Feature Guided Automated Collaborative Filtering

Yezdezard Zerxes Lashkari

B.Tech., Computer Science and Engineering,
Indian Institute of Technology, Bombay,
May 1991

M.S., Computer Sciences,
University of Wisconsin, Madison,
June 1993

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
In Partial Fulfillment of the Requirements for the degree of
MASTER OF SCIENCE IN MEDIA ARTS AND SCIENCES
at the
Massachusetts Institute of Technology
September 1995

Signature of Author _____

Program in Media Arts and Sciences
25 July 1995

Certified By _____

Pattie Maes
Associate Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted By _____

Stephen A. Benton
Chairperson
Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Feature Guided Automated Collaborative Filtering

Yezdezard Zerxes Lashkari

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on 25 July 1995
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

Abstract

Information filtering systems have traditionally relied on some form of content analysis of documents to represent a profile of user interests. Such **content-based filtering** is generally ineffective in domains with diverse media types such as audio, video, and images, because machine-analysis of such media is hard. Recently, information filtering systems relying primarily on human evaluations of documents have been built. Such **automated collaborative filtering** (ACF) systems work by discovering correlations in evaluations of documents amongst users, and by using these correlations to recommend new documents. However, such systems rely on the implicit assumption that the domain can be partitioned in some logical way (such as all articles in a particular newsgroup), prior to applying the ACF algorithm. This assumption breaks down in broad domains, (such as all documents in the World Wide Web), where no natural partition of the document space exists. This thesis claims that content-based and automated collaborative filtering are complementary techniques, and the combination of ACF with some easily extractable features of documents is a powerful information filtering technique for complex information spaces. Furthermore, systems using such **feature guided automated collaborative filtering** (FGACF) techniques can provide users with a very powerful **information retrieval** metaphor - **personalized querying by example**. WEBHOUND, an FGACF based system for personalized World Wide Web document recommendations was built to demonstrate the technique and test its effectiveness.

Thesis Supervisor: *Pattie Maes*

Title: *Associate Professor of Media Arts and Sciences*

This work was supported in part by a grant from the **News In The Future** consortium and a fellowship from the **Motorola** Corporation

Feature Guided Automated Collaborative Filtering

Yezdezard Zerxes Lashkari

The following people served as readers for this thesis:

*Reader*_____

Mitchell Kapor
Adjunct Professor
Program in Media Arts and Sciences
Co-Founder, Electronic Frontier Foundation

*Reader*_____

Tim Berners-Lee
Director, World Wide Web Consortium
Research Scientist, MIT Laboratory for Computer Science

*Reader*_____

Michael Hawley
Assistant Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Director, Personal Information Architecture Group, MIT Media Laboratory

Acknowledgements

I'd like to begin by thanking Pattie Maes, an excellent advisor, mentor, and friend. This research has benefited immeasurably from her unnering instinct for what is feasible, her guidance as to the real research issues, and, most importantly, her absolute faith even when I felt that the problem was too hard.

The insightful comments and discussions with my three readers, Tim Berners-Lee, Mitchell Kapor, and Mike Hawley, helped improve this thesis greatly.

This work would have been well nigh impossible without the rich and chaotic atmosphere that I always look forward to in the Agents group. Bruce, for being a good friend and counsellor; Mike, for invaluable system help; Brad, for helping me crystallize the ideas that form the backbone of this thesis, as well as being a great friend and late night conversationalist; Wex, who helped find some of the bugs (uh - features) in the interface; Lenny with his insightful comments; Henry, David and Akira for their patience and willingness to use the system in its early stages, and Christie, for making everything run smoothly. Lastly, and most importantly, I'd like to thank Max for being a patient and a close friend, implementing an elegant ACF server kernel and helping me debug at a time when I wasn't familiar with the code. I've been told we're like brothers too - and the thought scares the heck out of me :).

A big thanks to my "other" family - the **bawas** - especially Teresa, Jangoo, and Hanoz for making this year a blast.

One of the hard problems addressed by this research was the clustering of item feature values. I'd like to thank Roz Picard, Kris Popat, Baback Moghaddam, Trevor Darrell, Aaron Bobick, Andy Wilson and Josh Tenenbaum from MIT, and my friend V G Narayan from the Harvard Business School for all giving me large chunks of their precious time and valuable suggestions on how to solve this problem. My UROPs, Shawn Bolan and Michelle Eng, deserve special thanks.

Innumerable individuals, students, faculty, and staff, made my two year stay at MIT and especially the Media Lab intellectually challenging and thoroughly enjoyable ! The list is far too long to name each one individually but you know who you are. Thank you all.

Yezdi Lashkari,
Cambridge, MA,
July 1995.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 Related Work 16

1.2.1 Information Filtering and Retrieval 16

1.2.2 WWW Resource Discovery Mechanisms 19

1.3 Research Contributions 20

1.4 WEBHOUND 21

1.5 Guide To This Thesis 23

2 Automated Collaborative Filtering **24**

2.1 The ACF Algorithm 24

2.2 Mathematical Formulation 25

2.3 Specific ACF Algorithms 27

2.3.1 The Pearson r Algorithm 27

2.3.2 The Mean Squared Distance Algorithm 28

2.4 Limitations of ACF 30

2.4.1 Broad Domains 30

2.4.2 WWW Specific Limitations 32

3 Feature Guided Automated Collaborative Filtering **33**

3.1 Partitioning the Space 33

3.2 Mathematical Formulation 35

3.2.1 FGACF Specific Notation 36

3.2.2 Distance Between Users I and J 37

3.2.3 Predicting a rating for an item 39

3.3 Reduction to Standard ACF 39

3.3.1 Distance Between Users I and J 40

3.3.2 Predicting a rating for an item 40
3.4 FGACF Clustering Metrics 41
3.4.1 The Problem 41
3.4.2 Simple Average 42
3.4.3 Gaussian Distribution - Sample Estimators 43
3.4.4 Gaussian Distribution - Population Estimators 45
3.5 Item-Item Similarity 46
3.5.1 Tversky's Ratio Similarity Metric 46
3.5.2 Vector Space Metric 48

4 A Generic FGACF Engine **49**
4.1 Design Requirements 49
4.2 The FGACF Engine 50
4.2.1 Compatibility with ACF Engine 51
4.2.2 Domain Independence of Architecture 51
4.2.3 Types of Features 52
4.2.4 Domain Dependent Configuration 52
4.2.5 FGACF Algorithm Specific Module 54
4.2.6 Clustering Feature Values 54

5 WEBHOUND: A Personalized WWW Document Filtering System **56**
5.1 WEBHOUND Interface Functionality 56
5.1.1 Rating Documents 57
5.1.2 WWW Document Recommendations 59
5.1.3 E-Mail Based Periodic Recommendation Notifications 60
5.1.4 Administrative Functionality 60
5.1.5 Sampling the database 61
5.2 WWW Domain Specific Module 61
5.2.1 WEBHOUND Feature Extraction 62

6 Experimental Results **65**
6.1 The Data 65
6.2 Experimental Methodology 65
6.3 Evaluation Criteria 66
6.4 Results 67
6.4.1 The Base Algorithm: Averages 68

6.4.2 ACF 69
6.4.3 FGACF - Simple Average Based Clustering 70
6.4.4 FGACF - Gaussian (Sample Estimator) Clustering 72
6.5 Discussion 73
6.5.1 Effects of Number of Feature Value Clusters 74
6.5.2 Effects of Irrelevant Features 74
6.5.3 Effects of Quantization of Numerical Features 75
6.5.4 Using User Extracted Features 75

7 Conclusions and Future Work **76**
7.1 Distributed WEBHOUND 76
7.2 Filtering Search Engine Query Results 78
7.3 Filtering Objectionable Material Flexibly 78

List of Figures

1.1 The WEBHOUND WWW Interface 22

2.1 ACF Drawbacks in Broad Domains 31

3.1 Partitioned Item Space for Feature Value fv_1 34
3.2 Partitioned Item Space for Feature Value fv_2 34

4.1 Structure of the FGACF server engine 50

5.1 WEBHOUND Document Rating Scale 57
5.2 WEBHOUND Subscriber Functions 58
5.3 Structure of the WWW Domain Specific Module 62

6.1 Distribution of Rating Values in Test Data Set 66
6.2 Error Distribution for Base Algorithm 68
6.3 Error Distribution for ACF 70
6.4 Error Distribution for FGACF (simple average clustering) 71
6.5 Error Distribution for FGACF (gaussian sample estimator clustering) 72

Chapter 1

Introduction

1.1 Motivation

The increasing availability of inexpensive computing power coupled with the rapid proliferation of computer networks and connectivity has created a revolution in terms of an average user's access to vast amounts of information. However, this ease of access to vast quantities of information has sharply exacerbated a growing problem of *personal information overload.* While almost any information required by a user probably exists on-line somewhere, it is next to impossible for the average user to locate it, or to keep track of new related information.

The numbers below are indicative of the magnitude of this problem for some of the more popular information systems available. The World Wide Web (WWW) [4] is a system of hyperlinked multimedia documents spanning the global Internet that allows any user with access to the Internet to create and link in their own multimedia documents. The WWW, started in 1990, had grown to over 4600 servers by October 1994, and over 10,000 servers by December 1994. Together, these servers make over 1.05 million documents available for retrieval [23]. Growth of Usenet Net News is also exponential. There were an estimated 2.6 million users of Net News at approximately 87,000 sites around the world in May 1993. These users generated over 26,000 new articles a day, amounting to around 57 MBytes of data [31]. Just over a year later, some estimates put the number of users at over 10 million.

This problem is worsening every day. More and more organizations are putting their information on-line in the form of files, databases and Lotus Notes. In addition to documents, a vast variety of products and services can now be purchased electronically

over the WWW. Most significantly, an increasingly smaller fraction of this information universe is accurate, up to date, and relevant to a particular user.

The solution lies in developing increasingly sophisticated and personalized tools that can help users in filtering these vast quantities of information, and better retrieval methods to locate specific types of items. However, most current information filtering and retrieval techniques are keyword based [34, 37, 41] and hence don't work too well in information spaces containing documents of different media types (video, audio, graphics, text). More importantly, even the best such *content-based* techniques cannot give an opinion on the *quality* of a retrieved item (an inherently subjective notion) as perceived by a particular user [1]. While 200 documents may all match a given query equally, a user may only perceive 33 of them as being of high quality. The challenge is to somehow capture this notion of quality of an item. This notion is especially important in domains where the perceived quality of items fluctuates very widely, such as the WWW.

Recently, information filtering systems relying on human evaluations of items have been built. Such **collaborative filtering systems** work by using the opinions of human users for items and attempting to match a user's opinion or query with that of other like minded users. There are two main kinds of collaborative filtering systems.

Active Collaborative Filtering systems, such as Xerox PARC's pioneering Tapestry system [11] allow their users to annotate any document with their comments or opinions. Users can then construct sophisticated queries to retrieve certain types of documents that any of their colleagues like. An alternative model of active collaborative filtering is described by Maltz and Ehlrich [23]. Their system for Lotus Notes allows users to pro-actively annotate and forward documents to other users (or groups of users).

Automated Collaborative Filtering (ACF) [9] is an innovative technique for locating items of potential interest to users in any domain using human evaluations of items. It relies on a deceptively simple idea: if person A correlates strongly with person B in rating a set of items, then it is possible to predict the rating of a new item for A, given B's rating for that item. Since ACF does not rely on computer analysis of items, it is especially useful for domains where it is either too hard (or too computationally expensive) to analyze items by computer, such as information spaces containing images,

[1] Most retrieval systems rank the results of a query by degree of "match" - the degree to which a particular document matches a particular query ; this is not the same as the quality of the document for a user making that query.

movies, audio, etc. Furthermore, since ACF relies on human evaluations of items, it can provide some notion of the quality of a retrieved item for a particular user.

Various approaches have already been implemented to make it easier to find information on the WWW. Most of them attempt to create some form of index of WWW documents, which users may then query. Such solutions are necessarily non user-centered and possess numerous drawbacks from the viewpoint of an ordinary user. The basic assumption behind all indexing schemes is that users will somehow learn of the existence of the index and can then query it effectively. With the growing number of indices, it is no longer possible, even for expert users, to keep track of all the useful indices. Further, WWW indices vary widely in quality, indexing mechanisms, and coverage. Hence, simply locating an index isn't enough: a user must know the correct set of keywords to locate relevant documents from that index [2]. Furthermore, most WWW indices do not attempt to index non-text WWW documents. More importantly, even the best index cannot tell users which of a bunch of documents matching their query, they are likely to consider of high enough quality to read.

ACF is an attractive technique for the WWW. However, ACF assumes that the domain of items is narrow enough or alternatively, can be partitioned effectively (for example, all articles in a particular USENET newsgroup [32]) to make the technique effective. When the domain is broad (like the WWW), or no logical partition exists, it is very unlikely that two users will correlate highly over the entire domain. It then becomes necessary to determine combinations of features for items in the domain for which two users actually correlate. Hence, two users may both give a high evaluation to a particular document for completely different reasons: one due to the fact that it came from the **media.mit.edu** server and mentions **intelligent agents** and **Pattie Maes** in the text, the other because it mentions **intelligent agents**, **has links to other documents on agents**, and **has over 20 inline images**. Such simple feature information is available for most domains, and could be used to make much better recommendations. In the example above, the ACF system could determine that the two users correlate on documents which contain **intelligent agents** in their text. This is important in broad domains where two users may strongly agree in their opinions for certain sets of documents, and strongly disagree for others.

This thesis presents a novel technique for information filtering that attempts to address the problems faced by both ACF and content-based approaches by combining

[2]This set may differ from index to index depending on what information was used to construct the index

the two to make use of their complementary strengths. The technique we present, *Feature Guided Automated Collaborative Filtering* (FGACF), uses easily extractable features of items to dynamically partition the domain and so allow ACF to be applied relative to a set of features. By restricting itself to subsets of the entire space, the FGACF algorithm can calculate correlations between users for different classes of documents and make better recommendations. In addition, users can now retrieve items using example-based queries of the form:

"Recommend more items like this one"

where the FGACF system can determine what **like this one** means for a particular user and item (based on what features of an item the user seems to find significant). Since FGACF relies on simple item features as guides for an ACF algorithm, computationally expensive content analysis of items is not necessary.

1.2 Related Work

1.2.1 Information Filtering and Retrieval

Information Filtering (IF) refers to the filtering of a dynamic information stream based on a long term profile of the user's interests created and maintained by the system [3]. Various personalized filtering systems automatically create and maintain a profile of user interests using various machine learning techniques [41, 18]. Information Retrieval (IR) refers to the retrieval of specific information objects from a (relatively static) database in response to a particular (transient) user query. IF implies a longer term information interest, while IR normally implies a current information need of a particular user. However, Belkin and Croft [3] argue convincingly that these techniques are simply two different ways of looking at the same problem: how to translate a user's information interests into a profile (or query) that can then be used to retrieve the desired information objects, and indeed, the document representations used by both disciplines are remarkable similar.

Malone [22] identifies three important kinds of filtering techniques: cognitive, social, and economic.

Content-Based Filtering

Cognitive or content-based filtering involves the filtering of information based on its content, and is the standard technique used by most textual information filtering systems [37, 41]. Keyword-based filtering is an example of content filtering.

The profile (or query), is normally based on a variant of one of three standard models for textual documents.

- A query in the boolean model consists of keyword strings combined with the standard Boolean operators [36].
- The vector space model treats profiles (and documents) as weighted vectors of terms, the weights reflecting the relative importance of the terms [34]. The degree of "match" between a query and a document is the magnitude of the dot product of the query and document vector.
- The probabilistic model [33] uses term occurrences and a word independence assumption to compute the probability of a document's relevance to a query.

Profiles are updated as a result of user feedback by a variety of techniques such as relevance feedback [35] and reinforcement learning [41, 3].

Recently, more sophisticated query models that attempt to find some sort of higher level structure in documents have been proposed. The Latent Semantic Indexing (LSI) model [8] uses singular value decomposition on the term-document matrix to map document representations to a smaller dimensionality "concept" space and then perform retrieval on that space. It is still unclear whether these techniques offer any significant increases in retrieval accuracy.

Collaborative Filtering

Social or collaborative filtering techniques select articles based on correlations between people's subjective judgements. Active collaborative filtering, pioneered by the Tapestry system [11], is a form of social filtering in which users annotate documents by hand, actively decide whose opinions they are interested in, *and program arbitrarily complex filters* that are then run continuously over a document store. For example, a typical Tapestry filter may be a query of the form: **find all articles in the newsgroup** *comp.unix-wizards* **with the keywords** *UNIX* **and** *BSD* **in the subject that** *John Doe* **replied to.** Tapestry places the burden of identifying users with similar interests and

programming the appropriate filters entirely on the user. Such a solution only works well in a small group of computer literate users.

Another complementary model of active collaborative filtering is that presented by Maltz and Ehlrich in their system for Lotus Notes [23]. In their system, as in Tapestry, users annotate documents with their opinions. Unlike Tapestry however, users of their system then actively send the message out to people (or groups of people) they think may be interested in the message. The recipients of these annotated messages can then filter and sort these messages by different annotations. It is instructive to note that both systems emphasize different aspects of active collaboration; Tapestry is built around a *pull* model of filtering, while the system described in [23] focuses on a *push* model. Active Collaborative Systems are most effective in small workgroups where users know other users' interests, and can use that knowledge effectively.

Automated Collaborative Filtering (ACF), by contrast, refers to the system automatically determining correlations amongst users in their evaluations of items, and using these correlations to recommend interesting items. The GroupLens system [32] uses this model in the domain of USENET netnews. The algorithm is applied separately to readers within each newsgroup. Users evaluate articles using modified newsreader client programs. Grouplens provides a scalable architecture for propagating user evaluations from the clients to ACF servers that process these evaluations. Grouplens has been undergoing testing for over a year; initial results with a very limited set of users are encouraging.

