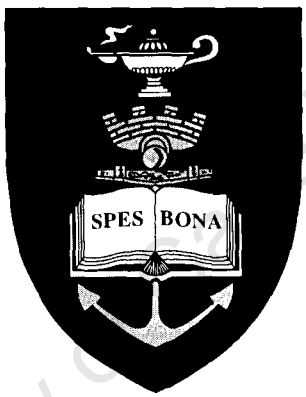
# Evaluating Collaborative Filtering Content Recommenders for Mobile Phones

Indika Weliwe Gamage Piyasena

PYSIND001






This thesis is submitted in partial fulfilment of the academic requirements
for the degree of
Master of Science in Electrical Engineering
in the Faculty of Engineering and The Built Environment
University of Cape Town
August 2007

# Abstract

The high adoption of mobile phones coupled with 3G technology can extend Internet access to new communities. Yet such access is currently impractical because mobile phone interfaces are cumbersome to use. In addition, hierarchical menus and search engines pose an interaction barrier to the unfamiliar. A content recommender is proposed to address these issues. Collaborative filtering is a technique developed to make predictions on unobserved items based on the preferences of similar users. User-based collaborative filtering has been identified as a simple, yet reasonably accurate scheme. An evaluation is conducted into how quickly this algorithm can identify preferred content based on user-content interactions. Two similarity measures between users are evaluated empirically in Matlab with the MovieLens and EachMovie datasets. Vector similarity significantly outperforms the Pearson correlation method during cold-start conditions. However, there is a high variance in the accuracy. Prior knowledge about the users is required to reliably recommend content.

# Declaration

I hereby declare that:

(1) the above thesis is my own unaided work, both in conception and execution, and that apart from the normal guidance of my supervisor, I have received no assistance apart from that stated below; (2) except as stated below, neither the substance or any part of the thesis has been submitted in the past, or is being, or is to be submitted for a degree in the University or at any other University.

I am now presenting the thesis for examination for the Degree of MSc in Electrical Engineering. I also grant the University free license to reproduce the above thesis in whole or in part, for the purpose of research.

Signed by candidate

Indika Weliwe Gamage PIYASENA

26 · 11 · 2007

Date

# Acknowledgements

Firstly I would like to thank my thesis supervisor Prof HA Chan for his constant support and always having his door open to me.

I would like to thank Gary Marsden for his input on mobile interface design and all my friends for supporting me during this dissertation.

I would like to acknowledge the National Research Foundation (NRF) for providing funding support during the period of this research. In addition, I would like to thank the Centre of Excellence (CoE) for sharing its equipment and space with me, and for making conference travel possible.

# Synopsis

Mobile phones coupled with 3G technology can provide technologically impoverished communities with access to the web. Access, however, is not practical. The interfaces on mobile phones make browsing the web challenging. Small screen sizes, cumbersome text input and the lack of a pointing device deter the exploration of the web [1]. In addition, web pages are filled with paraphernalia such as rich images and Adobe Flash to make them more attractive.

These limitations have been addressed by text-based mobile web pages with minimalist graphics. These pages are accessed via mobile search-interfaces and hierarchical-interfaces. Both these interfaces have been designed assuming that the users know what they are looking for, and thus poses an interaction barrier to the Internet naïve market.

Recommending mobile web content mitigates this interaction barrier. A software architecture is designed to do so, however, the identification of relevant content presents a challenging problem. Content-based filtering [2] can be employed to filter web content based on list of keywords, but require explicit queries from the user, which we would like to eliminate. Collaborative filtering [3] offers a means to recommend content purely on implicitly collected preferences.

Collaborative filtering operates on the axiom that like-minded individuals share similar interests. Predictions are made on unobserved items based on the preferences of similar individuals to the active user. Collaborative filtering has been identified as the most successful recommendation technique to date [4]. A multitude of these algorithms have been developed, and have been broadly grouped into two classes.

Memory-based collaborative filtering algorithms generate recommendations by creating a neighbourhood of similar users or items. The similarity measurement is computed using a Pearson correlation or with vector similarity. Although user-based collaborative filtering is one of the simplest implementations in this class, it is also a reasonably accurate scheme [3]. Scalability is an issue for schemes in this

class. Item-based collaborative filtering addresses this issue and thus is one of the most popular schemes [5], and has been implemented by Amazon.com [6].

Model-based collaborative filtering algorithms create a static user-item preference model prior to generating recommendations. The network operator can observe the reasons behind recommendations, thus offer the ability to scrutinize and make possible amendments. Personality diagnosis [7] operates on the notion that the preferences that user reports are manifestations of their underlying personality types. Clustering methods [8] attempt to identify natural clusters of users, thus offering a robust means of generating recommendations. Although model-based techniques are less accurate than memory-based techniques, less false positives are recommended and a deterministic query time is offered.

The feasibility of recommending content to the mobile device is in question, and therefore no actual usage data is available to the author. Human behaviour is modelled in Matlab using two datasets of user ratings towards movies. From each set of ratings, a probability mass function is created to model how likely each user will observe items recommended. User-based collaborative filtering is implemented to make predictions based on these observations. In addition, a preference learning scheme that randomly recommends items is implemented to benchmark the performance.



Results show that user-based collaborative filtering performed best when users had preferred 10% or more of a set of 1000 items. Here, half the items recommended were preferred in less than five sessions. Vector similarity outperforms the Pearson correlation method of computing user similarity when little user-preference information is available. However, in scenarios where users preferred less than 10%, employing collaborative filtering had no benefit over randomly recommending items and over 25 sessions were required to identify half the preferred items. The high variance in accuracy suggests that prior information about the user needs to be incorporated to design a robust recommender system.

The results of this research has been published at the 5$^{th}$ IEEE Consumer Communications and Networking Conference 2008. [9]

# Table of contents

# List of Figures



# List of Tables

# Abbreviations

| | |
|---|---|
| **3G** | 3rd Generation |
| **CF** | Collaborative Filtering |
| **EM** | Expectation Maximization |
| **GPRS** | General Packet Radio Service |
| **HTML** | HyperText Markup Language |
| **HTTP** | HyperText Transfer Protocol |
| **ID** | Identity |
| **LBS** | Location-based services |
| **MCMC** | Markov Chain Monte Carlo |
| **MLE** | Maximum Likelihood Estimate |
| **PDF** | Probability Density Function |
| **PMF** | Probability Mass Function |
| **RSS** | Really Simple Syndication |
| **SMS** | Short Message Service |
| **WAP** | Wireless Application Protocol |
| **XHTML** | Extensible HyperText Markup Language |
| **XML** | Extendable Markup Language |

# Variables

| | |
|---|---|
| $n$ | the total number of individuals in the community |
| $m$ | the total number of resources |
| $p_{a,j}$ | the predicted vote of the active user on resource j |
| $I_i$ | the set of resources that user $i$ has voted on |
| $U_i$ | the set of users that rated item $i$ |
| $v_{i,j}$ | the vote user $i$ made on item $j$ |
| $w_{a,i}$ | the similarity between the active user and user $i$ |
| $\mathbf{u}_i$ | the vector containing preferences on items for user $i$ |
| $\mathbf{i}_i$ | the vector containing all the user ratings on item $i$ |
| $c_{a,i}$ | the set of all co-rated items by user $a$ and user $i$ |
| $b_{a,i}$ | the set of all users that rated both items $a$ and $i$ |

# Chapter 1 Introduction

## 1.1 Mobile Internet Access

The proliferation of mobile phones throughout Africa can act as a bridge between the currently widening digital divide. These devices, coupled with GPRS or 3G technology, can enable technology impoverished communities to access the Internet.

While such a combination enables Internet access, access however is currently still not practical. The interfaces on mobile phones make browsing the web challenging. Small screen sizes, cumbersome text input and the lack of a pointing device deter the exploration of information [1].

In addition, web pages are filled with paraphernalia to make them more attractive. Large images, Adobe Flash, Ajax and JavaScript not only increase the size of web pages, thus increasing the cost of mobile data access, but are not readily and quickly processed by light weight web browsers found on mobile phones.

These limitations have been addressed by the mobile web, which is a small subset[1] of the web designed specially for mobile phone access. These pages are text-based with minimal use of images. Two interfaces are presented to the mobile phone user to access these pages:

- Aggregators: Mobile web content is collected by content providers and is arranged with an hierarchical menu framework. The South African market is

[1] Google estimates that 5 million of the 8 billion pages they index are Wireless Application Protocol (WAP) based

presented with aggregators such as Vodafone Live [10], MTN ICE [11] and ExactMobile [12].

- Mobile search engines: Content is retrieved from the mobile web from specified keywords. A popular mobile search engine is Google Mobile [13].

Interactions with hierarchical interfaces require that the user stores in his or her short term memory a map of the hierarchical structure under which the content is organised [1]. While the addition of this cognitive overhead is easily adopted by users accustomed to file systems found on operating systems, it is met with difficulty by the inexperienced [14].

Small screen impacts, cumbersome text input and slow mobile data access severely impact searching for content via the mobile device, often frustrating the user [1].

More fundamentally, both the hierarchical and the search-based interfaces rely on the user knowing what one is looking for. A user experienced with the Internet is conscious of the content present on the web, and can speculate what may exist on the mobile web. Based on this knowledge, the user may have the motivation to endure these cumbersome interfaces. Those who are unaware of the content may lack such a motivation, and furthermore a large portion of the African market can be considered Internet naïve.

Ignorance of web content may be attributed to lack of exposure to, and relevance of, such content. Web content originates from a socio-economic class of people who have access to fixed Internet connections. Such content may be irrelevant to technologically impoverished communities.

In summary, mobile phones coupled with 3G technology is a necessary, but insufficient criterion to provide Internet services. Both the interaction barrier and the lack of web relevance need addressing.

## 1.2 Problem Definition

### 1.2.1 Syndication from the mobile device

Historically, syndication is a term used to describe the process of publishing an article in many magazines or newspapers at the same time. Now with respect to the Internet, it refers to the process where content providers publish their content concurrently to many web pages.

The ability to syndicate content benefits both the content generating side and the content publishing side. Content generators benefit in their content being exposed to numerous online platforms. Publishers benefit from being able to present their visitors with dynamic content.

Extending the ability to syndicate from mobile phones would enable a community of individuals that do not have direct access to the Internet to publish information on the web. This syndicated content could exist on numerous typically rich web pages, and also as plain text pages suitable for the mobile web. Consequently, when considering the latter, the information available on the web would be more relevant to the publishing community. Figure 1-1 illustrates this mechanism. In this figure, the mobile web is decoupled from the Internet to highlight the possibility of syndication to multiple platforms concurrently.

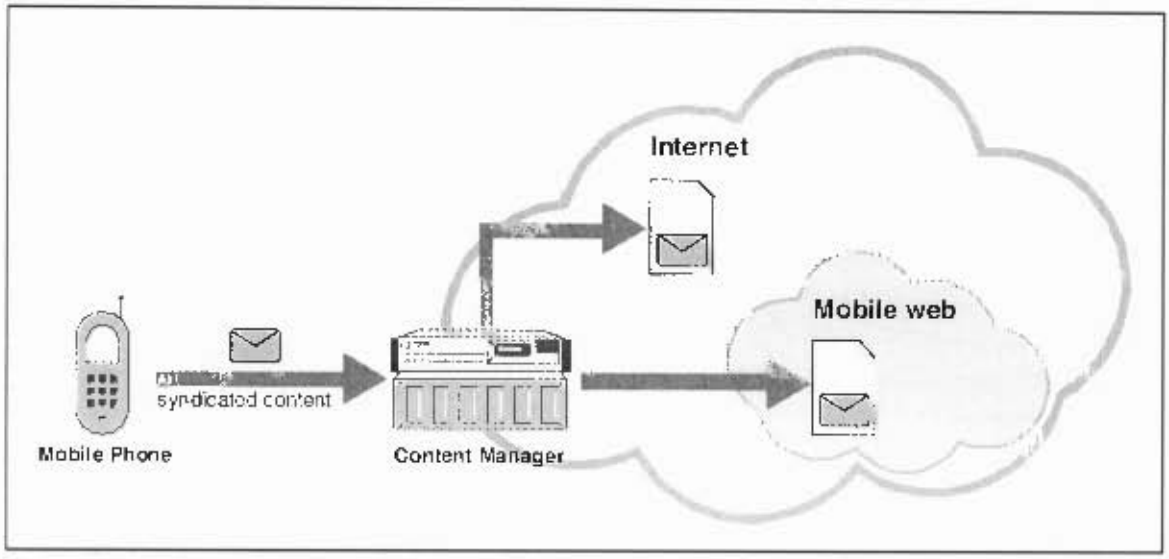



**Figure 1-1: Syndication from the Mobile Device**

A typical example would be where an owner of a company would like to advertise his or her service. The owner could simply achieve this by composing the SMS:

- City Rock: Beginner's Climbing Course: City Rock Observatory is offering a beginner's climbing course at 7pm to 9pm on Mondays. Cost is R50 per session. Must come in pairs. Call 021 650 8239 for more information.

This mechanism is beneficial for two reasons:

- The owner can syndicate content to the Internet without owning a computer with an Internet connection
- Any changes made to the content, will be immediately updated on all the locations where it has been published

The example used SMS as a form of content upload. Alternatively, an XHTML mobile web form can be developed to enable syndication from the mobile device. Such an implementation is more versatile than SMS, however, the interface would be less familiar.

Syndicated content can then simply be searched for via the web using popular search engines. A user specifying the keywords 'city rock', 'observatory' and '.za' should display the event details regarding 'City Rock'. The difficulty lies in the access to such content on mobile phones. Sheer volumes of content are available and need to be waded through using the limited interface.

### 1.2.2 Filtering and recommending content

Not all content present on the Internet is relevant to the mobile phone user. The mobile device can be perceived as a tool used to interact with one's environment. Hence, it is sensible to filter content on the Internet with respect to what is relevant to any particular mobile user. In addition to filtering content, relevant content can be recommended to the mobile user. A significant consequence of doing so is that the interaction barrier presented by hierarchical and search-based interfaces is eliminated. A mobile interface where content is recommended to the user is envisioned and is depicted in Figure 1-2.



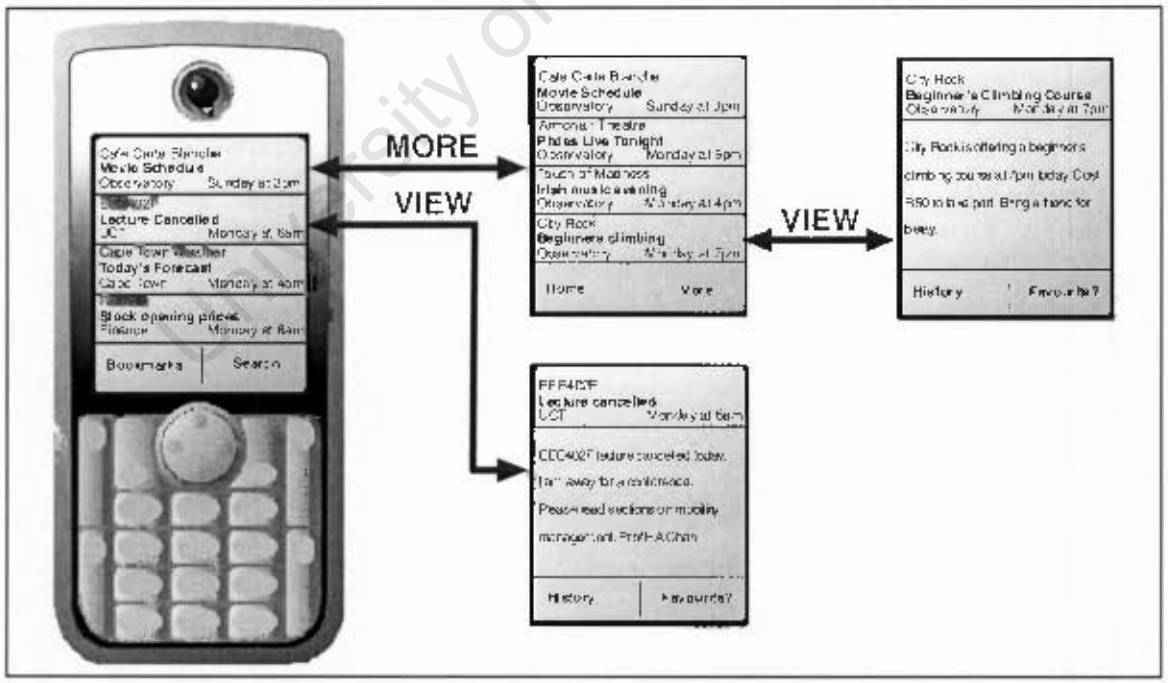



Figure 1-2: Envisioned mobile interface

With the envisioned mobile interface, after the user logs onto the content recommender application, headers are recommended to the user. These headers summarise the content of syndicated material. The user can then either view the content inside one of these headers, or view similar headers. Auxiliary functions such as book marking content headers and searching for headers can also be implemented.

### 1.2.3 Enabling architecture

The mobile interface can be constructed with XHTML or with JAVA. The syndication of content via SMS, and the filtering and projection of relevant content onto mobile phones, can be realised with the architecture proposed in Figure 1-3.

