of each user for each section.
- The table “prediction” maintains the current predictions for each news item for each registered user and the historical prediction changes (see section 4.10).
- The table “user_user_SI” keeps the similarities between any two users.

4.3 News Collector

The task of this module is to import the latest news from the web site at http://wire.ap.org/APticker/remote_data. The news collector first gets the content of this web page, parses each record, and fetches all useful characteristics of each news item. Every record in this web page is organized like the following example:

br/fhttp://wire.ap.org/APnews/?FRONTID=BUSINESS&STORYID=APIS7BACNK80/Wednesday's Most Active Stocks~

After splitting it, we can get the following attributes of this news item:

- News section = BUSINESS
- Story ID = APIS7BACNK80
- Title = “Wednesday's Most Active Stocks”
- URL = <http://wire.ap.org/APnews/?FRONTID=BUSINESS&STORYID=APIS7BACNK80>

Every news record found on the web page of http://wire.ap.org/APticker/remote_data will be compared with all existing news in the “news_item” table to see if it is a new one. If it is, we add

a new row to this table and insert all parsed attributes about this news item into each cell. We still need to verify that the news section of this news item is new. If so, the name of this new section is inserted into the table “news_section.”

Finally, using the URL of this news, “News Collector” can read the content of this story, and pass it to module “LSI Filter” so that predictions for this news article can be computed there for all registered users.

4.4 LSI Filter

“LSI filter” uses the LSI algorithm (section 3.2) to generate predictions for each registered user by comparing the content of the news story sent by “News Collector” with each user profile. A user profile is a collection of news articles which have been read by the current user recently. Users themselves can configure the size of their profiles to be large enough to represent their recent interests. To compare the content of the incoming news with the user profile, we need to pre-process it into a term-by-document matrix, and then use SVD to reveal the underlying associations between the news and all documents in the user profile (see Figure 4.2 for flowchart).

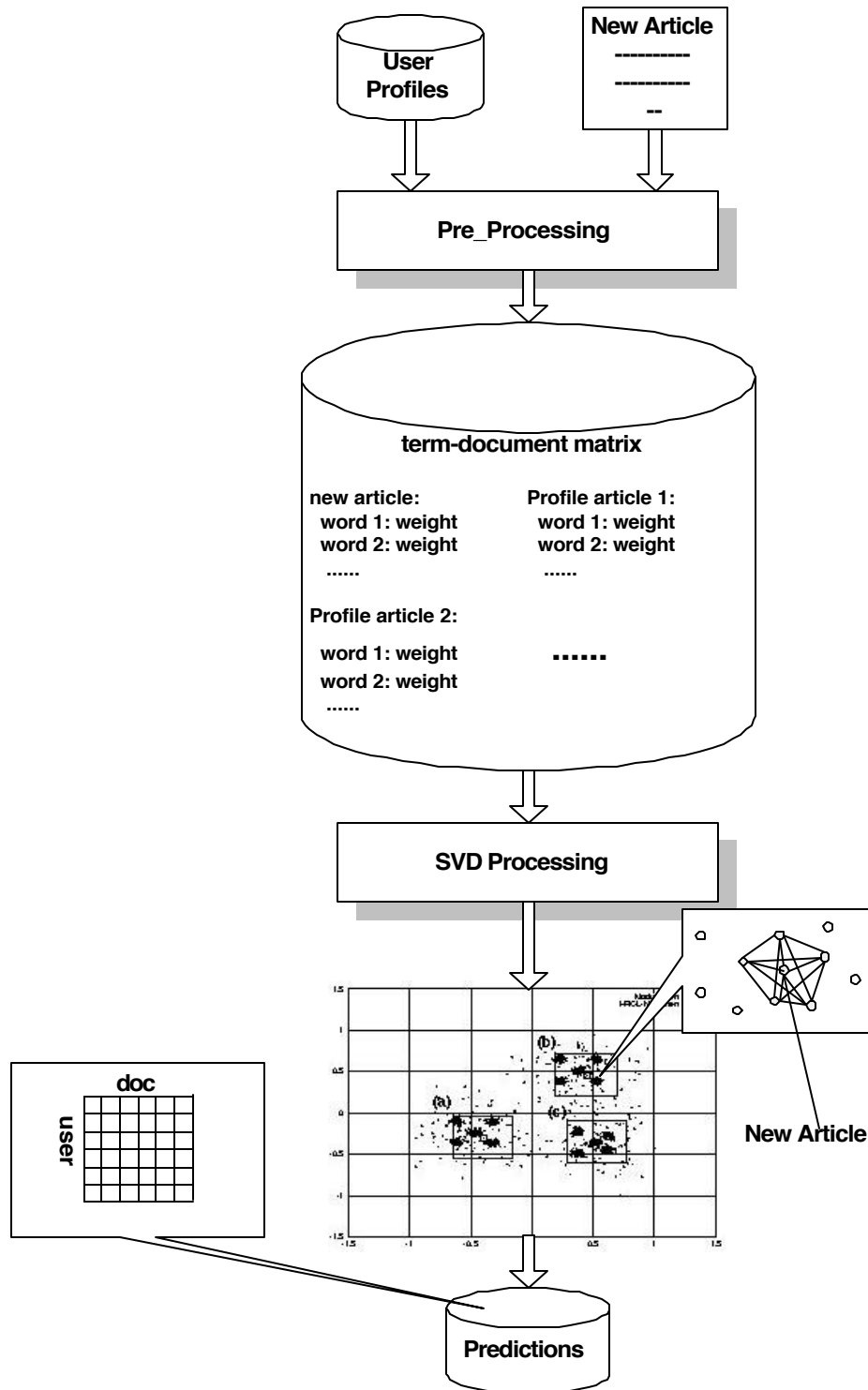


Figure 4.2 Flowchart of LSI Filter

4.4.1 Pre-Processing Module

Here we use the SMART system [8,9] (from the SMART research group at Cornell University) to pre-process the user profile and the incoming news to generate the term-by-document matrix. Given the locations of all documents and their formats, SMART will first convert them into a standard collection. All upper case characters are changed into lower case ones; all markups (such as delimiters and punctuations) are removed and only white spaces are used to separate terms. Finally, all common (useless) words are removed from the text.

SMART has a stemmer that can extract the stems of different words so that different forms of the same word (e.g. fact, facts, factual) will not be considered as different words in counting their frequency. To strip word endings, SMART maintains an exhaustive list of suffixes and tries to remove the longest possible one. Then, SMART will map each token (stem) into a “concept number” using a hash function, and the resulted hash table is called “dictionary.” Each document – a list of tokens will be matched against the dictionary, and then whole document collections can be represented as a term-by-document matrix, where each entry is the frequency at which a term occurs in a document.

SMART also offers a multitude of term-weighting schemas by using different combinations of weighting functions. Because all weighting functions can be divided into three groups (local, global, and normalization functions), SMART uses three character codes to specify a combination, each of which identifies a specific weighting function from each group. For example, the most useful weighting method “tf×idf” (which is also used in our project) can be represented as “*ltc*,” in which *l* stands for “Log” local function, *t* represents “Tfidf” global function, and *c* means “Cosine” normalized function. Its definition is

$$w_{ij} = \frac{(\log(tf_{ij}) + 1.0) * \log(num_doc / df_j)}{\sqrt{\sum_j [(\log(tf_{ij}) + 1.0) * \log(num_doc / df_j)]^2}} \quad (4-1)$$

where:

tf_{ij} = the frequency of term T_j in document D_i .

df_j = the number of documents in which term T_j ($1 \leq j \leq N$) appears

num_docs = the collection size

4.4.2 SVD Processing

Once the weighted term-by-document matrix is passed to the SVD sub-module [10,15], the matrix will be re-constructed into the best dimension-reduced approximation (see section 3.2.3). In this dimension-reduced space, each term and document can be represented as a rank- k vector.

The positions of term vectors reflect the correlations in their usages across all documents. For example, terms used in similar contexts should be close to each other (or have similar directions). Similarly, the locations of document vectors tell the correlations in their term usages. Thus, if an incoming news article is close to most documents in the user profile, we say that the chance that this user likes this news is large.

Suppose the singular value matrix is S and the left and right singular matrices are U and V respectively. We use rows of the matrix VS as coordinates for the corresponding documents, and the cosine values between the incoming news vector and each profile document vector as the similarities. We tried the following methods to generate predictions using these estimated similarities:

- Use the maximum similarity as the prediction: this approach is simple, but it has an obvious problem that can be seen from the following example. If the user has no interest in basketball, but accidentally tried an article A about it, one can imagine that all news items about basketball similar to article A would be recommended to this user after that. Because all basketball articles are very similar in a semantic sense, the user will be bothered by all these recommendations.
- Use the average of the similarities as the prediction: this method can overcome the shortcoming of the first one, but also has apparent drawbacks. If a news section is so unique that all news items within this section are very similar to each other, but are totally different from news in other sections, the prediction for all news within this section could be very low under some situations. For example, if one user likes the news from several news sections equally (generally speaking, M sections), and the similarities between any pair of news items in this unique section is 1, while the similarities between them and the news from the other sections are 0, then the prediction of any news of this section will not be larger than $1/M$.
- Another method is to use the average of the largest N similarities as the prediction (We set N as 10 (or 15) in our project). By using this method, we can overcome the

shortcomings of the first two methods. Hence, we choose this method to give the prediction.

4.4.3 Initial Prediction

At the beginning, user profiles are empty; the users have not read any articles. To do filtering for these users, we need to use initial configurations given by them (see section 4.6.1 for editing user preferences). Such information is stored in the table “user” in the database. Thus, if a news item belongs to one of the news sections chosen by the user, we will set the prediction as 0.5; otherwise, we may set it as 0.2. Furthermore, if the keywords specified by this user appear in the title of this news, we will increase the prediction by 0.3.

However, if a user has not said anything about his/her preferences, we will only give 0.2 as the prediction for every news item, which means that the recommendation for this user is only a random selection.

4.5 Similarity Comparison

The module “Similarity Comparison” in Figure 4.1 compares the similarities between any two users. In this case, all articles in the user profile are considered a single document, and the similarity between two users is the similarity between two profile documents. To estimate the similarity between any two user profiles, we also used the SMART system to convert them into two weighted vectors (see section 4.4.1) by applying the ‘*atc*’ weighting method defined as:

$$w_{ij} = \frac{(0.5 + 0.5 * (tf_{ij} / \max_tf)) * \log(num_doc / df_j)}{\sqrt{\sum_j [(0.5 + 0.5 * (tf_{ij} / \max_tf)) * \log(num_doc / df_j)]^2}} \quad (4-2)$$

where:

tf_{ij} = the frequency of term T_j in document D_i .

df_j = the number of documents in which term T_j ($1 \leq j \leq N$) appears

\max_tf = the maximum tf in document D_i .

$num_docs = 2$ (two user profiles)

We then compute the cosine of these two weighted vectors as the user similarity using formula 4-3.

$$\text{Similarity} = \sum_i \frac{v_{a,i}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{b,i}}{\sqrt{\sum_{k \in I_b} v_{b,k}^2}} \quad (4-3)$$

where:

$v_{a,i}$ and $v_{b,i}$ are the elements of the two document vectors respectively.

Finally the results will be inserted into the table “user_user_SI” in the database.

4.6 User Interface

The user interface to our news filtering system allows users to register, configure their preferences, browse the titles of recommended news, and select and read news stories. It has two separate parts: one is the registration/configuration window and the other is the display window – the tickertape.

4.6.1 Registration Window

The registration window consists of web pages accessible through a Web browser run by the users on their workstations or PCs (<http://aminor.cs.uiuc.edu/smart/news/web-prefs.cgi> and all links in this page). Users must register their username and full name (see Figure 4.3) the first time they use news agents. By clicking on “Add” button, a new user can add himself/herself into the system.

Upon the first login, users may also edit their preferences following the appropriate links. Within the preference page, users can choose which news sections they like by marking the corresponding checkboxes. In addition, users can also specify keywords in which they are interested (see Figure 4.4). The username and all initial preferences of a new user will be inserted into the table “user” in the database, and a user identity will be automatically assigned for this user.