The RINGO system [39] built at the MIT Media Lab uses ACF for making personalized music recommendations. RINGO consisted of a single central server and did not partition the item space in any way [40]. The RINGO system was rewritten and extended using a generic ACF engine [27] and now runs as the HOMR (**H**elpful **O**n-line **M**usic **R**ecommendations) system. At the time of writing, HOMR had a user population of 13,229 users who can add new items to the database as well as submit reviews of groups and albums. HOMR does have a single fixed hierarchical partitioning of the item space.

While systems applying either content or social filtering techniques exist, to date, no system has attempted to effectively combine the two. One such system currently being implemented, is the NewsWeeder system [18] for USENET netnews. However, the emphasis in NewsWeeder is in attempting to combine ACF with content representations, so as to discover new machine representations for text, and how peer evaluations can guide the learning of these representations.

Economic Filtering

Economic filtering techniques pick documents based on the costs and benefits of producing and reading them. Hence mass mailings that have a low production cost per addressee are normally of lower priority than a personally addressed message.

1.2.2 WWW Resource Discovery Mechanisms

WWW Indexes

Most previous attempts to tackle the information overload problem for the WWW have attempted to construct a WWW index of some sort using a variety of approaches such as individual web-robots [26, 29, 25], collaborative swarms of *ants* such as the CMU WebAnts project [25], distributed ARCHIE like indexing [16], collaboratively-maintained, hierarchical bulletin-boards of WWW resources [26, 44], generalized resource discovery architectures such as Harvest [5], and meta-indices such as [6, 42].

By far the most popular method for indexing the WWW has been to build a web robot [3] [24, 29]. Martijn Koster [17] maintains a fairly upto date list of currently known web robots. Most robots start of from a particular root document, extract all the hyperlinks, follow these links (in some order) to their documents, extract all the links from these documents, and so on. The result of this web traversal is normally a database of hyperlinks indexed by keywords extracted from the title, and sometimes the full text. Since it typically takes a robot over three months to simply traverse the WWW [21], collaborative exploration solutions are being proposed, such as the CMU WebAnts project [25] which plans to use an army of communicating ants.

Non-robot based indexing mechanisms normally consist of a distributed indexing mechanism of cooperating server sites. ALIWEB [16] is a system for producing ARCHIE-like indexing of WWW sites and relies on cooperating servers periodically sending information to a central site. The Harvest architecture [5] is a generalized resource discovery architecture that consists of distributed gatherers and brokers all with the ability to interoperate to make the indexing problem more efficient and manageable. A Harvest broker maintains information about a specific type of resource (for example, all software sites in the UK). A centralized registry keeps track of all current brokers and gatherers so as not to duplicate effort. Other distributed mechanisms normally consist of some form of collaboratively-maintained, hierarchi-

[3] Alternatively referred to as web crawlers, spiders, ants etc.

cal, categories of WWW resources such as GenVL [26], Da-Clod [38], Yahoo [44] etc. Finally, various meta-indices of resources such as [6, 42] have been constructed.

User-Centric Retrieval Mechanisms

A few systems take a more user-centric approach to the information retrieval problem. The Infobot hotlist database collects the contents of various hotlists mailed to it, and periodically forms a list of most popular documents [28]. The SIMON system [15] distributed a package that allowed users to associate keywords with various resources in their WWW browser hotlist and retrieve their documents by simply specifying keywords. In return for using this software, users were requested to send their hotlists (and associated keywords) periodically to a central site, where the data is summarized. The hope is that each user's hotlist can be used to construct *local maps* of WWW space, which can then all be linked somehow (it is not clear exactly how), into a unified *global map* of the WWW. The Fish-search robot [7] integrated with Mosaic allows users to specify keywords to search for, in documents reachable from a particular document. A robot is then invoked which searches out from the start document looking for documents containing certain keywords.

A very interesting approach to personalized spider searching is the LIRA system being developed at Stanford [2] that combines a personalized profile built and modified using reinforcement learning techniques very similar to the NewT system [41], with a spider that compares all documents it retrieves in a specified CPU time limit against a user's profile. While this is a sophisticated spider, it cannot really capture the notion of quality; furthermore, the approach is quite compute resource intensive.

Two other interesting approaches attempt to determine patterns in a user's behaviour as they browse the WWW by continually monitoring all actions of the user. The agents then start making suggestions as to which links to follow next for a particular page. Letizia [20] and WebWatcher [1] are two such agents.

1.3 Research Contributions

The research in this thesis has resulted in the following unique contributions:

- A formal framework for combining content-based and automated collaborative filtering techniques to leverage off of the complementary strengths of both techniques. The filtering framework presented, Feature Guided Automated Collab-

orative Filtering (FGACF) is general enough to apply to any domain containing items which can be represented as a set of features.

- General techniques for clustering feature values of items in domains where the number of feature values is very large and there exists no a priori model to group these feature values (for example, keywords in the title of WWW documents). The clustering problem is non-trivial, since users only submit ratings for items (which consists of sets of feature values).

- A framework for information retrieval using personalized example-based queries. A nice result of the FGACF framework is that a user can ask the system to recommend new items *similar to* a particular item. The notion of similarity is personalized to each user based on the features they seem to find more important in items. Such example based querying is a very powerful retrieval paradigm especially in domains when users cannot articulate what sort of item they desire in terms that the computer system may understand [4].

- A proof that the body of algorithms used by ACF techniques are simply special cases of the algorithms used by FGACF techniques. That is, for the special case of one identical feature value per item, FGACF reduces to ACF.

- Development of a personalized WWW document filtering system that provides on-demand or periodic WWW document recommendations to users using FGACF. This system, WEBHOUND, also supports example-based querying.

- Development of a generic FGACF server for any domain [5]. This architecture is based on the generic ACF server architecture developed by Max Metral [27].

1.4 WEBHOUND

The FGACF framework developed in this thesis applies to any domain where simple feature information is available. However, an important contribution of this thesis has been the development of a concrete implementation of this framework for the domain

[4]Conversely, such a query paradigm frees the system implementor to use any sorts of features they feel are appropriate for a particular domain, without having to worry about the end-user understanding what these features mean.

[5]The WWW (WEBHOUND) is just a particular application domain to the architecture.

of WWW documents. WEBHOUND, the resulting system provides personalized WWW document recommendations to each of its users as well as personalized query by example functionality. Figure 1.1 shows the main page of the WWW Interface to WEBHOUND [6]. WEBHOUND is presented in greater detail in Chapter 5.

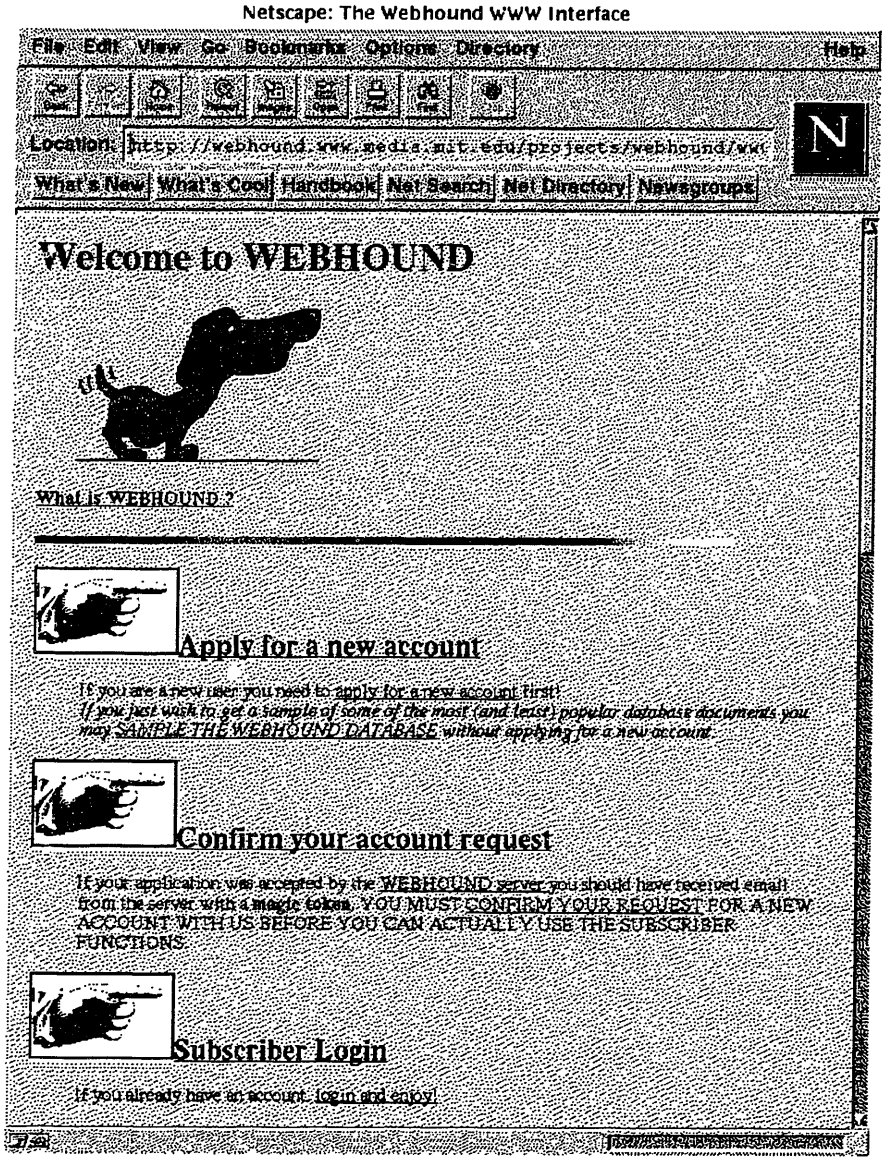

Figure 1.1: The WEBHOUND WWW Interface

[6] Accessible on-line at `http://webhound.www.media.mit.edu/projects/webhound/`

1.5 Guide To This Thesis

The rest of this thesis is arranged as follows. In Chapter 2 we present the ACF framework and then present some drawbacks of standard ACF in broad domains such as the WWW. Chapter 3 presents the FGACF framework, the item-item similarity framework and three schemes for clustering of feature values based on users' ratings of items in the domain. The reduction of FGACF to ACF is also presented. Chapter 4 presents the architecture of the generic FGACF server in terms of its structure, its relation to the generic ACF server architecture [27] and the domain specific parts that need to be added. In Chapter 5 we present WEBHOUND, a personalized WWW document filtering system built using the FGACF server. Chapter 6 presents the results of several experiments run on a subset of the data collected from WEBHOUND, and an analysis of those results. Finally, in Chapter 7 we conclude this report and identify promising directions for future work.

Chapter 2

Automated Collaborative Filtering

In this chapter, we present a formalization of the ACF algorithm. We then present detailed formulations for two ACF algorithms: the Pearson r and the Mean Squared Distance (MSD) ACF algorithms. Finally, we present some drawbacks of the standard ACF algorithm in broad domains.

2.1 The ACF Algorithm

Automated Collaborative Filtering algorithms exploit the similarities between the subjective tastes of different users in a particular domain to filter items in a personalized fashion for each user. They rely on the observation that if two people A and B share similar opinions about a number of items in a particular domain, and A likes a particular item that B hasn't rated, then B is probably likely to enjoy it too, and vice versa [1].

As an example, lets consider three friends, Arthur, Bob, and Catherine. Arthur and Catherine are avid Websurfers while Bob is just learning about the Web. Let us say that Bob asks his friends to recommend a particularly interesting Web document on his first exploration of the Web. Catherine recommends the Calvin and Hobbes Archive while Arthur recommends HotWired. Bob knows that Arthur and he share

[1] Vigilant readers will note that this observation is dependent on how broadly or narrowly we define a "domain". If the domain were all entertainment items in the universe, then the above will *most likely not hold* for any pair of users. If however, the domain were all action movies, then this observation will *most likely hold* between a pair of users.

similar tastes while Catherine and his tastes don't exactly match. Hence, Bob decides to follow Arthur's suggestions. This is an example of collaborative filtering.

ACF automates such a "word of mouth" process on a very large scale to make recommendations to users. However, instead of being constrained to just asking a few people for recommendations, the ACF algorithm essentially can search amongst thousands of users for people with similar tastes, and use their opinions to make recommendations to a particular user. Since the ACF algorithm is not constrained to only consider items a few trusted friends have recommended (as we are in daily life), the universe of items considered is much larger.

All ACF algorithms use the following steps in making a recommendation to a user:

1. **Construct a profile for a user**
   This profile normally consists of a user's ratings of some items in the domain. The ratings are normally captured on some numerical scale.

2. **Compare user's profile with profile of other users**
   Compare this profile to the profiles of all (or some subset of) the other users in the system and calculate a *similarity* between each pair of profiles compared. The actual algorithm used to determine similarity of user profiles may vary.

3. **Construct the set of nearest neighbors for this user**
   Take the $\mathcal{N}$ most similar user profiles for a particular user. These form this user's *nearest neighbors*. Weight each profile in the nearest neighbor set by the degree of similarity to the user profile.

4. **Use the Nearest Neighbor Set to Make Recommendations**
   Use the nearest neighbor set of weighted user profiles to calculate a *predicted rating* for a new item in the domain for the user. If the predicted rating exceeds a given threshold value, recommend this item to the user.

2.2 Mathematical Formulation

We now formalize the four steps. Before we do this however, we need to introduce some notation.

- $\mathcal{N}$ represents the maximum number of user profiles in a nearest neighbor set.
- R_{min} and R_{max} represent the minimum and maximum rating values respectively.

- $Similarity_{min}$ represents a minimum similarity value between a pair of profiles for users I and J for them to be neighbors.
- **User Indices** are uppercase subscripts: e.g. $User_I, User_J, User_K$
- **Item Indices** are lowercase subscripts: e.g. $Item_p, Item_q, Item_r$
- $R_{I,p}$ is user I's rating for item p.
  Note: $R_{I,p} \in \{R_{min}..R_{max}\}$
- $c_{I,p}$ indicates whether user I has rated item p.
  Note: $c_{I,p} \in [0, 1]$
- $P_{I,p}$ is our prediction of user I's rating for item p.
- The notation $|x|$ is used to denote the absolute value of x.
- The notation $||S||$ will be used to denote the number of elements in set S.
  For example $||Items||$ denotes the total number of items in the database.

The four step procedure of Section 2.1 can now be formalized as follows:

1. **Construct a profile for a user**
   The user profile for User I is the set of ordered pairs

$$U_I = \{< Item_p, R_{I,p} >\} \quad |c_{I,p} = 1 \tag{2.1}$$

2. **Compare user's profile with profile of other users**
   Define the similarity function $\theta(U_I, U_J)$ that returns a scalar similarity value between two user profiles U_I and U_J. The actual definition of $\theta(U_I, U_J)$ depends on the ACF algorithm used.

3. **Construct the set of nearest neighbors for this user**
   The neighbor set for a User I is the set of ordered pairs

$$Neighbors_I = \{< U_J, \theta(U_I, U_J) >\} \tag{2.2}$$

   where $\theta(U_I, U_J)$ lies in the top $\mathcal{N}$ similarity values between user I and any user J and

$$\theta(U_I, U_J) \geq Similarity_{min}$$

4. **Use the Nearest Neighbor Set to Make Recommendations**
   Calculate the predicted rating for user I for an item p by weighting the rating of all neighbors of this user who have rated this item.

$$P_{I,p} = \lambda(U_I) + \frac{\sum_{J=1}^{||Neighbors_I||} \kappa(R_{J,p}, Neighbors_I)) \times c_{J,p}}{\sum_{J=1}^{||Neighbors_I||} \alpha(R_{J,p}, Neighbors_I) \times c_{J,p}} \tag{2.3}$$

   where the **rating-weight combination function** $\kappa(R_{J,p}, Neighbors_I)$, the **prediction weighting function** $\alpha(R_{J,p}, Neighbors_I)$, and the **constant factor function** $\lambda(U_I)$ depend on the particular ACF algorithm used.

2.3 Specific ACF Algorithms

In this section we present two specific ACF algorithms - the Pearson r correlation and the Mean Squared Distance (MSD) ACF algorithms. These are by no means the only ACF algorithms possible - all sorts of variants of these as well as other techniques may be considered valid ACF algorithms. We present these two since the Pearson r algorithm will be used to illustrate some drawbacks of ACF in broad domains in Section 2.4. The MSD algorithm is presented in detail because it is the algorithm used in all the ACF experiments reported in this thesis, and will be used to demonstrate the reduction of FGACF to ACF in Chapter 3.

2.3.1 The Pearson r Algorithm

The Pearson r algorithm uses a standard Pearson r correlation coefficient to measure similarity between two user profiles U_I and U_J. The Pearson r correlation coefficient between two user profiles U_I and U_J is defined as:

$$r_{I,J} = \frac{Covariance(U_I, U_J)}{\sigma_I \times \sigma_J} \tag{2.4}$$

which can be rewritten as:

$$r_{I,J} = \frac{\sum_{p=1}^{||Items||}(U_I - \bar{U_{I,J}}) \times (U_J - \bar{U_{J,I}}) \times c_{I,p} \times c_{J,p}}{\sqrt{\sum_{p=1}^{||Items||}(U_I - \bar{U_{I,J}})^2 \times \sum_{p=1}^{||Items||}(U_J - \bar{U_{J,I}})^2 \times c_{I,p} \times c_{J,p}}} \tag{2.5}$$

where $\bar{U_{K,J}}$ represents the average rating for user K's profile with respect to user J's profile, and is calculated as:

$$\bar{U_{K,J}} = \frac{\sum_{p=1}^{||Items||} R_{K,p} \times c_{K,p} \times c_{J,p}}{\sum_{p=1}^{||Items||} c_{K,p} \times c_{J,p}} \tag{2.6}$$

Note that the value $\bar{U_{K,K}}$ simply represents the average rating for user K's profile. The value of $r_{I,J}$ always lies in the interval [-1, 1] where a negative r value indicates a negative correlation between two users, and a positive value indicates a positive correlation. An r value of 0 indicates no correlation. The greater the magnitude of $r_{I,J}$ the greater the similarity between the two users.