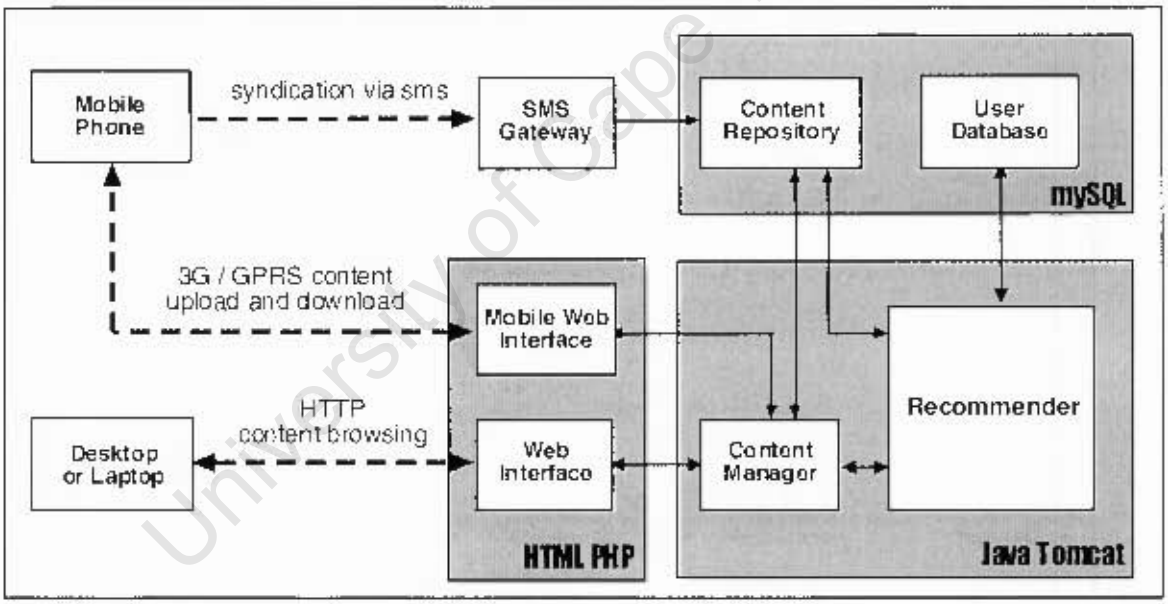



Figure 1-3: Proposed architecture for the content recommender system

All the components in the above architecture are deterministic in their implementation with exception of the recommender component. The function of the recommender component is to select a subset of the mobile web based on

information known about the user. The design of the recommender system is where the difficulty lies because each human being is different.

### 1.2.4 Content-based filtering

Approaching the problem from an idealistic perspective, if the user concerned has explicitly stated the mobile web content that he or she likes, the recommender would simply have to retrieve those resources from the content repository and present them to the user. Suppose that less information was provided, such as a list of keywords representing the user's interests, then content-based filtering [2] techniques such as Latent Semantic Indexing [15] can be applied. Content-based filtering identifies relevant content based on the properties of the item. This method of information filtering is a search-based method. It requires that the user explicitly states what he or she is looking for. Inputting keywords imposes additional interactions with the mobile device, which one would like to eliminate, and is thus not a suitable approach.

### 1.2.5 Collaborative filtering

From an alternative and idealistic perspective, if a person identical to the user concerned can be identified, then that person's preferred content can be recommended to the user concerned. It is this notion that leads to the development of information filtering via collaborative filtering [3]. Although identical people are non-existent, similar people are. Groupings of similar people can be identified analytically or sociologically. Based on the degree of similarity between two users, preferred content by the one can be recommended to the other. A simple, but highly visible example of this notion is:

"*Users, who are interested in content X,*

*are also likely to be interested in content Y*"

This method has two notable advantages over content-based filtering:

**Information is implicitly collected**

Interactions of the user with the web content can be observed passively, such as counting page visits. The user's preferences towards those resources can then be inferred from the page counts. Implicitly obtaining preferences is non-intrusive, and thus does not impose cumbersome interactions on the mobile device.

**Recommendations made on human judgments**

The relationship between similar users and similar content are based purely on the human interactions between them, and are not subject to algorithms that attempt to understand what the resources represent.

To summarise, a software architecture is developed that extends the creation of content to mobile phone users, thus facilitating the generation of more relevant content. Recommending content to the user is proposed to eliminate the interaction barrier presented by hierarchical and search-based interfaces. An investigation concerned with the suitability of employing collaborative filtering to recommend content to the mobile phone is required.



## 1.3 Significance of Problem

Recommender systems have been designed since the 90s beginning with Tapestry [16]. They have been used successfully by e-commerce websites such as Amazon.com, CDNow.com, and Levis.com [17] to recommend products to customers. The application of a recommender in the proposed mobile environment differs from web-based recommender services in three ways:

### 1.3.1 Recommendations are the primary aspect of the service

The applications of recommenders in web-based services have been a secondary function of these environments. Users are attracted to these web sites in the hope of serendipitous finds. Conversely, if a poor recommendation is made, the user would simply ignore it.

In the mobile phone environment, the recommender takes a primary, active role in the application taking full responsibility for the usability of the application. Thus, a robust recommender needs to be designed that has little tolerance to poor recommendations.

### 1.3.2 Preferences are implicitly collected

Web services sometimes require that the user explicitly rate items of preference before any recommendations are made. Such a process on a mobile device may deter users. Hence, a collaborative filtering recommender system that generates recommendations from purely implicitly collected data is required.

### 1.3.3 All preferred items are recommended

The motivation behind the design of recommenders has been from the e-commerce industry to recommend items for sale. E-commerce implementations of recommender systems primarily seek to identify undiscovered items. The design of the recommender for the mobile environment requires that both discovered and undiscovered items are presented to the user.

## 1.4 Hypothesis

This investigation is based on the hypothesis that collaborative filtering can be employed to effectively identify preferred content for the mobile phone user by observing the user's interactions with the recommended content.

## 1.5 Thesis Objectives

The purpose of the investigation is to:

- Review the various methods of implementing collaborative filtering and identify their relative advantages and disadvantages
- Determine which method will be most suitable for the proposed mobile phone application
- Identify the necessary conditions required to make accurate recommendations
- Implement some of these methods and evaluate how quickly they can accurately recommend content.

## 1.6 Scope and Limitations

### 1.6.1 Effectiveness of recommending content is assumed

The purpose of recommending content is to reduce the psychological distance between the user and the content that is imposed by search-based and hierarchically-based interfaces. The effectiveness of this method cannot be determined empirically. Such an evaluation would require that users be subjected to the three different interfaces and observed how easily they adopt them. Such an experiment is beyond the scope of this research, and thus the effectiveness of recommending content is left as an assumption.

### 1.6.2 Collaborative filtering evaluated on speculative data

The feasibility of implementing such a recommender system is in question, thus, no actual usage data is available to the author. Hence, datasets containing user preferences are used to evaluate the algorithms. These datasets can only approximate how users would interact with the recommender system. Thus, the evaluation of collaborative filtering is purely speculative.

### 1.6.3 Content filtering can be based on the user's location

It is assumed that content-based filtering can be applied to reduce the amount of content that collaborative filtering will operate on, hence improving the learning rate of these schemes. The most natural way that adheres to the proposed community centric design is to filter content based on the user's physical location.

Although content-based filtering has required the user to explicitly specify keywords of interest, it can be non-invasively implemented by integrating it with location based services (LBS). High-end mobile phones are being equipped with GPS chips which enable service providers from pinpointing the user's location[2]. Low-end devices can be located using radiolocation and trilateration based on the signal-strength of the closest cell-phone towers.

### 1.6.4 Cold-starting scenarios are investigated

A prior literature review of collaborative filtering algorithms suggests that a 60% accuracy rate of recommendations can be made. The concern when applying these algorithms to the proposed mobile environment is less about the accuracy that can be reached, but more about how quickly the accuracy rate can be reached. Hence, the evaluation of the collaborative filtering algorithms is limited to cold-start

[2] The Can Spam Act passed in 2005 requires that users specifically opt-in to enable service providers to query their current locations

scenarios. Cold-start refers to the situation where no information regarding the users' preferences are available.

## 1.7 Thesis Outline

An exploration into the various collaborative filtering recommendation techniques begins in Chapter 2. Two major classes of algorithms are identified, namely memory-based techniques and model-based techniques. The first of these classes is reviewed in this chapter, discussing in detail two popular techniques: user-based collaborative filtering and item-based collaborative filtering. Finally, their relative advantages and disadvantages are discussed.

In Chapter 3, the class of model-based collaborative filtering techniques is introduced. Two of the model based techniques are explained in detail. Personality diagnosis as a simple, yet effective technique and a generic clustering method of collaborative filtering is explained. Finally, the remaining set of model-based techniques of collaborative filtering is briefly reviewed.

An explanation of how collaborative filtering would be applied to a mobile phone interface is explained in Chapter 4. An evaluation method using datasets to simulate human behaviour is developed and key metrics to evaluate the effectiveness of the algorithms are defined. A discussion of the parameters that would influence the behaviour of the algorithms is speculated and finally a series of test cases are developed.

The most significant results, along with an analysis of these, are presented in Chapter 5. Here, observations are made on the effectiveness of the algorithms on different sets of users and different sizes of communities.

Based on the results analysed, conclusions are drawn on the effectiveness of employing collaborative filtering and are presented in Chapter 6. Recommendations for future work are made in Chapter 7.

Finally, in the appendix, the source code to the major simulation methods is included, with a detailed explanation of these methods and instructions on how to compile them in Matlab.

# Chapter 2 Memory-based Collaborative Filtering

Collaborative filtering is a technique developed to make predictions about an individual's preferences towards unobserved items. These predictions are based on the past preferences of the individual and the preferences of the other members of the community. It is based on the axiom that like-minded individuals share similar interests. Collaborative filtering has been identified as the most successful recommendation technique to date [4, 18].

Two distinctive classes of collaborative filtering have been identified [3], namely memory-based collaborative filtering and model-based collaborative filtering. Memory-based algorithms identify a neighbourhood of similar users around the active user. Two memory-based algorithms are explained in this chapter, namely user-based collaborative filtering and item-based collaborative filtering and finally their relative advantages and disadvantages are discussed.



## 2.1 Overview of the Collaborative Filtering Process

Collaborative filtering models a community of users and a pool of resources. A resource, in this model, is any object with which the user can have a subjective relationship. Typically these resources are articles, movies, music or books. Each user is different, thus each user will have certain preferences towards these resources. The process of collaborative filtering is performed in four phases and is depicted in Figure 2-1.

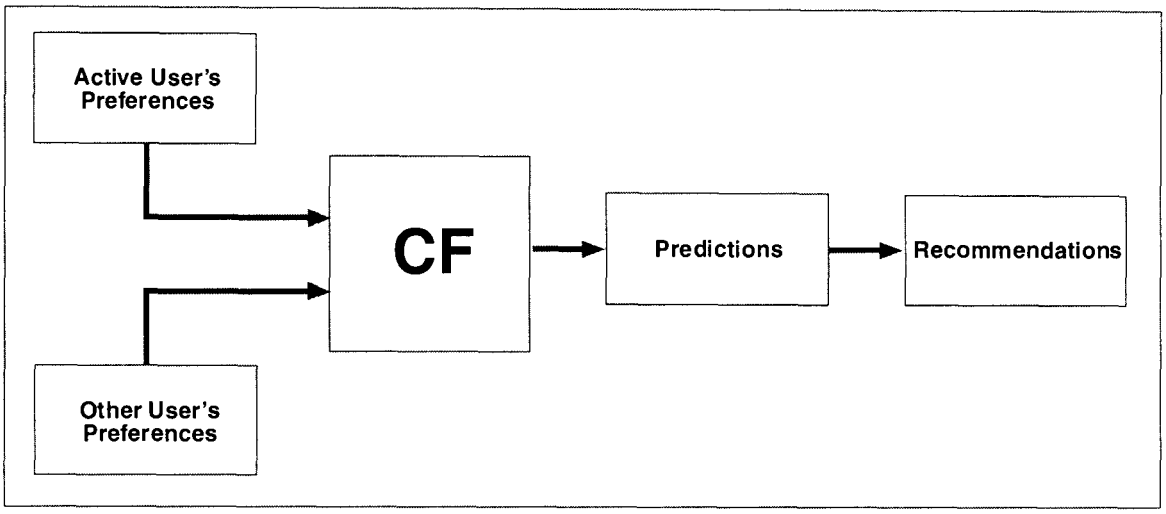



**Figure 2-1: The collaborative filtering (CF) process**

In the first phase of the collaborative filtering process, the preferences of the active user (the individual whom recommendations are made to) and the preferences of the other users in the community (collaborative information) are collected. Secondly, the preferences of the active user are compared against the preferences of the other users and a neighbourhood of similar users around the active user is constructed. Predictions of items that the active user has not observed are generated from the preferences of the users in his or her neighbourhood. Finally, the items with the highest predictions are recommended to the active user.

The details of this process are explained in the following sections.



### 2.1.1 Construction of Model

To provide a formal explanation of how collaborative filtering is performed, a model is constructed. Formally, let community $C$ be the group of $n$ individuals each denoted by $P_i$. Let there be $m$ resources in a pool $L$ associated with this community, each denoted by $R_j$. A particular individual is isolated from the

community and is denoted as $P_a$, the active user. Figure 2-2 depicts these constructions.

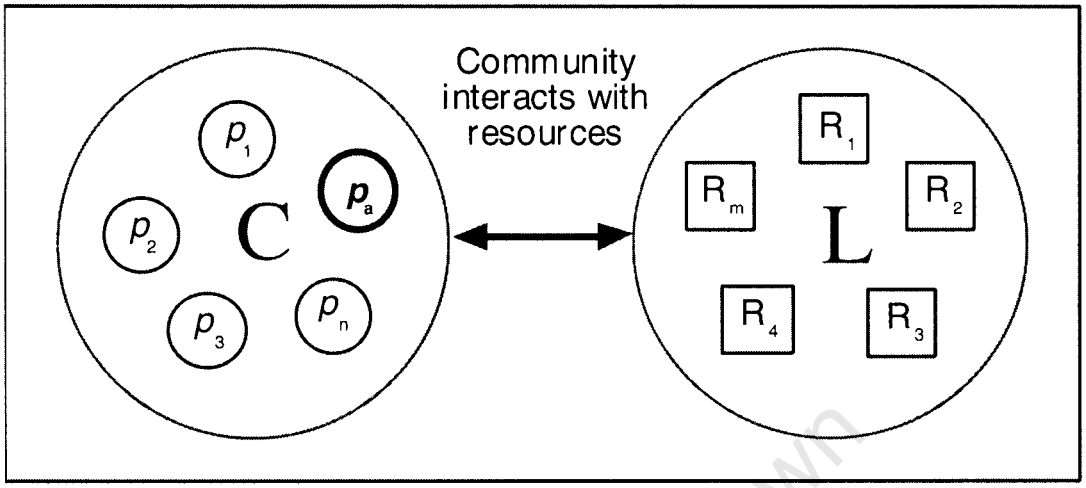



**Figure 2-2: A community of individuals with an associated pool of resources**

### 2.1.2 Data collection

User preferences towards resources are gathered either explicitly or implicitly. Explicit data collection includes

- Presenting resources to the user and requesting that they rate the resources.

Implicit data collection can be performed in numerous ways. Some means include:

- Recording the resources that the user views
- Observing the time spent viewing an item
- Making inferences based on stereotypes

Collaborative filtering that is applied to data implicitly collected is sometimes referred to as log-based collaborative filtering because the user's activities are logged.

These preferences are translated into a numerical scale. The range of this scale is from 0.0 to 1.0. Zero represents no information and 1.0 represents the highest preference. These preferences are stored in a user database referred to as the user-item matrix. In this database, the value $v_{i,j}$ represents the preference of user $i$ towards item $j$. Figure 2-3 depicts this database.

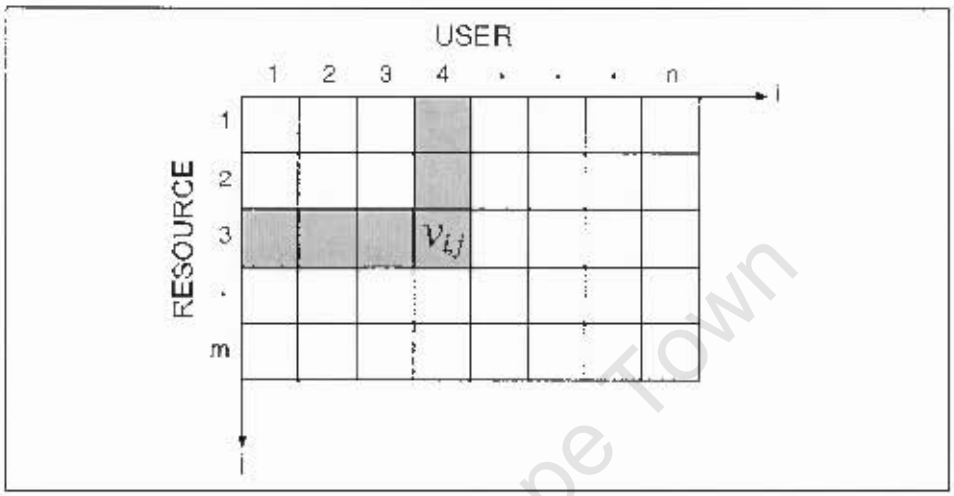



Figure 2-3: User-item matrix

This database is sparsely populated as each user may have only explicitly stated preferences to a small number of items. This matrix is primarily populated with zeros. Hence, the database is often encoded as a sparse matrix in order to save memory.

An individual's preferences towards a resource are a subjective quality of that individual. These preferences are translated into a numerical scale referred to as ratings. The obvious reason for differences in ratings is the differences in preferences of the individuals. There are, however, more subtle reasons for the differences. In the case of

- explicit ratings, how the user translates his or her preferences to a numerical value has an influence on the ratings

- implicit ratings, how often the user engages with resources that he or she likes has an influence on the ratings

Once preference information is collected from users, the collaborative filtering can be applied.