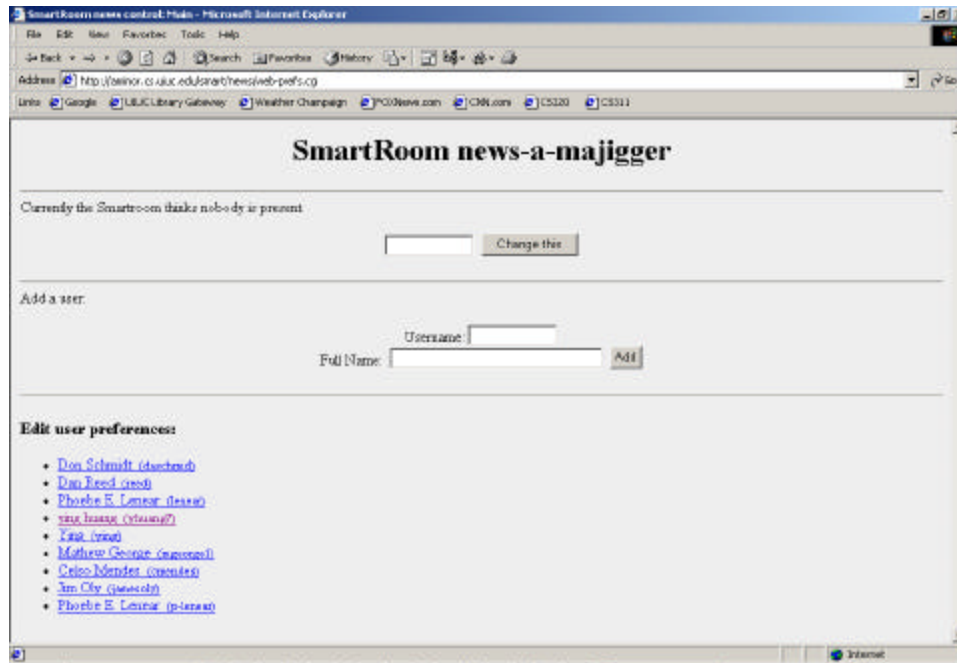


Figure 4.3 Registration window

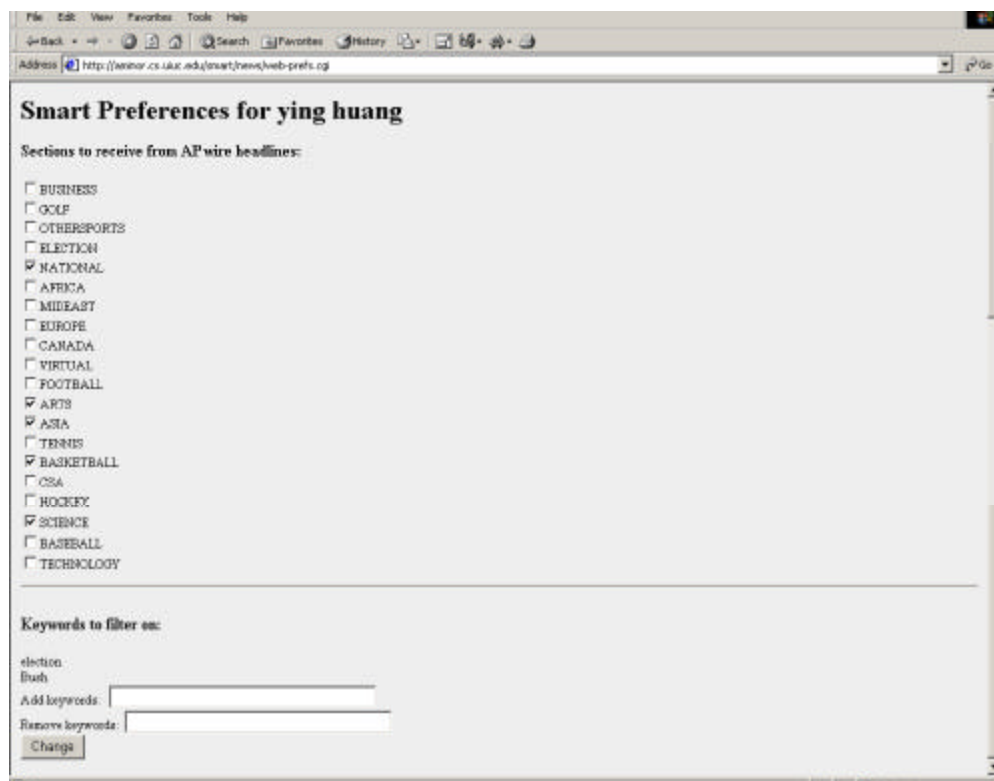


Figure 4.4 Editing the preference for user Ying Huang

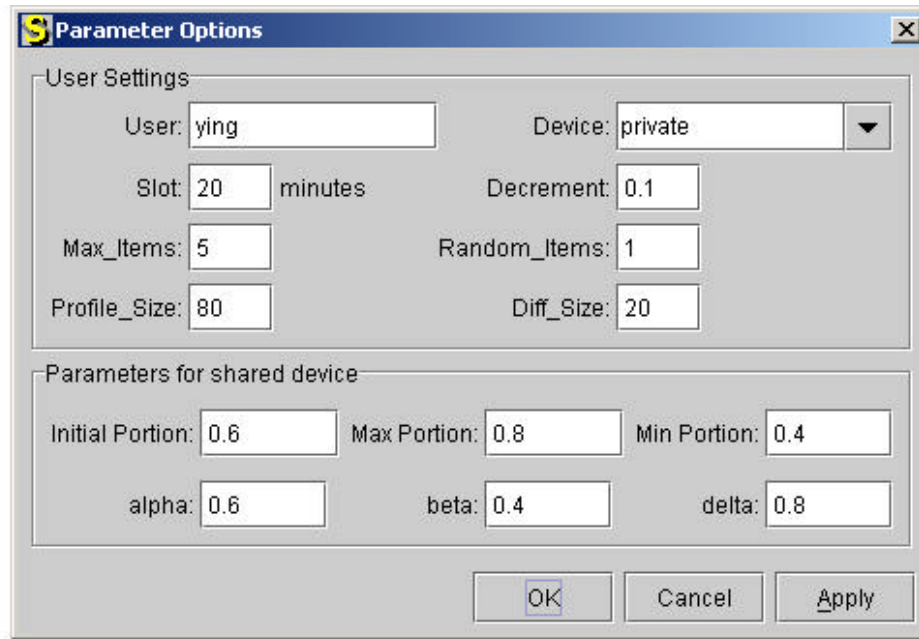
Using the usernames of all registered users, “LSI Filter” will be able to predict the incoming news item for each of them. Before receiving any user feedback, initial configurations will be used to generate predictions (see section 4.4.3).

4.6.2 Tickertape – The Display Window

We use tickertape windows to display the recommended news titles (with mimes). To use the tickertape in our system, we modified it to communicate with the “News Agent” and “Feedback” modules. Furthermore, we added one more configuration dialog containing the username and other performance parameters including the *slot* (the time interval during which messages scroll before fading away), *max_items* (the maximum number of news items recommended in each slot), *random_items* (the number of serendipitous items chosen in each slot), etc (see Figure 4.5). Therefore, the tickertape knows the characteristics of its owner, and can provide the news agent with these identification parameters. Thus, the news agent will be able to recommend news for this user. Once the user selects a news item, the tickertape will be able to inform the “Feedback” module that this user has read this article.

Later, we added a user preference dialog to the tickertape. It enables users to indicate their interest changes directly (see Figure 4.6). User can edit their preferences via this window and the preference values will be recorded in the table “user_favor”. Once users finish editing their preferences, predictions in the table “prediction” will be modified based on these preference changes. For example, if a user alters the preference value of the “Asia” section from 0.1 to 0.2, all predictions for news in section “Asia” will be increased by $(0.2 - 0.1) = 0.1$.

Thus, if a user dislikes news from some news sections, he/she can set smaller preference values to these sections so that predictions of articles in these sections will be decreased. Or if the user is especially in need of news from some sections, he/she can give larger values to increase predictions of news within those news sections.



Parameter Options

User Settings

User: Device:

Slot: minutes Decrement:

Max_Items: Random_Items:

Profile_Size: Diff_Size:

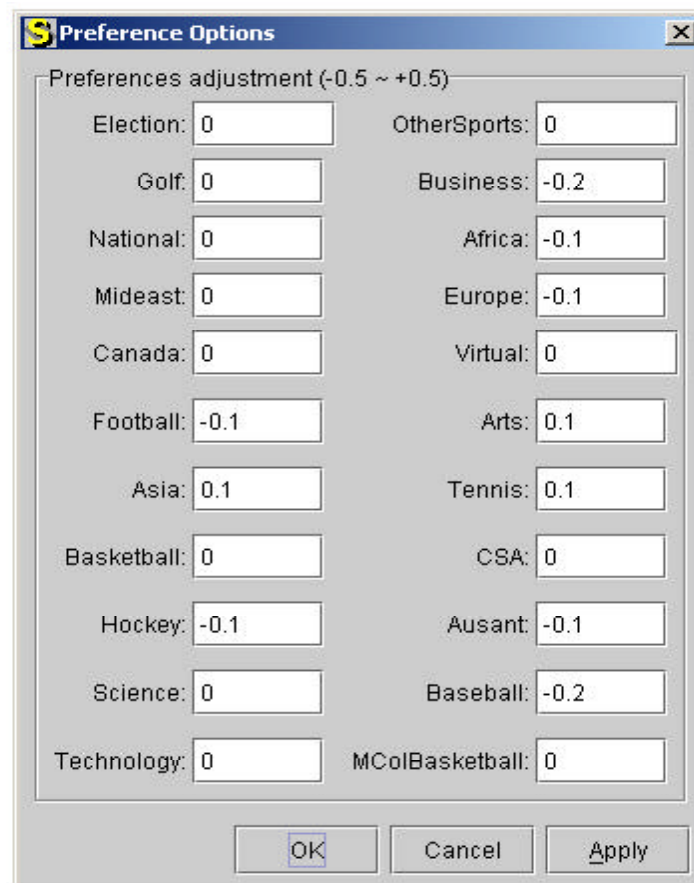
Parameters for shared device

Initial Portion: Max Portion: Min Portion:

alpha: beta: delta:

OK Cancel Apply

Figure 4.5 Configuration of parameters



Preference Options

Preferences adjustment (-0.5 ~ +0.5)

| | |
|---|--|
| Election: <input type="text" value="0"/> | OtherSports: <input type="text" value="0"/> |
| Golf: <input type="text" value="0"/> | Business: <input type="text" value="-0.2"/> |
| National: <input type="text" value="0"/> | Africa: <input type="text" value="-0.1"/> |
| Mideast: <input type="text" value="0"/> | Europe: <input type="text" value="-0.1"/> |
| Canada: <input type="text" value="0"/> | Virtual: <input type="text" value="0"/> |
| Football: <input type="text" value="-0.1"/> | Arts: <input type="text" value="0.1"/> |
| Asia: <input type="text" value="0.1"/> | Tennis: <input type="text" value="0.1"/> |
| Basketball: <input type="text" value="0"/> | CSA: <input type="text" value="0"/> |
| Hockey: <input type="text" value="-0.1"/> | Ausant: <input type="text" value="-0.1"/> |
| Science: <input type="text" value="0"/> | Baseball: <input type="text" value="-0.2"/> |
| Technology: <input type="text" value="0"/> | MColBasketball: <input type="text" value="0"/> |

OK Cancel Apply

Figure 4.6
preference

4.7 News

The “News
the server in

User
dialog

Agent

Agent” is
charge of

recommending news to different users. When a user (or a group of users) wants to browse some

news, what he/she needs to do is to open a tickertape window. The tickertape, a client of the news agent server, will establish a TCP connection to the server (the server IP address and port have already been configured in the tickertape). The main server process acts as a receptionist: it blocks itself on *accept()* and waits for new connections from clients. When a connection request comes in, the main server process spawns a child process and goes back to *accept()*. The newly created child process meanwhile has a copy of its parent's environment and shares all open file descriptors. Hence it is able to read from, and write to the new socket returned by *accept()*. Therefore, the child process can communicate with the tickertape through this new socket to receive all personal information (*username*, *slot*, *max_items*, *random_items* and other parameters). After receiving all the necessary parameters, this customized child process will be able to start recommending news.

If this user is a single user, the work of his/her server process is very simple. First, it will mark this user online by updating the “online” field of his/her entry in the “user” table to “Y.” Then, at the beginning of each slot, this server process takes the following steps:

- First, it will query the database for *max_items* news items with the highest predictions for this user;
- Second, it will choose *random_items* serendipitous news items from the news sections from which no news has been selected in step 1;
- Finally, it will send the news messages to the Elvin server using *tick* one by one (see section 2.4.3). The Elvin server will route these messages to the correct user.

If the user is a group of users (the username is a group name), the work of their server process is more complicated. At the beginning of each time slot, the server process executes the following procedures:

- First, it will query the database for all active users inside this group, and check who are currently using private news agents to recommend news for themselves (their “online” fields in table “user” are ‘Y’);
- Second, it will query the database to get the news ids of items read by this group in the previous time slot from the table “news_read,” and determine how many of them are shared news and how many of them are private news recommended for each user. Therefore, it can use the arbitration algorithm (see section 3.5) to calculate how many shared news (N_s) and private news ($N[i]$) should be recommended in this time slot.

- Third, it will retrieve N_s news items with the highest sum of predictions for the whole group as shared news, and it will also select $M[i]$ news items with the highest predictions for each user without private devices as his/her private news. Note that if the news has already been selected as a shared item or for other users, it's not necessary to select it again.
- Fourth, it will also choose *random_items* serendipitous news from news sections other than those from which news items had been selected in the previous steps;
- Last, it will send all recommended news messages to the Elvin server using *tick* one by one (see section 2.4.3). The Elvin server will route these messages to the correct group.

We recommend serendipitous news items because we want to give users a shortcut to change their profiles. By recommending some randomly selected news from different news sections, users will be able to expand their interests to new fields. After they read some randomly recommended articles from these new fields, they will get a chance to receive more recommendations in these areas.

If the user read the recommended news items, the entry for this news item and for this user will be deleted from the “prediction” table. On the other hand, if the recommended news has not been read at the end of the time slot for some reason (the user keeps busy with something else and ignores this news, or dislikes it), the prediction for this news and for this user will be decreased by a small value — the *Reduced_Rate*. Users can set and reset this parameter based on their using habits. If the users did not read this news only because they disliked it, they may set parameter *Reduced_Rate* very high so that the unread news will not be recommended to them any more. Otherwise, users may set the parameter a lower value, and the unread news items still have a chance to be recommended again.

Once the user has decided to shift focus away from the recommendation system, he/she can simply close the tickertape window. The tickertape will inform its server process that the user is going to leave. Thus, this child process will set the “redo” field (in the table “user”) to ‘Y’ for this user so that the “ReLSI” module will be able to detect this information and start to re-compute predictions for all unread news for this user. Finally, if the user is a single user, the child process will also mark this user offline before exiting, otherwise it just simply exits.

4.8 Feedback Module

The “Feedback” Module deals with users’ implicit feedback. To do so, the feedback module needs the user name and the URL of the news read by the user. In our implementation, we have the tickertape send such information to the “Feedback” module. Once a recommended news item has been read, “Feedback” takes the following actions.

If the user is a single user:

1. “Feedback” will query the “user” table to get the user_id of this specific user;
2. It will also check the table “news_item” to get the news_id of the news with the URL;
3. It will delete the entry <news_id, user_id, prediction> from the table “prediction” in the database;
4. It will attach the content of this news article to the user’s profile;
5. Finally, it will inform module “Collaborative Recommender” to spread the influence of this selection to other users with similar tastes.

If the user is a group of users:

1. “Feedback” will first query the database to get the news_id of the news with the specific URL;
2. It will get the user_ids for all users in this group, and it will delete the appropriate entry in the table “prediction” for each user;
3. It will insert this selection information into the table “news_read” by adding a new entry <news_id, username, agent> so that the news agent will be able to tell which news was selected during the previous time slot when it calculates the proportions using the arbitration algorithm (section 3.4);
4. It will attach the content of this news article to all group members’ user profiles;
5. Finally, it will inform module “Collaborative Recommender” to spread the influence of this selection to other users with similar tastes.

4.9 Collaborative Recommender

The “Collaborative Recommender” module uses the neighborhood-based collaborative filtering algorithm to adjust the predictions for similar users (See section 3.4 for details).

4.10 ReLSI Module

To detect users' changes in interests as soon as possible, the "ReLSI" module is used to redo the LSI filtering for all unread news for each user after he/she finished using this recommendation system and changed the user profile.

"ReLSI" is a loop that keeps checking the database to see whether the "redo" field of any user has been set "Y." If it finds one, it will re-compute predictions for the unread news for this user:

1. It will first get the IDs of the remaining news items of this user from the table "prediction";
2. It will use the LSI algorithm to generate prediction for each unread news article. The procedure used here is the same as what we have described in the "LSI filter" module.
3. Note that we still need to maintain the historical prediction changes: the part increased by "collaborative Recommender" because other similar users selected this article, and/or the part decreased by "News Agent" module because this user didn't read this recommended article. Such changes are cumulated in the "others" field in the table "prediction". Moreover, We need to consider the prediction changes made by the user (recorded in the table "user_favor"). Thus, we can use the sum of the new prediction generated by the LSI algorithm, the historical changes, and the preference value indicated by the user as the new prediction for each article.

CHAPTER 5

EXPERIMENTAL RESULTS

A personalized information filtering system must be able to learn and represent current user interests, detect interest changes, and adapt itself over time. Besides, the system must be capable to explore new domains and recommend serendipities. In this chapter, we present the tests on our system. Six real users used the system for 2 weeks evaluating the overall performance including stability and customization. We also used a simulated user to test the system's capability of specialization, adaptation and exploration.

5.1 Statistical Information of News Source

First of all, we describe the statistical characteristics of the incoming news. We observed all news items provided by the Associated Press (http://wire.ap.org/APticker/remote_data) from June 15th to July 14th. These news items are categorized into 17 news sections as shown in Table 5.1. (News items about Africa, Canada and Australia were ignored in our experiments because they rarely appeared.) Every day, we recorded each section's article number and computed the similarity of its articles (self-similarity) as well as the similarities to other sections.