We can now define the functions in the general ACF formulation of Section 2.2 as follows:

$$\theta(U_I, U_J) = r_{I,J}$$

$$\lambda(U_I) = \bar{U_{I,I}}$$

$$\kappa(R_{J,p}, Neighbors_I) = r_{I,J} \times (R_{J,p} - \bar{U_{J,I}})$$

$$\alpha(R_{J,p}, Neighbors_I) = |r_{I,J}|$$

Using the equations above, the predicted rating for user I for item p is given as:

$$P_{I,p} = \bar{U_{I,I}} + \frac{\sum_{J=1}^{||Neighbors_I||}(r_{I,J} \times (R_{J,p} - \bar{U_{J,I}})) \times c_{J,p}}{\sum_{J=1}^{||Neighbors_I||} |r_{I,J}| \times c_{J,p}} \tag{2.7}$$

2.3.2 The Mean Squared Distance Algorithm

The MSD algorithm calculates a *distance* value between every pair of user profiles in the database. The greater the distance, the lower the similarity between any pair of users. The specific MSD definitions of the functions of the general ACF formulation of Section 2.2 are as follows:

$$\theta(U_I, U_J) = \frac{1}{D_{I,J}}$$

$$\lambda(U_I) = 0 \quad \forall I$$

$$\kappa(R_{J,p}, Neighbors_I) = Wt_{I,J} \times R_{J,p}$$

$$\alpha(R_{J,p}, Neighbors_I) = Wt_{I,J}$$

where $D_{I,J}$ and $Wt_{I,J}$ are defined below.

Since we will be using the MSD algorithm as our example ACF algorithm, we present a detailed formulation of the steps in the MSD ACF algorithm below.

1. **Distance Between Users I and J**

   The *distance* between any two users I and J is given by the following formula:

$$D_{I,J} = \frac{\sum_{p=1}^{||Items||} (R_{I,p} - R_{J,p})^2 \times c_{I,p} \times c_{J,p}}{\sum_{p=1}^{||Items||} c_{I,p} \times c_{J,p}} \tag{2.8}$$

2. **Calculate Neighbor Set**

   User J is a possible nearest neighbor of user I iff

$$D_{I,J} \leq \mathcal{L}$$

   where $\mathcal{L}$ is a maximum distance apart two users can be for them to be neighbors for each other. For each user I we store the $\mathcal{N}$ nearest neighbors [2].

3. **Calculate weight of each Neighbor**

   Weight is given by the formula:

$$Wt_{I,J} = \frac{\mathcal{L} - D_{I,J}}{\mathcal{L}} \tag{2.9}$$

   Note that since thresholding guarantees that $D_{I,J} \leq \mathcal{L}$, the value of $Wt_{I,J}$ always lies in [0,1].

4. **Calculate Predicted Rating for Item p**

   The predicted rating for item p is calculated as:

$$P_{I,p} = \frac{\sum_{J=1}^{||Neighbors_I||} Wt_{I,J} \times R_{J,p} \times c_{J,p}}{\sum_{J=1}^{||Neighbors_I||} Wt_{I,J} \times c_{J,p}} \tag{2.10}$$

[2]The **mmouth** code uses finer tuned thresholding. Equation 2.8 can be thought of as:

$$D_{I,J} = \frac{Numerator_{I,J}}{Denominator_{I,J}}$$

where

$$Numerator_{I,J} = \sum_{p=1}^{||Items||} (R_{I,p} - R_{J,p})^2 \times c_{I,p} \times c_{J,p}$$

and

$$Denominator_{I,J} = \sum_{p=1}^{||Items||} c_{I,p} \times c_{J,p}$$

Given this formulation, **mmouth** applies the following two additional checks to the distance value from Equation 2.8 above.

(a) The two users need to have a minimum number of common rated items $\mathcal{C}$. i.e.

$$Denominator_{I,J} > \mathcal{C}$$

(b) The fraction of the number of commonly rated items to the total number of rated items for user I must exceed a certain threshold $\mathcal{P}$. i.e.

$$\frac{Denominator_{I,J}}{\sum_{p=1}^{||Items||} c_{I,p}} > \mathcal{P}$$

2.4 Limitations of ACF

2.4.1 Broad Domains

ACF techniques are extremely powerful information filtering techniques in domains which are traditionally hard to analyze by computers (movie, images, audio etc.), inherently subjective (music, movies etc.), or in which there is a very wide fluctuation in the perceived quality of items (documents in the Web etc.) because they use humans as subjective editors instead of attempting to analyze the contents of each item in the domain by computer. However, it is precisely this utter independence from any analysis of the items in a domain that serves as a limitation in very broad domains (all documents in the Web, all magazines on a newsstand etc.)

Before we use a numerical example to demonstrate this fact, lets return to our extremely simple example of the three friends from Section 2.1. Although Bob and Arthur have similar tastes their common space of documents has been mainly technical ; Furthermore, while Bob and Catherine's tastes in reading material don't always match, they share a similar quirky sense of humor. Given these facts, Bob may well decide to visit the Calvin and Hobbes Archive page first.

The example above illustrates two important facts about broad domains.

1. In broad domains, any two users may share similar opinions about *certain combinations of feature values* possessed by items both of them have rated. In our example above, Bob and Arthur share similar opinions for technical articles, while Bob and Catherine are likelier to have the same opinions on documents dealing with humor.

2. In broad domains, a statement such as *two users share similar tastes* or *two users don't seem to have any common tastes* may be quite meaningless (and even wrong), since any two users will now have multiple sets of (certain kinds of) items they both strongly agree on, strongly disagree on, and share no opinion on. Taking a simple average across all the common items wipes out this information, as illustrated below.

To illustrate the two points above let's consider two users $User_1$ and $User_2$ and a domain where the items all posses a single feature that can take one of two values (denoted by fv_1 and fv_2). For the purpose of this example let the minimum rating

(R_{min}) be 1 and the maximum rating (R_{max}) be 7 [3]. Let us say that the two users have rated the first four items (I_1, I_2, I_3, and I_4) in common, as shown in Figure 2.1 below. We now need to predict a rating for $User_2$ for items I_5 and I_6.

	Items						...
Users	$Item_1$	$Item_2$	$Item_3$	$Item_4$	$Item_5$	$Item_6$	...
$User_1$	7	7	1	1	1	7	
$User_2$	1	7	7	1	???	???	
⋮							

Figure 2.1: ACF Drawbacks in Broad Domains

Using the Pearson r ACF formulation and Equation 2.6, the average ratings for the two users are

$$\bar{U_{1,2}} = \bar{U_{2,1}} = \bar{U_{1,1}} = \bar{U_{2,2}} = 4$$

Assume furthermore that all items with odd numbered indices in the table of Figure 2.1 ($Item_1, Item_3, Item_5$) possess the value fv_1 for the single feature, while all items with even numbered indices in the table ($Item_2, Item_4, Item_6$) possess the value fv_2 for the feature. Note that ACF doesn't use this information in any way.

Continuing with the mathematics, the Pearson r correlation coefficient between $User_1$ and $User_2$, using Equation 2.5, is

$$r_{2,1} = 0$$

Hence, using Equation 2.7, the predicted ratings for $User_2$ for both $Item_5$ and $Item_6$ are

$$P_{2,5} = P_{2,6} = \bar{U_{2,2}} = 4$$

(i.e. $User_2$'s average rating).

This however is not the correct answer. Glancing at the table of Figure 2.1, and keeping in mind the feature values possessed by each item reveals the following significant patterns:

- The two users *disagree perfectly* on their opinions for all items possessing the feature value fv_1.

[3] All the experimental results use this 1-7 scale for ratings.

- The two users *agree perfectly* on their opinions for all items possessing the feature value fv_2.

Using these observations, the predicted ratings for items $Item_5$ and $Item_6$ for user $User_2$ should be closer to the following values:

$$P_{2,5} = 7 \quad \text{Perfect Disagreement}$$

and

$$P_{2,6} = 7 \quad \text{Perfect Agreement}$$

2.4.2 WWW Specific Limitations

As demonstrated in Section 2.4.1 above, even a small amount of feature information can provide a more accurate picture of exactly how two users' opinions match in a domain. This is especially important in broad domains such as all documents in the Web. For almost any given set of documents any pair of users can be expected to have very similar, very dissimilar, and neutral opinions *for different sets of common documents.* Standard ACF algorithms cannot capture this sort of information ; *in fact they tend to destroy such information since they average across all common items between a pair of users.* Hence, the measure of opinion overlap they provide is flawed, especially in broad domains.

Related to this, broad domains such as the WWW possess an *overlap problem.* For an ACF algorithm to make a reasonably good prediction for a user, it needs to use a weighted sum of the opinions of a number of users (nearest neighbors) who have rated a sufficient number of items in common with that particular user. However, due to the extreme sparsity of the ratings matrix, the average overlap of WWW documents rated in common by any pair of users can be very low, thus affecting the accuracy of the ACF predictions. In a study at Bellcore [12], researchers *found the average overlap to be no greater than 3 % of all documents rated by any user, even for users with similar interests.* This is normally not enough to make good predictions. The WEBHOUND system explicitly provides mechanisms to increase the overlap of rated documents between users and encourages users to explicitly use these mechanisms.

Chapter 3

Feature Guided Automated Collaborative Filtering

In this chapter, we present Feature Guided Automated Collaborative Filtering (FGACF). FGACF is a novel technique that uses simple feature information to dynamically partition the item space for each user, so as to apply an ACF algorithm more effectively. We begin with an example to demonstrate the use of feature information to partition the item space. We then present the FGACF formulation for the Mean Squared Distance (MSD) algorithm and show how it reduces to the ACF formulation for MSD in the special case of one identical feature for every item in the domain. We next present issues involved in clustering feature values in domains containing large numbers of feature values and with no a priori model of similarity between feature values and introduce three possible clustering metrics. We conclude this chapter by presenting the personalized item-item similarity metric that forms the basis of the *query by example* retrieval mechanism.

3.1 Partitioning the Space

Let us return briefly to the example of Section 2.4.1. Standard ACF predicted a rating of 4 for both items for the user $User_2$. Let us now use the feature information and partition the space of items into two distinct sets - those containing items having the value fv_1, and those containing items having the value fv_2 [1]. To predict a rating for

[1]The rationale for this partitioning is that since we know a particular item is *like* another (in terms of the features it possesses), it makes sense to use only the rating information for similar items.

a particular item, we use only the information of the item set it falls into. Hence, our original problem of predicting the rating for items $Item_5$ and $Item_6$ can be thought of as two separate problems:

1. Predicting a rating for $User_2$ for $Item_5$ using the rating information for $Item_1$ and $Item_3$.

2. Predicting a rating for $User_2$ for $Item_6$ using the rating information for $Item_2$ and $Item_4$.

We construct the following two tables from the table of Figure 2.1.

	Items			...
Users	$Item_1$	$Item_3$	$Item_5$	...
$User_1$	7	1	1	
$User_2$	1	7	???	
⋮				

Figure 3.1: Partitioned Item Space for Feature Value fv_1

	Items			...
Users	$Item_2$	$Item_4$	$Item_6$	...
$User_1$	7	1	7	
$User_2$	7	1	???	
⋮				

Figure 3.2: Partitioned Item Space for Feature Value fv_2

Let us now try to predict the rating for $Item_5$ for $User_2$ using the information in the partitioned space from Figure 3.1. From Equation 2.6, the average ratings for the two users are

$$\bar{U_{1,2}} = \bar{U_{2,1}} = \bar{U_{2,2}} = 4$$

From Equation 2.5, we have

$$r_{2,1} = -1.0$$

Using Equation 2.7, the predicted rating is

$$P_{2,5} = 4 + \frac{-1.0 \times (1-4)}{|-1.0|} = 7$$

Similarly, for $Item_6$ using only the information in Figure 3.2

$$\bar{U_{1,2}} = \bar{U_{2,1}} = \bar{U_{2,2}} = 4$$

Using Equation 2.5, we have

$$r_{2,1} = +1.0$$

and from Equation 2.7, the predicted rating is

$$P_{2,5} = 4 + \frac{+1.0 \times (7-4)}{|+1.0|} = 7$$

As the simple example above illustrates, knowing something about the items enables us to partition the item space so that the ACF algorithm can be applied effectively. In most interesting domains, an a priori partition of the space may not exist ; even if it does, it may be better to let the partition fall out of the data then impose an artificial one.

It is our claim that simple feature information of items combined with aggregate ratings of items in the database can be used to form effective dynamic partitions of the item space in a computationally tractable way. The technique we present, feature guided automated collaborative filtering (FGACF), combines simple feature information with user ratings to create such dynamic partitions of the item space and allow ACF to work effectively.

3.2 Mathematical Formulation

In this section we present a mathematical formulation for the FGACF technique for the specific case of the Mean Squared Distance (MSD) algorithm [2].

The idea behind the the FGACF algorithm is that users don't necessarily correlate on the item level but rather for certain combinations of values of features of these items. Thus the FGACF algorithm treats each item as consisting of a set of of *feature values* for a set of *features* defined in the domain.

Is is important for the reader to understand the difference between features and feature values. A feature is a certain characteristic of the item: (e.g. the *keywords in*

[2]Since all the results in Chapter 6 use the MSD algorithm formulation.

the title of a document is a feature in a document based domain). Feature values on the other hand, refer to values a certain feature can take for an item. For example, for the document titled "The MIT Media Laboratory", the keywords *MIT, Media* and *Laboratory* are **feature values** for the feature **title keywords** *for that particular document.*

Because some domains (like WWW documents) may easily have hundreds of thousands of feature values in the database for certain features (e.g. keywords in the body of documents), we approximate the ideal FGACF by clustering feature values into a fixed set of *feature value clusters* per feature. *Thus each item is now treated as consisting of a set of feature value clusters for the set of features in the domain.*

The quality of the clustering is obviously a big factor in the accuracy of FGACF. The clustering metrics used will be presented in Section 3.4. *However, when the number of feature values is small, each feature value forms its own cluster.* Hence we will use the term *cluster* to refer to these clusters throughout although the reader should keep in mind that a cluster for some features will consist of single feature values [3]. For example, WEBHOUND quantizes the feature *number of inline images in document* to one of four possible values: NONE, 1..5, 6..20, and ABOVE 20. This feature is always allocated four clusters (one feature value per cluster) - hence values for this feature are never clustered. On the other hand, the number of possible feature values for the feature *keywords in the body text* numbers in the hundreds of thousands. Clustering of these keywords into a fixed set of clusters is hence necessary.

3.2.1 FGACF Specific Notation

Since the FGACF algorithm operates on a vector of distances/correlations/weights (one for each cluster of each feature defined) we need to define the following extra notation to operate on these vectors.

- **Feature Indices** are denoted by Greek letter superscipts. E.g. $Item_p^{\alpha}$ refers to feature α of item p.

- **Feature Value Cluster Indices** are denoted by lowercase subscripts of their corresponding feature index. E.g. $Item_p^{\alpha_x}$ refers to cluster x of feature α of item p.

[3] In this case clustering is obviously not an issue.

- $\gamma_p^{\alpha_x}$ is a Boolean operator that indicates whether item p contains the feature value cluster x of feature α.

$$\gamma_p^{\alpha_x} \in [0,1]$$

- τ^{α_x} is a Boolean operator on a vector of values and indicates whether feature value cluster x for feature α of the vector contains a value that is defined. For a vector $\vec{V}$

$$\tau^{\alpha_x}(\vec{V}) \in [0,1]$$

3.2.2 Distance Between Users I and J

The FGACF algorithm computes a vector of *distances* between every pair of users (one value per feature value cluster). The values in this vector are then combined to come up with a single distance value. Each entry $\vec{D}_{I,J}^{\alpha_x}$ of the vector $\vec{D}_{I,J}$ is calculated as follows:

$$\vec{D}_{I,J}^{\alpha_x} = \begin{cases} \frac{\sum_{p=1}^{||Items||}(R_{I,p}-R_{J,p})^2 \times c_{I,p} \times c_{J,p} \times \gamma_p^{\alpha_x}}{\sum_{p=1}^{||Items||} c_{I,p} \times c_{J,p} \times \gamma_p^{\alpha_x}} & \text{iff } \sum_{p=1}^{||Items||} c_{I,p} \times c_{J,p} \times \gamma_p^{\alpha_x} \geq 1 \\ Undefined & \text{otherwise} \end{cases} \quad (3.1)$$

In addition, we define the following two concepts:

1. **Cluster Weights**

   A *cluster weight* for each feature value cluster is calculated for each user based on their ratings. The cluster weight is an indication of how important a particular user seems to find a particular feature value (cluster). The vector of cluster weights is defined as:

$$C\vec{W}'^{\alpha_x}_I = \begin{cases} \frac{\sum_{p=1}^{||Items||} R_{I,p} \times c_{I,p} \times \gamma_p^{\alpha_x}}{\sum_{p=1}^{||Items||} c_{I,p} \times \gamma_p^{\alpha_x}} & \text{iff } \sum_{p=1}^{||Items||} c_{I,p} \times \gamma_p^{\alpha_x} \geq 1 \\ 0.0 & \text{otherwise} \end{cases} \quad (3.2)$$

   Since, the number of clusters allocated per feature need not be the same, each value $C\vec{W}'^{\alpha_x}_I$ is then normalized to a value $C\vec{W}^{\alpha_x}_I$ such that

$$\sum_{x=1}^{||\alpha_x||} C\vec{W}^{\alpha_x}_I = 1.0$$

   (Note that $(C\vec{W}'^{\alpha_x}_I = 0.0) \Rightarrow (C\vec{W}^{\alpha_x}_I = 0.0)$)

2. **Feature Weights**

While the *cluster weights* provide an indication of the relative importance of each feature value cluster *for a feature*, so far we are still implicitly treating all features as equally important to every user. This is almost never the case. To account for the fact that users invariably consider certain features more important than others in evaluating items, we define the notion of a *feature weight*, which reflects the importance of that feature relative to the other features for a particular user.