## 2.2 User-based Collaborative Filtering

User-based collaborative filtering [3] takes user preference information and constructs a neighbourhood of similar users around the active user. Sometimes this scheme is referred to as the k-nearest neighbour scheme for the reason that k similar users are identified with the active user.

The difference in ratings imposed by different users employing different rating techniques is normalised by calculating the mean vote. If $I_i$ is the set of items user $i$ has voted, then we can define the mean vote for user $i$ as:

$$\overline{v}_i = \frac{1}{|I_i|}\sum_{j \in I_i} v_{i,j} \tag{2.1}$$

The essence of the assumption that collaborative filtering makes is that similar users share preferences towards similar resources. Hence, we assume that the predicted vote of the active user for item $j$, $p_{a,j}$ is a weighted sum of the votes of the other users:

$$p_{a,j} = \overline{v}_a + \kappa \sum_{i=1}^{n} w_{a,i}(v_{i,j} - \overline{v}_i) \tag{2.2}$$

These weights $w_{a,i}$ represent the similarity between the active user $a$ and user $i$. Two methods to calculate these weights are discussed in the following sections.

### 2.2.1 Pearson Correlation

The Pearson correlation has been employed to measure the strength of the similarity, represented by the weights $w_{a,i}$, between two individuals [4]. A correlation is a statistical relationship that indicates the strength of a linear relationship between two linear variables. The Pearson correlation is a specific type of correlation obtained by dividing the covariance of the two variables by the product of their standard deviations. Covariance is the measure of how much two random variables vary together.

Specifically, the computed correlation is:

$$w_{a,i} = \frac{\sum_j (v_{a,j} - \overline{v}_a)(v_{i,j} - \overline{v}_i)}{\sqrt{\sum_j (v_{a,j} - \overline{v}_a)^2 \sum_j (v_{i,j} - \overline{v}_i)^2}} \tag{2.3}$$

The above equation evaluates the correlation between the active user $a$ and user $i$. The variable $j$ represents the set of items that both the active user $a$ and user $i$ have rated. The maximum possible value is 1 and this represents a perfect positive correlation, whilst the minimum possible value is -1 and this represents a perfect negative correlation.

Two adaptations to the Pearson correlation method are discussed below.

#### 2.2.1.1 Significance weighting

Co-rated resources refer to resources that the active user and the other user in question have rated. If two users have a small number of co-rated resources, it is possible that their correlations will be high simply by chance. Such a case would lead to the identification of false neighbours and consequently generate poor recommendations.

An approach to alleviate this situation is by implementing a technique called significance weighting [19]. With this method, if the number of co-rated resources $m_c$ is less than $N$, where $N$ is some predetermined number, then the resultant weight from the correlation computation is multiplied by $m_c / N$.

#### 2.2.1.2 Default voting

Default voting is a technique used to deal with the sparsity present in the user-item matrix. The correlation computation is performed on the intersection of ratings of the resources between the active user and the other user. If the intersection is small, then the correlation computation will result in a poor similarity measure. Default voting extends the computation of the correlation to the union of ratings of the active user and the other user. For the unobserved items for both users, represented by $n$, a default vote $d$ is inserted into $k$ of these items.

$$w(a,i) = \frac{(n+k)(\sum_j v_{a,j} v_{i,j} + kd^2) - (\sum_j v_{a,j} + kd)(\sum_j v_{i,j} + kd)}{\sqrt{((n+k)(\sum_j v_{a,j}^2 + kd^2) - (\sum_j v_{a,j} + kd)^2)((n+k)(\sum_j v_{i,j}^2 + kd^2) - (\sum_j v_{i,j} + kd)^2)}} \quad (2.4)$$

### 2.2.2 Vector Similarity

Vector similarity has been used in the comparison of documents in the domain of web-based search engines. Comparisons are made based on a table of word frequencies [20]. This method has been adopted for collaborative filtering [3]. Here, the frequency of resource observations is measured, and the cosine between the resultant vectors is measured. This is performed by calculating the dot product between the vectors and normalising the result with the Euclidean distances of the vectors.

$$
\begin{aligned}
w_{a,i} &= \cos(\mathbf{u}_a, \mathbf{u}_i) = \frac{\mathbf{u}_a \cdot \mathbf{u}_i}{\|\mathbf{u}_a\|_2 \times \|\mathbf{u}_i\|_2} \\
&= \sum_{j \in b_{a,i}} \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}}
\end{aligned}
\tag{2.5}
$$

A cosine value of zero implies that the two vectors were orthogonal and thus there is no similarity between the two given individuals. The squared terms are present to prevent users that have observed more resources than others from influencing the similarity measure.

### 2.2.2.1 Inverse user frequency

Inverse user frequency is a modification used to improve the similarity measured by vector similarity. Words that are universal should not affect the similarity measure. Some words are present in all documents and thus do not provide information. Likewise for collaborative filtering, resources that all users like provide no information about the individuality of the user.

Let $f(j)$ be the frequency of votes on item $j$. If everyone has rated item $j$, then $f(j)=0$. Integrating this notion into the weights, we have:

$$
w(a,i) = \frac{\sum_j f_j \sum_j f_j v_{a,j} v_{i,j} - (\sum_j f_j v_{a,j})(\sum_j f_j v_{i,j})}{\sqrt{UV}} \tag{2.6}
$$

where

$$
U = \sum_j f_j (\sum_j f_j v_{a,j}^2 - (\sum_j f_j v_{a,j})^2)
$$

and

$$
V = \sum_j f_j (\sum_j f_j v_{i,j}^2 - (\sum_j f_j v_{i,j})^2)
$$

### 2.2.3 Case Amplification

Case amplification can be performed in conjunction with Pearson correlation or vector similarity methods when calculating the weights. With this modification, the values of the ratings are transformed to emphasize weights that are closer to one, and punish weights that are closer to zero.

Breese transforms the votes as follows:

$$w'_{a,i} = \begin{cases} w_{a,i}^{\rho} & \text{if } w_{a,i} \geq 0 \\ -(-w_{a,i}^{\rho}) & \text{if } w_{a,i} < 0 \end{cases} \tag{2.7}$$

A typical value for $\rho$ used is 2.5

This variable is referred to as the case amplification power. Its purpose is to reduce noise in the data. If the correlation is high, such as $w_{a,i} = 0.9$, then after case amplification, the correlation remains high at $w'_{a,i} = 0.9^{2.5} \simeq 0.8$. If the correlation is low, such as $w_{a,i} = 0.1$, then after case amplification, the correlation becomes negligible at $w'_{a,i} = 0.1^{2.5} \simeq 0.003$.

## 2.3 Item-based Collaborative Filtering

Unlike user-based collaborative filtering, as discussed in the previous section, where similarity computations are performed between users, item-based collaborative filtering [5] performs these computations between resources (items).

The purpose of this method is to predict the preference of the active user towards some unobserved item. The essence of this technique is to represent the unobserved item (referred to as the target item in this model) as a conglomerate of observed items. This notion is depicted in Figure 2-4.

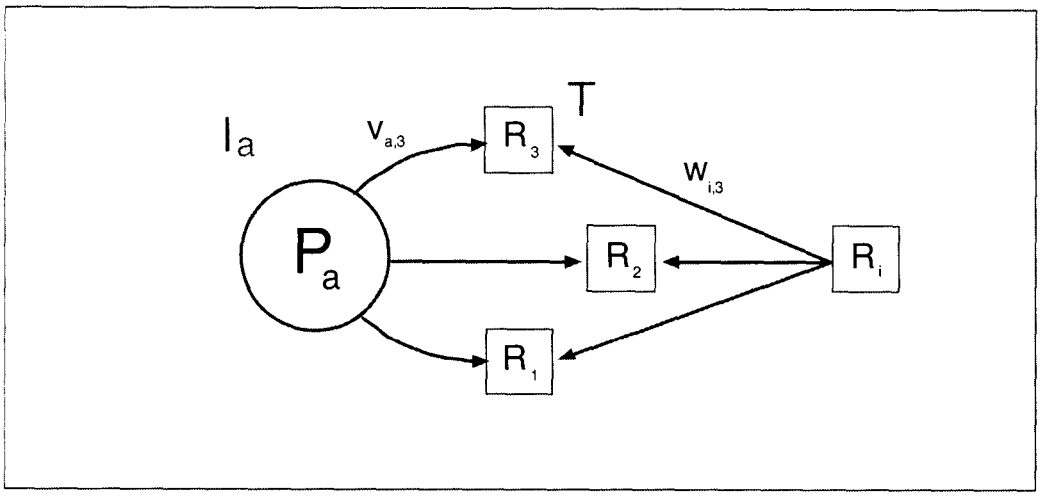



**Figure 2-4: The unobserved item is represented as a conglomerate of observed items**

In the above figure, the unobserved item $R_i$ is represented by a subset of observed items $T$ (where $I_a$ is the set of all observed items by user $P_a$). A conglomerate items is created by computing the item-to-item similarity between the observed items and the unobserved item represented by the weights $w_{i,j}$.

This process of predicting the preference towards the target item is achieved in two stages:

1. Similarities are computed between the set of items that the active user has rated, and the target item.
2. The predicted vote that the active user would have on the unobserved item concerned is calculated based on this user's ratings of the $k$-most similar items to the target item.

### 2.3.1 Item Similarity Computation

In the first stage of item-based collaborative filtering, the set of items that the active user has rated $I_a$ is identified. A similarity computation is then calculated

between each element of the set, and a target item $i$. Suppose an item $j$ is within the set of items that the active user has rated, then all users that have rated both the target item $i$ and item $j$ are identified. These users are said to have co-rated items $i$ and $j$. Their ratings are used in the computation of the similarity $w_{i,j}$. These similarity measures are nearly identical to those used in user-based collaborative filtering. There is a slight change in nomenclature and a slight modification to the Pearson correlation.

This similarity is computed in one of the following manners:

#### 2.3.1.1 Cosine-based similarity

The cosine-based similarity measured is exactly the same as the one in user-based collaborative filtering, except with a slight change in notation.

Each item can be observed as a vector in a $n$-dimensional space whose positions are determined by the ratings each user has assigned to the item. The cosine of the angle between item $i$ and item $j$ can be used to represent similarity.

$$\begin{aligned} w_{i,j} &= \cos(\mathbf{i}_i, \mathbf{i}_j) = \frac{\mathbf{i}_i \cdot \mathbf{i}_j}{\|\mathbf{i}_i\|_2 \times \|\mathbf{i}_j\|_2} \\ &= \sum_{k \in c_{a,i}} \frac{v_{k,i}}{\sqrt{\sum_{g \in U_i} v_{i,g}^2}} \frac{v_{k,j}}{\sqrt{\sum_{g \in U_j} v_{j,g}^2}} \end{aligned} \tag{2.8}$$

#### 2.3.1.2 Correlation-based similarity

As used in the user-based correlation techniques, the Pearson correlation can be computed to indicate similarity. The set of users that co-rated items $i$ and $j$ are denoted by $c_{i,j}$.

$$w_{i,j} = \frac{\sum_{u \in c_{i,j}} (v_{u,i} - \overline{v}_i)(v_{u,j} - \overline{v}_j)}{\sqrt{\sum_{u \in c_{i,j}} (v_{u,i} - \overline{v}_i)^2 \sum_{u \in c_{i,j}} (v_{u,j} - \overline{v}_j)^2}} \tag{2.9}$$

#### 2.3.1.3 Adjusted cosine similarity

The similarity computations in the case of the item-based scheme are calculated along the columns of the user-item matrix, unlike in the user-based scheme, where the computations are calculated along the rows. In the user-based case, the bias of the rating scale used by each user is mitigated by subtracting the mean vote of each user. In the item-based case, when two items are in comparison, more than two users are involved in the calculation. Hence, the corresponding user average from each co-rated pair is subtracted.

$$w_{i,j} = \frac{\sum_{u \in U} (v_{u,i} - \overline{v}_u)(v_{u,j} - \overline{v}_u)}{\sqrt{\sum_{u \in U} (v_{u,i} - \overline{v}_u)^2 \sum_{u \in U} (v_{u,j} - \overline{v}_u)^2}} \tag{2.10}$$

Here $\overline{v}_u$ is the average of the ratings of the $u$-th item.

### 2.3.2 Prediction Computation

Following the similarity computations between each item the active user has rated with the target item, the $k$-most similar items can be identified. With this technique the unobserved target item is represented as a conglomerate of items that the active user has observed.

The rating that the active user would have towards the unobserved item is equivalent to the rating he or she would have towards the conglomerate set, and can be calculated in two ways:

#### 2.3.2.1 Weighted Sum

Let $T$ be the set containing the $k$-most similar items to the target (unobserved) item given from the item-similarity computation. The active user's preferential perception of the unobserved item is calculated by considering his or her perception of the weighted sum. Technically, this is done by calculating a weighted sum of the user's preferences towards the set $T$. Specifically:

$$p_{a,i} = \frac{\sum_{k \in T} w_{i,k} \times v_{a,k}}{\sum_{k \in T} |w_{i,k}|} \tag{2.11}$$

#### 2.3.2.2 Linear Regression

This method of prediction computation is exactly the same as the weighted sum method discussed in the previous section, except, instead of using the active user's ratings towards the set of similar items, approximated ratings are used. These approximated ratings are derived from a linear regression.

The disadvantage of this method is that if there are merely 1000 items, there will be one million linear regressions and up to two million regressors. The consequences are an increase in space requirements, a decrease in response time and data overfitting. A response to this problem is suggested by Lemire et. al. [21] et al with their Slope One schemes and by Vucetic et. al. [22] .

## 2.4 Discussion

Memory-based collaborative filtering generates recommendations by identifying a neighbourhood of similar users to the active user (or similar items to the target item). This process is conducted by computing a similarity measure across the entire user-item matrix.

Although more accurate schemes exist, user-based collaborative filtering using a Pearson correlation with case amplification has been shown to be a reasonably accurate scheme [3]. These methods, however, suffer from scalability issues. Both the response time of user queries and the space requirement increases exponentially with each additional user or item.

Item-based collaborative filtering provides a scalable system with respect to the addition of users. This scheme has been reported to work best [5] and has been commercially implemented by Amazon.com [6]. This scheme uses linear regression to make predictions, and simpler yet effective means to do so have been suggested via Slope One schemes [21].

Memory-based collaborative filtering schemes, however, do not offer the network operator any intuitive reasoning behind the recommendations made. Model-based collaborative filtering methods allow the operator to observe the underlying relationship between users and items, thus promoting scrutiny of recommendations and possible amendments.

# Chapter 3 Model-based Collaborative Filtering

Model-based collaborative filtering has been developed on the lemma that a deeper understanding of the factors that affect the behaviour of individuals would help make more reliable recommendations of content. Unlike memory-based collaborative filtering, as described in the previous chapter, model-based schemes compile a model prior to making recommendations.

Two model-based schemes are explained in this chapter, namely Personality Diagnosis and a generic clustering method.

## 3.1 Personality Diagnosis

The technique of collaborative filtering named 'Personality Diagnosis' was developed by Pennock et al. [7]. In this model, a postulation is made that the preferences that each user reports regarding their preferences towards resources are a manifestation of their underlying personality types. As the name implies, the reported ratings are "symptoms" of an underlying "disease" and the objective of the scheme is to diagnose the individual into a particular personality type.

Behind the preferences reported exists a personality type. These personality types are described by a vector:

$$\mathbf{R}_i^{true} = \left\langle R_{i1}^{true}, R_{i2}^{true}, \ldots, R_{im}^{true} \right\rangle \tag{3.1}$$

The vector represents the "true" preferences of user $i$ towards the resources present in the community. This model incorporates the concept that a given user will report his or her preferences differently on varying occasions, influenced by

factors such as mood and biases. The word "true" is used to differentiate between the preferences that have been reported by the user, and the preferences that the user actually holds. The variations from the true preferences are modelled with Gaussian noise where the true preference is the mean of an independent normal distribution.

$$P(R_{ij} = x \mid R_{ij}^{true} = y) \propto e^{-\frac{(x-y)^2}{2\sigma^2}} \tag{3.2}$$

For example, when user $i$ reports that his or her preference toward item $j$ is $x$, he or she is modelled to be drawing a random sample from an independent normal distribution with mean $R_{ij}^{true} = y$. In other words, each rating that the user reports is a fluctuation from the mean. A free parameter, $\sigma$, is introduced into the model. Pennock et. al. initially sets this value to the variance of the ratings data, and then tunes it to a value of 2.5.

A hypothesis is then asserted. The true personality of the active user is representative of another user in the database. There are $n$ personality types: $\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n$ - one for each user. Initially, it is equally probable that the active user belongs to one of these personality types.

$$P(\mathbf{R}_a^{true} = \mathbf{R}_i) = \frac{1}{n} \tag{3.3}$$

The above probability measure represents the prior probability. This is the probability that the active user belongs to one of the personality types without any additional information.

After the active user has reported his or her ratings, the posterior probability of the asserted hypothesis can be calculated using Bayes' Rule.

$$P(\mathbf{R}_a^{true} = \mathbf{R}_i \mid R_{ai} = x_1, \ldots, R_{am} = x_m)$$
$$\propto P(\mathbf{R}_a^{true} = \mathbf{R}_i) \cdot P(R_{ai} = x_1 \mid R_{a1}^{true} = R_{i1}) \cdots P(R_{am} = x_m \mid R_{am}^{true} = R_{im}) \quad (3.4)$$

The posterior probability is computed between the active user and each other user in the database. This value represents the probability that the active user's ratings are representative of another user in the database. These posterior probabilities are used to construct a probability distribution of unobserved items for the active user. Specifically,

$$P(R_{aj} = x_j \mid R_{a1} = x_1, \ldots, R_{am} = x_m)$$
$$= \sum_{i=1}^{n} P(R_{aj} = x_j \mid \mathbf{R}_a^{true} = \mathbf{R}_i) \quad (3.5)$$
$$\cdot P(\mathbf{R}_a^{true} = \mathbf{R}_i \mid R_{a1} = x_1, \ldots, R_{am} = x_m)$$

where all $j$ are elements of the set of items that the active user has not rated.