Table 5.1 News IDs and Sections

| Section ID | Section Name | Section ID(continued) | Section Name(continued) |
|------------|--------------|-----------------------|-------------------------|
| 1 | Election | 10 | Asia |
| 2 | Other Sports | 11 | Tennis |
| 3 | Golf | 12 | Basketball |
| 4 | Business | 13 | CSA |
| 5 | National | 14 | Hockey |
| 6 | Mideast | 15 | Science |
| 7 | Europe | 16 | Baseball |
| 8 | Football | 17 | Technology |
| 9 | Arts | | |

5.1.1 Article Number of Each News Section

The analysis of the article numbers shows three different varying trends: 1) news sections such as "Election," "Business," "National," "Arts," and "Technology" have fewer articles on weekends than weekdays; 2) the numbers of daily news articles in sections "Mideast," "Asia," "Europe," "CSA," and "Science" are somewhat fixed compared with the sections in case 1; 3) the number of news items in sport sections may vary dramatically because of game seasons or special events.

For example, Wimbledon 2001 (June 25th ~ July 08th) and the NBA draft (June 28th) make the “Tennis” and “Basketball” article numbers up and down. Therefore, three groups of curves are shown in the following figures to illustrate these trends.

Case 1: weekends and weekdays variation

Figure 5.1 shows the variations of the article numbers of “Election,” “Business,” “National,” “Arts,” and “Technology” from June 15th to July 14th. It is obvious that these sections have fewer news items on weekends.

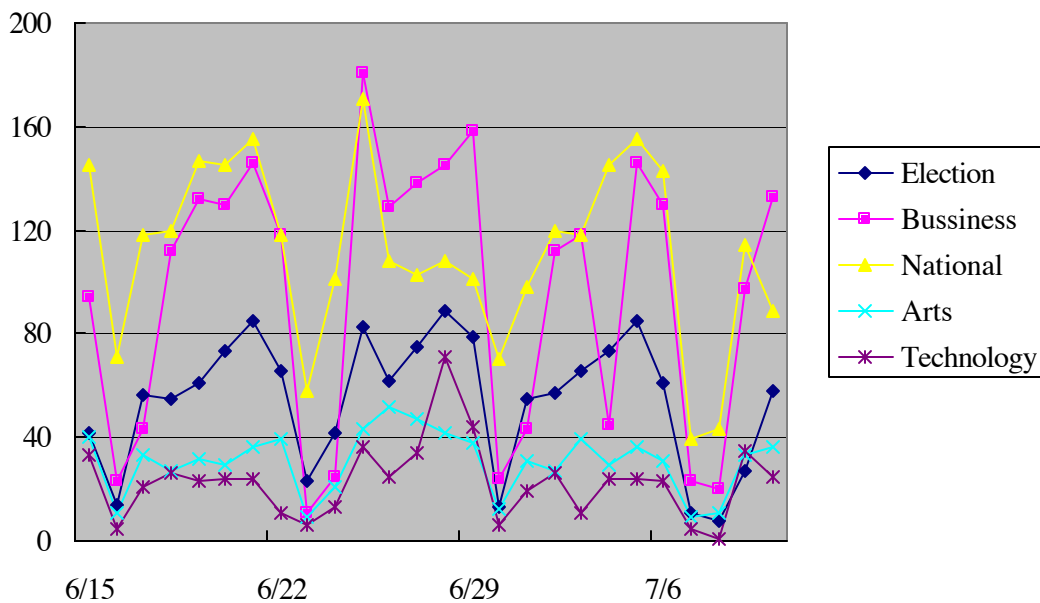


Figure 5.1 Numbers of articles in “Election,” “Business,” “National,” “Arts” and “Technology”

Case 2: randomly and slowly variation

The numbers of articles in sections “Mideast,” “Europe,” “Asia,” “CSA” and “Science” vary more randomly and slowly than those in case 1 (see Figure 5.2). They are mostly influenced by the number and importance of events happening on each day.

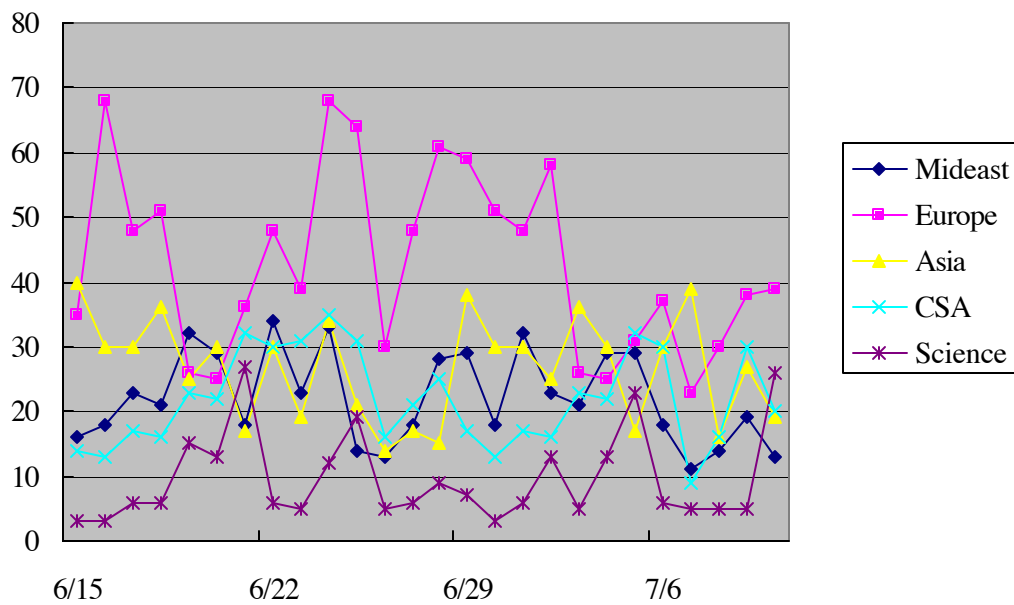


Figure 5.2 Numbers of articles in “Mideast,”
“Europe,” “Asia,” “CSA,” and “Science”

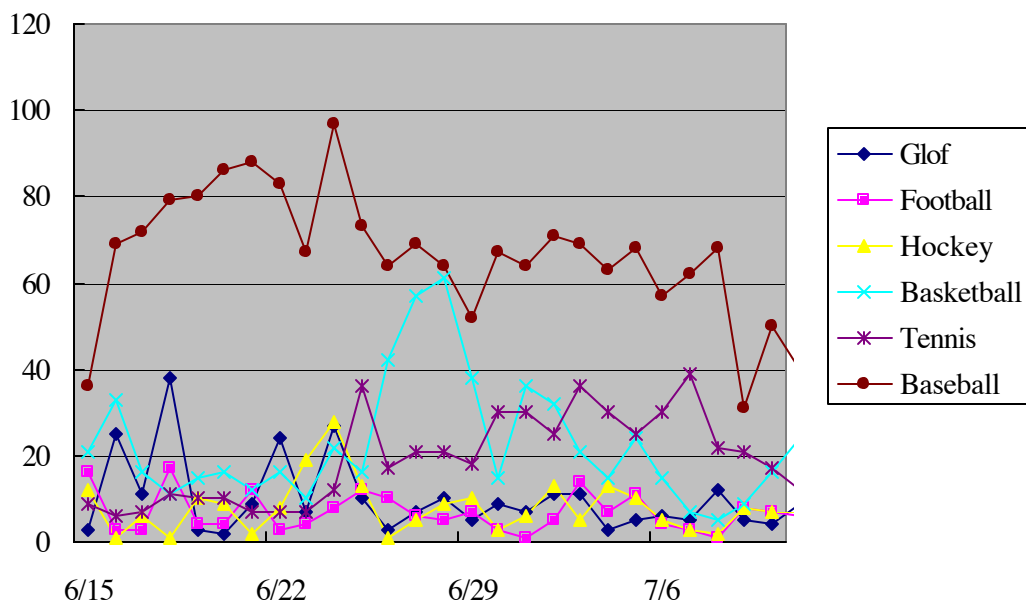


Figure 5.3 Number of articles in sport sections

Case 3: sport sections influenced by events

The numbers of articles in sport sections are influenced by sport seasons and special events (see Figure 5.3). Because football season and hockey season are over and it is now time for baseball, “Football” and “Hockey” have only a few articles, while the “Baseball” section has many.

Besides, the number of golf articles was a little more than usual during the 34th PGA CPC (June 21-24, 2001). The Wimbledon 2001 (June 25th ~ July 08th) and the NBA draft (June 28th) also increased the article numbers of “Tennis” and “Basketball” respectively.

5.1.2 Similarity Within News Sections

The similarity of articles within a news section (self-similarity) indicates correlations. We use the LSI algorithm (see section 3.2) to compute the similarity between any two articles in the section and use the average of these similarities as this section’s self-similarity.

If a news section has a high self-similarity, it is likely that more articles in this section will be recommended to the user if he/she has read one from it. In other words, the system can easily specialize itself to such kind of sections (user interests).

Figure 5.4 shows the self-similarity of each news section fluctuating around its average value. It is noticeable that sport sections usually have higher self-similarities than the others. To see it clearly, we show their temporal average values in Figure 5.5.

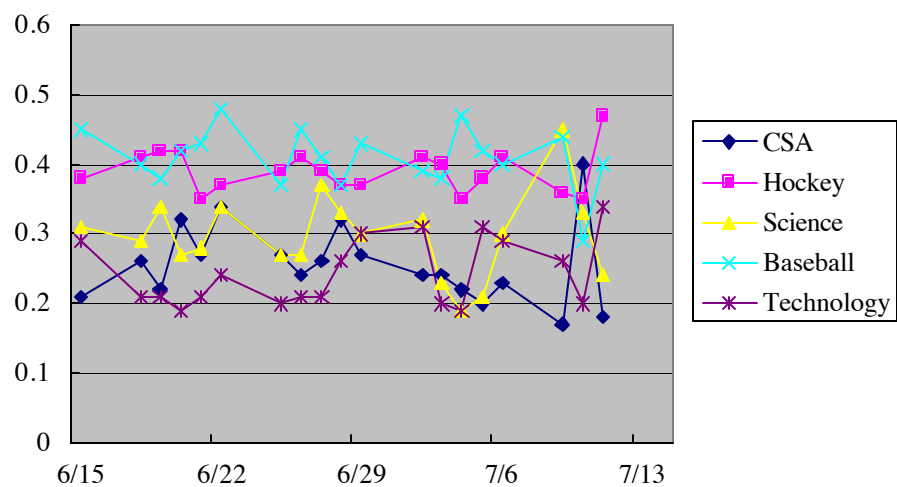
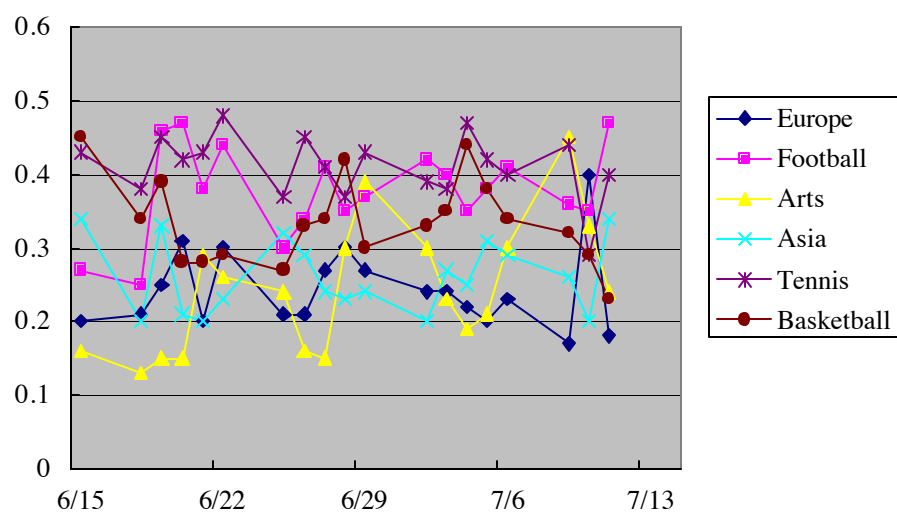
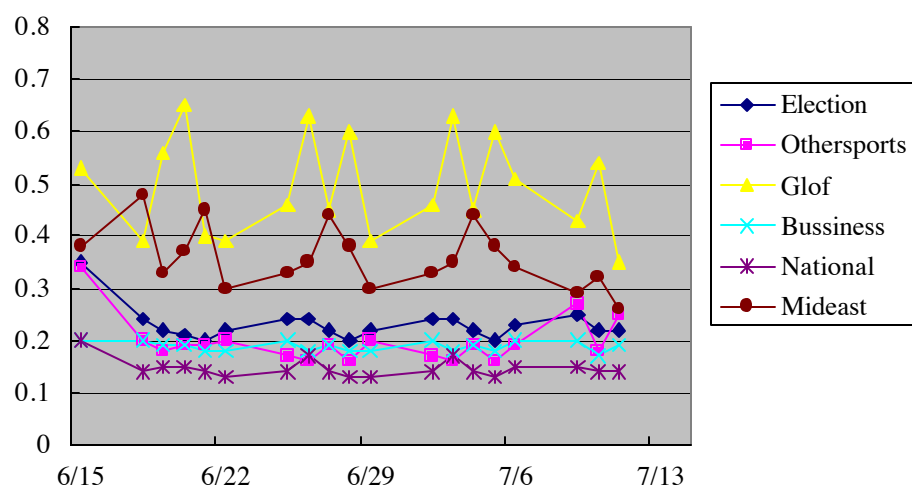


Figure 5.4 Similarities within each news section

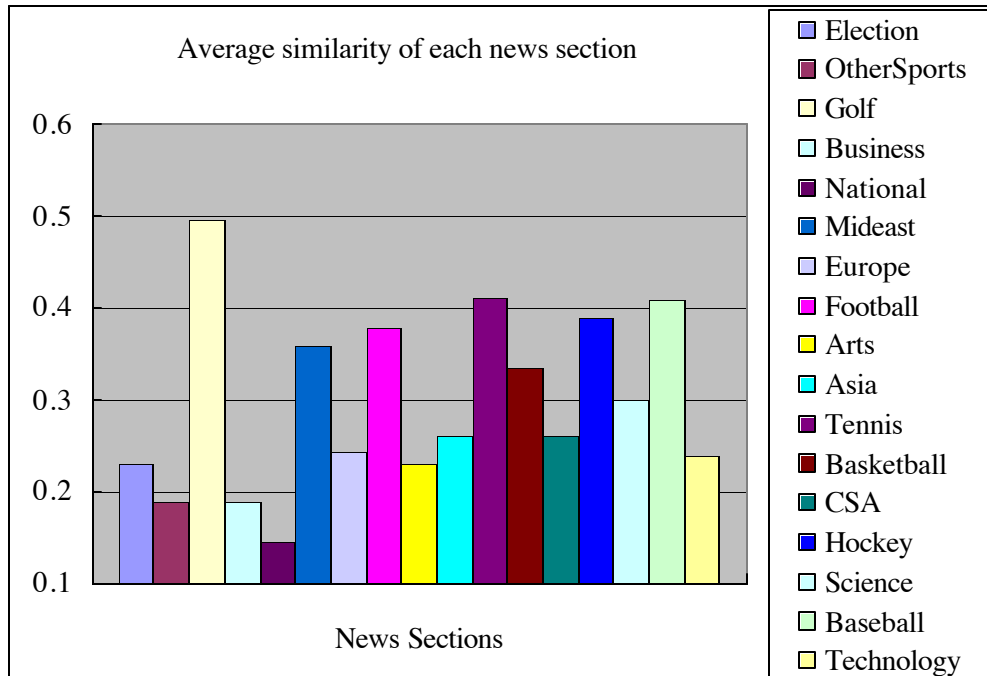


Figure 5.5 Average self-similarity of each news section

5.1.3 Similarity Between News Sections

The similarity between sections A and B is computed as the average of similarities between article pairs across A and B . If two news sections are highly similar, a user who loves news from one section will likely be recommended articles from the other. Figure 5.6 shows the temporal averages of similarities between news sections from June 15th to July 14th.