One method of estimating a relative weight is to use the following argument: *"If the variance in the normalized cluster weights for a particular feature is low, then that feature is probably not considered too important by the user"*. The rationale behind the statement being the fact that all the feature value clusters have approximately the same importance to the user.

Using the above argument, we calculate the unnormalized feature weights as:

$$\vec{FW'}_I^{\alpha} = \frac{StandardDev\ (\vec{CW}_I^{\alpha})}{Mean\ (\vec{CW}_I^{\alpha})} \tag{3.3}$$

and then normalize the values of $\vec{FW'}_I^{\alpha}$ to the values $\vec{FW}_I^{\alpha}$ such that

$$\sum_{\alpha=1}^{||FeaturesDefined||} \vec{FW}_I^{\alpha} = 1.0$$

Note: This is just one possible way to weight the features per user. Another way may be to simply weight all the features equally. i.e.

$$\forall \alpha \quad \vec{FW}_I^{\alpha} = \frac{1}{||FeaturesDefined||}$$

The **distance** between two users I and J is calculated as:

$$D_{I,J} = \sum_{\alpha=1}^{||FeaturesDefined||} \vec{FW}_I^{\alpha} \times (\sum_{\alpha_x=1}^{||\alpha||} \vec{D}_{I,J}^{\alpha_x} \times \vec{CW}_I^{\alpha_x} \times \tau^{\alpha_x}(\vec{D}_{I,J})) \tag{3.4}$$

As in standard ACF, user J is a possible nearest neighbor of user I iff

$$D_{I,J} \leq \mathcal{L}$$

Note that an alternative way to think of a distance *between users is to discard the idea of absolute neighbors for a user, and calculate a set of neighbors* relative to the set of feature values possessed by the item whose rating we are trying to predict. *For various computational reasons, this was not implemented in the engine - however, it is a more intuitive notion of distance given the FGACF framework.*

3.2.3 Predicting a rating for an item

To predict a rating for an item, for each neighbor J of user I that has rated the item, the algorithm calculates a *weight* and multiplies the neighbor's rating by this weight.

The predicted rating for item p for user I is calculated as follows:

1. **Calculate a per item distance between Users I and J**
   The distance between users I and J for item p is:

$$d_{p,I,J} = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \left(\sum_{\alpha_x=1}^{||\alpha||} \vec{D}_{I,J}^{\alpha_x} \times C\vec{W}_I^{\alpha_x} \times \tau^{\alpha_x}(\vec{D}_{I,J}) \times \gamma_p^{\alpha_x} \right) \quad (3.5)$$

2. **Calculate a per item weight for Neighbor J**
   The neighbor weight for neighbor J of user I for item p is:

$$Wt_{p,I,J} = \begin{cases} \frac{\mathcal{L} - d_{p,I,J}}{\mathcal{L}} & \text{iff } d_{p,I,J} \leq \mathcal{L} \\ 0.0 & \text{otherwise} \end{cases} \quad (3.6)$$

3. **Calculate Predicted Rating for Item p**
   The predicted rating for item p is calculated as:

$$P_{I,p} = \frac{\sum_{J=1}^{||Neighbors_I||} Wt_{p,I,J} \times R_{J,p} \times c_{J,p}}{\sum_{J=1}^{||Neighbors_I||} Wt_{p,I,J} \times c_{J,p}} \quad (3.7)$$

3.3 Reduction to Standard ACF

Standard ACF happens to be a very special case of FGACF when the number of features defined on the domain is 1 and each item has the same value for that one feature. In that special case FGACF reduces to standard ACF as shown below.

The special case results in the following tautologies:

1. The total number of features defined is 1.
2. The total number of feature value clusters defined is 1. i.e. $\forall_\alpha \ \sum_{\alpha_x=1}^{||\alpha||} 1 = 1$
3. This feature value cluster is always present for all items. i.e. $\forall_p \ \gamma_p^{1_1} = 1$
4. The vectors in equations 3.1 and 3.2 reduce to scalars. Hence

$$\forall_{I,J} \ \tau^{1_1}(\vec{D}_{I,J}) = 1$$

5. The value of the per user cluster weight is 1.0 after normalization. i.e.

$$\forall_I \ \vec{CW}_I^{1_1} = 1.0$$

6. The value of the per user feature weight is 1.0 after normalization. i.e.

$$\forall_I \ \vec{FW}_I^{1} = 1.0$$

3.3.1 Distance Between Users I and J

Using the results above Equation 3.1 is transformed to

$$\vec{D}_{I,J}^{1_1} = \frac{\sum_{p=1}^{||Items||} (R_{I,p} - R_{J,p})^2 \times c_{I,p} \times c_{J,p}}{\sum_{p=1}^{||Items||} c_{I,p} \times c_{J,p}} \tag{3.8}$$

Using the fact that $\forall_I \ \vec{CW}_I^{1_1} = 1.0$ and $\forall_I \ \vec{FW}_I^{1} = 1.0$, and substituting values in equation 3.4, we get:

$$D_{I,J} = \vec{D}_{I,J}^{1_1} \tag{3.9}$$

which is identical to the way the standard ACF distance is computed (Equation 2.8).

Hence user J is a possible nearest neighbor of user I iff

$$D_{I,J} \leq \mathcal{L}$$

3.3.2 Predicting a rating for an item

1. **Calculate a per item distance between Users I and J**
   Using the results above, the per-item distance is identical to the user distance. i.e.

$$d_{p,I,J} = D_{I,J}$$

2. **Calculate a per item weight for this Neighbor**
   Using the results above, the per-item neighbor weight is identical to the standard ACF user weight (Equation 2.9). i.e.

$$Wt_{p,I,J} = Wt_{I,J}$$

3. **Calculate Predicted Rating for Item p**
   Hence, the predicted rating for item p is

$$P_{I,p} = \frac{\sum_{J=1}^{||Neighbors_I||} Wt_{I,J} \times R_{J,p} \times c_{J,p}}{\sum_{J=1}^{||Neighbors_I||} Wt_{I,J} \times c_{J,p}}$$

   which is identical to the standard ACF formulation for the MSD algorithm (Equation 2.10).

3.4 FGACF Clustering Metrics

3.4.1 The Problem

Ideally, we'd like to be able to calculate a vector of distances between every pair of users, one for each possible feature value defined in the database. For various reasons this may not be possible (or desirable).

- In some domains, the number of possible feature values is very large. For example, for the WWW document domain (WEBHOUND) the number of unique keywords extracted from the body of the documents numbers over 50,000 for just 5,000 documents [4]. Clearly, some form of clustering is needed.
- Clustering may be desirable even if the computational resources to deal with a large space of feature values may be available, since the distribution of feature values is extremely sparse. Clustering may help discover similarities amongst feature values that are not immediately obvious.

We assume that we have no a priori model of similarity between feature values in most domains. We need to define a similarity metric between feature values that falls out of the users' ratings. The problem of course is that users don't actually rate individual feature values. Hence, while a metric for clustering items (or users) is obvious, one for clustering feature values of items is not.

Notation and Formalization of Problem

Given the set

$$FV^{\alpha} = FV_1^{\alpha}, FV_2^{\alpha}, ..., FV_{||FV^{\alpha}||}^{\alpha}$$

of feature values for feature α, the problem is to cluster the elements of the set FV^{α} into $\mathcal{K}$ exclusive partitional clusters.

Most standard partitional clustering algorithms require the points to be clustered to be represented as vectors in some metric space. Given this fact, a distance function Δ can be defined for any two points in the space as well as a vector combination function Ω that combines any two vectors in the space to produce a third point in the space that in some way represents the average of the points [5].

[4] This is after keyword stemming and selecting only the best 15 keywords per document

[5] This is used to compute the centroids of the clusters by the clustering algorithm.

Let us assume each feature value FV_x^α can be represented by a corresponding vector

$$\vec{FV}_x^\alpha = < \vec{v}_I^{\alpha_x} > \quad \forall I \in ||Users||$$

in some metric space. Furthermore, we let the operator $\Gamma_p^{\alpha_x}$ indicate the presence or absence of feature value FV_x^α in Item p.

$$\Gamma_p^{\alpha_x} \in [0, 1]$$

Below we present three possible formulations of the vector element $\vec{v}_I^{\alpha_x}$ and the corresponding Δ and Ω functions.

3.4.2 Simple Average

We assume that the dimensionality of each of the vectors is equal to the number of users in the database. i.e.

$$\mathcal{M} = ||Users||$$

That is, each user forms a separate (hopefully orthogonal) dimension of *opinions* on a particular feature value [6].

We define the notion of *similarity* between any two feature values as how similarly they have been rated by the same user, across the whole spectrum of users and items. Of course users don't directly rate feature values - but they do rate items which are composed of sets of feature values.

Hence we define the value $\vec{v}_I^{\alpha_x}$ to be the mean of the ratings given to each item containing feature value FV_x^α that user i has rated. Using the notation of Sections 2.2 and 3.2.1:

$$\vec{v}_I^{\alpha_x} = \begin{cases} \frac{\sum_{p=1}^{||Items||}(R_{I,p} \times c_{I,p} \times \Gamma_p^{\alpha_x})}{\sum_{p=1}^{||Items||}(c_{I,p} \times \Gamma_p^{\alpha_x})} & \text{iff } (\sum_{p=1}^{||Items||} c_{I,p} \times \Gamma_p^{\alpha_x}) \geq 1 \\ Undefined & \text{Otherwise} \end{cases} \tag{3.10}$$

The distance metric used is a standard Euclidean distance metric adapted to handle missing values [7]. Let the operator $\eta_I^{\alpha_x}$ indicate whether $\vec{v}_I^{\alpha_x}$ is defined.

$$\eta_I^{\alpha_x} = \begin{cases} 1 & \text{if } (\sum_{p=1}^{||Items||} c_{I,p} \times \Gamma_p^{\alpha_x}) \geq 1 \\ 0 & \text{Otherwise} \end{cases} \tag{3.11}$$

[6] In real life some of these users may be correlated but we ignore that fact for now.

[7] Since our vectors are sparse.

The per-user dimension squared distance between vectors $\vec{FV}_x^{\alpha}$ and $\vec{FV}_y^{\alpha}$ for user dimension I is:

$$\delta_I^{\alpha_{x,y}} = \eta_I^{\alpha_x} \times \eta_I^{\alpha_y} \times (\vec{v}_I^{\alpha_x} - \vec{v}_I^{\alpha_y})^2 \tag{3.12}$$

The total distance between the two feature value vectors is:

$$\Delta(\vec{FV}_x^{\alpha}, \vec{FV}_y^{\alpha}) = \sqrt{(\frac{||Users||}{\sum_{I=1}^{||Users||} \eta_I^{\alpha_x} \times \eta_I^{\alpha_y}}) \times (\sum_{I=1}^{||Users||} \delta_I^{\alpha_{x,y}})} \tag{3.13}$$

where, the term

$$\frac{||Users||}{\sum_{I=1}^{||Users||} \eta_I^{\alpha_x} \times \eta_I^{\alpha_y}}$$

represents the adjustment for missing data.

The combination function for two vectors is:

$$\Omega(\vec{FV}_x^{\alpha}, \vec{FV}_y^{\alpha}) = < \omega_I^{\alpha_{x,y}} > \forall_{I \in ||Users||} \tag{3.14}$$

where

$$\omega_I^{\alpha_{x,y}} = \begin{cases} \frac{(\vec{v}_I^{\alpha_x} + \vec{v}_I^{\alpha_y})}{2} & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 1 \\ \vec{v}_I^{\alpha_x} & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 0 \\ \vec{v}_I^{\alpha_y} & \text{if } \eta_I^{\alpha_x} = 0 \text{ and } \eta_I^{\alpha_y} = 1 \end{cases} \tag{3.15}$$

3.4.3 Gaussian Distribution - Sample Estimators

An objection raised against the simple average metric presented in Section 3.4.2 is that with a large number of vectors and user dimensions, the averaging done wipes out all the interesting information and the clusters formed may be quite meaningless as a consequence.

Furthermore, the simple average euclidean distance ignores the number of sample points used to arrive at the mean value. Intuitively if a value was arrived at with a larger number of sample points and has a lower variance it is likely to be more accurate than one arrived at from a smaller sample size or with a larger variance.

If we assume that the number of values used to compute $\vec{v}_I^{\alpha_x}$ in Equation 3.10 above is "sufficiently large" (on the average) then we can use the Central Limit Theorem to claim that whatever the underlying distribution from which $\vec{v}_I^{\alpha_x}$ was generated, we can approximate it by a Gaussian distribution.

Since the Gaussian distribution can be effectively characterized by its mean, variance and sample size, each entry $\vec{v}_I^{\alpha_x}$ is now a triplet:

$$\vec{v}_I^{\alpha_x} = < \mu_I^{\alpha_x}, \sigma^{2}{}_I^{\alpha_x}, N_I^{\alpha_x} >$$

where

$$\mu_I^{\alpha_x} = \frac{\sum_{p=1}^{||Items||}(R_{I,p} \times c_{I,p} \times \Gamma_p^{\alpha_x})}{\sum_{p=1}^{||Items||}(c_{I,p} \times \Gamma_p^{\alpha_x})}$$

is the sample mean of the population,

$$\sigma^{2}{}_I^{\alpha_x} = \frac{\sum_{p=1}^{||Items||}((R_{I,p} - \mu_I^{\alpha_x})^2 \times c_{I,p} \times \Gamma_p^{\alpha_x})}{\sum_{p=1}^{||Items||}(c_{I,p} \times \Gamma_p^{\alpha_x})}$$

is the variance of the sampling distribution, and

$$N_I^{\alpha_x} = \sum_{p=1}^{||Items||}(c_{I,p} \times \Gamma_p^{\alpha_x})$$

is the sample size.

Since each entry $\vec{v}_I^{\alpha_x}$ now represents a gaussian distribution the functions Δ and Ω change to reflect that.

The per-user dimension squared distance between vectors $F\vec{V}_x^{\alpha}$ and $F\vec{V}_y^{\alpha}$ for user-dimension I is

$$\delta_I^{\alpha_{x,y}} = \frac{(\eta_I^{\alpha_x} \times \eta_I^{\alpha_y}) \times (\mu_I^{\alpha_x} - \mu_I^{\alpha_y})^2}{\sigma^{2}{}_I^{\alpha_{x,y}} + 1.0} \tag{3.16}$$

where $\sigma^{2}{}_I^{\alpha_{x,y}}$ is the variance of the difference of two gaussian sample means and is calculated as:

$$\sigma^{2}{}_I^{\alpha_{x,y}} = \frac{\sigma^{2}{}_I^{\alpha_x}}{N_I^{\alpha_x}} + \frac{\sigma^{2}{}_I^{\alpha_y}}{N_I^{\alpha_y}} \tag{3.17}$$

This scheme for calculating $\delta_I^{\alpha_{x,y}}$ favors the means generated with a smaller variance. The 1.0 in the denominator of Equation 3.16 is added for the case when the value of the variance $\sigma^{2}{}_I^{\alpha_{x,y}}$ is 0.

Analogous to Equation 3.13, the total distance between the two feature value vectors is:

$$\Delta(F\vec{V}_x^{\alpha}, F\vec{V}_y^{\alpha}) = \sqrt{(\frac{||Users||}{\sum_{I=1}^{||Users||}\eta_I^{\alpha_x} \times \eta_I^{\alpha_y}}) \times (\sum_{I=1}^{||Users||}\delta_I^{\alpha_{x,y}})} \tag{3.18}$$

The feature value vector combination function Ω combines the corresponding triplets from the two vectors by treating them as gaussians. Hence Ω can be written as:

$$\Omega(F\vec{V}_x^{\alpha}, F\vec{V}_y^{\alpha}) = \begin{cases} < \mu_I^{\alpha_{x,y}{}'}, \sigma^{2}{}_I^{\alpha_{x,y}{}'}, N_I^{\alpha_{x,y}{}'} > & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 1 \\ < \mu_I^{\alpha_x}, \sigma^{2}{}_I^{\alpha_x}, N_I^{\alpha_x} > & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 0 \\ < \mu_I^{\alpha_y}, \sigma^{2}{}_I^{\alpha_y}, N_I^{\alpha_y} > & \text{if } \eta_I^{\alpha_x} = 0 \text{ and } \eta_I^{\alpha_y} = 1 \end{cases} \tag{3.19}$$

where

$$\mu_I^{\alpha_{x,y}'} = \frac{(N_I^{\alpha_x} \times \mu_I^{\alpha_x}) + (N_I^{\alpha_y} \times \mu_I^{\alpha_y})}{(N_I^{\alpha_x} + N_I^{\alpha_y})}$$

represents the mean of the new population,

$$\sigma^{2\alpha_{x,y}'}_I = (\frac{(N_I^{\alpha_x} \times \sigma^{2\alpha_x}_I) + (N_I^{\alpha_y} \times \sigma^{2\alpha_y}_I)}{(N_I^{\alpha_x} + N_I^{\alpha_y})}) + (\frac{(N_I^{\alpha_x} \times N_I^{\alpha_y}) \times (\mu_I^{\alpha_x} - \mu_I^{\alpha_y})^2}{(N_I^{\alpha_x} + N_I^{\alpha_y})^2})$$

represents the variance of the combined population, and:

$$N_I^{\alpha_{x,y}'} = (N_I^{\alpha_x} + N_I^{\alpha_y})$$

is the new sample size.