The above probability distribution represents the probability that the active user will like any given item. Recommendations are made by simply identifying the top-$n$ items that have not been rated by the user.

## 3.2 Clustering Methods

Clustering methods of collaborative filtering strive to fit the active user into a predefined cluster of other users, and recommendations are made to the active user based on the preferences of the other users in that cluster. Ungar et. al. [8] explores a method of clustering which seeks to identify natural clusters of people and items. He postulates that if these natural clusters can be identified, then reliable recommendations with less false positives, can be made.

The notion of classifying users and items is depicted in Figure 3-1.

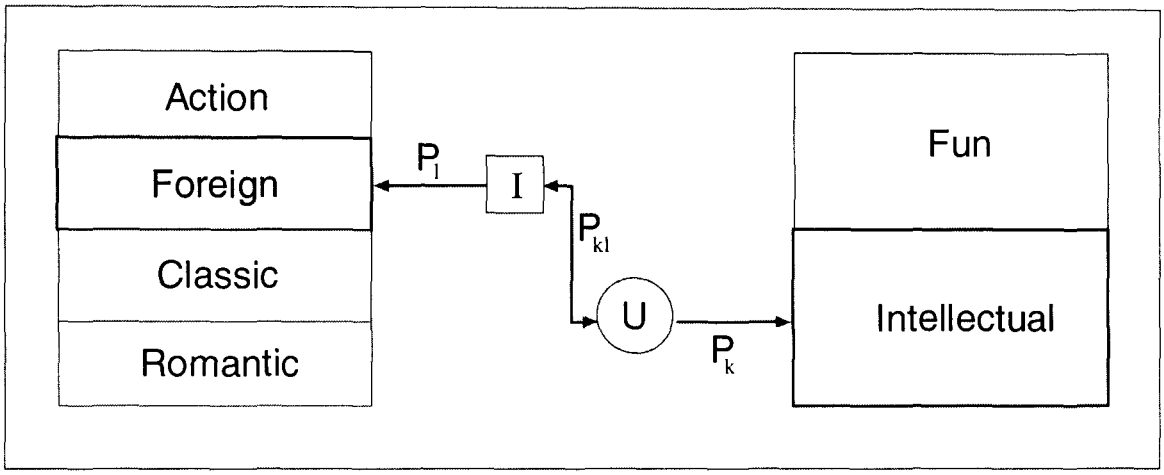



**Figure 3-1: Grouping items and users into classes**

In the above figure, four classes of items (specifically movies) and two classes of people are identified. The relationship between these classes and the users / items are described by the following model parameters.



### 3.2.1 Model parameters

This method of collaborative filtering introduces three parameters. The first two are base rates and the third is a link probability.

#### 3.2.1.1 Base rates

The base rate is the probability that a person is in class $k$ or an item is in class $l$. $P_k$ is calculated by observing the fraction of people in the community that belong to class $k$. $P_l$ is calculated by observing the fraction of items in the community that belong to class $l$.

#### 3.2.1.2 Link probability

The link probability $P_{kl}$ is the probability that a person in class $k$ observed an item in class $l$. It is calculated by counting the number of people in class $k$ who

observed an item in class $l$ and dividing by the sample space. The sample space is the number of people in class $k$ multiplied by the number of items in class $l$.

### 3.2.2 K-means clustering

One means to identify user-item memberships to classes is with K-means clustering [23, 24]. This method is used to cluster objects into $k$ partitions. This is accomplished by minimising the total intra-cluster variance.

$$V = \sum_{i=1}^{k} \sum_{j=1}^{n} | \mathbf{u}_j - \mu_i |^2 \tag{3.6}$$

where $\mu_i$ are the centroids of the clusters and the Euclidean distance is measured between the user-item vectors. Identifying the clusters can be performed in the following steps [25]:

1. Choose the number of clusters, $k$
2. Randomly generate $k$ random points as cluster centres
3. Assign each user / item to the cluster centre
4. Re-compute the new cluster centres
5. Repeat the two previous steps until the cluster assignment does not change

The k-means clustering process is conducted on users and items:

- People are clustered based on the items they observed
- Items are clustered based on the people that observed them

Then a re-clustering process is conducted:

- People are re-clustered based on the number of items in each item cluster they observed
- Items are re-clustered based on the number of people in each cluster that observed them

There is no formal method of deciding how many times to re-cluster the data. Re-clustering helps when the data is sparse, however, re-clustering too many times results in an over-generalisation of the data and thus the individuality of the users is lost.

### 3.2.3 Gibbs sampling

Gibbs sampling offers an alternative means of identifying user-item class memberships. An advantage that this method offers is that a person or item can be updated one at a time. Gibbs sampling is an algorithm that generates a sequence of samples from a probability distribution. This method of sampling is performed in two phases, namely the assignment phase and the model estimation phase.

#### 3.2.3.1 Assignment phase

In this phase, a person or item is picked at random and is assigned to a class with a probability proportional to the following expression.

$$P_k \prod_l P_{kl}^{\sum_{j:C_j=l} Y_{ij}} (1-P_{kl})^{\sum_{j:C_j=l} (1-Y_{ij})} \tag{3.7}$$

where $Y_{ij}$ is the observed data, $C_i$ the class that person $i$ is in, $C_j$ the class the movie $j$ is in. $C_i = k$ is the probability that person $i$ is in class $k$.

#### 3.2.3.2 Model estimation phase

In the model estimation phase, $P_k, P_l$ and $P_{kl}$ are picked with probabilities proportional to the likelihood of them generating the data. The link probabilities are drawn out of a beta distribution as follows:

$$P_{kl} = \beta(X_{kl} + 0.5\delta_{kl}, N_{kl} - X_{kl} + 0.05\delta_{kl}) \tag{3.8}$$

where

$X_{kl}$ = number of items in class $l$ observed by people in class $k$

$N_{kl} - X_{kl}$ = number of items in class $l$ not observed by people in class $k$

The class membership probabilities are drawn from a multivariate beta distribution. Observations from multivariate beta distributions can be represented by gamma distributions.

$$P_k = \frac{\gamma(count_k + 0.5)}{\sum_k \gamma(count_k + 0.5)} \tag{3.9}$$

and

$$P_l = \frac{\gamma(count_l + 0.5)}{\sum_l \gamma(count_l + 0.5)} \tag{3.10}$$

where $count_k$ is the number of people in class $k$.

## 3.3 Other methods

The statistical foundations of collaborative filtering have been investigated [26], based on the assumption that no other information regarding the users is available except for the preferences that have been expressed. An aspect model is designed where individual preferences are considered as a convex combination of preference factors. A latent class variable is associated with each pair of users and items observed. Latent variables are variables that are not directly observed, but rather inferred. Personality Diagnosis discussed at the beginning of this chapter, is a special case where there is one latent variable influencing the ratings. A class of latent variable models where multiple hidden factors influence each user-item tuple have been researched [27].

A flexible mixture model for collaborative filtering is looked at where the constraint that each user or items needs to belong to one cluster is dropped [28, 29]. These mixture models are summarised and reviewed [30].

The clustering methods can be perceived as a method used to reduce the dimensionality of the user-item matrix, thus reducing the sparsity presented by the problem. Other means of reducing the dimensionality have been suggested with the use of Principle Component Analysis [31] and Latent Semantic Indexing [32, 33]. To prevent potentially useful information being lost during dimensionality reduction, associative-retrieval techniques are explored [34].

Content-boosted collaborative filtering methods [35, 36] incorporate additional information about the user to generate recommendations. These are hybrid methods developed to take advantages from both collaborative filtering and content-based filtering.



## 3.4 Discussion

The Personality Diagnosis method can be seen as a specific type of clustering method, with only one individual per cluster. This method is simple to implement and only has one free parameter to be optimised by the network operator. The Bayesian inference used in this method resembles that of a Naïve Bayes classifier. Recent research suggests that the Naïve Bayes approach work much better in real-world situations [37].

The generic clustering method imposes on the network operator having to decide how many times to cluster the data and to choose how many clusters to create. For these methods, the network designer needs intuition regarding the data. The advantage of this method is that missing data is easily handled. K-means as a

method to identify clusters is an ad-hoc method of identifying centroids, and the same centroids are not always found. Gibbs sampling makes it possible to easily add new data, however, this method is computationally expensive.

Model-based approaches require that the model is created offline, and the additions of new items are only reflected in recommendations each time the model is recreated. Nevertheless, constructing a model provides a deterministic query time.

The definite strength of the model-based approaches is that they provide the network operator with meaningful probabilistic interpretations of the data. The relationships between the users and the recommendations can be observed, and modified if necessary, thus providing a robust means of making recommendations.

# Chapter 4 Simulation

The feasibility of recommending content to the mobile phone is in question. While literature suggests that the collaborative filtering algorithms provide a reasonable accuracy rate, little is known about how quickly such accuracy rates can be reached. It is crucial that the behaviour of these algorithms is observed during cold-start scenarios because of the impact they have on the usability of the proposed application. Thus, the purpose of the simulation is to evaluate how well recommendations can be generated during cold-start conditions.

## 4.1 Modelling human behaviour

The feasibility of implementing the proposed recommender system is in question; hence no actual usage data is available to the author. Thus, an empirical approach is conducted with publicly available datasets to model the problem and observe the behaviour of the collaborative filtering algorithms. These datasets are used to model human behaviour. The following sections describe what these datasets are and explain how they are used.



### 4.1.1 Simulation datasets

Two publicly available datasets are used to model human behaviour, namely the MovieLens and the EachMovie datasets. Each of these datasets consists of a large collection of individuals and their ratings of movies.

#### 4.1.1.1 The MovieLens dataset

The MovieLens dataset, courtesy of GroupLens [38], consists of two collections of individual ratings towards movies. The first MovieLens dataset consists of 80 000

ratings of 1 682 movies by 943 users. The second MovieLens dataset consists of approximately 1 million ratings of 3 900 movies by 6 040 users.

The ratings present in the dataset are encoded as integers ranging from one to five.

#### 4.1.1.2 The EachMovie dataset

This dataset was made available by the Compaq Systems Research Center[3]. This dataset consists of 1 261 movies rated by 74 424 users. The ratings are encoded as real numbers ranging from 0.0 to 1.0 in increments of 0.2. A significant difference this dataset has in comparison with the MovieLens dataset is that the number of users is much greater than the number of items.

### 4.1.2 Probability Mass Function

The datasets contain large numbers of explicit user ratings towards movies. Each individual rating towards a movie can be interpreted as the probability that that user will watch that movie. Using all the ratings from a particular individual, a probability mass is constructed that will determine the likelihood of that user watching the movie if it is recommended.

The movies analogously represent the resources discussed in the explanation of the collaborative filtering algorithms. The strategy of evaluating the performance of the algorithms is to use them to identify a set of resources to be recommended, and note which items were observed based on the probability mass function created.

### 4.1.3 Monte Carlo Methods

[3] As of October, 2004, HP retired this dataset and is no longer available for download; however, the dataset is still available within the research community.

The employment of a probability mass function to determine whether the user observes an item or not, subjects the results of the evaluation to a high variance. The probabilistic variance of the results can be countered with the use of Monte Carlo methods. These methods generally refer to a class of computational algorithms for simulating behaviour of various systems. Here, these techniques are applied by repetitively repeating the evaluation, and on multiple users to obtain the true behaviour of the algorithms.

## 4.2 Implemented Algorithms

### 4.2.1 Justifying user-based collaborative filtering

Item-based collaborative filtering has found a wide acceptance due to its scalability. It has been implemented by Amazon.com. Scalability, however, is not an issue that requires consideration when evaluating of collaborative filtering algorithms during cold-start conditions. The important criterion that needs to be evaluated is how quickly accurate recommendations can be made, and not computation speed.

The strength of Personality Diagnosis is its capacity to counteract the fluctuations in the ratings that a user reports which have been biased by their moods. The simulation dataset only models users' behaviour towards items recommended based on their preferences towards these items. Thus, the users' mood is not accommodated in the model. Adding Gaussian noise to the simulation data to model mood and removing this noise with the Personality Diagnosis method would be superfluous.

The clustering approaches to collaborative filtering offer a more robust means to making recommendations. It provides the ability for the network operator to observe the clusters of users in the database and make amendments to the

generated model based on his or her intuition. This approach to recommending content would be taken if the data available was real.

User-based collaborative filtering employing a Pearson correlation and the use of case amplification has been shown to be a reasonably accurate scheme [3]. Although these methods are computationally expensive, the evaluation is aimed at determining the feasibility of recommending content to the mobile device, and hence is not an issue with which to be concerned.

For the above reasons, user-based collaborative filtering has been chosen to evaluate the feasibility of employing collaborative filtering for the proposed application.

### 4.2.2 Simple preference learning scheme

A scheme to benchmark the performance of the user-based collaborative filtering algorithms is required. A simple preference learning algorithm is designed to do so. This scheme simply chooses a random selection of resources to be recommended to the user. If the user observes any of the recommended resources, then the probability of recommending those resources is increased. The scheme eventually recommends all possible resources, and the probability that a given resource will be recommended matches exactly the probability mass function created to model human behaviour.



## 4.3 Simulation Environment

Matlab was chosen as an environment to implement and evaluate the collaborative filtering algorithms. This environment provides a numerical computing workspace. It has a built-in programming language and is supported with a rich collection of

mathematical libraries. In addition, it provides access to databases, and tools to visualise and analyse data.

With this environment, mathematical sequences can be instructed concisely, thus providing a quick means to implement the algorithms. The disadvantage of Matlab is that it is an interpreter and therefore algorithms run slowly. Although routines can be coded in C using the MEX encoder, as explained earlier in this chapter, evaluating the accuracy of the algorithms is the purpose of the simulation and computation speed is not a prerogative.

## 4.4 Simulation method

Making and improving the recommendations presented to the user is an iterative process. As more preference information becomes available, better recommendations can be made. Each one of these iterations in this evaluation model is referred to as a session. The following section elaborates on this.

### 4.4.1 Session operation

The user connects to the recommender service application and evokes a login command providing the application with a unique user identification number. This user's preferences of resources are queried from the user database. The recommender algorithm is then evoked with this user's preferences and the preferences of the other users in the community. The most suitable set of resources are identified to be recommended. How this is achieved is discussed in the following section. The user is presented with a set of headers with their associated content on the mobile device. The client side application (the interface to the recommender service on the mobile phone) notes which headers were observed and these details are sent back to the recommender service application. Finally, when the user logs

off or is disconnected, the headers that were observed are written to the database. This entire process is referred to as a session and is made visible in Figure 4-1.

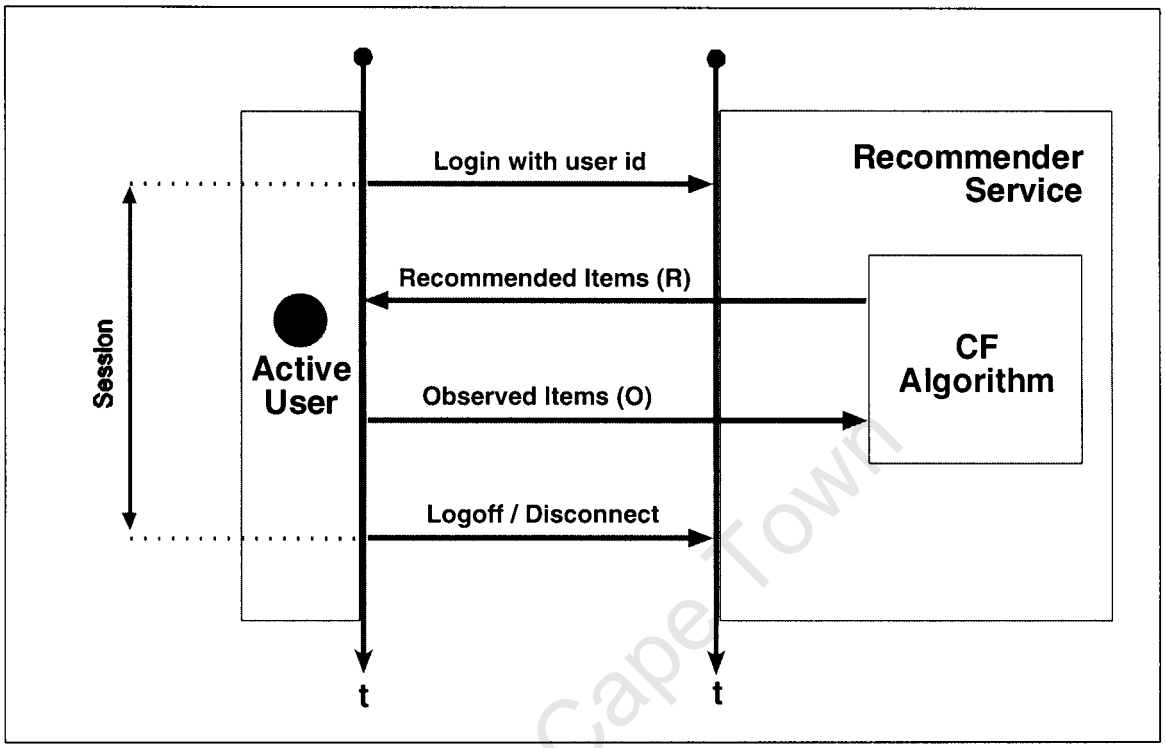



Figure 4-1: Definition of a session



### 4.4.2 Identifying suitable content

After each session, the user has provided implicit feedback on his or her preferred content. This feedback is used to improve the recommendations.