As shown in the contour plot, the warm-color represents high similarities. Based on how similar they are, the 17 news sections can be roughly put into two groups: politics and sports. In the political group, “Election” (ID=1), “Business” (ID=4), “National” (ID=5), and “Europe” (ID=7) are the most similar sections to one another (see Figure 5.7). Besides, “Mideast” (ID=6), “Asia” (ID=10), and “CSA” (ID=13) are also quite similar to those four sections respectively (see Figure 5.8~5.10).

On the other hand, the sport sections are also very similar to one another except “Golf.” Figure 5.11 shows their correlations. Among them, “Football,” “Basketball” and “Hockey” are more similar to each other than to the rests.

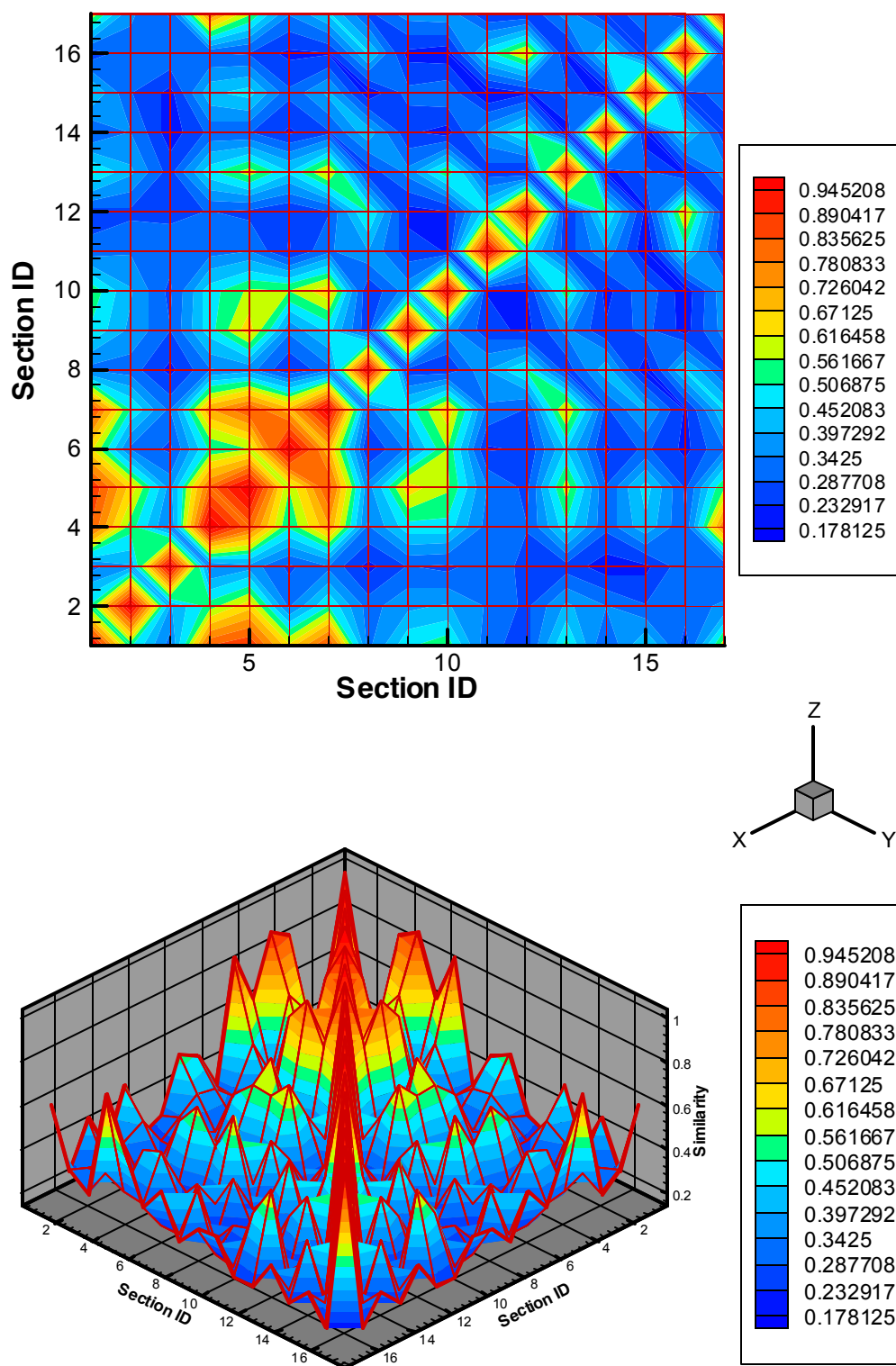


Figure 5.6 Temporal average similarity of each pair of sections (2D and 3D)