3.4.4 Gaussian Distribution - Population Estimators

When the sample size of the population is small, the variance of the sampling distribution $\sigma^{2\alpha_x}_I$ is not an accurate indicator of the underlying population variance [8]. A more accurate estimator of the population variance is given by the term:

$$S^{2\alpha_x}_I = \frac{\sum_{p=1}^{||Items||}((R_{I,p} - \mu_I^{\alpha_x})^2 \times c_{I,p} \times \Gamma_p^{\alpha_x})}{(\sum_{p=1}^{||Items||}(c_{I,p} \times \Gamma_p^{\alpha_x})) - 1}$$

where $S^{2\alpha_x}_I$ is known as the *sample variance* and is an accurate estimator of the underlying population variance.

Hence, we redefine the operator $\eta_I^{\alpha_x}$ of Equation 3.11 as:

$$\eta_I^{\alpha_x} = \begin{cases} 1 & \text{if } (\sum_{p=1}^{||Items||} c_{I,p} \times \Gamma_p^{\alpha_x}) > 1 \\ 0 & \text{Otherwise} \end{cases} \tag{3.20}$$

and the triplet of Section 3.4.3 to be the triplet:

$$\vec{v}_I^{\alpha_x} = < \mu_I^{\alpha_x}, S^{2\alpha_x}_I, N_I^{\alpha_x} >$$

where $\mu_I^{\alpha_x}$ and $N_I^{\alpha_x}$ are defined exactly as in Section 3.4.3, and:

$$S^{2\alpha_x}_I = \frac{\sum_{p=1}^{||Items||}((R_{I,p} - \mu_I^{\alpha_x})^2 \times c_{I,p} \times \Gamma_p^{\alpha_x})}{N_I^{\alpha_x} - 1}$$

[8] In statistical terminology, the variance of the sampling distribution $\sigma^{2\alpha_x}_I$ is a *biased estimator* of the true population variance.

The sample variance and the variance of the sampling distribution for a finite population are related by the following relationship:

$$\sigma^2 = (\frac{N-1}{N}) \times S^2 \tag{3.21}$$

Using Equation 3.21 above, Equation 3.17 is transformed to:

$$\sigma_I^{2^{\alpha_{x,y}}} = (\frac{N_I^{\alpha_x} - 1}{(N_I^{\alpha_x})^2}) \times S_I^{2^{\alpha_x}} + (\frac{N_I^{\alpha_y} - 1}{(N_I^{\alpha_y})^2}) \times S_I^{2^{\alpha_y}} \tag{3.22}$$

Analogous to Equation 3.19, the feature value vector combination function is defined as:

$$\Omega(\vec{FV}_x^{\alpha}, \vec{FV}_y^{\alpha}) = \begin{cases} < \mu_I^{\alpha_{x,y}'}, S_I^{2^{\alpha_{x,y}'}}, N_I^{\alpha_{x,y}'} > & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 1 \\ < \mu_I^{\alpha_x}, S_I^{2^{\alpha_x}}, N_I^{\alpha_x} > & \text{if } \eta_I^{\alpha_x} = 1 \text{ and } \eta_I^{\alpha_y} = 0 \\ < \mu_I^{\alpha_y}, S_I^{2^{\alpha_y}}, N_I^{\alpha_y} > & \text{if } \eta_I^{\alpha_x} = 0 \text{ and } \eta_I^{\alpha_y} = 1 \end{cases} \tag{3.23}$$

where $\mu_I^{\alpha_{x,y}'}$ and $N_I^{\alpha_{x,y}'}$ are defined as before and:

$$S_I^{2^{\alpha_{x,y}'}} = (\frac{N_I^{\alpha_{x,y}'}}{N_I^{\alpha_{x,y}'} - 1}) \times \sigma_I^{2^{\alpha_{x,y}'}}$$

and $\sigma_I^{2^{\alpha_{x,y}'}}$ is defined as in Equation 3.19.

3.5 Item-Item Similarity

The FGACF representation of an item as a set of feature values allows the application of various feature-based similarity metrics between items. The feature value clustering presented in Section 3.4 above allows us to perform *fuzzy feature matching* - i.e. two items may not share any identical feature values and still be considered quite similar to each other if they share some feature value clusters. While this kind of similarity is less obvious in some cases, its generality makes it powerful in domains where the distribution of feature values is extremely sparse.

3.5.1 Tversky's Ratio Similarity Metric

Tversky's seminal paper [43] shows that similarity between two items p_1 and p_2, where P_1 and P_2 represent the corresponding sets of feature values possessed by these items, can be represented as some function F of the following three sets:

1. The number of common feature values shared by the two items

2. The number of feature values that p_1 possesses that p_2 doesn't.

3. The number of feature values that p_2 possesses that p_1 doesn't.

Thus, the similarity between the two items, denoted by $S(p_1, p_2)$ is represented as:

$$S(p_1, p_2) = F(P_1 \bigcap P_2, P_1 - P_2, P_2 - P_1) \tag{3.24}$$

Tversky presents various general forms for the **matching function** F of Equation 3.24 above. The model currently implemented by the FGACF server code is the ratio model:

$$S(p_1, p_2) = \frac{f(P_1 \bigcap P_2)}{f(P_1 \bigcap P_2) + \alpha f(P_1 - P_2) + \beta f(P_2 - P_1)} \tag{3.25}$$

$$\alpha,\ \beta \geq 0$$

where the similarity value is normalized so that S lies between 0 and 1, f is a non-negative scale, and S and f are interval scales.

The ratio model generalizes several set-theoretical models of similarity proposed in the cognitive science literature.

There are obviously many ways to define the scale f. We enumerate some possibilities, and the reasons for choosing them, below.

Feature Value Cluster Matches Only

If we treat each item as a vector of feature value clusters, f for user I is defined as:

$$f(P_1 \bigcap P_2) = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \sum_{\alpha_x=1}^{||\alpha||} (C\vec{W}_I^{\alpha_x} \times \gamma_{p_1}^{\alpha_x} \times \gamma_{p_2}^{\alpha_x})$$

$$f(P_1 - P_2) = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \sum_{\alpha_x=1}^{||\alpha||} (C\vec{W}_I^{\alpha_x} \times \gamma_{p_1}^{\alpha_x} \times (1 - \gamma_{p_2}^{\alpha_x}))$$

$$f(P_2 - P_1) = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \sum_{\alpha_x=1}^{||\alpha||} (C\vec{W}_I^{\alpha_x} \times (1 - \gamma_{p_1}^{\alpha_x}) \times \gamma_{p_2}^{\alpha_x})$$

This metric is personalized to each user since, the feature weights and cluster weights reflect the relative importance of a particular feature value for a user.

Feature Value Cluster and Exact Feature Value Matches

A problem with the definition of f above is that two pairs of items may wind up having the same similarity values even though one pair has numerous *identical feature values* while the other has none. If two items share a number of identical feature values, we'd like them to be considered more similar than if they don't. To capture this notion, we redefine $f(P_1 \bigcap P_2)$ as:

$$f(P_1 \bigcap P_2) = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \left(\sum_{\alpha_x=1}^{||\alpha||} (C\vec{W}_I^{\alpha_x} \times \gamma_{p_1}^{\alpha_x} \times \gamma_{p_2}^{\alpha_x}) + \sum_{i=1}^{||FV^{\alpha}||} (\Gamma_{p_1}^{\alpha_x} \times \Gamma_{p_2}^{\alpha_x}) \right)$$

Since we add the extra terms to both the numerator and denominator of Equation 3.25, the similarity value is still guaranteed to lie in the range $[0, 1]$. Furthermore, using the identity:

$$\frac{a}{b} \leq \frac{(a+c)}{(b+c)}$$
$$\forall \ a, b, c > 0$$

this definition for $f(P_1 \bigcap P_2)$ guarantees that a pair of items with some identical feature values will always have a greater similarity then a pair having the same feature value clusters as the original pair but no identical feature values.

3.5.2 Vector Space Metric

Instead of using the ratio model, we can treat each item as a vector of feature value clusters (or alternatively as a vector of feature values), and then compute the weighted dot product of the two vectors. Using the notation of Section 3.5.1:

$$S(p_1, p_2) = g(P_1 \bigcap P_2) \tag{3.26}$$

Note that this definition of similarity is guaranteed to be symmetric - i.e.

$$S(p_1, p_2) = S(p_2, p_1)$$

If we treat each item as a vector of feature value clusters, the function g for user I is defined as:

$$g(P_1 \bigcap P_2) = \sum_{\alpha=1}^{||FeaturesDefined||} F\vec{W}_I^{\alpha} \times \sum_{\alpha_x=1}^{||\alpha||} (C\vec{W}_I^{\alpha_x} \times \gamma_{p_1}^{\alpha_x} \times \gamma_{p_2}^{\alpha_x})$$

Note that this value is always guaranteed to lie in the range $[0, 1]$.

Chapter 4

A Generic FGACF Engine

This chapter presents the design of the generic FGACF server engine. The WEBHOUND WWW document filtering system presented in Chapter 5 is built using this generic engine as a core. Readers not interested in the technical details can skip directly to Chapter 5.

4.1 Design Requirements

The design of the FGACF engine was influenced by several important requirements.

- **Compatibility with generic ACF server**
  The architecture had to be compatible with the generic ACF server architecture [27] on top of which it was built. That is, any new versions to the generic ACF server architecture would still work smoothly with any version of the FGACF server architecture.

- **Complete domain independence**
  The FGACF engine only deals with **features**, **feature values**, **users**, and **items**. Although features and feature values are domain specific, the engine simply treats them as database identifiers.

- **Ease of domain definition**
  It should be easy for a new domain to be defined and implemented by a programmer. All it currently takes is the specification of a feature definition file defining some information about the features in a domain, and the definition of

a **feature extraction** function that is called by the FGACF engine code to extract features from an item.

- **Flexibility of clustering schemes**
  Since FGACF relies on clustering of feature values, domain implementors must be given the flexibility of choosing whichever clustering schemes they feel are most appropriate for each of the features defined. The engine currently provides three types of clustering schemes for feature values.

4.2 The FGACF Engine

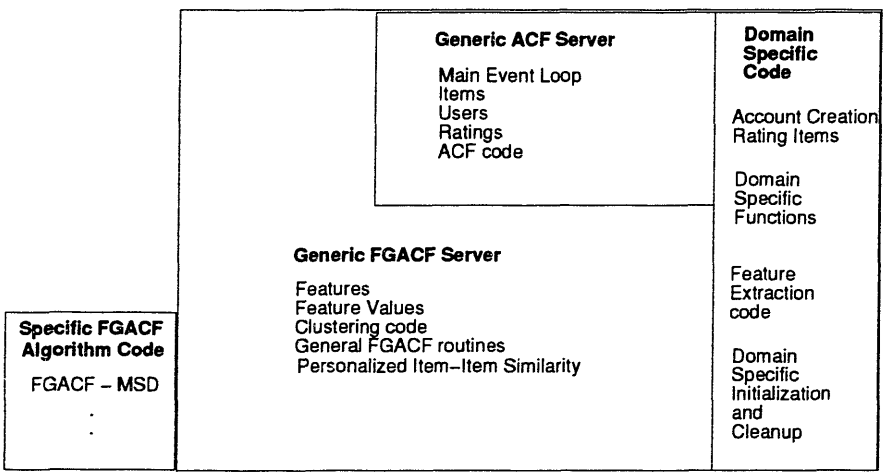

Figure 4.1: Structure of the FGACF server engine

Figure 4.1 shows the structure of the generic FGACF server engine. The core of the engine is the generic ACF server designed by Max Metral [27]. This contains the main server event loop as well as definitions for the databases containing user, item and rating information. The general FGACF engine module surrounds this core and provides the abstraction of features, feature values and general clustering routines for feature values, as well as personalized item-item similarity code. Interfacing with both the ACF and FGACF modules are domain specific routines such as initialization, cleanup and feature extraction routines. A clean separation between FGACF algorithm specific (MSD, Pearson r etc) code and general FGACF functionality is maintained.

4.2.1 Compatibility with ACF Engine

The FGACF engine implementation uses the following mechanisms to maintain consistency with the generic ACF engine architecture [27].

- **Identical Command Interface**
  The command interface for the FGACF server is identical to the ACF server. Commands are four character acronyms sent via a socket connection to the server by any external process. The command protocol is a request-reply protocol. In addition, all the commands recognized by the ACF server are supported.

- **Subclasses of ACF code classes**
  The ACF code defines various classes of objects. FGACF classes defined are subclassed off of the corresponding ACF classes (for example, the FGACFUser class is a subclass of the ACFUser class). This allows the code the flexibility to choose which functionality (ACF or FGACF) to use without any duplication of code.

- **Consistency of Recommendation Interface**
  When asked for n recommendations for a particular user, the ACF server normally returns $2 \times n$ items with a predicted rating for each. The convention followed is that the first n are the actual recommendations (highest n predicted ratings), while the next n are items that the user may wish to avoid (lowest n predicted ratings). To preserve consistency of interface this is also the behavior of the FGACF server.

4.2.2 Domain Independence of Architecture

The FGACF server makes absolutely no assumptions about the domain it is making recommendations for. The server deals with **users** and **items**. The domain has a fixed set of **features** defined at startup. Each of the items is treated as a set of **feature values**. The feature values for each item are extracted by a domain specific **feature extraction routine**.

Feature values for each feature are clustered into a number of **feature value clusters** by the server, where both the type of clustering as well as the number of clusters desired can be configured on a per-feature basis for the domain. These clustered feature values are used to form **cluster bit vectors** for each item in the database ;

the bits that are turned on corresponding to a cluster for a particular feature value present in that item.

4.2.3 Types of Features

Features can be classified along various dimensions.

Item Extracted and User Provided Features

1. **Item Extracted Features**
   These are features that are inherently present in the item and can be extracted from the item at any time. **Keywords in the title** of a WWW document are an example of an item extracted feature.

2. **User Provided Features**
   These are features whose value for a particular item is provided by the users of the system. **User annotations** of a particular WWW document are an example of a user-extracted feature. Obviously, the feature value for such features for an item is dependent on when the feature values are extracted *and the feature values for a particular item may change over time.* While the FGACF engine has provisions for such features, at the current time, we have no good mathematical model for how to deal with these types of features.

Number of Feature Values Per Item

Certain types of features are guaranteed to only contain a single value per item. The **server domain** of the WWW server serving a particular WWW document is an example of such a *singleton valued feature.* Other features generally have a set of values present per item. The **keywords in the title** of a document is an example of such a *multiple valued feature.*

The current FGACF formulation makes absolutely no distinction between these two types of features.

4.2.4 Domain Dependent Configuration

To configure the FGACF engine for a new domain, the following three tasks need to be completed:

1. **Create a feature definition file**

   The FGACF engine parses a domain specific feature definition file at startup. The feature definition file contains the following information per feature defined: the feature name, a textual description of what this feature corresponds to, whether the feature is item extracted or user provided, the type of clustering function to be used for feature values of this feature, and the number of feature value clusters (partitions) desired for values of this feature.

2. **Define Feature Extraction Function**

   The FGACF engine expects the domain dependent code to provide a function, which when given an item can extract values for the features defined and return a list of feature value identifiers corresponding to the feature values present in that item. To allow for all sorts of domains, this function is required to act in one of three modes:

   (a) **Initialize Feature Value Counts Only**

   This mode requires the function to initialize whatever tables it may need to extract the appropriate features for the item.

   (b) **Extract Features Only**

   This mode requires the function to perform the actual feature value extraction for the item. It is assumed that the function has been called previously in Initialize mode on that item.

   (c) **Initialize Counts and Extract Features**

   Combine the two steps above.

   The reason for requiring this sort of flexibility is because certain types of features require a sweep through the entire item database to initialize tables before they are able to actually extract feature values from any item. Keywords in documents are an example of such a feature. The first pass initializes the term frequency and document frequency tables for every possible feature value and item. The extraction pass then chooses only the best keywords (in terms of highest term frequency inverse document frequency product) as feature values for that document.

3. **Define Domain Dependent Commands**

   While the FGACF engine implements a set of domain independent commands,

domain dependent commands that the server understands must still be defined by the designer of the server for a particular domain. In addition the engine provides hooks to link in domain specific initialization and cleanup functions which are called at server startup and shutdown respectively.

4.2.5 FGACF Algorithm Specific Module

Since FGACF refers to a whole body of algorithms, the FGACF server is designed to run one of a number of FGACF algorithms. The algorithm to be run can be specified at startup. Currently, only the MSD based FGACF algorithm presented in Section 3.2 is implemented. However, hooks to link in any number of specific FGACF algorithms exist in the code. As shown in Figure 4.1, the FGACF engine code itself has been deliberately partitioned so that any dependencies on a specific FGACF algorithm are abstracted away in the main engine code.

4.2.6 Clustering Feature Values

Clustering feature values of any feature is an extremely important part of the FGACF algorithm. The FGACF architecture was designed to provide domain implementors the ultimate flexibility in choosing what sort of clustering scheme they feel best suits a particular feature. The engine code itself provides three general kinds of clustering functions for features where no a priori model of feature value similarity exists:

1. **Average Based Clustering**
   This corresponds to the distance formulation for feature values presented in Section 3.4.2.

2. **Gaussian Based Clustering**
   This corresponds to the distance formulation for feature values presented in Section 3.4.3.

3. **Fixed Clustering**
   This is not really a clustering metric. Fixed clustering is used for features where the total number of feature values is always less than (or equal to) the total number of feature value clusters allocated to that feature. In other words, each feature value cluster contains exactly one feature value.

Because the number of feature values to be clustered is often too large to fit in memory, the FGACF engine performs several clever optimizations in feature value pattern representation to do the clustering. Furthermore, an *incremental reclustering* is performed periodically to assign any new feature values to the best possible clusters (shortest distance from the centroid of the feature value cluster). At longer intervals, *full reclustering* needs to be performed to make sure that the clustering is not too out of date.