A probability mass function $\delta$ is created for each user. This function encodes the probability that any given resource will be recommended to the user. The initial probabilities that this user will observe a particular resource are each assigned to $1/m$ (where $m$ is the number of resources), thus forming a uniform distribution. A random sample of $n$ resources (where n is the number of items that are to be recommended) is drawn out of this distribution. This set of resources recommended

to the user is marked with a vector $\mathbf{R}$. Specifically, if item $j$ is recommended to user $i$ during session $k$, then $\mathbf{R}_i(j,k)=1$.

None, some or all of the recommended items are observed by the user. These observations are marked with a vector $\mathbf{O}$. A perceived behaviour $\mathbf{B}$ is then calculated by finding the ratio of the number of times a resource was observed, given that it was recommended. The following equation formally defines this matrix.

$$\mathbf{B}_{i,j} = \sum_{k=1}^{s} \frac{\mathbf{O}_i(j,k)}{\mathbf{R}_i(j,k)} \quad \exists \quad \mathbf{R}_i(j,k) \neq 0 \tag{4.1}$$

The perceived behaviour of user $i$ on item $j$, $\mathbf{B}_{i,j}$, is only calculated if item $j$ has been recommended to the user, hence $\mathbf{R}_i(j,k) \neq 0$.

This perceived behaviour is fed into the user-based collaborative filtering algorithm. The details of this algorithm have been described in Section 2.2. In the model described, collaborative filtering has been applied to items that the users have explicitly rated. In this simulation, the algorithm is applied to the perceived preference of the items, translated into a 'rating' by computing the perceived behaviour.

In addition to the perceived behaviour, the set of all observations made from other users is fed into the collaborative filtering algorithm. This process is depicted in Figure 4-2.

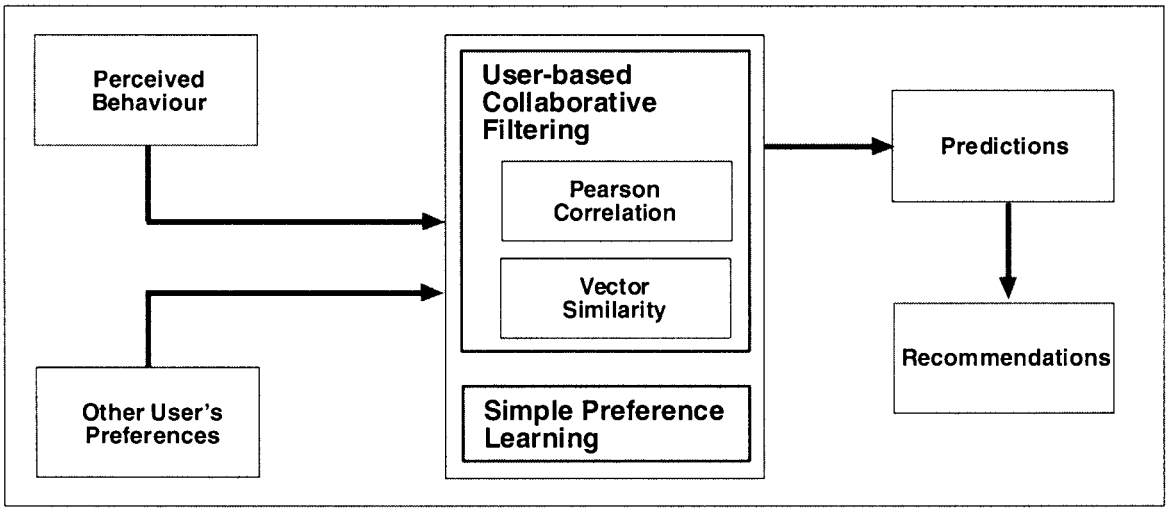



**Figure 4-2: Applying collaborative filtering to perceived behaviour**

User-based collaborative filtering is employed with two methods of computing similarity; Pearson correlation and vector similarity. Both these schemes return a series of predictions towards unobserved items. The variance of these predictions is centred on the mean of the perceived behaviour. The items with the highest $n$ predictions are taken and the corresponding probabilities in $\delta$ (the probability mass function that governs which items are recommended) are increased.

In the simple preference learning case, the process of collaborative filtering is omitted, and the probabilities in $\delta$ are increased purely based on the items that were observed.

This process is repeated for each session. Figure 4-3 provides an overview of the evaluation method. The full source code and a thorough explanation of the code are both provided in Appendix A.

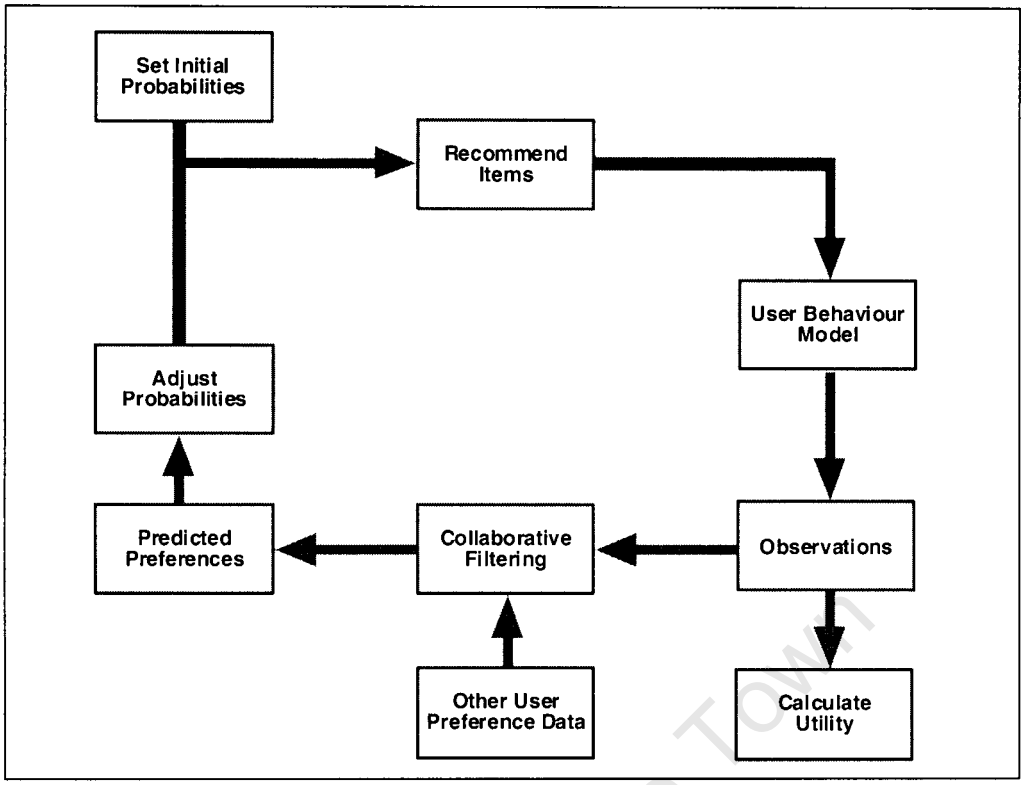



**Figure 4-3: Overview of the evaluation method. Each cycle represents a session.**



## 4.5 Evaluation Metrics

The purpose of simulation is to determine the effectiveness of employing collaborative filtering to recommend items to the mobile device. To quantify this effectiveness, two metrics are developed.

#### 4.5.1.1 Utility

The gratification of the content recommended per session which cannot be measured directly, can however be determined indirectly by measuring the content's usage. The utility of the session can be determined by calculating the ratio between the number of items observed with the number of items recommended. Specifically, for session $s$, the utility metric $U_s$ is defined as:

$$U_s = \sum_{k=1}^{m} \frac{\mathbf{O}_i(k,s)}{\mathbf{R}_i(k,s)} \tag{4.2}$$

#### 4.5.1.2 Learning rate

An important metric to be measured is the learning rate. This metric determines how quickly preferred resources are identified. This metric influences the usability of the proposed recommender system. Two specific conditions are of concern:

- The number of sessions taken to reach 30% accuracy
- The number of sessions taken to reach 60% accuracy

## 4.6 Influential parameters

The following test cases are designed with the intention of providing a well-rounded evaluation of how well the user-based collaborative filtering algorithms behave under various conditions. This investigation particularly focuses on the performance of the algorithms during cold-start scenarios. The performance of these algorithms is governed by certain parameters which are identified and discussed as follows.

The three major factors that influence the behaviour of these algorithms is:

1. Items: the number of resources present in the community
2. The active user: the interest the user has with the items present in the community
3. Collaborative information: the quantity of preference information that is known from other users in the community

### 4.6.1 Items

The number of resources present in the community has an influence on the performance of making recommendations. A large diversity of items decreases the probability that the recommendation algorithms will identify a resource that the

user would prefer. In this context, diversity refers to the presence of resources that are irrelevant to the active user.

Content-based filtering (as described in Section 1.2.4) is a technique used to reduce the diversity of the available resources. With this technique, a subset of all resources is drawn by restricting content to those which contain certain keywords. It is assumed that content-based filtering can be integrated with Location Based Services and applied to the proposed recommender system to filter content based on the user's current location. Thus, the diversity of the available content that the collaborative filtering algorithms operate on is reduced.

The proposed application is aimed at recommending content to small communities. An evaluation will be performed on community sizes in the order of one hundred to one thousand available resources.

### 4.6.2 Active user

The performance of the recommender algorithms are measured with respect to how many items the active user observes. Hence, the engagement of the active user with the available resources in the community will have an influence on the accuracy of the recommendation algorithms. The active user's engagement with these resources can be quantified with the following metric.

Coverage of the user is defined as the proportion of the total resources that the active user has rated. This metric provides an indication of the depth the active user has/will engage with the resources. The user's coverage will influence both the probability of making an observation of a recommended resource and also the information available to the collaborative filtering algorithms.

A series of cases will be simulated to observe their influence on the utility. These are specifically, where users had a preference towards 2.5%, 5%, 10%, 20%, 30% and 40% of the items in the community. These experiments will be conducted on sets of users, numerous times, and the average utility will be calculated.

### 4.6.3 Collaborative Information

The collection of ratings from the other users in the community will have an influence on how well the collaborative filtering algorithms make predictions. Certainly, if there are no other users in the community that have observed the items that the active user has observed, applying collaborative filtering has no advantage.

An experiment will be conducted where the amount of collaborative information present is varied. This will be achieved by randomly removing users from the collaborative information set such that the amount of information available represents 5%, 10%, 20% and 50% of the original set. Again, the experiment will be repeated numerous times to observe the true behaviour of the algorithms.

# Chapter 5 Analysis of Results

The influential parameters that affect the performance of the user-based collaborative filtering algorithms were identified in the previous chapter. The effects of these parameters have been observed, and these results along with an analysis of these results are presented in this chapter. Testing was time consuming, hence only significant cases were considered for analysis.

## 5.1 User Coverage

User coverage, that is the fraction of items that the user was interesting in, had the primary influence on the utility measured per session. No matter how good the algorithms are, if the user is not interested in the content, then the utility measured will be poor. Different sets of users were chosen from the datasets that had varying coverages towards the data. The results from these sets of users were averaged out and are presented below.



### 5.1.1 2.5% coverage and a comparison with EachMovie

In the first experiment, users from the MovieLens dataset with a coverage of 2.5% towards the 1000 available resources were considered. This equates to any particular user having a preference towards 25 resources. The collaborative information consists of a community of 943 other users providing a total of 74,000 ratings. Five items were recommended per session. Figure 5-1 depicts the results of this experiment.

The above experiment was repeated with the above configuration except with data extracted from the EachMovie dataset. Figure 5-2 depicts the results of this experiment.

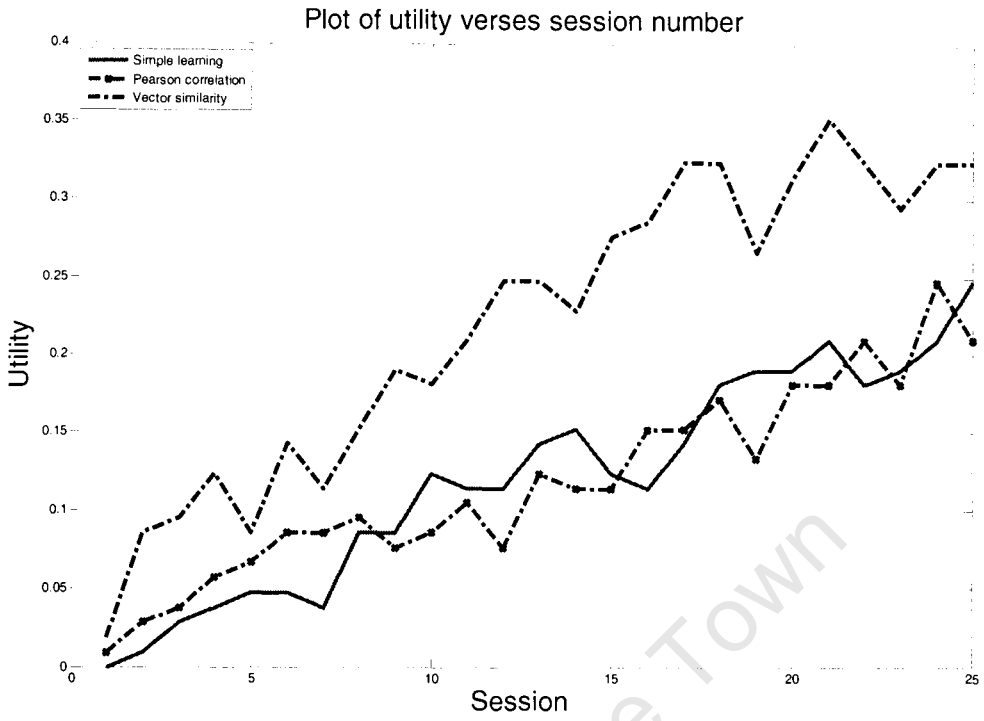



**Figure 5-1: 2.5% Coverage of 1000 items (MovieLens dataset)**

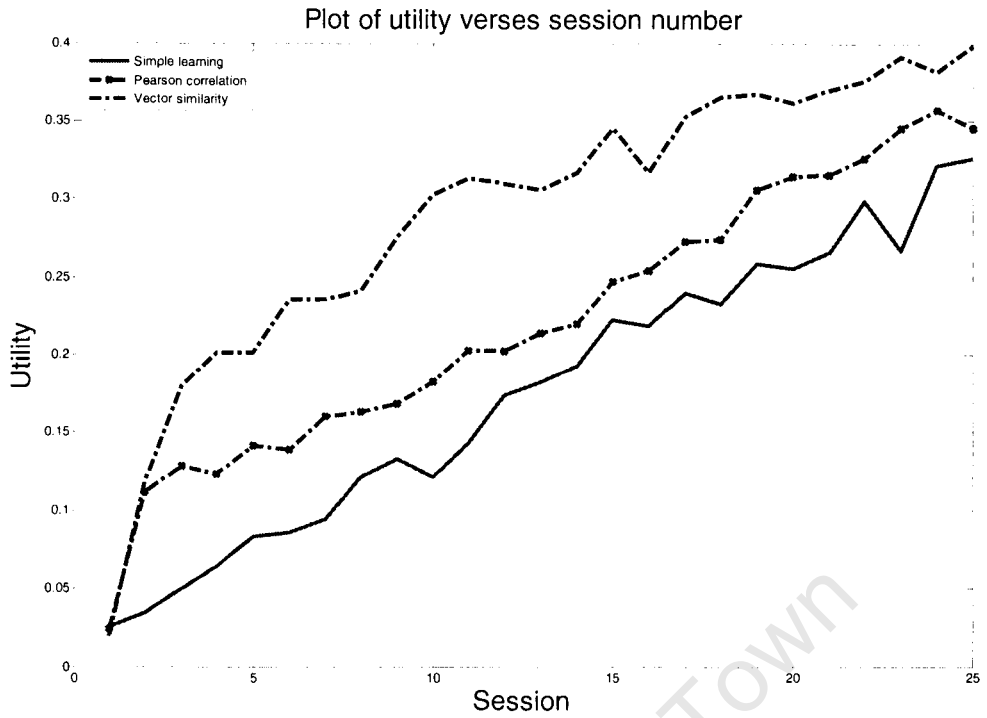



**Figure 5-2: 2.5% Coverage on 1000 items (EachMovie dataset)**

The graphs show the results of three schemes: Simple learning, Pearson correlation and vector similarity. Vector similarity outperforms both the Pearson correlation scheme and the simple learning scheme. It takes about 15 sessions to reach a utility of 0.3. This means that after 15 sessions, one third of the resources that are recommended to the user are preferred. The Pearson correlation technique barely outperforms the simple preference learning scheme. The consistency of the learning rate observed in the EachMovie dataset suggests that the general behaviour of the collaborative filtering algorithms is being correctly observed.

What can decisively be concluded from the results of both experiments is that the vector similarity method of identifying preferred content is more suitable during the cold-start condition than the Pearson correlation method.

### 5.1.2 10% coverage

The same experiment was conducted, however, on a set of users with a 10% coverage. Figure 5-3 depicts the results of this experiment.