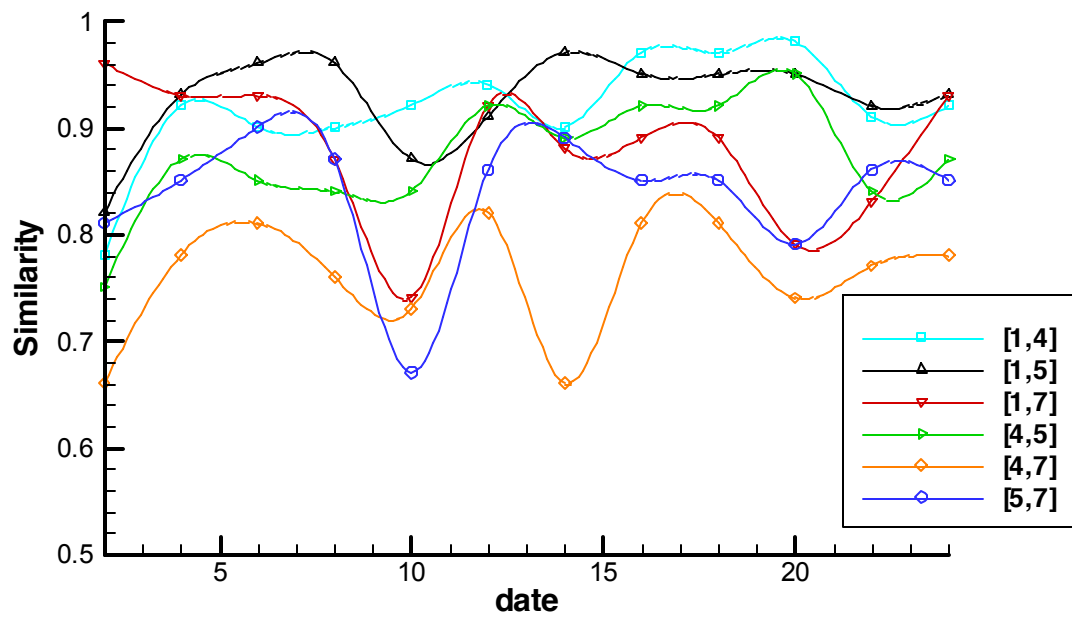


Figure 5.7 Similarities between any two sections with ID in {1,4,5,7}

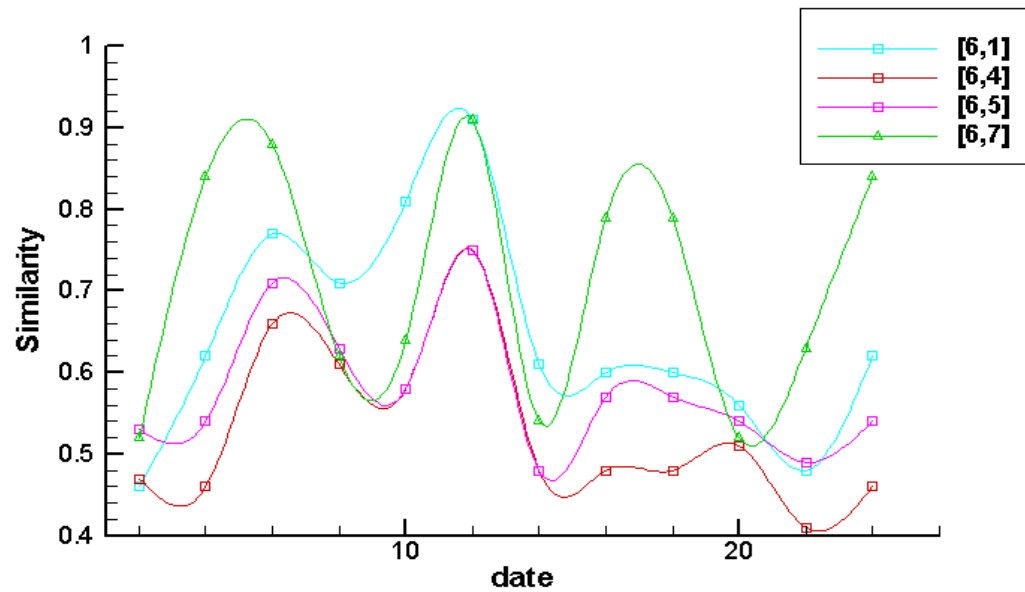


Figure 5.8 Similarity between section 6 and section 1,4,5,7 over time

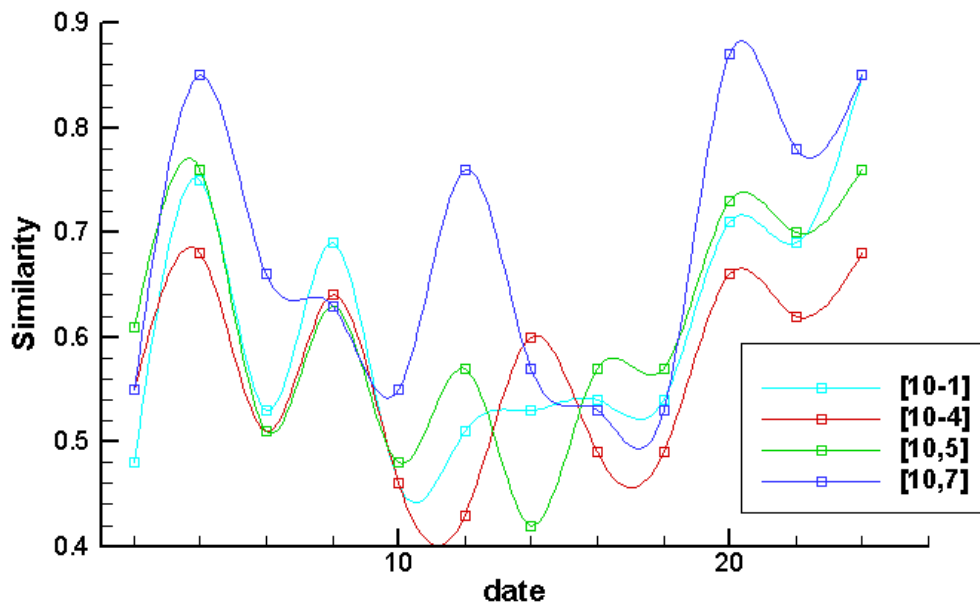


Figure 5.9 Similarities between section 10 with section 1,4,5,7 over time

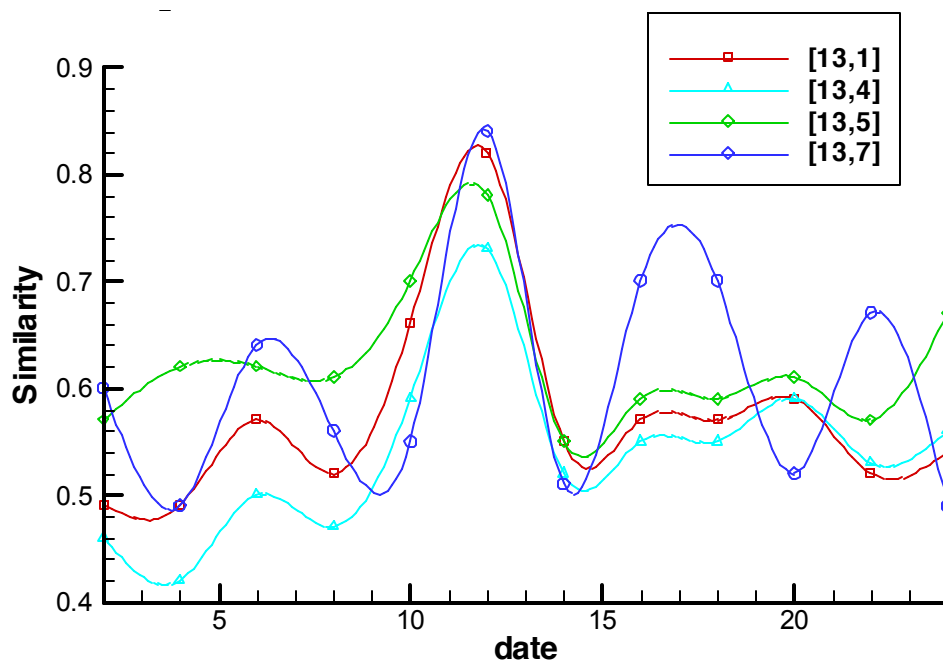


Figure 5.10 Similarities between section 13 with section 1,4,5,7 over time

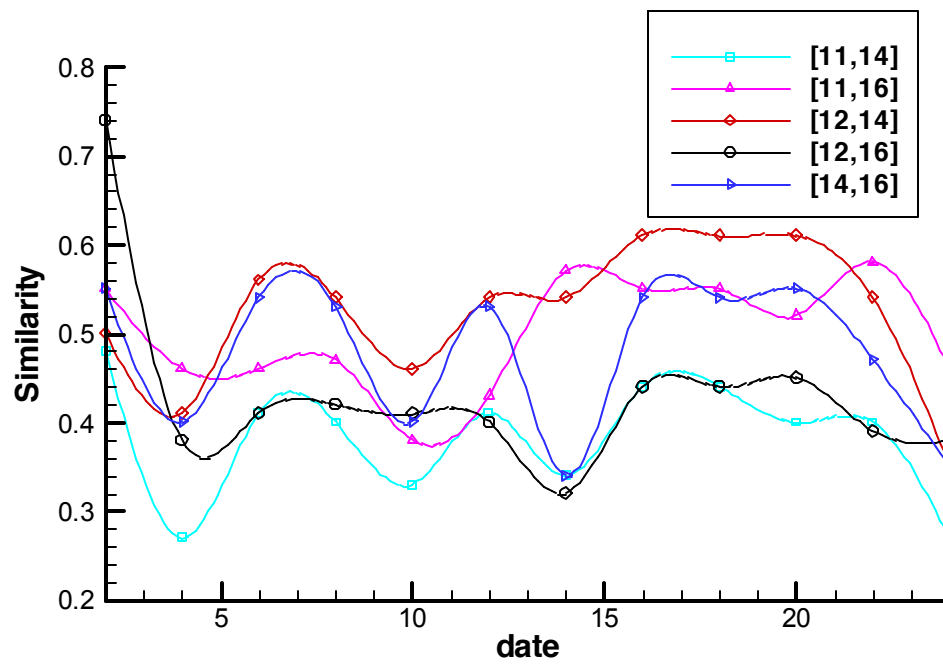
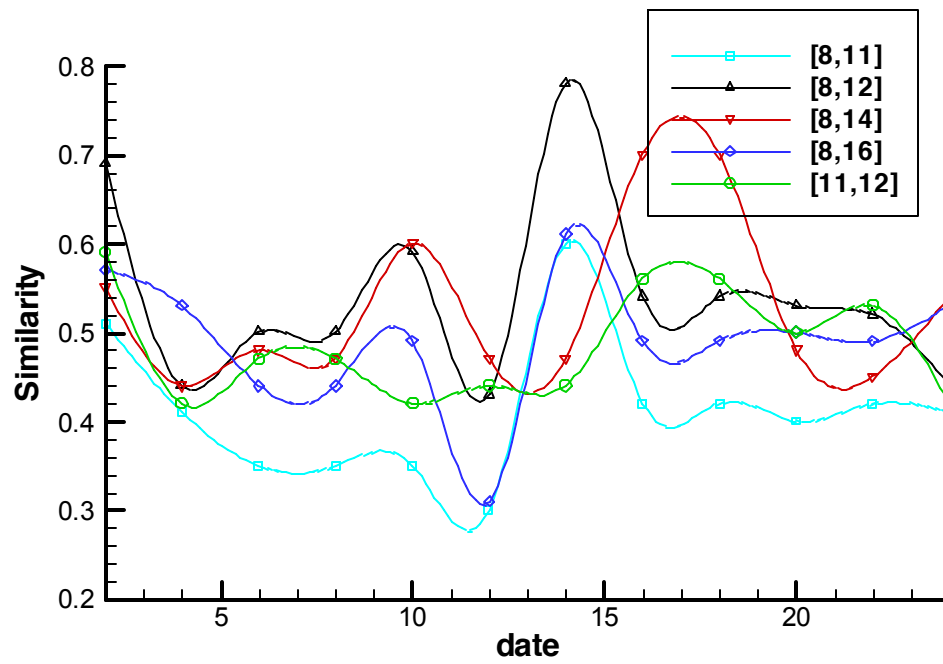


Figure 5.11 Similarities between sport news sections over time

5.2 User Experiments

Six volunteers used our system for two weeks. During the first week, we got plenty of comments from them, which helped us improve our system. Two problems were solved based on their feedback.

The first one was there were articles with the same title recommended more than once. On June 26th, one user complained that the system recommended the article “*Court Reverses Break-Up of Microsoft*” in one time slot twice. We checked his log file as well as the database and found that they were indeed two news items with different story-IDs and received timestamps. And when the user tried to browse these two news stories, we found that the earlier one yielded an “Error” page. We guessed that the new article replaced the old one in the news source, while our database did not update itself to this change. Therefore, we added one more check for incoming news. If a news item with the same title exists, all corresponding entries of it will be deleted from the database before the system inserts the new one.

The second problem was that the system kept recommending articles in some fields in which users had no interest. This problem was raised after the first week’s testing. One user interested in basketball complained that many baseball articles had been recommended to him although he never read one. Another user asked how he would be recommended only “National” news, but no “Mideast” or “Europe” articles. This problem is caused by the fact: articles in political group are similar to each other and so are articles in sports group.

Unfortunately, both LSI and the neighborhood-based collaborative algorithm used in our system depend heavily on term analysis. For example, it cannot tell whether a car accident happened on a street of America or Germany, and it may confuse the games in basketball with those in baseball. To solve this problem, we added a preference editor dialog on Tickertape so that users can set negative preference values to disliked sections (see Figure 4.6). The more negative the value for a section, the fewer the articles in the section will be recommended. In this way, users would not be bothered by the news that they really dislike.

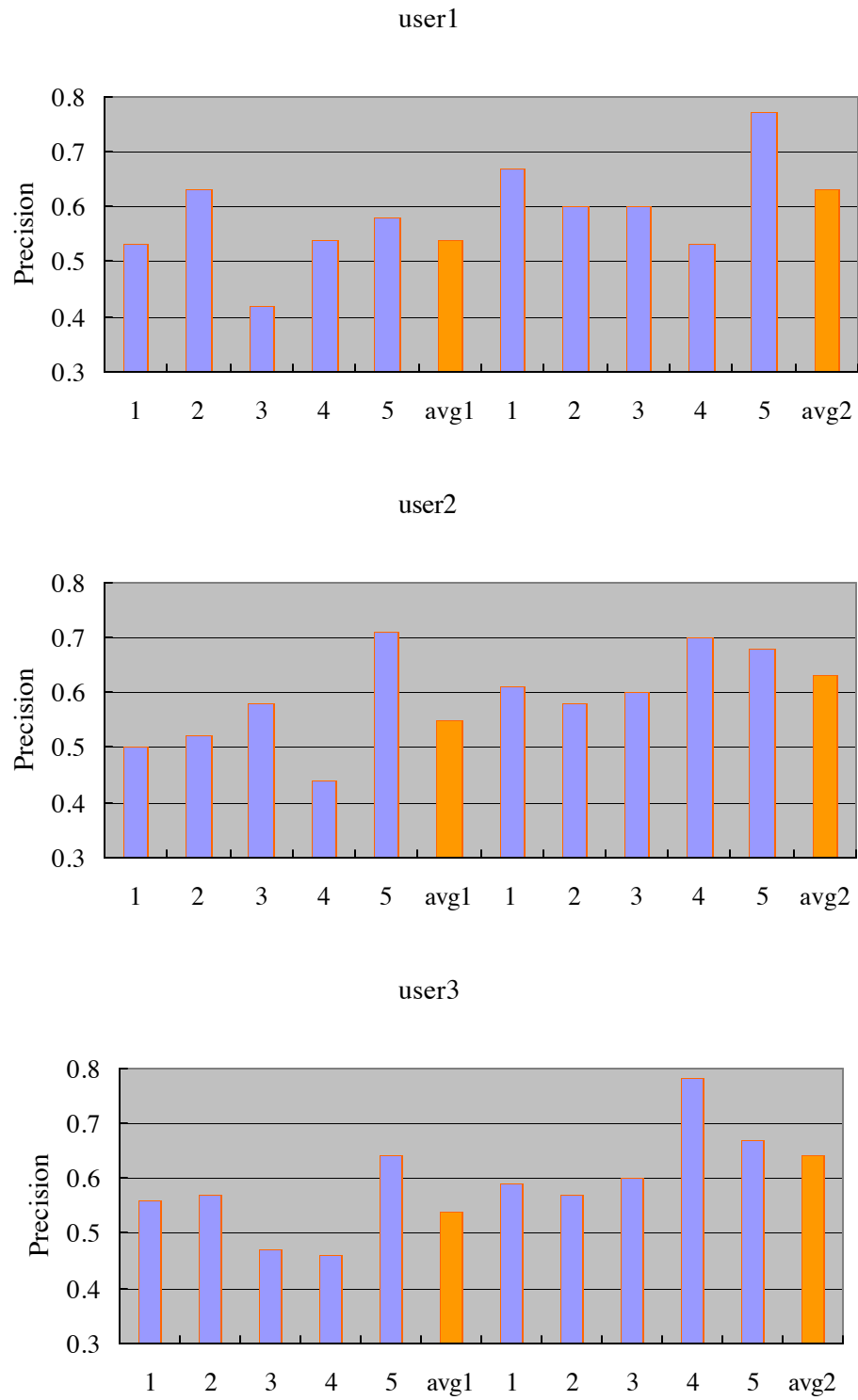


Figure 5.12 Real user experimental results

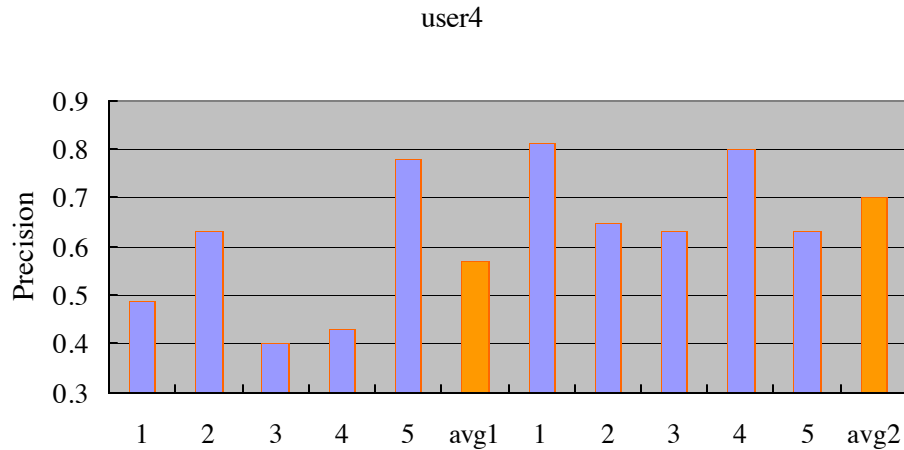


Figure 5.12 Real user experimental results (continued)

Figure 5.12 shows the results of 4 users who used the system almost every day. Here, the precision is defined as the proportion of articles read to all retrieved articles. 1~5 represent Monday ~ Friday. The precision value of “avg1” (or “avg2”) is the average of the precisions of the first (second) week. From this figure, we can see that the system precision becomes higher after the first week.

The results from the real user experiments show that our system's performance is satisfactory. The users consider that the system is intelligent, interface-friendly, and controllable. The analysis of the user log files shows that the system can successfully capture users' interests as well as adapt to their interest changes.

5.3 Simulated User Tests

We did a number of experiments using simulated user inputs under tailored situations to test the system's capability of specialization, adaptation and exploration respectively. This approach allows us to focus on the performance of each individual component. We describe the experiments and results in the following as well as the sample scenarios presented along with the experiments as vehicles for introducing user contexts.

5.3.1 Given Condition of the Experiments

Database

Article number of each section in the database is shown in Figure 5.13. There are totally 360 articles in all 17 sections. During the experiments, the database was unchanged: no article was inserted or deleted for the simulated user.

User Profile

When an initial user profile is needed, it is composed of news articles from the past several days except for the ones in the database (i.e. in the last 24 hours). For example, if the user profile is supposed to contain two “National” news items, we pick two articles from the out-of-date “National” news section and put them into the profile.

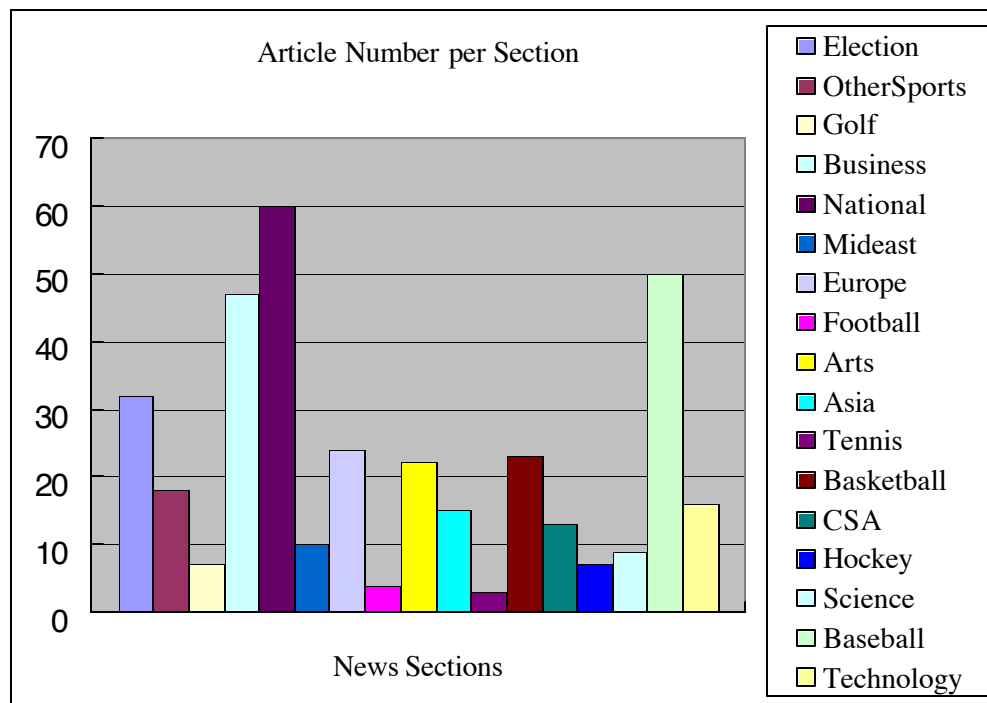


Figure 5.13 Breakdown of the database in our experiments

5.3.2 Specializing to User Interests

When a user has a known interest and provides consistent feedback, a responsive system should converge to the terms defining this interest. The following experiments were conducted with the aim of evaluating our system’s ability to specialize to user interests. Heuristically, if the user interest lies in a section with a high self-similarity, the system will easily specialize itself. Thus,

we chose “Baseball” and “National” as two user interests to test our system respectively. “Baseball” has a high self-similarity, while the similarity within “National” is the smallest (see Figure 5.5). The following experiments demonstrate how self-similarity can affect the system’s capability of specialization.

Scenario 1

Tom is a fan of Baseball. Only articles referring to Baseball are interesting to him.

Experiment 1

Suppose Tom is a new user and he has not edited his preferences. We did a series of tests using different user profile sizes ($n = 0,2,5,8,10,12$). In each test case, the number of articles recommended to him in one time slot (*max_items*) is changed to measure the system performance. We keep each test going until all 50 baseball articles are retrieved.

Results

The performance measures used here are precision and recall. These measures are well known and commonly used to evaluate the performance of information retrieval systems. Precision is defined as the proportion of relevant articles to all retrieved articles. Recall is defined as the proportion of relevant articles retrieved to all relevant articles. In this case, only articles from the “Baseball” section are considered to be relevant.

Figure 5.14 shows the results as the precision-recall curves for different profile sizes ($n = 0,2,5,8,10,12$). The curves are parametric curves drawn by connecting all possible precision-recall pairs with the parameter being the number of articles recommended in a time slot.