The actual clustering is performed using a standard K-Means partitional clustering algorithm [14] adapted to handle missing data values.

Chapter 5

WEBHOUND: A Personalized WWW Document Filtering System

This chapter introduces WEBHOUND, a Personalized WWW Document Filtering System using the FGACF engine presented in Chapter 4. We begin by presenting an overview of WEBHOUND's interface functionality. In the next section we present the domain specific module of Figure 4.1 as implemented for WEBHOUND, with details on the features of WWW documents used, the use of external libraries to perform specific tasks such as WWW document retrieval and parsing, URL canonicalization, and keyword stemming.

5.1 WEBHOUND Interface Functionality

WEBHOUND is currently accessible to any user with a WWW browser through a WWW interface [1]. A screen shot of the *subscriber functions* page of the WEBHOUND WWW interface is shown in Figure 5.2.

Users need to create an account with the WEBHOUND server before they can access its functionality. The WEBHOUND WWW Interface provides a subscriber with numerous functions described below.

[1] Available on-line at `http://webhound.www.media.mit.edu/projects/webhound/`

5.1.1 Rating Documents

Users rate documents on a seven point scale. The scale is the one used by the generic ACF server code and was chosen for consistency. The scale represents the *"interestingness"* of a particular WWW document to a user. A scale that we provide as a guideline is reproduced below in Figure 5.1.

```
7: Top notch! I WANT more documents like this!
6: Very interesting
5: Good stuff
4: Fair to middling
3: Doesn't do me
2: Not my type of document
1: BORING!
```

Figure 5.1: WEBHOUND Document Rating Scale

In addition to associating a numerical rating with documents, users can associate their personal annotations (personalized keywords) to any document if they choose. While the current set of features does not include these user annotations of documents, we hope to use them as features soon.

Because typing in a long URL for a document is *not* the way most users want to rate documents, WEBHOUND provides a variety of mechanisms for users to quickly tell it about documents they have opinions on. The important ones are listed below.

Hotlist Snarfing

Most WWW browsers provide a *hotlist* or *bookmark* facility for documents that a particular user may visit frequently (and hence finds interesting or useful or both). Because each user's hotlist represents a very detailed personalized profile of their tastes, WEBHOUND allows users to paste their entire contents of their hotlist/bookmark text files into a window, associate a default rating with all the documents in the list, and submit the entire list with a single button click. WEBHOUND then attempts to parse all the URLs out of the list [2], and verify that actual documents corresponding

[2]Current Hotlist bookmark formats the hotlist parser recognizes are **Mosaic (up to Version 2.4)**, **Netscape Bookmarks**, and **General HTML Lists**.

to the URLs exist for any new URLs. The documents parsed successfully are then returned to the user with any previous ratings and annotations the user may have given them or the default rating if a previous rating did not exist. The user can then change the ratings of individual documents, annotate individual documents and/or choose to have some documents removed from their list of ratings.

Hotlist snarfing is an easy way to quickly inform WEBHOUND about a large set of personally relevant documents. However, for WEBHOUND to do a good job a user needs to also rate a few documents that other users have rated.

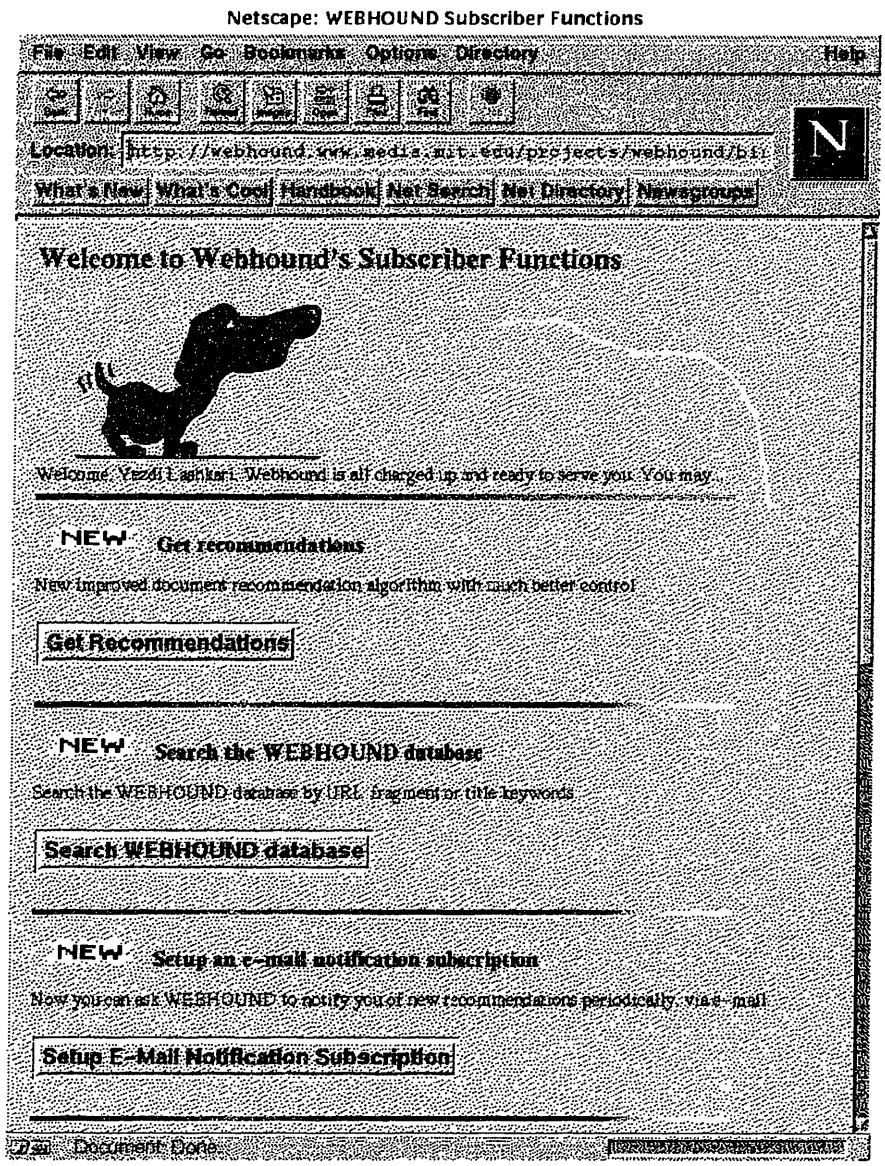

Figure 5.2: WEBHOUND Subscriber Functions

Rating Documents Based on Other Users Ratings

WEBHOUND provides various options for new users to rate documents already in the database that other users have rated according to some criterion. This is one way for users to help avoid the *WWW overlap problem* mentioned in Section 2.4.2. The three criteria currently provided to users are the following:

1. **Most Often Rated Documents**
   Users can rate documents that have been rated by the most number of users but not by them.

2. **Most Recently Rated Documents**
   Users can rate documents that have been rated most recently by other users.

3. **Personalized Selection**
   WEBHOUND selects documents that most of a user's nearest neighbors have rated that that user hasn't.

Rating Documents Matching a Search Criterion

Users can search the WEBHOUND database for documents containing a particular URL fragment (for example, the URL fragment `media.mit.edu` matches all documents in the WEBHOUND database residing on any web server at the MIT Media Laboratory) or by keywords in the title (for example, the search keyword **movie** matches all documents having the keyword **movie** in their title) and rate them.

5.1.2 WWW Document Recommendations

WEBHOUND provides users three types of personalized recommendation mechanisms. For each of these options, users can customize the minimum number of neighbors who must have rated a recommended document and/or the maximum number of recommendations returned, although reasonable defaults are provided.

Recommendations using ACF Algorithm

Users can ask WEBHOUND to recommend documents using simple ACF. This is normally faster than an FGACF based recommendation request but the recommendations returned may not necessarily be that good.

Recommendations using FGACF Algorithm

Users can ask WEBHOUND to recommend documents using FGACF. This is slower than a simple ACF based recommendation request but the recommendations are of better quality.

Recommendations for documents "similar" to a particular document

Users can ask WEBHOUND to recommend documents *similar* to a particular document. The similarity is computed using variants of the formulas presented in Section 3.5. Users can ask for *"like this one"* recommendations for a document they found matching a certain criterion by searching the database, or for a document that was recommended to them by WEBHOUND.

WEBHOUND users can also ask WEBHOUND to explain why it considers a certain document to be similar to another one. The explanation facility attempts to condense the mathematics of the similarity formula into three lines of English, followed by a more detailed explanation for interested users.

5.1.3 E-Mail Based Periodic Recommendation Notifications

Rather than visiting the WEBHOUND WWW page regularly for document recommendations, users can choose to set up a *document recommendation notification subscription* with the WEBHOUND server. Users can choose one of a set of four options for the frequency of the notifications as well as the minimum and maximum number of recommendations wanted. WEBHOUND then notifies each user having such a subscription of any new document recommendations periodically, *via e-mail.* The current notification periods users can choose are one of

1. Daily notification
2. Weekly notification
3. Fortnightly notification
4. Monthly notification

5.1.4 Administrative Functionality

Various other administrative functions are also provided to users.

Setting and Changing Account Password

Users can choose to protect their accounts using a password of their choice.

Viewing or Changing Old Ratings

As users taste change, they may wish to change some of their old ratings for some documents. Furthermore, most users may wish to periodically review the list of their ratings that WEBHOUND knows about to check that it still fits in with what they are interested in.

5.1.5 Sampling the database

Since the documents in the WEBHOUND database are provided entirely by its user community, users (as well as people not currently subscribed to WEBHOUND) can sample the current database to get some idea of the interests of the user community for a particular WEBHOUND server. The database can be sampled in one of the following two ways:

1. **Average Rating**
   Documents currently having the highest (and lowest) average ratings.

2. **Number of Ratings**
   Documents currently having the highest (and lowest) number of ratings.

Although WEBHOUND currently consists of a centralized server, the idea is to provide a distributed scalable architecture of a federation of WEBHOUND servers all over the internet. It should be trivial for anyone possessing the computational resources to set up a WEBHOUND server to serve either a particular community or anyone desiring to join the user population. In such cases, it becomes crucial to provide prospective new users with a way to roughly gauge the interests of the community around a particular WEBHOUND server. The database sampling functions are a step in that direction.

5.2 WWW Domain Specific Module

As mentioned earlier, the WWW is simply another application domain to the FGACF server code. Figure 5.3 shows the structure of this module for the WWW domain.

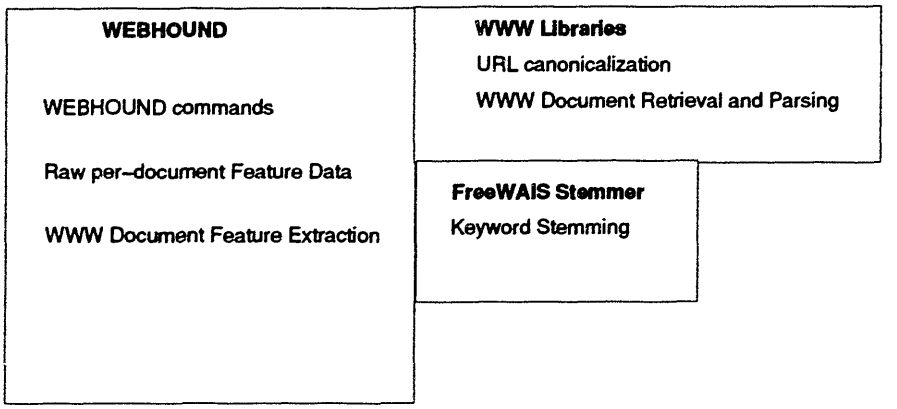

Figure 5.3: Structure of the WWW Domain Specific Module

The module consists of three distinct parts: general WEBHOUND commands and feature extraction routines that interface with the FGACF server, code to retrieve WWW documents and parse them for features that utilizes the CERN WWW Library of Common Code [10], and an efficient keyword stemming routine that was extracted from the FreeWAIS code distribution that uses the Porter Suffix Stripping Algorithm [30] to stem keywords.

5.2.1 WEBHOUND Feature Extraction

The module marked "WEBHOUND" in Figure 5.3 above, contains the code to interface with the FGACF server. It implements all the WEBHOUND specific commands used to communicate with the WEBHOUND server along with domain specific initialization and cleanup routines.

The WEBHOUND module implements a feature extraction routine as required by the FGACF server interface, which, when given an item (WWW document) can extract the relevant features from it and return a list of features and corresponding feature values for that item.

For all candidate feature values for a *keyword type* feature of a document, WEBHOUND follows the following steps to determine whether the candidate is a valid feature value for that document:

1. Stem candidate keyword using the FreeWAIS stemmer.

2. Check stemmed keyword is not a member of a list of *stopwords*. Stopwords are words that commonly occur in most textual documents (such as **the**, **and**, **an**

etc.) and should not be treated as keywords.

3. Compute the *product of term frequency and inverse document frequency* for that keyword for that document. The *term frequency* of a particular keyword term in a document is the number of times that term occurs in that document. The *document frequency* of a particular keyword term is the number of documents in which that term appears in a particular database of documents. The *inverse document frequency* is the reciprocal of the document frequency. The product of the two frequencies (also known as the *tfidf product* for that keyword) is a standard information retrieval measure of the effectiveness of a particular keyword term as a search term for a document [37].

4. Order stemmed keywords in descending order of *tfidf product* value.

The features currently used by WEBHOUND are

- **Title Keywords**
  The top 5 keywords in the title text (if any) of the document.

- **Body Keywords**
  The top 15 keywords in the body text (if any) of the document.

- **Anchor Keywords**
  The top 10 keywords in the anchor text (if any) of the document.

- **Server Domain**
  The domain of the web server on which the document resides.

- **Number of Inline Images**
  The number of inline images in the document is extracted and one of the following four possible values is used as the feature value: NONE, 1..5, 6..20, and ABOVE 20.

- **Number of Hypertext Links**
  The number of hypertext links in the document is extracted and one of the following four possible values is used as the feature value: NONE, 1..5, 6..20, and ABOVE 20.

The reason for allowing only a small number of feature values per document (only 15 body text keyword terms) is twofold. The size of the feature value database is

relatively manageable which makes clustering tractable. More importantly, we would like to test the hypothesis that extensive content-analysis of items is not necessary using the FGACF technique.

Note also that, although WEBHOUND allow users to annotate a document with their own keywords, it currently *does not* use user-annotations as a feature. This is because this feature is an example of a *user extracted feature* (refer Section 4.2.3), and we currently do not have a good mathematical model to deal with such feature values.

WEBHOUND uses the CERN **libwww** Library of Common Code [10] to retrieve and parse WWW documents as well as canonicalize URL references (since the Items database is keyed by URL).

Chapter 6

Experimental Results

In this chapter, we present some preliminary experimental results using a subset of the WEBHOUND data. The aim of the experiments was to determine the accuracy of various algorithms in being able to predict ratings.

6.1 The Data

At the time of writing WEBHOUND had 1460 users and 13882 documents in its database. Our experiments were carried out on a smaller subset of the data (666 users and 5235 documents). The total number of ratings in the test data set was 18491. The distribution of ratings for the data set is shown in Figure 6.1.

Note that the rating distribution is greater toward the higher values (5, 6, 7). This is probably because WEBHOUND users can submit their entire hotlists and associate a default rating with all the documents in the hotlist.

6.2 Experimental Methodology

We followed the methodology of [39] and divided the data into two sets:

1. **Training Set**
   80 % of the ratings of each user were chosen randomly as the set of ratings an algorithm could use to build up its profile of the user's interests. This comprised the training set of ratings.

2. **Test Set**
   The remaining 20 % of the ratings for each user were chosen as test values for

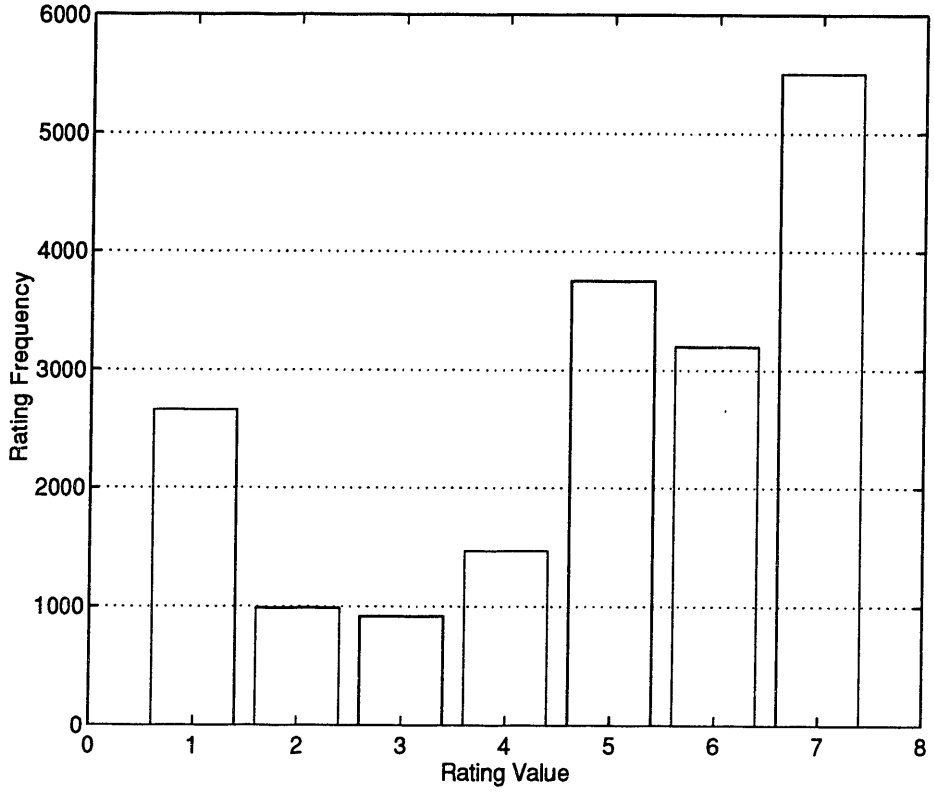

Figure 6.1: Distribution of Rating Values in Test Data Set

the algorithms to predict. This comprised the test set of ratings.