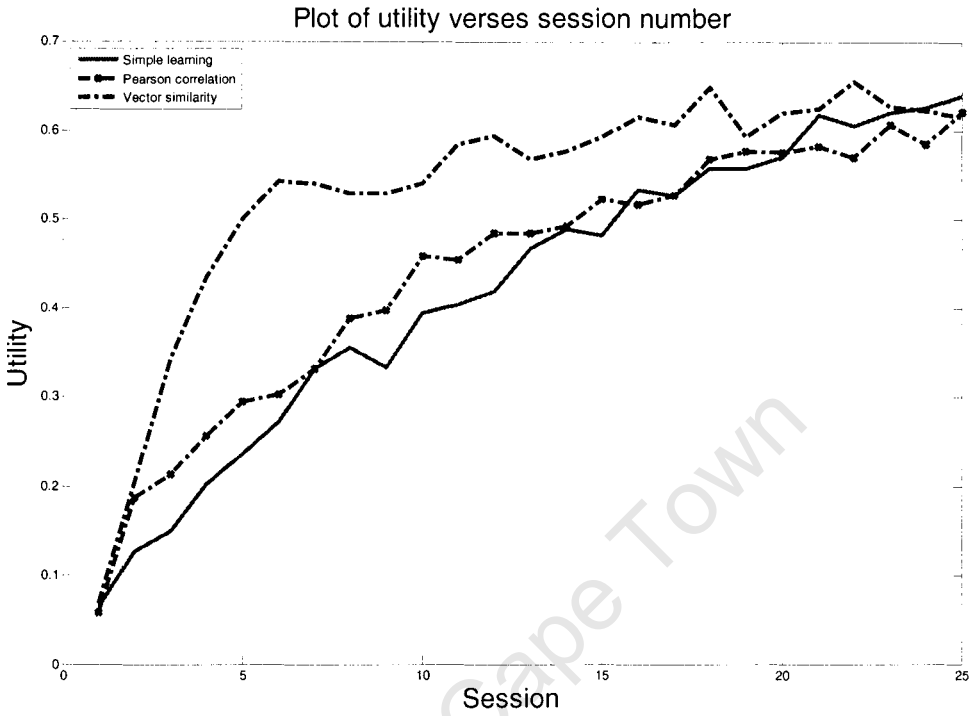



**Figure 5-3: 10% Coverage on 1000 items**

The advantage of employing user-based collaborative filtering with vector similarity is clear in the results depicted in the above figure. In four sessions, half the recommended items identified by the vector similarity method were preferred by the users. In this scenario, 10% of the items were preferred, which implies that 100 out of 1000 items were preferred. This ratio is considered quite high.

After 20 sessions a convergence in utility is seen between the three methods. This trend was confirmed by performing the experiment over 100 sessions. The results of this experiment are depicted in Figure 5-4.

## 5.2 Small communities

An experiment was performed where the number of items in the community was extremely small. Specifically, cases were chosen where only 100 items were extracted from the datasets. Figure 5-5 depicts the results of this experiment.

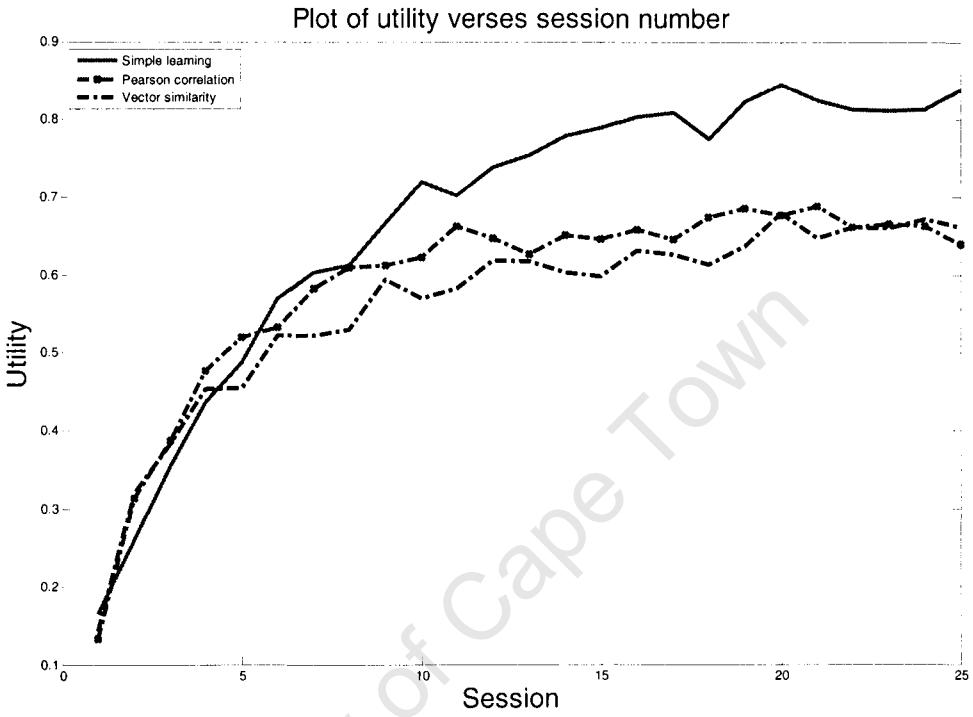



**Figure 5-5: 20% coverage of 100 items**

In this experiment, the user had a coverage of 20%, which implies that 20 items out of the 100 were preferred. The application of collaborative filtering provided a very slight advantage (in the first five sessions) over recommending items randomly. Given the reasons discussed earlier in Section 5.1.2, the simple preference learning methods begins to become more effective than the collaborative filtering algorithms after 8 sessions.

## 5.3 Comparing the effects of varying collaborative information

To observe the effect of the amount of collaborative information (ratings from other users in the community), the best performing configuration was chosen and the number of users contributing towards the collaborative information was varied. Specifically, the experiment was run on a community of 1000 items where the users had a preference towards 20% of the items and vector similarity was used to compute similarity. Random subsets of users were chosen from the original dataset. The experiment was repeated numerous times where the subsets were sized at 10%, 20%, 50% and 100% of the original user-item matrix. Figure 5-6 depicts the results of this experiment.

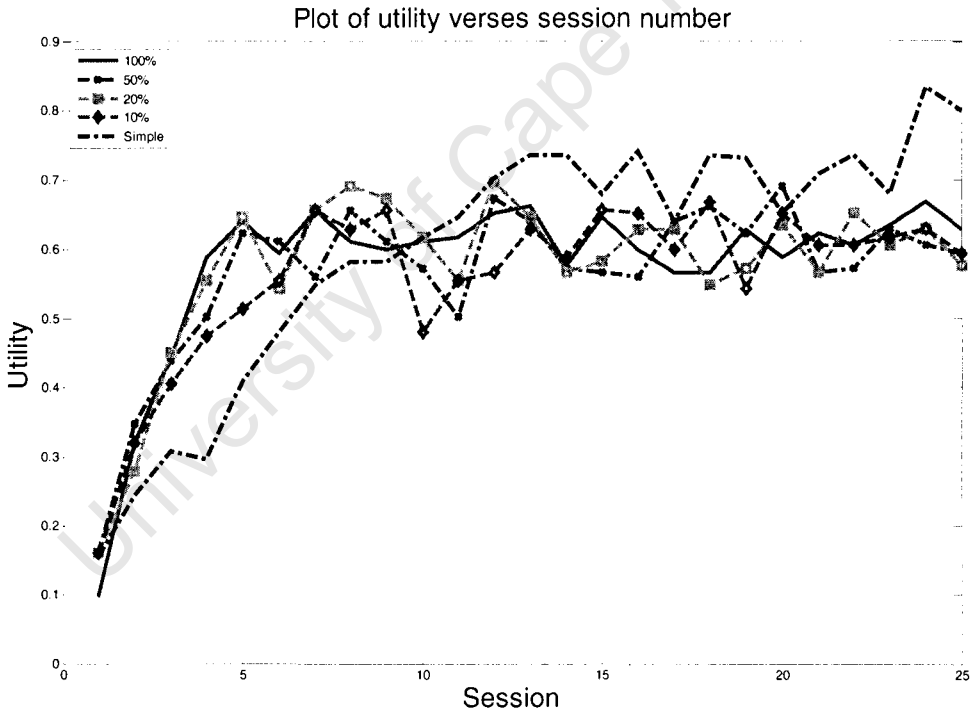



**Figure 5-6: Varying the amount of collaborative information**

The effect of all the variances of the amount of collaborative information is negligible in the first four sessions. A minor effect on the utility can be seen where

only 10% of the collaborative information was used between sessions four and six. Nevertheless, the lack of influence of removing collaborative information suggests that 90% of the data is redundant. Conversely, only a few users in the community make useful contributions to the recommendations generated for the active user.

# Chapter 6 Conclusions

From the review of the literature in the field of collaborative filtering, and the analysis of the results from the simulation conducted, the following conclusions are drawn.

## 6.1 User-based collaborative filtering is simple, yet effective

Literature reports that user-based collaborative filtering using a Pearson correlation to compute the similarities, and having the ratings transformed with case amplification, provides a reasonably effective scheme [3]. Although more accurate schemes exist, this scheme can be quickly implemented to gauge the effectiveness of collaborative filtering as a recommendation technique. For real world implementations, item-based collaborative filtering offers a similar accuracy rate, but also has the advantage of being scalable. Slope one scheme can be applied to improve the query response time with a slight compromise of accuracy.



## 6.2 Vector similarity outperforms during cold-start

Although, it has been shown that the Pearson correlation technique generally achieves a higher performance than the vector similarity method [3], from the results of the simulation it is observed that the vector similarity method of computing similarity is far more effective than the Pearson correlation method during cold-start conditions. The Pearson correlation method determines similarity by applying a linear regression between the preference vectors of the users. Linear regressions are more sensitive to noisy data. Hence, implementing user-based

collaborative filtering with vector similarity is more suitable during cold-start situations.

## 6.3 Few users contribute towards identifying preferred content

Excluding 90% of the collaborative information had a negligible effect on the accuracy of recommendations made. This observation suggests that only a few users in the community make useful contributions towards identifying preferred content for the active user.

## 6.4 Implicit observations of behaviour is insufficient to make reliable recommendations

User-based collaborative filtering with vector similarity was shown to identify half the preferred content in five sessions, which is a very reasonable learning rate for the proposed application. However, this rate was only obtained when the users had a preference towards more than 10% of the items a community of 1000. Users that preferences towards less than 10% resulted in cases where more than 25 sessions were needed to identify half the preferred content.

The sensitivity to the amount of preferred content affects the rate at which preferred content can be identified. Expecting some users to interact with the application for more than 25 sessions for relevant content to be recommended is unreasonable. Poor recommendations during cold-start conditions are to be avoided because it leaves the user with a negative stigma towards the application. Employing pure collaborative filtering on implicit observations of preferences is insufficient to make reliable recommendations.

## 6.5 Model-based schemes are advantageous if prior knowledge is available

Generating accurate recommendations during cold-start conditions requires an approach that extends beyond purely analysing behaviour. Prior preference information of the users, despite being difficult to obtain, is necessary to reliably make recommendations. Model-based schemes allow the integration of prior knowledge with observed behaviour.

Clustering methods, as an example of model-based schemes, seeks to place individuals into intuitive groups. Each of these groups are associated with a set of resources. The grouping of users and resources are based on human judgements, thus a reasonable accuracy can be ensured. In addition, these methods simultaneously deal with the problem of data sparsity, scalability and query response time. Thus a robust recommender system can be created. Implementation of such schemes, however, requires real data.

# Chapter 7 Recommendations

This thesis represents a pilot study of making recommendations onto mobile phones. Although pure collaborative filtering has shown not to be a feasible approach to making recommendations onto the mobile device, this author still feels that it is crucial to do so for certain markets of mobile phone users. The following recommendations are made for future work on this subject.

## 7.1 Evolve from current hierarchical interfaces

Content aggregators such as MTN ICE, Vodafone Live and ExactMobile provide mobile web content to South African subscribers with the user of hierarchical interfaces. Although these interfaces are not ideal, and impose an interaction barrier, they are nevertheless functional. The evaluation of collaborative filtering has been conducted on datasets in the domain of movie preferences. While this evaluation has concluded that the pure use of collaborative filtering alone is insufficient to make reliable recommendations, this conclusion is on speculative data. Further investigations should focus the evaluations on real data. Such evaluations can be conducted if usage patterns are passively collected from these aggregators for research purposes.

Furthermore, collaborative filtering can be applied to generate recommendations as an auxiliary function of these interfaces – similar to implementations on e-commerce web sites. The lack of accuracy of these algorithms will not have a crucial influence of the usability of the interfaces because their presence is passive.

## 7.2 Conduct a sociological study

If recommendations are to be an active, primary aspect of the mobile phone interface, then prior knowledge of the user is required to make accurate recommendations. Such information can be obtained by conducting a sociological study of the groups of people and what their associated interests are. The most obvious method to begin such a study is to segment the population based on location, adhering to a community-centric design. Any new user can be stereotyped into one of these groups and recommendations can initially be made based on the preferences of the other individuals in these groups. Collaborative filtering can then be applied to fine-tune the recommendations as the user engages with the recommender service.

## 7.3 Research Learning Bayesian Networks

Bayesian learning networks [39] are a deep and powerful branch of statistics that offers an iterative machine learning process. Although this branch of recommending content has not been covered by this investigation due to time limitations, literature suggests that these techniques can elegantly combine prior knowledge of a user (from a sociological investigation) and implicit behaviour information.

## 7.4 Other considerations

In the design of a fully functional recommender system, prioritisation of items is necessary. Newer items, popular items and regularly updated items certainly need to be prioritised. In addition, the time sensitivity of the popularity of items can be taken into account – certain items become more relevant during certain hours or on certain days.

Also, social networks have recently gained immense popularity. Collaborative filtering generates recommendations by analytically determining a neighbourhood of similar users. Social networks provide an alternative organisation structure that neighbourhoods of similar users can be drawn from. Research is being conducted to combine the two [40].

Given that a popular mobile recommender system is developed, the vulnerability to shilling attacks would have to be addressed. Here, malicious users are inserted into the system to push the popularity of some targeted items. Algorithms are being developed to combat such users [41, 42], and the integration of such algorithms should be researched.

Integrating all these factors may realise a robust recommender system for mobile phones. Unfortunately, all such efforts are speculative unless a real recommender system is first manifested. With a real system, the effectiveness of applying each of these schemes can be observed.

# References


[1] M. J. a. G. Marsden, *Mobile Interaction Design*: West Sussex, England: John Wiley and Sons, 2006.

[2] J. M. Raymond and R. Loriene, "Content-based book recommending using learning for text categorization," in *Proceedings of the fifth ACM conference on Digital libraries* San Antonio, Texas, United States: ACM Press, 2000.

[3] J. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *In Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 43-52.

[4] R. Paul, I. Neophytos, S. Mitesh, B. Peter, and R. John, "GroupLens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work* Chapel Hill, North Carolina, United States: ACM Press, 1994.

[5] S. Badrul, K. George, K. Joseph, and R. John, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web* Hong Kong, Hong Kong: ACM Press, 2001.

[6] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *IEEE Internet Computing,* vol. 7, pp. 76-80, 2003.

[7] D. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, "Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-based Approach," *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI 2000,* pp. 473-480, 2000.

[8] L. Ungar and D. Foster, "Clustering Methods For Collaborative Filtering," *Proceedings of the Workshop on Recommendation Systems,* 1998.

[9] I. Piyasena and H. A. Chan, "Bridging the Interaction Barrier with Mobile Phones by Recommending Content," in *Fifth IEEE Consumer Communications & Networking Conference*, Las Vegas, 2007.

[10] "Vodafone Live." vol. 2007, 2007.

[11] "MTN ICE." vol. 2007, 2007.

[12] "Exactmobile." vol. 2007, 2007.

[13] "Google Mobile," 2007.

[14] S. Kent, "The Windows 95 user interface: iterative design and problem tracking in action," in *Proceedings of the 19th international conference on*

*Software engineering* Boston, Massachusetts, United States: ACM Press, 1997.

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

[16] G. David, N. David, M. O. Brian, and T. Douglas, "Using collaborative filtering to weave an information tapestry," *Commun. ACM,* vol. 35, pp. 61-70, 1992.

[17] J. B. Schafer, K. Joseph, and R. John, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic commerce* Denver, Colorado, United States: ACM Press, 1999.

[18] U. Shardanand and P. Maes, "Social Information Filtering: Algorithms for Automating ``Word of Mouth''," *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems,* vol. 1, pp. 210-217, 1995.

[19] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval,* pp. 230-237, 1999.

[20] G. a. M. Salton, M.J, *Introduction to Modern Information Retrieval*: McGraw-Hill, 1983.

[21] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," *SDM '05: Proceedings of SIAM Data Mining,* 2005.

[22] Z. Obradovic and S. Vucetic, "A regression-based approach for scaling-up personalized recommender systems in e-commerce," 2000.

[23] L. K. Désiré Luc Massart, *The Interpretation of Analytical Chemical Data by the Use of Cluster Analysis*: Wiley, 1983.

[24] M. S. A. Roger K. Blashfield, *Cluster Analysis*: Sage Publications, 1984.

[25] J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations," in *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability.* vol. 1 Berkeley: University of California Press, 1967, pp. 281-297.

[26] H. Thomas and P. Jan, "Latent Class Models for Collaborative Filtering," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*: Morgan Kaufmann Publishers Inc., 1999.

[27] M. Benjamin and S. Z. Richard, "The multiple multiplicative factor model for collaborative filtering," in *Proceedings of the twenty-first international conference on Machine learning* Banff, Alberta, Canada: ACM Press, 2004.

[28] E. Savia, K. aki , J. Sinkkonen, and S. Kaski, "Two-way latent grouping model for user preference prediction," in *Proceedings of UAI 2005*, 2005.

[29] L. Si and R. Jin, "A Flexible Mixture Model for Collaborative Filtering," in *Proceedings of the Twentieth International Conference on Machine Learning (ICML 2003)*, 2003.