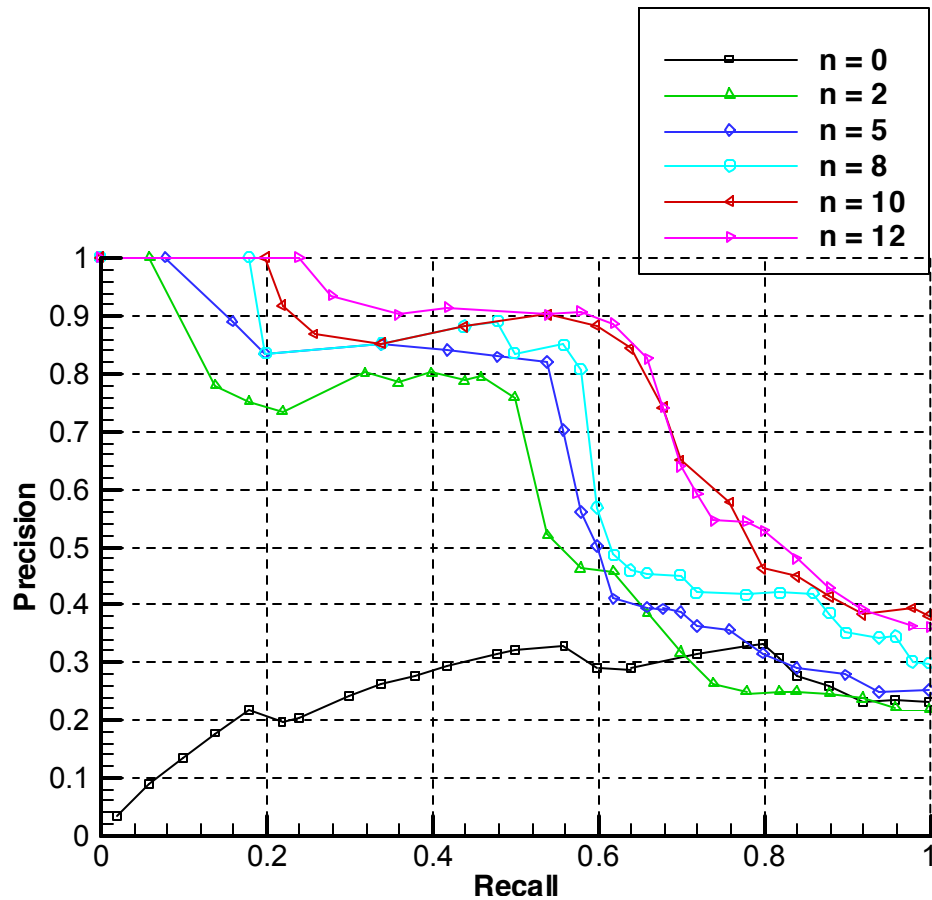


Figure 5.14 Specializing to user interests (experiment 1)

Because Tom did not read anything before, his profile is empty initially and all articles are scored 0.2 according to our initial prediction algorithm (see section 4.3.3). Therefore, articles are randomly retrieved at the beginning based on their position in the database (see curve $n = 0$). Once Tom reads some articles, the user profile will be updated accordingly. We expect that the system is going to give more sensible recommendations with an improved user profile.

In fact, in our experiment, the improvement is obvious and remarkable as we can see from the $n > 0$ curves. The precisions jump to roughly unity for a certain range of recalls. Even when $n = 2$, the system can catch the user's interest quickly and correctly. For $n = 10$ and 12, the precision can be maintained at a large value for large recalls.

We notice that precision drops dramatically when the recall is larger than 0.6, and believe it is because around 60% baseball articles are very similar and retrieved easily right after the user profile had been updated. After that, the system starts recommending more articles in other sport sections, which are quite alike to baseball ones in many features (see Figure 5.11). Therefore, the precision drops to only 40%~50% after the “easy” articles are retrieved.

Scenario 2

Fernando is from South America. He finds that the society of the United States is quite different from that in his home country. He wants to learn more about American society so the articles of the “National” section interest him very much.

Experiment 2

Suppose Fernando’s initial profile is empty and he has not edited his preferences. We did this experiment in the same way as in experiment 1 except that the self-similarity of the “National” section is much lower than that of “Baseball.” This time, there are totally 60 articles in the “National” section.

Results

The results shown in Figure 5.15 are also presented as precision-recall curves with different profile sizes ($n = 0,5,8,10$). Figure 5.15 shows the same trend of improvement as in experiment 1. However, because the self-similarity of “National” is much lower than that of “Baseball,” the precision is not as good as that in experiment 1 for the same recall. After recall is greater than 0.2, the system starts recommending articles from other political sections such as “Election” and “Europe” because they are quite similar to “National” (see Figure 5.7, 5.8). This is reflected in Figure 5.15 in that the precision drops to only 0.3~0.4 for recalls larger than 0.2.

The above experiments demonstrate that our system can successfully specialize itself to the known user interests.

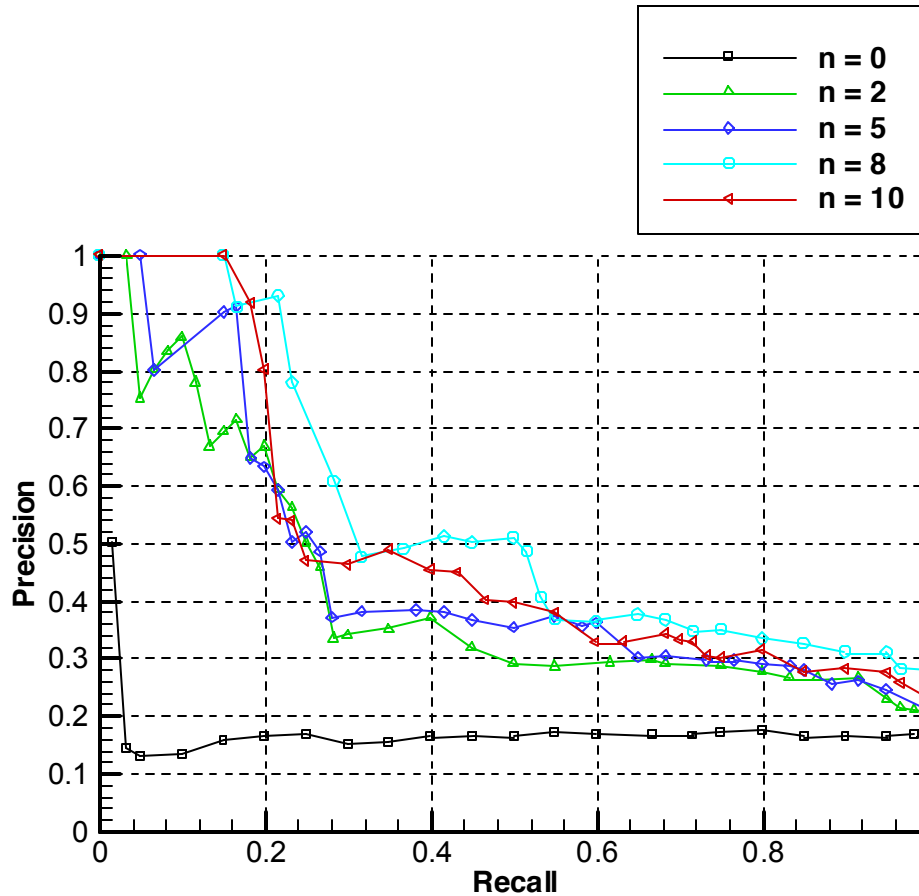


Figure 5.15 Specializing to user interests (experiment 2)

5.3.2 Adapting to User Interest Changes

A filtering system repeatedly interacts with users, so it should be able to follow users' needs over time. The next two experiments are conducted to test how well the system can detect user interest changes and adapt to them. One is to change a user's interest from "Basketball" to "Baseball," while the other is to change from "Basketball" to "National." Note that the average similarity between "Basketball" and "Baseball" is as high as 0.637, and only 0.27 between "Basketball" and "National." It is reasonable to guess that the system will adapt itself to "Baseball" much easier than to "National."

Scenario 1

Suppose Tom was a basketball fan and he only read articles about basketball. However, he was disappointed that his favorite team did not win the championship last season, so he decided not to read basketball articles and switched to baseball.

Experiment 1

We randomly choose 10 basketball articles from the old news as Tom's user profile. Now the size of user profile is fixed to be 10. A series of tests using different user profile makeup are conducted: the number of "Baseball" articles in the user profile is 0, 2, 5, and 10 respectively. In each case, we vary the value of *max_items* to get different precision-recall pairs (recall goes from 0.0 to 1.0).

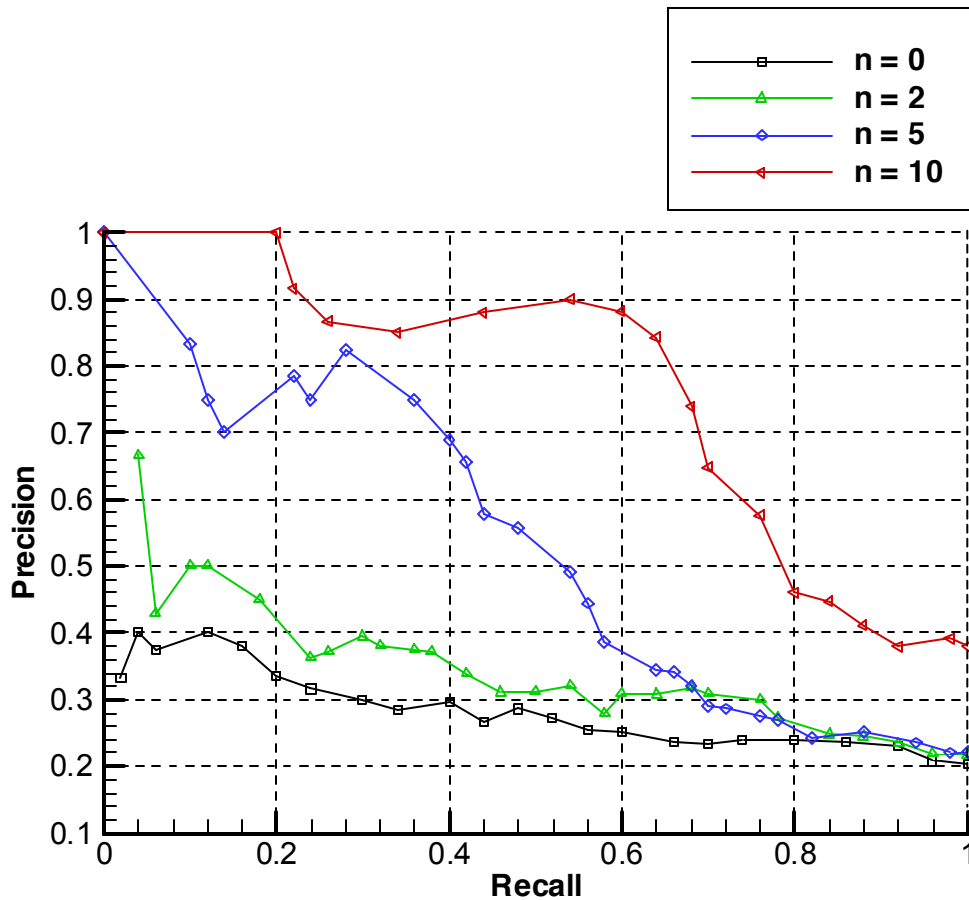


Figure 5.16 Adapting to interest changes (experiment 1)

Results

Precision-recall curves with different n serve to present the results again in Figure 5.16. Here n is the number of the "Baseball" articles in the user profile. A trend similar to that of experiment 1 in section 5.3.1 is observed. However, there are some differences we need to highlight.

1. In the previous experiment, the system knows nothing about the user preference at the beginning, and the recommendations are only random selections so that the precision-recall curve totally depends on the position distribution of baseball articles in the database. While in the current experiment, the initial profile is not empty, but has a previous bias for basketball. As mentioned before, in addition to basketball articles, the baseball ones are also similar to the user profile. This can be seen from Figure 5.16 that the precision is larger than 0.3 before the recall reaches 0.4.
2. As n grows larger, the prediction for baseball articles increases and more and more baseball articles are retrieved. This is demonstrated in Figure 5.16 that as n goes up the precision increases for the same recall value.

Scenario 2

Suppose Tom now will have to focus on his research on American culture and society because the basketball season is over and he has a paper due soon. He decides to read only articles about national events. Now He is expecting the system to be specialized to “National.”

Experiment 2

We also randomly pick 10 basketball articles as Tom’s user profile from the old news. The profile size is fixed to be 10. We conduct a series of tests using different user profile makeups: the number of the “National” articles in the user profile is 0, 2, 5, and 10 respectively. In each case, we vary the value of *max_items* to get different precision-recall pairs.

Results

The results shown in Figure 5.17 are also the precision-recall curves with different n . Here n is the number of “National” articles in the user profile. Not surprisingly, the system performs worse because national articles are less similar to the user profile than baseball ones. We can see this from the $n = 0$ curve, the precision is much lower than that in experiment 1 for the same recall value. Another reason causing the inaccuracy is that the self-similarity of “National” is also lower than that of “Baseball.” Thus, as n goes up, the precision does not increase as dramatically as it does in experiment 1.

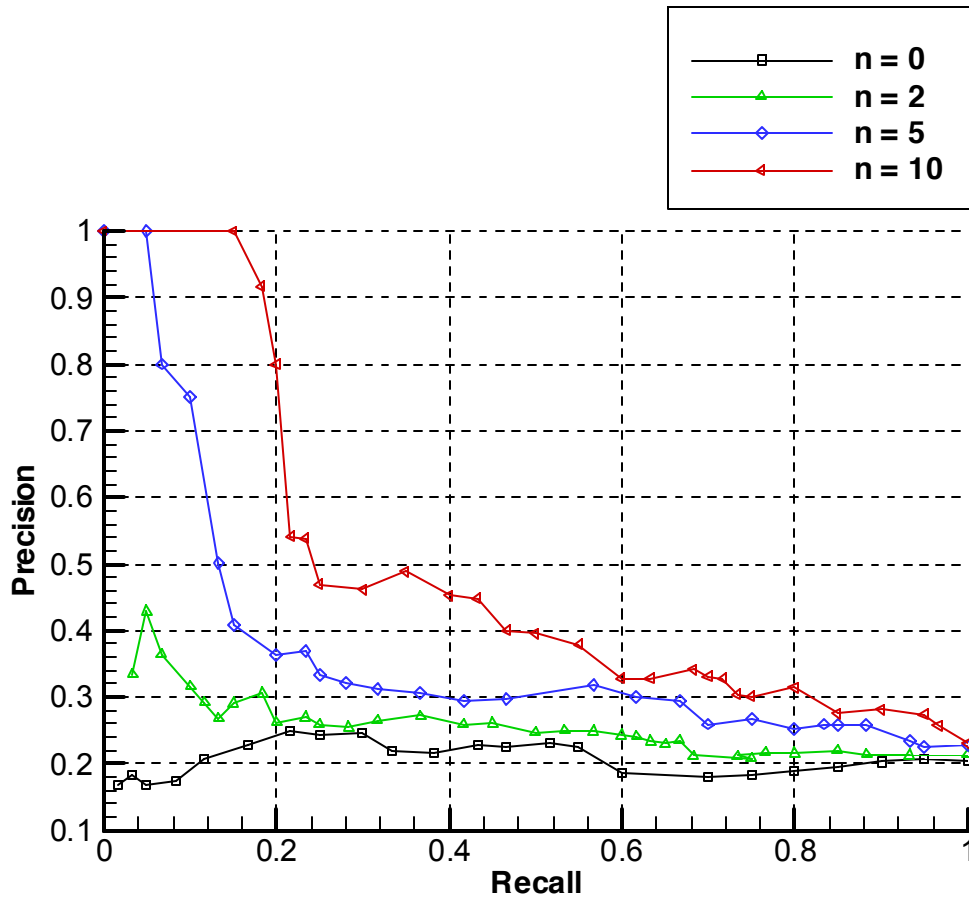


Figure 5.17 Adapting to interest changes (experiment 2)

5.3.3 Exploring New Domains

Exploring new information domains can either precede or succeed changes in user interest. When it precedes the change, the system apparently exhibits a serendipity behavior. Conversely, when the change precedes the exploration, it appears as if the system exhibits a better adaptive behavior in response to changes in user interests.