6.3 Evaluation Criteria

Let $R = \{r_1, r_2, ...r_N\}$ represent the actual ratings in the test set. Let $P = \{p_1, p_2, ...p_N\}$ represent the predicted values for the test set. We define the error set $E = \{\epsilon_1, \epsilon_2, ...\epsilon_N\} = \{p_1 - r_1, p_2 - r_2, ...p_N - r_N\}$.

The experiments were designed to evaluate the algorithms on the following criteria.

- **Minimum Mean Absolute Error**

  The mean absolute error of the predicted ratings should be as low as possible. The mean absolute error is defined as

$$\left|\overline{E}\right| = \frac{\sum_{i=1}^{N} |\epsilon_i|}{N} \tag{6.1}$$

  In general, the lower the mean absolute error, the more accurate the algorithm.

- **Minimum Standard Deviation of Error**
  The lower the standard deviation of the error, the more consistently accurate the algorithm. The standard deviation of the errors is given by

$$\sigma = \sqrt{\frac{\left(\sum_{i=1}^{N} (E - \overline{E})^2\right)}{N}} \tag{6.2}$$

- **Maximum Correlation Coefficient of Predictions**
  Following the methodology of [13], the higher the correlation of the predicted ratings with the actual ratings, the better the algorithm. The correlation coefficient is given by

$$r = \frac{\sum_{i=1}^{N} Covariance(r_i, p_i)}{\sigma_r \times \sigma_p} \tag{6.3}$$

  where σ_p and σ_r represent the standard deviations of the predicted and actual ratings in the test set respectively.

Shardanand [39] asserts that the predictions of an algorithm for *extreme values* (ratings in the test set above **6** or below **2**) are probably more important than for other values, since these values indicate strong user preferences. Hence, a good algorithm should optimize the three criteria above for extreme values.

To try and keep the comparison as fair as possible, the parameters used by both the ACF and FGACF algorithms tested were identical. Although each of the algorithms performs optimally at a different setting for different parameters, to compare them, we chose the exact same values for the following parameters:

- **Maximum number of nearest neighbors**
  The value of $\mathcal{N}$ was fixed at 200 for all the algorithms.

- **Maximum neighbor distance**
  The value of $\mathcal{L}$ was fixed at 10 for all the algorithms.

It may be noted that the *mmouth* code [27] uses finer tuned thresholding to select its neighbor set (Section 2.3.2), while neither of the FGACF algorithms do this. In this respect there exists a slight difference between the initial conditions of the algorithms.

6.4 Results

Because FGACF is such a rich body of techniques, different methods of clustering, updating cluster weights and feature weights, etc., may all have significant impact on

the accuracy of the algorithms. However, due to the lack of time and computational resources only one variant of each type of scheme was tested.

All the algorithms tested were of the Mean-Squared Difference (MSD) type.

6.4.1 The Base Algorithm: Averages

The first algorithm tested was to simply use the average rating for a document as the predicted rating. Hence, using the notation of Section 2.2

$$P_{I,p} = \frac{\sum_{J=1}^{||Users||} c_{J,p} \times R_{J,p}}{\sum_{J=1}^{||Users||} c_{J,p}} \quad \forall I$$

This is the simplest algorithm and is computationally quite efficient. The predicted rating for a document is *not personalized*, but is identical for every user. The error distribution is shown in Figure 6.2.

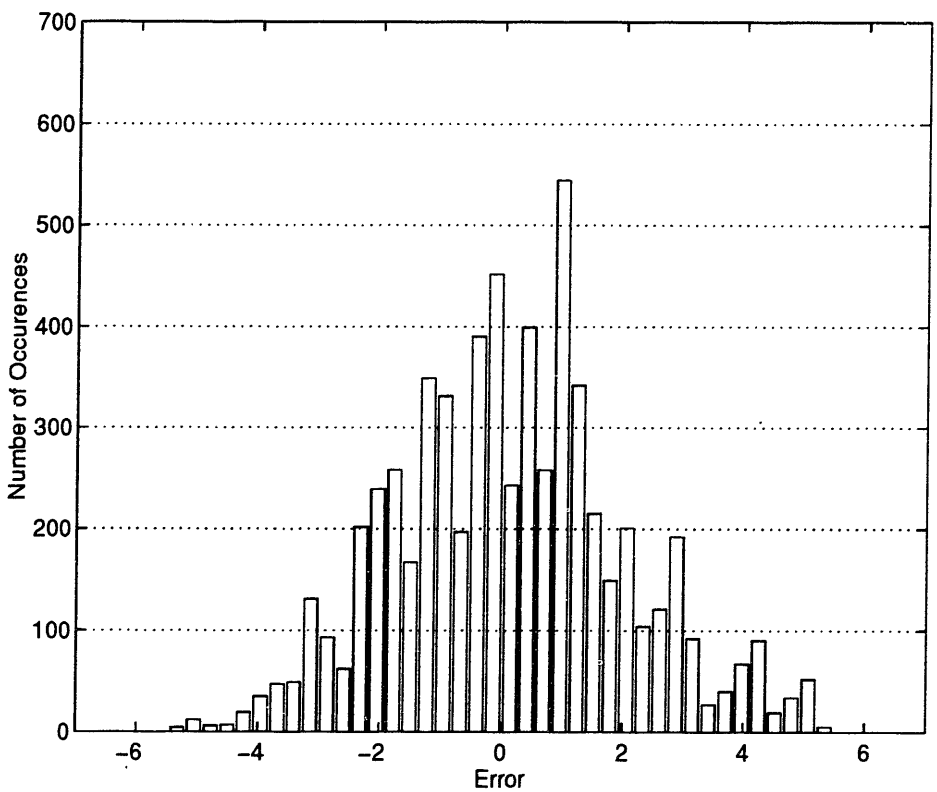

Figure 6.2: Error Distribution for Base Algorithm

The performance of the base algorithm is summarized in Table 6.1. The percentages in parentheses next to the Mean Absolute Error represent the Percentage Mean Absolute Error (a Mean Absolute Error of 7 represents 100 %).

Metric	All Values	Extreme Values
Mean Absolute Error ($\overline{\lvert E \rvert}$)	1.5138 (21.63 %)	1.7416 (24.88 %)
Std Deviation (σ)	1.8982	2.1233
Correlation Coeff (r)	0.5661	0.6227

Table 6.1: Base Algorithm Performance Summary

6.4.2 ACF

The next algorithm tested was the ACF MSD algorithm implemented in the *mmouth* code. Note that this is **not** an FGACF algorithm. The ACF algorithm was chosen as a reference algorithm against which to compare the performance of the FGACF algorithms. Figure 6.3 shows the distribution of the errors for all test set values. The error distribution has a bell-shape curve. It is also interesting to note the spikes in the error distribution around the integer error points (i.e. -6, -5, -4 ..., 4, 5, 6). This is most likely due to integer predicted ratings (based on a very small number of neighbors) for a document.

The performance of the ACF MSD algorithm is summarized in Table 6.2.

Metric	All Values	Extreme Values
Mean Absolute Error ($\overline{\lvert E \rvert}$)	1.5483 (22.12 %)	1.6506 (23.58 %)
Std Deviation (σ)	2.0511	2.2114
Correlation Coeff (r)	0.5491	0.5976

Table 6.2: ACF Performance Summary

Comparing the results in Tables 6.1 and 6.2 reveals that *the base algorithm outperforms standard ACF on almost every performance measure!* This is a startling although not wholly unexpected result for the following reasons:

- The data set used for testing is still comparatively small. In general, standard ACF performs better with a larger data set.
- The WWW domain is extremely sparse. Standard ACF does not perform well in such a domain.

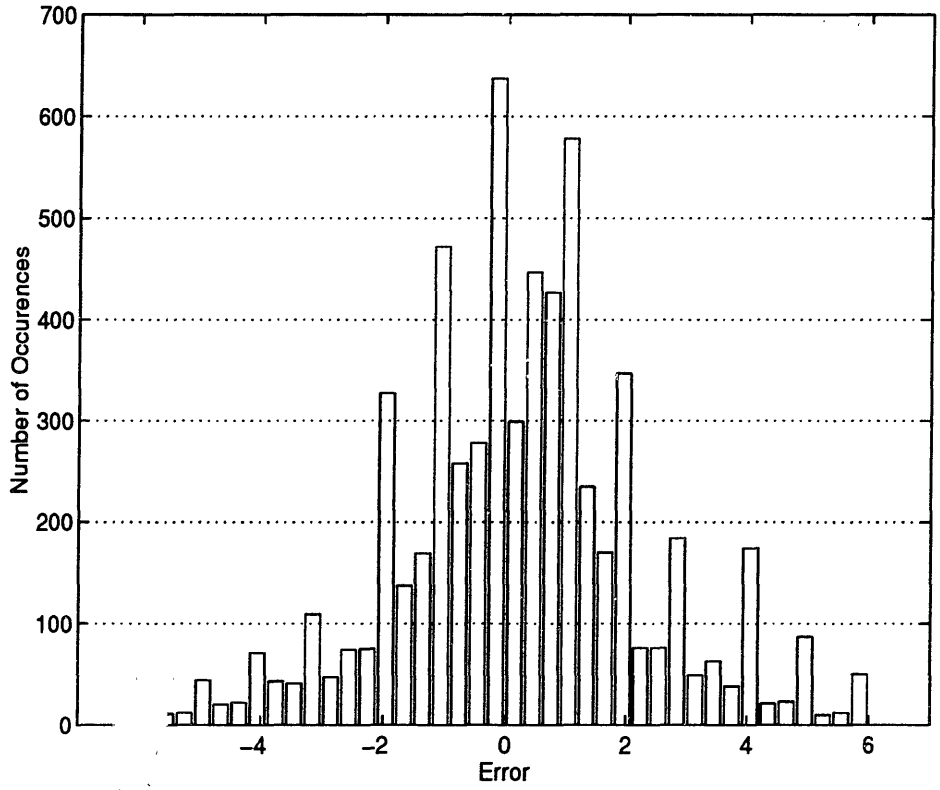

Figure 6.3: Error Distribution for ACF

- The distribution of ratings may also play a part in the effectiveness of the prediction algorithm.

As can be seen from the table the correlation of standard ACF with the actual values in the test set is a little over 0.5. The performance of the algorithm does improves on the base algorithm when we consider the average absolute error for only the extreme values.

6.4.3 FGACF - Simple Average Based Clustering

The next algorithm tested was the FGACF MSD algorithm using the simple average based clustering for feature values presented in Section 3.4.2. Figure 6.4 shows the distribution of the errors for all test set values.

Here too, the error distribution has a bell-shape curve. The performance of the FGACF algorithm with mean clustering of feature values is summarized in Table 6.3.

The FGACF algorithm with mean clustering outperforms standard ACF on almost every metric used (except for the absolute error for extreme values). For both the

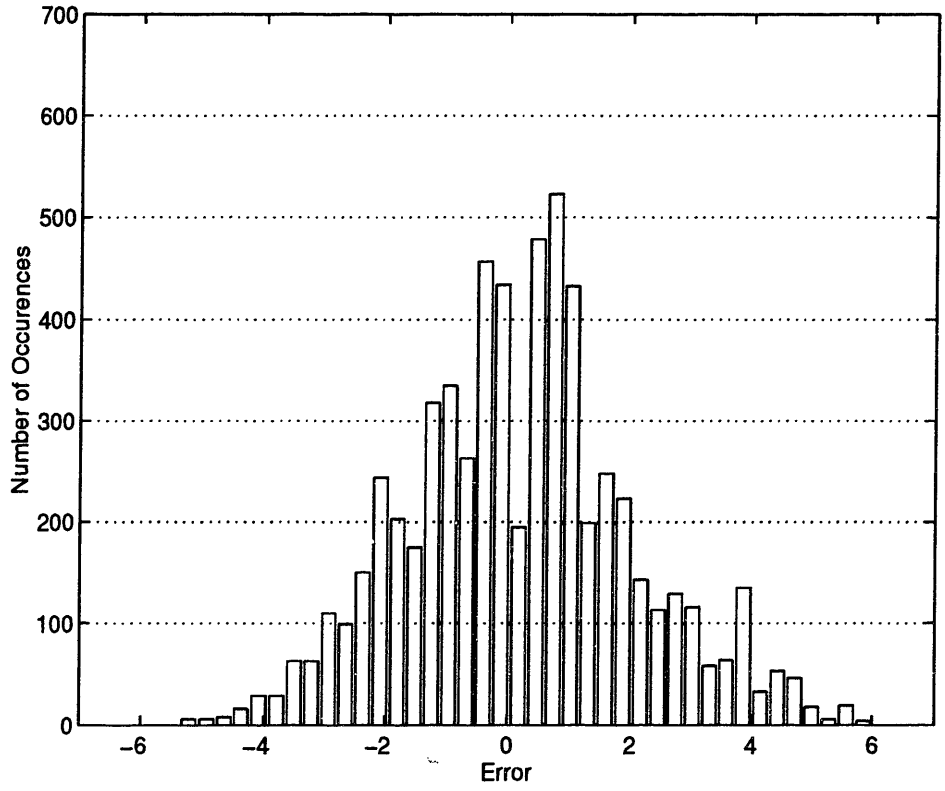

Figure 6.4: Error Distribution for FGACF (simple average clustering)

standard deviation of the errors, as well as in the degree of correlation between the predicted ratings and the actual ratings in the test set, it is significantly better than the standard ACF algorithm. It is also interesting to note that although this FGACF algorithm has a slightly worse mean absolute error for extreme values than standard ACF, *it still has a lower standard deviation of that error, as also a higher degree of correlation between the predicted ratings and the actual ratings for extreme values in the test set.*

Metric	All Values	Extreme Values
Mean Absolute Error ($\lvert\overline{E}\rvert$)	1.4825 (21.18 %)	1.6755 (23.94 %)
Std Deviation (σ)	1.8753	2.0808
Correlation Coeff (r)	0.5907	0.6455

Table 6.3: FGACF Performance Summary (Average Clustering)

FGACF with average based clustering also outperforms the base algorithm on each of the six evaluation criteria.

6.4.4 FGACF - Gaussian (Sample Estimator) Clustering

The error distribution for the FGACF MSD algorithm using the Gaussian distribution based clustering (with sample estimators) presented in Section 3.4.3 is shown in Figure 6.5.

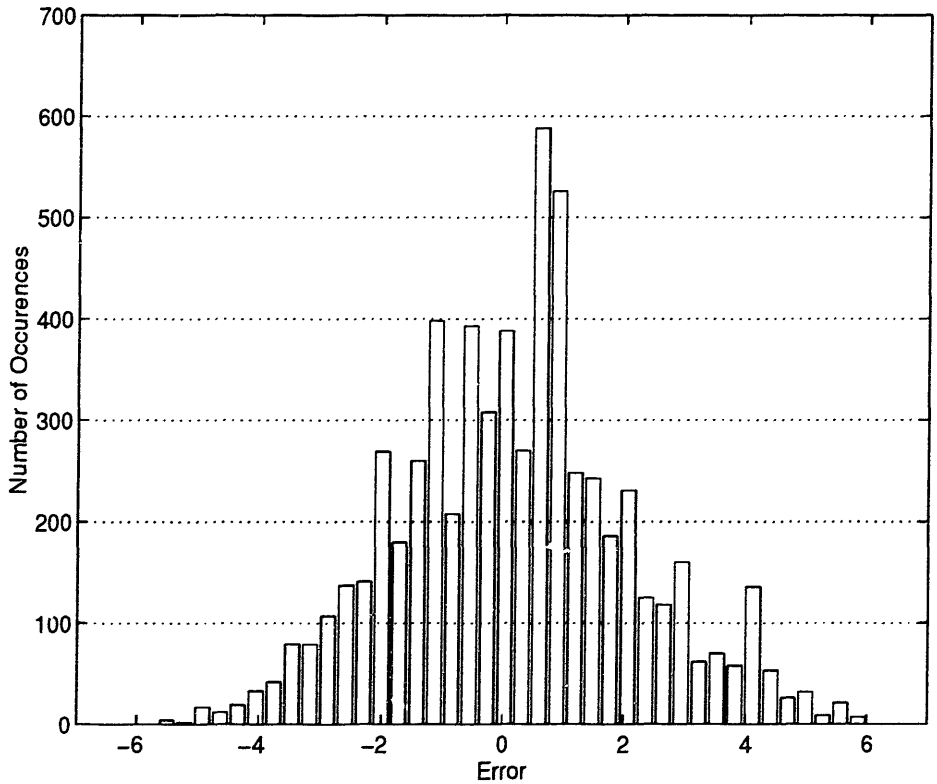

Figure 6.5: Error Distribution for FGACF (gaussian sample estimator clustering)

The performance of the algorithm is summarized in Table 6.4.

This FGACF algorithm performs worse than standard ACF on Mean Absolute Error for extreme values. However, it still has a lower standard deviation of the errors than standard ACF. In addition the correlation coefficient of the predictions is higher than standard ACF. It too is outperformed by the base algorithm (although it does do better on Mean Absolute Error for extreme values).

One reason for these results may be the fact that the training set does not contain enough data for the gaussian clustering to be applicable (recall from Section 3.4.3,

Metric	All Values	Extreme Values
Mean Absolute Error ($\lvert\overline{E}\rvert$)	1.5371 (21.96 %)	1.7340 (24.77 %)
Std Deviation (σ)	1.9374	2.1485
Correlation Coeff (r)	0.5599	0.6140

Table 6.4: FGACF Performance Summary (Gaussian Sample Estimator Clustering)

that the idea behind the gaussian clustering was the fact that there were a sufficiently large number of ratings per feature value per user to approximate the population by a gaussian distribution (applying the **Central Limit Theorem**)). If the sample size is small (as is the case with the data in the training set), this assumption does not really hold. As the amount of data increases we expect the gaussian clustering based metrics to get better in terms of accuracy.