[30] R. Jin, L. Si, and C. X. Zhai, "A Study of Mixture Models for Collaborative Filtering," in *Information Retrieval Journal 2006.* vol. Issue Volume 9, Number 3, 2006, pp. 357-382.

[31] G. Ken, R. Theresa, G. Dhruv, and P. Chris, "Eigentaste: A Constant Time Collaborative Filtering Algorithm," *Inf. Retr.,* vol. 4, pp. 133-151, 2001.

[32] D. Fisher, K. Hildrum, J. Hong, M. Newman, M. Thomas, and R. Vuduc, "SWAMI: a framework for collaborative filtering algorithm developmen and evaluation," *Research and Development in Information Retrieval,* pp. 366-368, 2000.

[33] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender systems-a case study," 2000.

[34] H. Zan, C. Hsinchun, and Z. Daniel, "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering," *ACM Trans. Inf. Syst.,* vol. 22, pp. 116-142, 2004.

[35] B. Kim, Q. Li, C. Park, S. Kim, and J. Kim, "A new approach for combining content-based and collaborative filters," *Journal of Intelligent Information Systems,* vol. 27, pp. 79-91, 2006.

[36] M. Prem, J. M. Raymod, and N. Ramadass, "Content-boosted collaborative filtering for improved recommendations," in *Eighteenth national conference on Artificial intelligence* Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002.

[37] H. Zhang, "The optimality of naive Bayes," in *Proceedings of the Seventeenth Florida Artificial Intelligence Research Society Conference*: The AAAI Press, 2004, pp. 562-567.

[38] "GroupLens Research." vol. 2007: Department of Computer Science and Engineering at the University of Minnesota, 2007.

[39] D. Heckerman and D. Geiger, "Learning Bayesian Networks," Microsoft Research, Redmond, WA December 1994.

[40] K. Henry, S. Bart, and S. Mehul, "Referral Web: combining social networks and collaborative filtering," *Commun. ACM,* vol. 40, pp. 63-65, 1997.

[41] P. O. M. Michael, H. Neil, and C. M. S. Guenole, "Promoting Recommendations: An Attack on Collaborative Filtering," in *Proceedings of*

*the 13th International Conference on Database and Expert Systems Applications*: Springer-Verlag, 2002.

[42] C. Paul-Alexandru, N. Wolfgang, and Z. Cristian, "Preventing shilling attacks in online recommender systems," in *Proceedings of the 7th annual ACM international workshop on Web information and data management* Bremen, Germany: ACM Press, 2005.

[43] G. Lebanon, "C/Matlab Toolkit for Collaborative Filtering." vol. 2007, 2003.

[44] J. v. d. Geest, "Shake.m." vol. 2007, 2006.

# Appendix A: Source code

## A.1 Compiling source code

1. Unzip the source code zip file into a MATLAB\work\ubcfEval directory.
2. Start Matlab R7.0.1 or later and navigate to the above directory.
3. Some of the functions have been written in C to improve computation time. Recompile the mex files using the command: "mex -O memoryBasedModels.c". The -O flag indicates the optimization level.
4. Edit the cfevaluation.m and testusers.m file to specify parameters. These parameters are explained in Section A.2.1 and A.2.2 respectively. The initial parameters provided in the code in Appendix A.3 are suitable. Ensure that the numberOfSessions parameter in the cfevaluation.m file, and the totalSessions parameter in the testUsers.m file are the same.
5. Run the cfevaluation.m file to begin the test suite. Please be patient, each experiment can take up to a few hours to complete depending on the parameters set.



## A.2 Explanation of code

### A.2.1 Cfevaluation.m

The overall method in the collaborative filtering evaluation is **cfevaluation.m**. In this method, tests of the collaborative filtering methods and the simple preference learning scheme are triggered on the users. The results are collected over a number of sessions specified by the *sessions* parameter, learning benchmarks are determined and graphs are plotted.

The tests on users with the desired collaborative filtering methods are spawned with the **testUsers.m** function. When calling this function, a parameter is specified indicating which collaborative filtering method (or just the simple preference learning method) is employed.

- Method 0 represents the simple preference learning method where no collaborative information is used.
- Method 1 represents user-based collaborative filtering with a Pearson correlation.
- Method 2 represents user-based collaborative filtering with a vector similarity computation.

### A.2.2 testUsers.m

The **testUsers.m** function performs the collaborative filtering evaluation depending on the method specified in **cfevaluation.m.**

Firstly, loading of the user-preference data from the datasets triggered. The parameter *dataset* indicates which dataset to load. Dataset 1 refers to the MovieLens dataset and Dataset 2 refers to the EachMovie dataset. This process is handled by **loadData.m**, which subsequently calls **readLens.m** and **readEM.m** to load the MovieLens and EachMovie datasets respectively. The number of users to load, and the number of items to load, are respectively specified by variables $U$ and $L$. The user preferences are loaded into a Matlab sparse matrix $P$ (to save space, because the dataset are mostly empty) and users that have seen less than 3 movies are removed.

Once the user-preference data is loaded, set of users is demarcated in the dataset that the collaborative filtering algorithms will be evaluated on. These users have a specific engagement with the data, referred to as *coverage*, as discussed in Section

4.6.2. Users with a specific coverage are identified using the **withRatings.m** function. The parameter of this function specifies the coverage desired and the *spread* parameter indicates the number of users that are to be extracted. This method makes use of the **ratingsHist2.m** function.

This method has three main nested loops. They are as follows:

1. Current user loop
2. The Monte Carlo loop
3. Sessions loop

### A.2.2.1 Current user loop

The current user loop repeats the simulation for each user that has been chosen. These users have a specific coverage towards the items in the community, and have been chosen with the methods described in the previous paragraph. The ratings of the chosen user are removed from the user-item matrix using the **removeUser.m**. A user behaivour model *uModel* is constructed using the **extractUser.m** method and finally the current user is labelled as the active user.

### A.2.2.2 Monte Carlo loop

Once the active user has been assigned, the simulation for this user is repeated via the Monte Carlo loop. Here, all the results of the utility calculation for each session are simply averaged using an array called *monteAppreciation*. For each iteration, matrices are defined to store the items recommended $R$ and the items that have been observed $O$. In addition, a matrix called *observedBehaivour* is created to store the perceived behaviour of the user. The concept of perceived behaviour is described in detail in Section 4.4.1.

### A.2.2.3 Sessions loop

The deepest loop in the method is the sessions loop, and this process is repeated depending on the number specified in the variable *session*. The computation of recommendations are performed within this loop. A series of random number are generated under the probability distribution governed by *delta*. Before the sessions loop commences, this distribution is set to a uniform distribution. Out of the random numbers generated, the highest $n$ values are picked. These represent the items to be recommended, and the number of such recommendations is specified by the variable *numberOfRecommendation*.

Depending on the user behaviour model, *uModel*, some, none or all of the items recommended are observed. After the observations made by the user are determined, the *perceivedBehaivour* is calculated.

Collaborative filtering is then performed on the *perceivedBehaivour* and the ratings present in the *otherMat* matrix. The *perceivedBehaivour* matrix is converted into a vector train using **sparseMat2CellVec.m** that can be fed into the collaborative filtering method **memoryBasedModels.c**. This C optimised Pearson correlation and vector similarity method, and the function to convert the sparse matrix into a vector train are adopted from a collaborative filtering toolkit developed by Guy Lebanon [43]. Finally, predictions are computed using the **predictPreferenceMemBased.m** method.



## A.2.3 CfevaluationSubsets.m

The **cfevaluationSubsets.m** method is written to compare the collaborative filtering algorithms on varying amounts of user-item preferences. A method written called **ratingsSubset.m** is designed to randomly select subsets of the user-item

preference matrix. This method relies on a **shake.m** method written by Jos van der Geest [44].

# A.3 Source code

## cfevaluation.m

```
% cfevaluation.m
% this is the main method in the collaborative filtering evaluation
% this method triggers the testUsers.m methods with the desired
% collaborative filtering method, collects the results in an array, and
% plots the results
% method 0 refers to the simple preference learning scheme
% method 1 refers to the Pearson correlation method
% method 2 refers to the vector similarity method


warning off MATLAB:divideByZero

numberOfSessions = 25;

R = zeros(3,numberOfSessions);

% trigger testUser methods
R(1,:) = testUsers(0, 100);
R(2,:) = testUsers(1, 100);
R(3,:) = testUsers(2, 100);

% determine learning benchmarks
alpha = 0;
beta = 0;
for i = 1 : numberOfSessions,
   if ((R(1,i) > 0.3) & (alpha == 0))
      alpha = 1;
      disp('Random process reaches 0.3 at ')
      i
   end;
   if ((R(1,i) > 0.6) & (beta == 0))
      beta = 1;
      disp('Random process reaches 0.6 at ')
      i
   end;
end;
if (alpha == 0),   disp('Random process does not reach 0.3'); end;
if (beta == 0),    disp('Random process does not reach 0.6'); end;

alpha = 0;
beta = 0;
for i = 1 : numberOfSessions,
   if ((R(2,i) > 0.3) & (alpha == 0))
      alpha = 1;
      disp('Pearson correlation reaches 0.3 at ')
      i
   end;
   if ((R(2,i) > 0.6) & (beta == 0))
      beta = 1;
      disp('Pearson correlation reaches 0.6 at ')
      i
   end;
end;
if (alpha == 0),   disp('Pearson correlation does not reach 0.3'); end;
if (beta == 0),    disp('Pearson correlation does not reach 0.6'); end;


alpha = 0;
beta = 0;
for i = 1 : numberOfSessions,
   if ((R(3,i) > 0.3) & (alpha == 0))
      alpha = 1;
      disp('Vector similarity reaches 0.3 at ')
      i
```

```
    end;
    if ((R(3,i) > 0.6) & (beta == 0))
        beta = 1;
        disp('Vector similarity reaches 0.6 at ')
        i
    end;
end;
if (alpha == 0),    disp('Vector similarity does not reach 0.3'); end;
if (beta == 0),    disp('Vector similarity does not reach 0.6'); end;


% plot the results
x = [1:numberOfSessions];

plot(x,R(1,:), 'r','LineWidth',3)
hold on
plot(x,R(2,:), '-.xb','LineWidth',3, 'MarkerSize', 8)
hold on
plot(x,R(3,:), '-.k','LineWidth',3)
hold off
XLABEL('Session', 'fontsize', 20)
YLABEL('Utility', 'fontsize', 20)
title('Plot of utility verses session number', 'fontsize', 20)
h = legend('Simple learning','Pearson correlation','Vector
similarity',2);
set(h,'Interpreter','none')
```

## cfevaluationSubsets.m

```
% cfevaluationSubsets.m
% this method is different from cfevalution.m because subsets of the
other
% matrix are chosen

warning off MATLAB:divideByZero

numberOfSessions = 25;

R = zeros(5,numberOfSessions);

% trigger testUser methods
R(1,:) = testUsers(2, 100);
R(2,:) = testUsers(2, 50);
R(3,:) = testUsers(2, 20);
R(4,:) = testUsers(2, 10);
R(5,:) = testUsers(2, 5);


% plot the results
x = [1:numberOfSessions];

plot(x,R(1,:), 'r','LineWidth',3)
hold on
plot(x,R(2,:), '-.xb','LineWidth',3, 'MarkerSize', 8)
hold on
plot(x,R(3,:), '--gs','LineWidth',3)
hold on
plot(x,R(4,:), '--md','LineWidth',3)
hold on
plot(x,R(5,:), '-.k','LineWidth',3)
hold off

XLABEL('Session', 'fontsize', 20)
YLABEL('Utility', 'fontsize', 20)
title('Plot of utility verses session number', 'fontsize', 20)
h = legend('100%','50%','20%', '10%', '5%',2);
set(h,'Interpreter','none')
```

## extractUser.m

```
% this function extracts a user from a sparse matrix
```

```
function [L] = extractUser(userNum, P)

Pmod = full(P);
L = Pmod(userNum,:);
```

## loadData.m

```
function Pmod = loadData(U, L, dataset)
% This function loads the data from the respective datasets
% dataset 0 refers to the movielens dataset
% there are 943 users in the dataset, but something is wrong with
the
% algorithm and can only
numUsersInMovieLensDataset = 942;

% dataset 1 refers to the eachmovie dataset
numUsersInEachMovieDataset = 1000;

if (dataset == 0),
   % then load the movielens dataset
   P = readLens(numUsersInMovieLensDataset);
end;

if (dataset == 1),
   % then load the movielens dataset
   P = readEM(numUsersInEachMovieDataset);
end;

disp('The number of ratings in the original data set');
nnz(P)
T = full(P);

% now create a new matrix with only rows P and columns I
Pmod = T(U,L);

Pmod = sparse(Pmod);
disp('The number of ratings in the modified data set');
nnz(Pmod)
```