In the previous section, we designed the experiment to test how well the system can adapt to new interests. Now we investigate how well the system can explore new information domains and recommend serendipities based on the currently known user interests. In our approach, we use both collaborative filtering and random selection methods to realize this feature. Because it is difficult to simulate the behavior of random selection, we only did a test to prove that collaborative filtering method support recommending serendipities.

Scenario

There are two students, George is an American and Wang is from China. They both like competitive sports such as basketball and hockey. Wang did not watch football before he came to the United States because football is not popular in China, while George is a big fan of football. The collaborative filter will recommend some football articles to Wang because they have similar preferences in sports. If Wang feels football is interesting and reads the recommended articles, the system will update his user profile to reflect his new sport interest.

Experiment

In this experiment, we suppose that George's profile contains 10 football, 10 basketball, and 10 hockey articles, while Wang's profile only contains 10 articles about basketball and 10 about hockey. The similarity between Wang and George is computed to be 0.880793. All incoming news articles are compared with the two user profiles respectively, and the averages of the highest 10 predictions in each section of "Basketball," "Hockey" and "Football" are illustrated in Figure 5.18 for both George and Wang.

Now suppose George used our recommendation system and read 20 recommended news items including 11 about football, 6 about Basketball, and 3 about Hockey. Because of their similarity, the system will try to recommend to Wang what George read by increasing the predictions of these articles for Wang. Because only two users are in the community, the parameter δ is set a little lower (0.5 here). (i.e., Using Formulae 3-11 and 3-12, we can get that $\text{increment} = \delta * \text{similarity} = \delta * 0.88$.)

Results

Figure 5.19 shows the changes of the predictions for Wang. It shows that articles from the "Football" section will be recommended to Wang. If Wang has no interest in football, he would ignore such recommendations and there will not be any influence on his user profile.

On the other hand, if he finds football interesting, he may read several recommendations and all these selections will be recorded. After that, the system will adapt to his interest change and recommend more about football. For example, Wang uses this recommendation system and read 5 recommended Football articles, as a response, the predictions for all remaining articles will be changed as shown in Figure 5.20. Clearly, the system successfully finds a new interest for Wang.

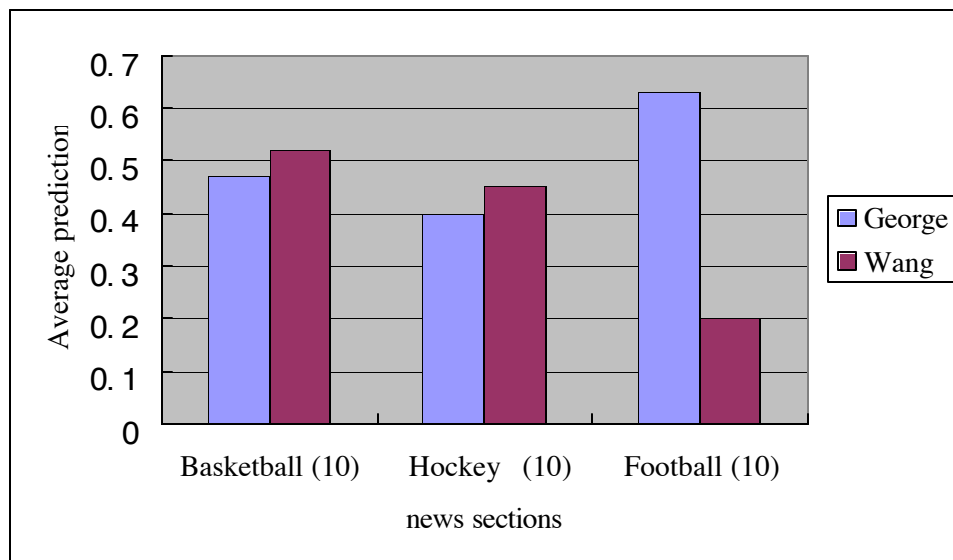


Figure 5.18

Average predictions for George and Wang

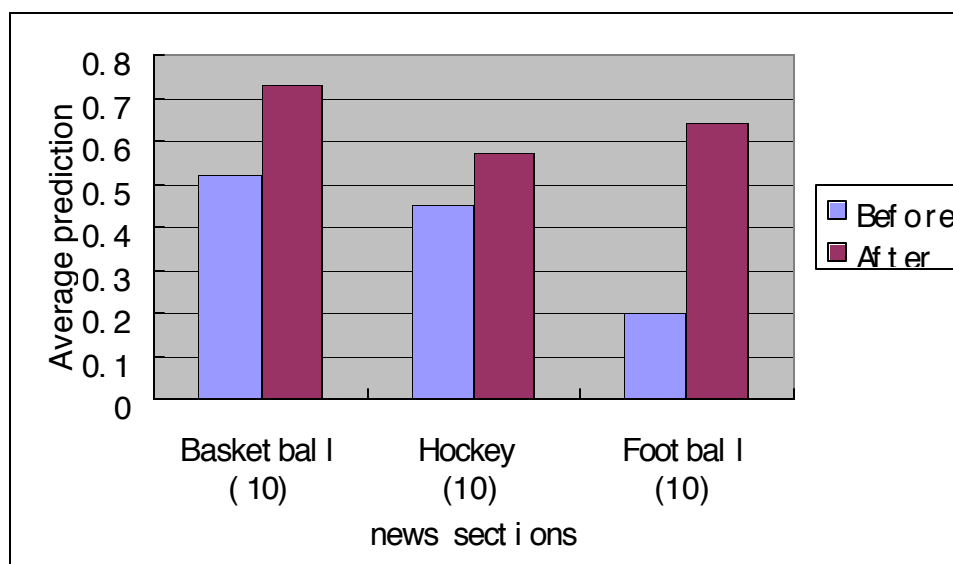


Figure 5.19 Predictions for Wang based on George's opinion

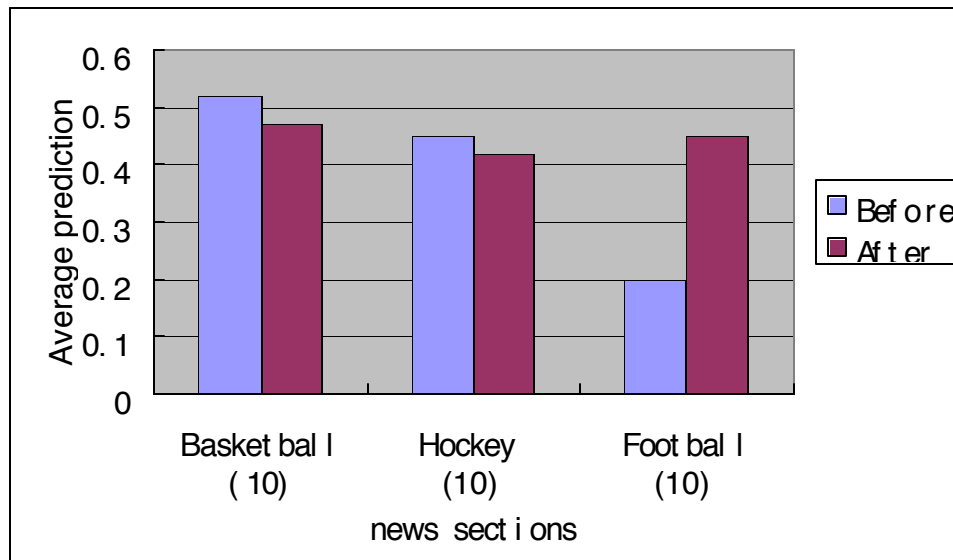


Figure 5.20 Predictions for Wang after he read 5 Football articles

CHAPTER 6

CONCLUSIONS

A personalized information filtering system must be tailored to user interests, and must adapt itself as these interests change. Our implementation of personalized information filtering agents demonstrates the advantages of using a combination of content-based and collaborative algorithms. We used the Latent Semantic Indexing algorithm as the primary method to evaluate predictions, and the neighborhood-based collaborative filtering algorithm as a complement that can adjust users' predictions based on their neighbors' opinions/selections.

By using LSI in the content-based filters, the synonymy and polysemy problems of the standard vector-space methods are solved. With the help of singular-value decomposition techniques, LSI can exploit the underlying semantic structures to make better recommendations. Using the collaborative filtering method, we can take advantage of the strength of human intelligence in understanding information content so that we can improve the accuracy of our model.

There is a big concern about customized information filtering, which is serendipity. The question is, how can the system recommend relevant articles that the user could not possibly have anticipated in the first place? Research groups have tried to solve this problem using collaborative filtering methods that provide serendipitous recommendations [6,12]. However, this approach is only a partial solution to the serendipity problem because collaborative filtering has an "early rater" problem, if nobody has ever tried an item, the item will not be recommended to anyone.

Therefore, in our approach, besides using collaborative filtering, we also provide some randomly selected news items among the recommendations so that users will be able to find their interests in some new fields. If users select these randomly recommended items, they can receive more recommendations in those areas. In this way, we give users a way to explore new interests.

Another feature unique to our system is an arbitrator that allows a group of users to share a display device and can best accommodate the preferences of all active users. Most current information filtering systems usually only work for individuals by tracing their specific interests [12,16,23]. However, in the real world it is often the case that users may want to, or have to, share a display device with others. Therefore an arbitrator will be necessary to adjust the recommendation of news items to best satisfy all sharing users. Using this arbitration system, the

group preference is the combination of all present members so that any number of users can form a group and share the filtering system.

Our experiments gave satisfactory results. The system can quickly capture users' interests and represent them using user profiles, and it can also detect changes in users' interests and adapt itself to follow the new interests. Also, our tests showed that the system has the capability to recommend serendipities. However, it still has some drawbacks because our approach is heavily based on content processing.

Content-based systems work fairly well for most newswire articles that typically describe particular events, persons, places, or things and enough keyword clues. However, there are difficulties on the types of concepts that cannot be expressed in terms of keywords, for example, humor and satire. By incorporating collaborative filtering, human intelligence can be employed in processing content, which may help solving this problem.

Another problem, which is also an inherit feature of content-based systems, is that only the articles similar to what the user has read will be recommended. The more similar the new article is, the bigger the chance it will be recommended. However, although two identical articles have the largest similarity, users will not enjoy reading the same material twice.

In our approach, we leave this task to users. We recommend similar items like other filtering systems, assuming that after users feel they have read enough about an event, they will ignore other similar articles. Therefore, the predictions for the remaining articles will be reduced and eventually they will not be recommended to these users any more.

CHAPTER 7

FUTURE WORK

The explosive growth of online information demands new techniques for prioritizing and presenting items of potential interests to users. Based on the inadequacies in our system, several avenues for future work can be pursued.

7.1 Filtering Module

In our approach, we use the LSI algorithm to generate predictions. Although LSI is a well-known, successful approach applied in information retrieval/filtering systems, it cannot achieve 100% precision in evaluating inter-document similarity. Therefore we would like to explore or adopt some precision-enhancing methods.

Rie Kubota Ando proposed a better algorithm that creates document vectors via dimensional reduction for a relatively small document set. In his paper [24], he showed that his algorithm could yield inter-document similarities, with an average precision up to 17.8% higher than that of SVD used in LSI. Because the number of articles contained in the user profile is small, we may try to implement this algorithm for better accuracy. We may also want to try different methods using natural language processing [25,26] or neural networks and specifically spreading activation models [27].

Taking a step further, we may try to implement more than one news agents to recommend documents for a single user. There are two interesting directions for future work in this area. One is to implement different agents using different approaches. Each agent would generate its own evaluation for an item and the weighted combination of evaluations from all agents would be used as the prediction. The weights assigned to different agents would be adaptively adjusted according to user feedback. The more similar the recommendations made by the agent compared to the selection made by the user, the larger the weight would be. Because different filtering approaches may be better at filtering items in different domains, the advantage of this implementation is that the user would have a common interface to support his/her diverse filtering needs.

The other direction is to have different news agents filter news articles from different news sections. Correspondingly, different user profiles would be maintained for a user, each for a different news section. One of the advantages of this approach is scalability. For example, if there

is a lot of incoming news, we could generate predictions for news from different sections in parallel. The other advantage is that less time would be needed to finish the prediction because no time would be wasted computing the similarities between articles about football and political, scientific, or other articles in the user profiles.

7.2 Implicit Feedback

In our approach, we used implicit feedback – reading news or ignoring it as two ratings. However, it is very likely that after a user launches a web browser to read a news article, he/she will find that it is not as interesting as he/she had expected. But since the user has selected it, the system thinks the user likes it and adds it to the user profile. What is worse is that the article could also be recommended to other similar users, which would enlarge the system inaccuracy/errors. Besides, users may like some articles more and others less. Thus the fact that only two scales are used in our system is a cause of system imprecision.

One solution to shortcomings in our filtering system is to compare the time spent on reading with the article length as a measure of enjoyment. However, this approach has its own problem: computers cannot tell whether users are reading or just leaving the browser open. Combining this method with other techniques could help overcome this shortcoming: 1) we can use sensors or computer vision methods to locate the user, and if the location is in front of the computer, we will assume that the user is reading; 2) or we can compute the time needed to read the article based on the average reading speed of the user, if there are no changes (no mouse actions, no scrolling) for a period of time much longer than the expected time, we could assume that the user is not reading.

7.3 User Interface

Users would feel more convenient if the interface were more intelligent and versatile. For example, some users would prefer to read textual news, hear it if they are driving, view it if it involves video and so on. An interesting problem would be to design an intelligent agent that can learn users' habits and offer the user a seamless switchover across different media.

APPENDIX A

EXAMPLE OF FILTERING USING LSI

Figure A.1 is Ben's user profile containing 9 documents. All the documents are classified into two clusters, one is in the field of combustion on stretched/strained flames (labeled C1-C5) and the other is on the research of viscous flows that is an active area in fluid dynamics (labeled m1-m4). Only the words that occur in more than one document will be considered as terms (which are bold in the documents). C6 is an incoming document in the field of combustion. The term-by-document matrix (12*10) of C6 and all profile documents is shown in Figure A.1.