6.5 Discussion

Table 6.5 below summarizes the results for all of the algorithms tested (FGACF 1 refers to the FGACF algorithm with average based clustering, FGACF 2 refers to the FGACF algorithm with gaussian sample estimator clustering).

	Base	ACF	FGACF 1	FGACF 2
$\lvert\overline{E}\rvert$ (All Values)	21.63 %	22.12 %	21.18 %	21.96 %
$\lvert\overline{E}\rvert$ (Extreme Values)	24.88 %	23.58 %	23.94 %	24.77 %
σ (All Values)	1.8982	2.0511	1.8753	1.9374
σ (Extreme Values)	2.1233	2.2114	2.0808	2.1485
r (All Values)	0.5661	0.5491	0.5907	0.5599
r (Extreme Values)	0.6227	0.5976	0.6455	0.6140

Table 6.5: Summary of Experimental Results

Standard ACF performed the best in terms of the Mean Absolute Error ($\lvert\overline{E}\rvert$) for extreme values, although FGACF with simple average clustering performed nearly as well on that metric. On all the other metrics, both the FGACF algorithms out-

performed standard ACF, in some cases, significantly ; for example, the FGACF algorithm using simple average clustering has a Mean Absolute Error of an entire percentage point less than the ACF algorithm.

Both ACF algorithms also had a lower standard deviation of the errors than the ACF algorithm for both extreme and all test set values indicating that they are more consistently accurate predictors than standard ACF. Furthermore, the correlation coefficient (r) between the actual and predicted ratings, another indicator of the accuracy of a prediction algorithm, was better for both the FGACF algorithms.

It is interesting to note that the base algorithm outperformed both standard ACF as well as FGACF with gaussian clustering. We believe that this is due to the size and sparsity of the data set as well as the distribution of ratings. As the size of the data set grows we expect the performance of both ACF and FGACF with gaussian clustering to improve.

These preliminary sets of experiments indicate that FGACF is a more promising technique than standard ACF, even using simple feature information. Although the experiments did not bear out this trend, we believe that as the amount of data increases, FGACF algorithms using the gaussian clustering techniques of Sections 3.4.3 and 3.4.4 will start outperforming those using simple average clustering techniques.

There are still a number of parameters that need to be tested to understand the effect of various different concepts that the FGACF formalism introduces.

6.5.1 Effects of Number of Feature Value Clusters

In theory the number of clusters allocated to a particular feature value cluster should not affect the predicted value, since each cluster is weighted by a corresponding cluster weight. However, for features having a very large number of feature values, providing a larger number of clusters generally allows a better discrimination amongst feature values while clustering, and probably, more accurate predictions.

6.5.2 Effects of Irrelevant Features

While the feature weights are supposed to handle the effect of irrelevant features for each user, we have no way of telling if our current method of calculating these feature weights correctly captures the notion of the *importance* of a particular feature for the user. It is certainly worthwhile exploring the performance of FGACF algorithms by selectively removing features from the set of features defined for a domain.

6.5.3 Effects of Quantization of Numerical Features

Quantizing numerical features is a standard way to deal with an unbounded space of feature values. However, how exactly the quantization is performed may be crucial to the performance of the FGACF algorithm. For example, early on, a design decision was made to quantize the feature *number of links in document* to one of four possible values, NONE, 1..5, 6..20, and ABOVE 20. This resulted in 6..20 and ABOVE 20 being the highest weighted values for most users, and since most WWW documents have greater than six links, this feature (*number of links in document*) wasn't too effective in partitioning the document space. In retrospect, a finer grained quantization may have yielded better results.

6.5.4 Using User Extracted Features

User extracted features such as *user annotations* are potentially invaluable sources of information for an FGACF algorithm. However, as explained in Section 4.2.3, we currently do not have a good mathematical model for such features.

Chapter 7

Conclusions and Future Work

This research has introduced the concept of *Feature Guided Automated Collaborative Filtering* (FGACF). FGACF refers to a body of information filtering algorithms that are applicable in almost any domain where the number of potential choices of items is very large, users' perception of the quality of items is largely subjective, and some feature information is available (or can be extracted) for items in the domain. While initial experiments with FGACF indicate it is a promising technique, much work still needs to be done to exploit its full potential.

This research also led to the development of WEBHOUND, a personalized information filtering system for WWW documents. Currently, WEBHOUND is implemented as a single server which can only be accessed through a WWW interface front end. This decision was made to make the development and testing of new ideas relatively easy, as also to avoid the software support role that comes with distributing a front end. However, if a system like WEBHOUND is to be put to widespread use (and there is a very real need for something along these lines) several challenging issues still need to be explored. Furthermore, there are many potential directions where such technology can be successfully applied.

In the following few sections, we identify some directions for future research as well as potential application domains for the technology.

7.1 Distributed WEBHOUND

For WEBHOUND to be a serious and widely used system, instead of the research prototype it currently is, it needs to be designed to scale rapidly along two dimensions: number of users and number of WWW documents, both of which are increasing at

tremendous rates. The solution of simply throwing more hardware at the problem will only buy a small amount of time given the growth rates of the WWW and the user population.

To make a system like WEBHOUND as ubiquitous as say, USENET or the Domain Naming System, it has to be completely decentralized and distributed. Users should be allowed to connect to any of a set of WEBHOUND servers (each with a different virtual community). A user may still have most of their ratings on a single server. However, an efficient server-server query handoff protocol needs to be defined such that when a given server knows that it doesn't possess sufficient information to satisfy a given user's query adequately, it can hand off the query to a set of peer servers (along with a summary of relevant information). Although this sounds simple in the abstract, there are a whole host of issues that need to be solved to allow a scheme like this to work:

- A given FGACF (ACF) server must be able to realize when it does not have enough information to make a given prediction or recommendation for a given user. Currently, the FGACF formalism incorporates no notion of confidence in a prediction. However, such a notion is crucial for the handoff to work successfully.

- The protocol amongst servers should be sufficiently high-level and abstract to allow for any type of underlying scheme. That is, no assumption regarding the types of features, method, etc., should creep into the server-protocol.

- The ratings server essentially acts as a personalized agent for a user. A ratings server talks to a set of peers for query handoffs. Not all these peers may give the same quality of ratings (as perceived by the querying ratings server) for a particular user. Even a rudimentary modelling of the quality of *trust* in the predictions of the peer servers along the lines of [19] may be extremely helpful.

- It may also be extremely useful for ratings servers to somehow exchange summaries of their user populations' interests in WWW document space. This would enable the discovery that certain users in a given virtual community are most like another bunch of users in a different virtual community for certain kinds of documents, and hence requests for predictions for those kinds of documents could be handed off directly to the peer server.

7.2 Filtering Search Engine Query Results

WEBHOUND is primarily an information filtering service. Popular WWW search engines such as Lycos [24], WebCrawler [29], Yahoo [44], etc. are primarily information retrieval engines (as opposed to information filtering systems). The two are complementary - a WEBHOUND like front-end to a popular search engine such as Lycos, could enable users with WEBHOUND accounts to filter the results of their searches on the extensive databases compiled by these search engines in a personalized fashion. As a concrete example, let's say a user is searching for documents on Indian Cooking. He types the keywords Indian Cooking into the Lycos search form. The number of documents matching both keywords numbers in the hundreds. Even though any good search engine will order the matches in descending order of match, there are still too many documents for the average user to go through. However, if the user had a WEBHOUND account, the resulting matches could be filtered through WEBHOUND and only the top ranked ones (in terms of predicted rating) need be returned.

7.3 Filtering Objectionable Material Flexibly

With more and more ordinary citizens migrating to the Internet in greater numbers, the issue of protecting minors from objectionable material (along with the on-going debate on *cyberporn*) has suddenly become very important. The computer industry is rushing to implement rating protocols for on-line documents that browsers could filter on. All the proposals so far follow variants of two common threads:

- Voluntary ratings by publishers of *adult* material on their documents. It is highly doubtful that all publishers of *adult* material will label their sites as such. In addition, the issue is clouded by the fact that what is considered acceptable in Sweden may be considered pornographic in the United States, and analogously, material considered acceptable by American standards may be totally unacceptable by Japanese standards. Furthermore, to view the information filtering problem as simply to filter *adult* material is too narrow a perspective - the broader view is to filter out *objectionable material* for a certain user (for example, a mother may not wish to have her ten-year old daughter accidentally stumble over the Aryan Nation HomePage).

- The second scheme is to have a certain trusted entity surf the net and create *blacklists* (lists of documents that should be blocked), and/or *whitelists* (lists of documents that are suitable for viewing). This solution has numerous drawbacks - if a certain document isn't on the blacklist it may be that it just wasn't evaluated - not because it doesn't contain objectionable material. More significant than this omission problem is the fact that none of these lists are personalized to a particular user's preferences; a user has to essentially buy into the opinions of the entity providing the lists.

FGACF and ACF can address the problems with both the approaches above - if a document isn't on a standard set of lists it could be sent to multiple FGACF servers that can compute a *view/don't view* decision on the document by consulting users in much larger virtual communities that share the same opinions as a particular user. This solution is personalized, community oriented, and scalable.

This thesis has presented a novel information filtering technique, Feature Guided Automated Collaborative Filtering (FGACF). FGACF draws on the complementary strengths of content-based and automated collaborative information filtering techniques. Since it is a hybrid, FGACF can deal more successfully with the drawbacks of either pure content filtering or pure ACF. FGACF is especially useful in broad domains where computer analysis of items is hard. Furthermore, the FGACF formalism allows for the utilization of domain knowledge (when available), since it does not depend on the type of clustering scheme used. However, it can also handle domains where little or no domain knowledge is available.

This thesis has also produced a generic FGACF server implementation that can easily be modified and setup for an application domain. The engine provides a number of clustering schemes to application domain designers, and is modular and efficient.

WEBHOUND, a personalized WWW document filtering system was built using the generic FGACF engine core. WEBHOUND also provides the powerful information retrieval metaphor of *query by example.*

Initial experiments with some FGACF algorithms indicate it is a powerful information filtering technique.

Bibliography

[1] Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T., *WebWatcher: A Learning Apprentice for the World Wide Web*, in Proceedings of the AAAI Spring Symposium Workshop on *Information Gathering from Large Heterogenous Environments*, March 1995, Stanford, CA.

[2] Balabanovic, M., and Shoham, Y., *Learning Information Retrieval Agents: Experiments with Automated Web Browsing*, in Proceedings of the AAAI Spring Symposium Workshop on *Information Gathering from Large Heterogenous Environments*, March 1995, Stanford, CA.

[3] Belkin, N. J., and Croft, B. W., *Information Filtering and Information Retrieval: Two Sides of the Same Coin?*, CACM, 35 (12), Dec 1992, pp 29-38.

[4] Berners-Lee, T., Cailliau, R., Groff, J-F., and Pollermann, B., *World-Wide Web: The Information Universe*, Electronic Networking: Research, Applications and Policy, 1 (2), Meckler, CT, 1992.

[5] Bowman, C. M., Danzig, P. B., Hardy, D. R., Manber, U., and Schwartz, M. F., *The Harvest Information Discovery and Access System*, in Proceedings of the Second International World Wide Web Conference, Chicago, Illinois, Oct 1994.

[6] *CUI W3 Catalog*, available on-line at
`http://cui_www.unige.ch/w3catalog`

[7] DeBra, P., Houben, G-J., and Kornatzky, Y., *Navigational Search in the World-Wide Web*, available on-line at
`http://www.win.tue.nl/help/doc/demo.ps`

[8] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R., *Indexing by Latent Semantic Analysis*, Journal of the American Society for Information Science, 41 (6), pp 391-407, 1990.

[9] Feynman, C., *Nearest neighbor and maximum likelihood methods for social information filtering*, Internal Document, MIT Media Lab, Fall 1993.

[10] Frystyk, H., and Lie, H., *Towards a Uniform Library of Common Code: A Presentation of the CERN World-Wide Web Library*, in Proceedings of the Second International World Wide Web Conference, Oct 17-20 1994, Chicago, IL, USA.

[11] Goldberg, D., Nichols, D., Oki, B., and Terry, D., *Using Collaborative Filtering to Weave an Information Tapestry*, CACM, 35 (12), Dec 1992, pp 61-70.

[12] Hill, W., *Personal Communication*, Feb 1995.

[13] Hill, W., Stead, L., Rosenstein, R., and Furnas, G., *Recommending and Evaluating Choices in a Virtual Community of Use*, in Proceedings of CHI '95, Denver, CO, May 1995.

[14] Jain, A, and Dubes, R, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, 1988.

[15] Johnson, M., *SIMON - System of Internet Mapping for Organized Navigation*, available on-line at
`http://www.elec.qmw.ac.uk/simon/welcome.html`

[16] Koster, M., *ALIWEB - Archie-Like Indexing in the WEB*, in Proceedings of the First International World Wide Web Conference, CERN, Geneva, May 1994.

[17] Koster, M., *World Wide Web Robots, Wanderers, and Spiders*, available on-line at
`http://web.nexor.co.uk/mak/doc/robots/robots.html`

[18] Lang, K., *NewsWeeder: An Adaptive Multi-User Text Filter*, Research Summary, Aug 1994.

[19] Lashkari, Y., Metral, M., and Maes, P., *Collaborative Interface Agents*, in Proceedings of the National Conference on Artificial Intelligence, AAAI '94, Aug 1994, Seattle, WA.

[20] Lieberman, H., *Letizia: An Agent That Assists Web Browsing*, to appear in Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI-95, Montreal, Aug 1995.

[21] Mallery, J. C., *Indexing and Retrieval For the World Wide Web*, Intelligent Information Infrastructure Project, MIT AI Lab, Sept 1994, draft.

[22] Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., and Cohen, M. D., *Intelligent Information Systems*, CACM, 30 (5), May 1987, pp 390-402.

[23] Maltz, D., and Ehlrich, K., *Pointing the Way: Active Collaborative Filtering*, in Proceedings of CHI '95, Denver, CO, May 1995.

[24] Mauldin, M., *The Lycos Spider*, available on-line at `http://lycos.cs.cmu.edu/`

[25] Mauldin, M. L., and Leavitt, J. R., *Web Agent Related Research at the Center for Machine Translation*, in Proceedings SIGNIDR-94, Aug 1994, McLean Virginia.

[26] McBryan, O., *GENVL and WWWW: Tools for Taming the Web*, in Proceedings of the First International World Wide Web Conference, CERN, Geneva, May 1994.

[27] Metral, M., *MotorMouth: A Generic Engine for Large-Scale, Real-Time Automated Collaborative Filtering.* SM Thesis, Dept of Media Arts and Sciences, MIT, May 1995.

[28] Mueller, P., *Infobot Hotlist Database*, available on-line at `ftp://ftp.netcom.com/pub/ksedgwic/hotlist/hotlist.html`

[29] Pinkerton, B., *The Web Crawler*, available on-line at `http://www.webcrawler.com/`

[30] Porter, M., *An Algorithm for Suffix Stripping*, Program 14 (3), pg 130-137, 1980.

[31] Reid, B., *Usenet Readership Summary Report For May 93*, Usenet news.lists, June 1993.

[32] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J., *GroupLens: An Open Architecture for Collaborative Filtering of Netnews*, in Proceedings CSCW '94.

[33] Robertson, S. E., *The Probability Ranking Principle in Information Retrieval*, J. Doc, 33 (4), Dec 1977, pp 294-304.

[34] Salton, G., *Automatic Information Retrieval*, IEEE Computer, Sept 1980, pp 41-56.

[35] Salton, G., and Buckley, C., *Improving Retrieval Performance by Relevance Feedback*, J. Am. Soc. for Information Science, 41 (4), 1990, pp 288-297.

[36] Salton, G., Fox, E., and Wu, H., *Extended Boolean Information Retrieval*, CACM, 26 (11), Nov 1983, pp 1022-1036.

[37] Salton, G., and McGill, M. J., *Introduction to Modern Information Retrieval*, McGraw-Hill, 1983.

[38] Sengupta, S., *Project DA-CLOD: Distributedly Administered Collaborative List of Documents* available on-line at
`http://schiller.wustl.edu/DACLOD/daclod/`

[39] Shardanand, U., *Social Information Filtering for Music Recommendation*, SM Thesis, Dept of EECS, MIT, Sept 1994.

[40] Shardanand, U., and Maes, P., *Social Information Filtering: Algorithms for Automating "Word of Mouth"*, in Proceedings of CHI '95, Denver, CO, May 1995.

[41] Sheth, B. D., *A Learning Approach to Personalized Information Filtering*, SM Thesis, Department of EECS, MIT, Feb 1994.

[42] *SUSI - Simple Unified Search Index*, available on-line at
`http://web.nexor.co.uk/susi/susi.html`

[43] Tversky, A., *Features of Similarity*, Psychological Review, Vol 84 (4), July 1977.

[44] *YAHOO - A guide to WWW*, available on-line at
`http://www.yahoo.com/`

THESIS PROCESSING SLIP

FIXED FIELD: ill. _____ name _____

index _____ biblio _____

▶ COPIES: (Archives) Aero Dewey Eng Hum

Lindgren Music (Rotch) Science

TITLE VARIES: ▶☐ _____

NAME VARIES: ▶☐ _____

IMPRINT: (COPYRIGHT) _____

▶ COLLATION: 83 p

▶ ADD. DEGREE: _____ ▶ DEPT.: _____

SUPERVISORS: _____

NOTES:

cat'r: date:

page: 526

▶ DEPT: MediaA&S

▶ YEAR: 1995 ▶ DEGREE: M.S.

▶ NAME: LASHKARI, Yezdezard Zerxes