User Profile Documents:

C1: Numerical model of flame stretching

C2: Numerical simulation of **premixed deflagrations** under variation of the rate of **stretch**

C3: Extinction of strained premixed deflagrations of hydrogen/air/steam mixture

C4: Effects of straining on the local structure of freely propagating **premixed flames**

C5: Flame extinction in **stretched** PMMA diffusion **flames**

M1: Numerical investigation of high **Re-number** internal **flow**

M2: 1DFLOW – a model for **viscous** incompressible **flow**

M3: Study of the viscous effects on the vortex/wall interaction

M4: Viscous cavity **flow** under different **Re-number**

Incoming document:

C6: Dynamics of edge flames

| Terms | Documents | | | | | | | | | |
|--------------|-----------|----|----|----|----|----|----|----|----|----|
| | C6 | C1 | C2 | C3 | C4 | C5 | M1 | M2 | M3 | M4 |
| numerical | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| model | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| flame | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| stretch | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| premix | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| deflagration | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| Extinction | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| strain | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| effect | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| Re-number | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| flow | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| viscous | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

Figure A.1 Term-by-document matrix of both the incoming document and profile documents

Assuming this term-by-document matrix is A_θ . Its factorization is $A_\theta = U_\theta S_\theta V_\theta^T$, where

$U_\theta =$

| | | | | | | | | | |
|--------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0.2779 | 0.2487 | -0.0611 | 0.5565 | -0.1393 | -0.3624 | 0.0351 | 0.3628 | 0.3027 | 0.1224 |
| 0.1624 | 0.2445 | 0.1524 | 0.0838 | -0.4309 | 0.3629 | 0.5884 | 0.1483 | -0.1232 | 0.1194 |
| 0.6686 | -0.1850 | 0.4801 | -0.2815 | 0.1437 | -0.1454 | 0.0669 | -0.2400 | 0.2337 | 0.2247 |
| 0.4204 | -0.0151 | 0.1485 | 0.3631 | -0.1673 | 0.1362 | -0.3561 | -0.0757 | -0.5443 | -0.3395 |
| 0.3074 | -0.0842 | -0.5699 | -0.0106 | -0.0828 | -0.0842 | 0.0751 | -0.4403 | -0.0697 | -0.2015 |
| 0.1914 | -0.0339 | -0.4518 | 0.2246 | 0.0460 | 0.2872 | -0.2026 | -0.1158 | 0.3558 | 0.3723 |
| 0.2644 | -0.1261 | -0.0910 | -0.1183 | 0.4246 | 0.4517 | -0.1106 | 0.5964 | 0.0777 | -0.2344 |
| 0.2014 | -0.1012 | -0.3971 | -0.2783 | 0.0636 | -0.0723 | 0.4322 | 0.1784 | -0.3522 | 0.1579 |
| 0.1333 | 0.0284 | -0.1516 | -0.4253 | -0.4268 | -0.4212 | -0.2254 | 0.3609 | 0.1212 | -0.2550 |
| 0.0539 | 0.4149 | -0.0372 | 0.0155 | 0.4637 | -0.3393 | -0.0995 | 0.0848 | -0.4383 | 0.3397 |
| 0.0809 | 0.6163 | -0.0179 | -0.0712 | 0.2859 | 0.0280 | 0.2227 | -0.2163 | 0.2698 | -0.5271 |
| 0.0617 | 0.5066 | -0.0300 | -0.3796 | -0.2721 | 0.3243 | -0.4065 | -0.0629 | -0.0350 | 0.2961 |

$S_\theta =$

| | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 3.3609 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2.6059 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 2.3258 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2.0576 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1.5317 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1.3953 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1.1207 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.6594 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.3971 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.3589 |

$V_\theta =$

| | | | | | | | | | |
|--------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 0.1989 | -0.0710 | 0.2064 | -0.1368 | 0.0938 | -0.1042 | 0.0597 | -0.3640 | 0.5884 | 0.6262 |
| 0.4550 | 0.1125 | 0.3095 | 0.3508 | -0.3877 | -0.0062 | 0.2983 | 0.2964 | -0.3301 | 0.3539 |
| 0.3562 | 0.0443 | -0.4017 | 0.5509 | -0.2242 | -0.0166 | -0.4002 | -0.4079 | 0.1122 | -0.1290 |
| 0.2870 | -0.1326 | -0.6491 | -0.0888 | 0.2947 | 0.4173 | 0.1731 | 0.3316 | 0.0291 | 0.2626 |
| 0.3900 | -0.1312 | -0.2745 | -0.4839 | -0.1974 | -0.5182 | 0.3112 | -0.2140 | -0.1690 | -0.2060 |
| 0.6016 | -0.1961 | 0.4376 | -0.1547 | 0.3557 | 0.2129 | -0.2971 | 0.0617 | 0.0018 | -0.3468 |
| 0.1228 | 0.4912 | -0.0500 | 0.2434 | 0.3985 | -0.4828 | 0.1412 | 0.3508 | 0.3381 | -0.1810 |

0.0907 0.5248 0.0449 -0.1783 -0.2723 0.5126 0.3611 -0.1986 0.2812 -0.3111
0.0580 0.2053 -0.0781 -0.3911 -0.4563 -0.0694 -0.5638 0.4520 0.2171 0.1144
0.0585 0.5902 -0.0366 -0.2115 0.3118 0.0093 -0.2528 -0.2949 -0.5122 0.3029

If we only keep the first two largest singular values of S_0 , we can get A_θ 's two-dimensional approximation $A = USV^T =$

$$\begin{bmatrix} 0.28 & 0.16 & 0.67 & 0.42 & 0.31 & 0.20 & 0.26 & 0.20 & 0.13 & 0.05 & 0.08 & 0.06 \\ 0.25 & 0.24 & -0.19 & -0.02 & -0.08 & -0.03 & -0.13 & -0.10 & 0.03 & 0.41 & 0.62 & 0.51 \end{bmatrix}^T$$

$$* \begin{bmatrix} 3.3609 & 0 \\ 0 & 2.6059 \end{bmatrix} *$$

$$\begin{bmatrix} 0.20 & 0.46 & 0.36 & 0.29 & 0.39 & 0.60 & 0.12 & 0.09 & 0.06 & 0.06 \\ -0.07 & 0.11 & 0.04 & -0.13 & -0.13 & -0.20 & 0.49 & 0.52 & 0.21 & 0.59 \end{bmatrix}$$

=

| Terms | Documents | | | | | | | | | |
|--------------|-----------|------|------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| | C6 | C1 | C2 | C3 | C4 | C5 | M1 | M2 | M3 | M4 |
| numerical | 0.14 | 0.50 | 0.36 | 0.18 | 0.28 | 0.43 | 0.43 | 0.42 | 0.19 | 0.44 |
| Model | 0.06 | 0.32 | 0.22 | 0.07 | 0.13 | 0.20 | 0.38 | 0.38 | 0.16 | 0.41 |
| Flame | 0.48 | 0.97 | 0.78 | 0.71 | 0.94 | 1.45 | 0.04 | - 0.05 | 0.03 | - 0.15 |
| stretch | 0.28 | 0.64 | 0.50 | 0.41 | 0.56 | 0.86 | 0.15 | 0.11 | 0.08 | 0.06 |
| premix | 0.22 | 0.45 | 0.36 | 0.33 | 0.43 | 0.66 | 0.02 | - 0.02 | 0.01 | - 0.07 |
| deflagration | 0.13 | 0.28 | 0.23 | 0.20 | 0.26 | 0.40 | 0.04 | 0.01 | 0.02 | - 0.01 |
| extinction | 0.20 | 0.37 | 0.30 | 0.30 | 0.39 | 0.60 | - 0.05 | - 0.09 | - 0.02 | - 0.14 |
| Strain | 0.15 | 0.28 | 0.23 | 0.23 | 0.30 | 0.46 | - 0.05 | - 0.08 | - 0.01 | - 0.12 |
| Effect | 0.08 | 0.21 | 0.16 | 0.12 | 0.17 | 0.26 | 0.09 | 0.08 | 0.04 | 0.07 |
| re-number | - 0.04 | 0.20 | 0.11 | - 0.09 | - 0.07 | - 0.10 | 0.55 | 0.58 | 0.23 | 0.65 |
| Flow | - 0.06 | 0.30 | 0.17 | - 0.13 | - 0.10 | - 0.15 | 0.82 | 0.87 | 0.35 | 0.96 |
| viscous | - 0.05 | 0.24 | 0.13 | - 0.12 | - 0.09 | - 0.13 | 0.67 | 0.71 | 0.28 | 0.79 |

Figure A.2 Two-dimensional approximation A of the original term-by-document matrix A_0

We can see from Figure A.2, we can guess that the document vectors for c1~c6 should have similar directions. To prove it, we can compute the vectors VS .

$$\begin{aligned}
 VS &= \begin{bmatrix} 0.20 & 0.46 & 0.36 & 0.29 & 0.39 & 0.60 & 0.12 & 0.09 & 0.06 & 0.06 \\ -0.07 & 0.11 & 0.04 & -0.13 & -0.13 & -0.20 & 0.49 & 0.52 & 0.21 & 0.59 \end{bmatrix}^T \\
 &\quad * \begin{bmatrix} 3.3609 & 0 \\ 0 & 2.6059 \end{bmatrix} \\
 &= \begin{bmatrix} 0.67 & 1.53 & 1.20 & 0.96 & 1.31 & 2.02 & 0.41 & 0.31 & 0.20 & 0.20 \\ -0.19 & 0.29 & 0.12 & -0.35 & -0.34 & -0.51 & 1.28 & 1.37 & 0.54 & 1.54 \end{bmatrix}^T
 \end{aligned}$$

To make it clearer, we re-write the document vectors and similarities in Figure A.3.

| C6 | C1 | C2 | C3 | C4 | C5 | M1 | M2 | M3 | M4 |
|-------|------|------|-------|-------|-------|------|------|------|------|
| 0.67 | 1.53 | 1.20 | 0.96 | 1.31 | 2.02 | 0.41 | 0.31 | 0.20 | 0.20 |
| -0.19 | 0.29 | 0.12 | -0.35 | -0.34 | -0.51 | 1.28 | 1.37 | 0.54 | 1.54 |

↓

| C6~C1 | C6~C2 | C6~C3 | C6~C4 | C6~C5 | C6~M1 | C6~M2 | C6~M3 | C6~M4 |
|--------|--------|--------|--------|--------|--------|---------|--------|---------|
| 0.8964 | 0.9337 | 0.9973 | 0.9999 | 0.9998 | 0.0420 | -0.0504 | 0.0795 | -0.1422 |

FigureA.3 Results of the document vectors and the similarites.

We can see that document C6 fits in Ben's first preference well. Thus, C6 should be filtered in. If we use the average of the largest 5 values as the prediction, we will have *prediction* = 0.9654 .

REFERENCES

- [1] G. Salton, M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw Hill Publications, 1 edition, 1893.
- [2] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. *Journal of the Society for Information Science*, 41(6):391-407, 1990.
- [3] Landauer, T. K., Foltz, P. W., & Laham, D. Introduction to Latent Semantic Analysis. *Discourse Processes*, 25:259-284, 1998.
- [4] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An Algorithmic Framework for Predicting Collaborative Filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*. August 1999.
- [5] Breese, J., Heckerman, D. and Kadie, C. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*. San Francisco, CA: Morgan Kaufmann, 43-52. 1998.
- [6] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, etc. Combining Content-Based and Collaborative Filters in an Online Newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, August 1999.
- [7] Balabanovic, M. and Shaham, Y. Fab. Content-Based, Collaborative Recommendation. *CACM*. 40(3):66-72, March 1997.
- [8] Hans Paijmans. Smart Tutorial for beginners.
In <http://pi0959.kub.nl:2080/Paai/Onderw/Smart/hands.html>
- [9] ftp://ftp.cs.cornell.edu/pub/smart/smart_linux.tgz documents
- [10] Michael Berry, Theresa Do, Gavin O'Brien, Vijay Krishna Varadhan. SVDPACKC (version 1.0) Users' Guide. Technical Report UT/CS/93/194, Department of Computer Science, University of Tennessee, April 1993.
- [11] Nicholas J. Belkin, W. Bruce Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM*, 35(12):29-38, December 1992
- [12] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstorm, and John Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM CSCW'94 Conference on Computer Supported Cooperative Work*, 175-186, 1994
- [13] Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. In *Proceeding of the ACM Conference on Computer Supported Cooperative Work (CSCW)*, 1998

- [14] K.H. Packer and D. Soergel. The Importance of SDI for Current Awareness in Fields with Scatter of Information. *Journal of American Society of Information Science*, 30:125-135, 1979.
- [15] Iain Duff, Roger G. Grimes, and John G. Lewis. Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release I). Technical Report TR/PA/92/86, CERFACS, October 1992.
- [16] Joseph F. McCarthy and Theodore D. Anagnost. MUSICFX: An Arbiter of Group Preferences for Computer Supported Collaborative Workouts. *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work (CSCW '98)*, 363-372.
- [17] <http://www-pablo.cs.uiuc.edu/Project/SmartSpaces/SmartSpaceOverview2.htm>
- [18] Yan, T.W. and Garcia-Molina, H. SIFT- A Tool for Wide-Area Information Dissemination. *Proceeding of the 1995 USENIX Technical Conference*, 177-186
- [19] W.C. Hill, J.D. Hollan, D. Wroblewski, and T. McCandless. Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, 35(12):61-70, December 1992.
- [20] Curt Stevens. Automating the Creation of Information Filters. *Communications of the ACM*, 35(12):48, December 1992.
- [21] Morita, M. and Shinoda, Y. (1994), Information Filtering Based on User Behavior Analysis and Best Match Text Retrieval, *Proceedings of the 17th ACM Annual International Conference on Research and Development in Information Retrieval (SIGIR'94)*, Dublin, Ireland, Springer-Verlag, 272-281.
- [22] Phong Le and Makoto Waseda, [Curious Browsers](#), Major Qualifying Project MQP-DCB-9906, Computer Science Department, Worcester Polytechnic Institute, Spring 2000 (Advisors: Professor Dave Brown and Professor Mark Claypool).
- [23] Upendra Shardanand and Patti Maes. Social Information Filtering: Algorithm for Automating "Word of Mouth". *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, 210-217, 1995.
- [24] Rie Kubota Ando. Latent Semantic Space: Iterative Scaling Improves Precision of Inter-document Similarity Measurement. [SIGIR 2000](#): 216-223.
- [25] Paul S. Jacobs and Lisa F. Rau. Natural Language Techniques for Intelligent Information Retrieval. *11th International Conference on Research and Development in Information Retrieval*, 85-99, Grenoble, France, June 1998. Presses University de Grenoble.
- [26] Ashwin Ram. Interest-based Information Filtering and Extraction in Natural Language Understanding Systems. In *Proceedings of the Bellcore Workshop on High Performance Information Filtering*, Nov 1991.
- [27] Andrew Jennings, Hideyuki Higuchi. A User Model Neural Network for a Personal News Service. *User Modeling and User-Adapted Interaction*, 3(1):1-25, 1993.

[28] Bill Segall and David Arnold, Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching. In <http://elvin.dstc.edu.au/doc/papers/auug97/AUUG97.html>

[29] Sara Parsowith, Geraldine Fitzpatrick, Simon Kaplan and Bill Segall, Tickertape: Notification and Communication in a Single Line. In <http://elvin.dstc.edu.au/doc/papers/apchi98/apchi98.html>

[30] Furnas, G.W., Landauer, T.K., Gomez, L.M., & Dumais, S.T. Statistical Semantics: Analysis of the Potential Performance of Keyword Information Systems. *Bell System Technical Journal*, 62(6):1753-1806, 1983.