## memoryBasedModels.c

```
/*-------------------------------------------------------
  Functions for predicting user's prefernce based on
  the memeory based models defined in Breese et al.

  - Guy Lebanon, July 2003.
  -------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mex.h"

#define EPS 0.00001

/*-------------------------------------------------
  a structure that represents a (single) vote on a
  specific item by a specific user.
  -------------------------------------------------*/
typedef struct vote {
  int itemID;
  int value;
} vote;

typedef struct user {
  int numItems;
  int defaultVote;
  int numVotes;
  vote* votes;
```

```
} user;


/*-----------------------------------------------
  Computes the average vote for a specific user.
  If defaultVote>0, the unvoted items get a vote
  of defaultVote.
  -----------------------------------------------*/
double AverageVote(const user u)
{
  int i;
  double res=0;
  for (i=0;i<u.numVotes;i++)
    res += u.votes[i].value;
  if (u.defaultVote>0)
    return (res+u.defaultVote*(u.numItems-u.numVotes))
      /( (double)u.numItems);
  else return res / ( (double)u.numVotes);
}

/*-----------------------------------------------
  Computes the sum of squared votes for a specific user.
  If defaultVote>0, the unvoted items get a vote
  of defaultVote.
  -----------------------------------------------*/
double SumSquaresVote(const user u)
{
  int i;
  double res=0;
  for (i=0;i<u.numVotes;i++)
    res += pow(u.votes[i].value,2);
  if (u.defaultVote>0)
    return res+pow(u.defaultVote,2)*(u.numItems-u.numVotes);
  else return res;
}
/*-----------------------------------------------
  Computes the variance vote for a specific user.
  If defaultVote>0, the unvoted items get a vote
  if defaultVote.
  Note: This is non-normalized (not divided by the
  number of votes).
  -----------------------------------------------*/
double VarianceVote(const user u)
{
  int i;
  double avg=AverageVote(u);
  double res=0;
  for (i=0;i<u.numVotes;i++)
    res += pow(u.votes[i].value-avg,2);
  if (u.defaultVote>0)
    return res + pow(u.defaultVote-avg,2) *
      (u.numItems-u.numVotes);
  else return res;
}

/*-----------------------------------------------
  Computes the Pearson's correlation between two users
  (equation (2) at Breese et al.).
  If defaultVote>0, the unvoted items get a vote
  defaultVote.

  IMPORTANT:
  The votes of users u,v are assumed to be sorted
  according to itemID.
  -----------------------------------------------*/
double Correlation(const user u, const user v)
{
  double avg1,avg2,cov=0,var1=0,var2=0;
  int i=0,j=0,currID1,currID2,intersectionSize=0,nonUnionSize;
  avg1=AverageVote(u); avg2=AverageVote(v);
  var1=VarianceVote(u);var2=VarianceVote(v);
  while (1) {
    if (i>=u.numVotes || j>=v.numVotes) break;
    currID1=u.votes[i].itemID;
```

```
  currID2=v.votes[j].itemID;
  if (currID1<currID2) {
   if (v.defaultVote>0)
        cov += (u.votes[i].value-avg1)*(v.defaultVote-avg2);
   i++;
  } else if (currID1>currID2) {
   if (u.defaultVote>0)
        cov += (u.defaultVote-avg1)*(v.votes[j].value-avg2);
   j++;
  } else { /* The two items are equal */
   cov += (u.votes[i].value-avg1)*(v.votes[j].value-avg2);
   i++; j++; intersectionSize++;
  }
 }
 if (u.defaultVote>0) {/* add default vote to unvoted items*/
  if (i<u.numVotes)
   for (;i<u.numVotes;i++) cov += (v.defaultVote-
avg2)*(u.votes[i].value-avg1);
  if (j<v.numVotes)
   for (;j<v.numVotes;j++) cov += (u.defaultVote-
avg1)*(v.votes[j].value-avg2);
  nonUnionSize = u.numItems - u.numVotes - v.numVotes +
intersectionSize;
  cov += nonUnionSize*(u.defaultVote-avg1)*(v.defaultVote-avg2);
 }

 if (cov == 0)
  return cov;
 else if (var1==0 || var2==0)
  return cov/fabs(cov);
 else
  return cov / sqrt(var1 * var2);
}
/*-------------------------------------------------------------
 Vector similarity between two users (equation (3) in Breese
 et al.). If defaultVote>0, the unvoted items get a vote
 defaultVote.

 IMPORTANT:
 The votes in user1,user2 are assumed to be sorted
 according to itemID.
 -------------------------------------------------------------*/
double VectorSimilarity(const user u, const user v)
{
 double normU,normV,dotProd=0;
 int i=0,j=0,currID1,currID2,intersectionSize=0,nonUnionSize;

normU=sqrt(SumSquaresVote(u));normV=sqrt(SumSquaresVote(v));
 while (1) {
  if (i>=u.numVotes || j>=v.numVotes) break;
  currID1=u.votes[i].itemID;
  currID2=v.votes[j].itemID;
  if (currID1<currID2) {
   if (v.defaultVote>0)
        dotProd += u.votes[i].value * v.defaultVote;
   i++;
  } else if (currID1>currID2) {
   if (u.defaultVote>0)
        dotProd += v.votes[j].value * u.defaultVote;
   j++;
  } else { /* The two items are equal */
   dotProd += u.votes[i].value * v.votes[j].value;
   i++;j++; intersectionSize++;
  }
 }
 if (u.defaultVote>0) {/* add default vote to unvoted items*/
  if (i<u.numVotes)
   for (;i<u.numVotes;i++) dotProd += u.votes[i].value *
v.defaultVote;
  if (j<v.numVotes)
   for (;j<v.numVotes;j++) dotProd += v.votes[j].value *
u.defaultVote;
  nonUnionSize = u.numItems - u.numVotes - v.numVotes +
intersectionSize;
```

```
    dotProd += nonUnionSize * u.defaultVote * v.defaultVote;
  }
  return dotProd / (normU*normV);
}

/*-------------------------------------------------------
  Case amplification weight transform (section 2.2.3 in Breese
  et al.).
  -------------------------------------------------------*/
double CaseAmplification(const double weight, const double rho)
{
  if (weight >= 0) return pow(weight,rho);
  else return - fabs(pow(weight,rho));
}

/*-------------------------------------------------------
  Computes memory based similarity of one user (u) to several
  (vsLen) other users (stored in vs). The result will be stored
  in res, which need to be pre-allocated before function call.
  -------------------------------------------------------*/
void
computeSimilarityOne2Many(const user u, const user* vs, int vsLen,
                          int method, double* res)
{
  int i;
  for (i=0;i<vsLen;i++) {
    switch(method) {
    case 1:
      res[i]=Correlation(u,vs[i]);
      break;
    case 2:
      res[i]=VectorSimilarity(u,vs[i]);
      break;
    default:
      res[i]=-1;
      break;
    }
  }
}


/*-------------------------------------------------------
  Computes memory based similarity between users us
  to users vs. The result will be stored
  in res, which need to be pre-allocated before function call.
  -------------------------------------------------------*/
void
computeSimilarityMany2Many(const user* us, int usLen,
                           const user* vs, int vsLen,
                           int method, double* res)
{
  int i,j;
  for (i=0;i<usLen;i++) {
    for (j=0;j<vsLen;j++) {
      switch(method) {
      case 1:
           res[j*usLen+i]=Correlation(us[i],vs[j]);
           break;
      case 2:
           res[j*usLen+i]=VectorSimilarity(us[i],vs[j]);
           break;
      default:
           res[j*usLen+i]=-1;
           break;
      }
    }
  }
}

void FreeUsers(user* u, int len) {
  int i;
  for (i=0;i<len;i++) mxFree(u[i].votes);
  mxFree(u);
}
```

```
/*-------------------------------------------------------------------
  The input arguments are
  user1, user2, similarity method, defaultVote, numItems.

  Each of the first input arguments have to be a 1-Dim column cell
arrays
  that represents the votes of the train users and the test users.
  Each cell entry representes votes of a specific user in the following
  format:
  itemID vote
  itemID vote
  .
  .
  .

  The last three input arguments are scalars reprsenting the method
  of computing the similarity, the defaultVote (applies if >0) and
  the total number of distinct items.
  -------------------------------------------------------------------*/
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
  int usSz, vsSz,i,j,simMethod;
  user* us, *vs;
  double* res, *matlabVotes;

  /* Check number of input and output arguments */
  if (nrhs != 5) mexErrMsgTxt("Five input arguments required");
  if (nlhs != 1) mexErrMsgTxt("One output argument required");
  if ( ! mxIsCell(prhs[0]) || ! mxIsCell(prhs[1]))
    mexErrMsgTxt("First two input arguments have to be cell arrays");

  usSz = mxGetM(prhs[0]);
  vsSz = mxGetM(prhs[1]);
  simMethod = mxGetScalar(prhs[2]);

  /* read the two users into struct arrays */
  us = mxCalloc(usSz,sizeof(user));
  for (i=0;i<usSz;i++) {
    us[i].defaultVote = mxGetScalar(prhs[3]);
    us[i].numItems = mxGetScalar(prhs[4]);
    us[i].numVotes = mxGetM(mxGetCell(prhs[0],i));
    matlabVotes = mxGetPr(mxGetCell(prhs[0],i));
    us[i].votes = mxCalloc(us[i].numVotes,sizeof(vote));
    for (j=0;j<us[i].numVotes;j++) {
      us[i].votes[j].itemID = (int)matlabVotes[j];
      us[i].votes[j].value  = (int)matlabVotes[j+us[i].numVotes];
    }
  }
  vs = mxCalloc(vsSz,sizeof(user));
  for (i=0;i<vsSz;i++) {
    vs[i].defaultVote = mxGetScalar(prhs[3]);
    vs[i].numItems = mxGetScalar(prhs[4]);
    vs[i].numVotes = mxGetM(mxGetCell(prhs[1],i));
    matlabVotes = mxGetPr(mxGetCell(prhs[1],i));
    vs[i].votes = mxCalloc(vs[i].numVotes,sizeof(vote));
    for (j=0;j<vs[i].numVotes;j++) {
      vs[i].votes[j].itemID = (int)matlabVotes[j];
      vs[i].votes[j].value  = (int)matlabVotes[j+vs[i].numVotes];
    }
  }
  /* compute the similarities and store it in the output argument */
  plhs[0] = mxCreateDoubleMatrix(usSz,vsSz,mxREAL);
  res = mxGetPr(plhs[0]);
  computeSimilarityMany2Many(us,usSz,vs,vsSz, simMethod, res);
  FreeUsers(us, usSz);
  FreeUsers(vs, vsSz);
  return;
}
```

## predictPreferenceMemBased.m

```
function predPref=predictPreferenceMemBased(simVec, usersMat,
activeAvg)
% FUNCTION predPref=predictPreferenceMemBased(simMat,
usersMat, activeAvg)
%
% simVec        simVec(j) contains the similarity of the active user
%               and user j. The similarity may be computed using any
%               of the methods in Breese et al.
% usersMat      A sparse matrix representing the votes of the non-
active
%               users (rows) on the different items (columns).
% activeAvg     The mean vote of the active user (scalar).
%
% Guy Lebanon, August 2003.

if size(simVec,2) > 1, simVec=simVec';end

[numUsers,numItems]=size(usersMat);
sumAbs=sum(abs(simVec));
if sumAbs == 0,
   predPref = ones(1,numItems) * activeAvg;
else
   simVec = simVec / sumAbs; % make sure the similarities are
normalized
   usersAvg = full(sum(usersMat')./sum(spones(usersMat')));
   usersMat = usersMat -
spdiags(usersAvg',0,numUsers,numUsers)*spones(usersMat);
   predPref = sum(spdiags(simVec,0,numUsers,numUsers) *
usersMat) + activeAvg;
end
```

## ratingsSubset.m

```
function [Pmod]=ratingsSubset(percentage, P)
% this function extracts a subset of the user-item ratings

Pfull = full(P);
temp = size(P);
num = round(temp(1) * percentage / 100);
a = [1:temp(1)]';
as = shake(a);
am = as(1:num);
Pnew = P(am, :);
Pmod = sparse(Pnew);
```

## readLens.m

```
function userVoteMat=readLens(numUsersLimit)

% load movielens data into matlab
load u1.ch;

% select only user movie rating and create matrix P
% P stands for preferences
mat = u1(:,1:3);

% now only select numUsersLimit
lineNumber = 0;
lastUser = 1;

temp = size(mat);
flag = 0;

for i = 1 : temp(1)
   lineNumber = lineNumber + 1;
   currentUser = mat(i,1);
   if (currentUser > lastUser),
      lastUser = currentUser;
      if ((lastUser > numUsersLimit) & flag == 0),
         lastLine = lineNumber - 1;
         flag = 1;
```

```
      end;
   end;
end;

newMat = mat(1:lastLine, :);

userVoteMat=spconvert(newMat);
%userVoteMat=compressUserVoteMat(userVoteMat);

clear mat;
clear newMat;


function userVoteMat=compressUserVoteMat(userVoteMat);
% Removes extremely unseen movies (seen by < 3 users)
% and users with few votes

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary)<3);
userVoteMat(:,ind)=[];

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary,2)<4);
userVoteMat(ind,:)=[];

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary)==0);
userVoteMat(:,ind)=[];
return
```

## readEM.m

```
function userVoteMat=readEM(numUsersLimit)
% Reads the ./Votes.txt file of the eachmovie dataset into
% a sparse matrix, whose rows indicate users and columns
% indicte movies.
% written by Guy Lebanon

fid=fopen('Vote.txt');
mat=[];
currUser=0;
numUsers=0;

while 1
   tline = fgetl(fid);
   if ~ischar(tline), break, end
   numLine=sscanf(tline,'%f');
   numLine=transformVote(numLine);
   if currUser~=numLine(1),
      numUsers=numUsers+1;
      currUser=numLine(1);
   end
   if numUsers>numUsersLimit, break;end
   mat=[mat;numLine(1:3)'];
end

userVoteMat=spconvert(mat);
userVoteMat=compressUserVoteMat(userVoteMat);
fclose(fid);
return

function numLine=transformVote(numLine)
% Transforms the original eachmovie votes
% (0,0.2,0.4,0.6,0.8,1) to the scale 1-6.
switch numLine(3),
case 0
   numLine(3)=1;
case 0.2
   numLine(3)=2;
case 0.4
   numLine(3)=3;
case 0.6
   numLine(3)=4;
```

```
case 0.8
   numLine(3)=5;
case 1
   numLine(3)=6;
end
return

function userVoteMat=compressUserVoteMat(userVoteMat);
% Removes extremely unseen movies (seen by < 3 users)
% and users with few votes

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary)<3);
userVoteMat(:,ind)=[];

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary,2)<4);
userVoteMat(ind,:)=[];

userVoteBinary=spones(userVoteMat);
ind=find(sum(userVoteBinary)==0);
userVoteMat(:,ind)=[];
return
```

## ratingsHist2.m

```
% this function returns a histogram with the number of ratings per
user
% the function takes a sparse matrix of user ratings called P

function [H] = ratingsHist2(P)

fullP = full(P);
temp = size(fullP);
numUsers = temp(1);
H = zeros(1,numUsers);

for i = 1 : numUsers,
   uModel = extractUser(i, P);
   H(i) = nnz(uModel);
end;
```

## removeNaN.m

```
function [V]=removeNaN(T)
% removes all the NaN's from a column vector

temp = size(T);
V = zeros(temp(1), 1);

for i = 1 : temp(1),
   if (isnan (T(i))),
      V(i) = 0;
   else
      V(i) = T(i);
   end;
end;
```

## removeUser.m

```
function [Pmod]=removeUser(P, userNumber)
% This function removes a user from a sparse matrix

Pmod = full(P);
Pmod(userNumber,:) = 0;
Pmod = sparse(P);
```

## sparseMat2CellVec.m

```
function cellVec=sparseMat2CellVec(sparseMat)
% function cellVec=sparseMat2CellVec(sparseMat)
%
% converts a sparse matrix S to a cell vector c in the
% following manner:
%
% c{i} represents the non-zero entries in S(i,:) in an nnz(i)x2 table

for i=1:size(sparseMat,1),
   ind=find(sparseMat(i,:)>0);
   cellVec{i,1}=[ind' full(sparseMat(i,ind)')];
end
```

## testUsers.m

```
function [results]=testUsers(method, subset)
% this is the method collaborative filtering evaluation function
% in this method, the loading of user-preference data is triggered
% dataset 1 refers to the movieLens dataset
% dataset 2 refers to the eachMovie dataset
% a set of users from these datasets is chosen to be evaluated
based on
% their coverages
% two main loops are present: the first to average data from each
user, and
% the other to repeat the experiment numerous times to obtain the
true
% behaivour of the algorithms

warning off MATLAB:divideByZero
totalSessions = 25;
numberOfRecommendations = 5;
nExplore = 1;
%monteTimes = 500;
monteTimes = 25;
numItems = 1000;
spread = 7;

g = 1; % what is g?
currentSession = 1;

allUsersAppreciation = zeros(1,totalSessions);

% dataset = 1 for movieLens and dataset = 2 for eachMovie
dataset = 1;

if (dataset == 1),
   U = [1 : 942];
   L = [1 : 1000];
   Po = loadData(U, L, 0);
end;

if (dataset == 2),
   % load, the eachMovie dataset
   % how many items and users are there in this dataset?
   % NB: this eachMovie configuration does not have more than 990
items
   U = [1 : 854];
   L = [1 : 990];
   Po = loadData(U, L, 1);
end;

% here is where I specify the coverage
L = withRatings(25, Po);
```

```
% here the set of users is cycled through
for k = 1 : spread,
   % now I have to remove the active user from P
   activeUser = L(k);
   uModel = extractUser(activeUser, Po);
   uModel = uModel';

   coverage = nnz(uModel) / numItems

   if (dataset == 1),
      uModel = uModel / 5;
   end;
   if (dataset == 2),
      uModel = uModel / 6;
   end;

   P = removeUser(Po, activeUser);
   % to keep the user-item matrix so that it can be replaced when
subset
   % removal is performed
   Pstored = P;

   % BEGIN MONTE CARLO LOOP
   monteAppreciation = zeros(1,totalSessions);
   for monte = 1:monteTimes,

      % here is where I select a subset of the other matrix
      P = Pstored;
      if (subset < 100),
         P = ratingsSubset(subset, P);
         disp('Number of ratings after removal ')
         size(P)
      end;

      observedBehaivour = zeros(numItems,1);
      sumObs = zeros(numItems,1);
      sumRec = zeros(numItems,1);

      % now create a probability matrix for one of the users
      delta = zeros(numItems,1);

      % create a vector of known items
      temp = zeros(numItems,1);
      known = logical(temp);
      tryItems = 0;

      % now create a observation matrix
      O = zeros(numItems,totalSessions,'uint8');
      % and a matrix which keeps track of which items were
recommended
      R = zeros(numItems,totalSessions,'uint8');

      for currentSession = 1:totalSessions,
         % generate a random number
         Ra = (rand(numItems,1) - .5) / 5;
         %       Ra = (rand(numItems,1) - .5);
         pos = delta + Ra;

         % now decide which items to recommend
         t = pos(:,1);
         [y,i] = sort(t, 'descend');

         if (tryItems == 0),
            recommendedItems = i(1:numberOfRecommendations);
         else,
            tempA = i(1:numberOfRecommendations - nExplore);
            %          recommendedItems = [tempA, tryItems];
            recommendedItems =
zeros(numberOfRecommendations,1);
            recommendedItems(1:numberOfRecommendations -
nExplore) = tempA;

recommendedItems(numberOfRecommendations:numberOfRecom
mendations + nExplore) = tryItems;
```

```
        end;

        % now add the recommended items to R
        for i = 1:numberOfRecommendations,
           R(recommendedItems(i), currentSession) = 1;
        end;

        % now, which items were observed?
        x = rand(1);
        for i = 1:numberOfRecommendations,
           if (x <= uModel(recommendedItems(i))),
              recommendedItems(i);
              O(recommendedItems(i),currentSession) = 1;
           end;
        end;

        appreciation(currentSession) = sum(O(:,currentSession)) /
numberOfRecommendations;
        monteAppreciation(currentSession) =
monteAppreciation(currentSession) + appreciation(currentSession);

        % now, add a decay function to the items that were observed
        for i = 1:numberOfRecommendations,
           if (O(recommendedItems(i),currentSession) == 1),
              %delta(recommendedItems(i),:) = sa2();
              delta(recommendedItems(i),:) = 1;
              known(recommendedItems(i),:) = 1;
           end;
        end;

        % Perform collaborative filtering
        if (method > 0),
           % !! the following line might have to be modified for the
           % eachmovie case
           behaivour = removeNaN((sum(O.')).' ./ ((sum(R.')).')) * 5;
           activeMatTrain = sparse(behaivour');
           activeCellVecTrain = sparseMat2CellVec(activeMatTrain);
           otherMat = P;

           otherCellVec = sparseMat2CellVec(otherMat);
           [numActive,numItems] = size(activeMatTrain);

           % Obtain similarity matrices for memory base prediction
           simMethod = method;
           if (simMethod == 1),
              % then the method is a Pearson correlation method
              % there is no default voting

simMat=memoryBasedModels(activeCellVecTrain,otherCellVec,sim
Method,-1, numItems);
           end;

           if (simMethod == 2),
              % then the method is a vector similarity method
              % don't use default voting

simMat=memoryBasedModels(activeCellVecTrain,otherCellVec,sim
Method, -1, numItems);
           end;

           ind=find(activeMatTrain(1,:)>0);
           activeUserMean = full(mean(activeMatTrain(1,ind)));
           predPref=predictPreferenceMemBased(simMat(1,:),
otherMat, activeUserMean);

           %          predPref

           shift = (predPref - activeUserMean)';

           unknown = not(known);
           mostProb = shift .* unknown;
           [y,i] = sort(mostProb, 'descend');
           tryItems = i(1:nExplore);
```

```
        end;

      end;

   end;
   %END MONTE CARLO LOOP

   for i = 1:totalSessions,
      monteAppreciation(i) = monteAppreciation(i) / monteTimes;
   end;

   allUsersAppreciation = allUsersAppreciation + monteAppreciation;

end;

allUsersAppreciation = allUsersAppreciation ./ spread;

results = allUsersAppreciation;
```

## withRatings.m

```
function [L] = withRatings(n, P)
% this function returns a list of users that have made n number of
ratings

spread = 7;

H = ratingsHist2(P);
[y, userIndex] = sort(H);

diff = abs(y - n);
minValue = min (diff);
temp = size(diff);
numItems = temp(2);

posBegin = 0;
posEnd = 0;

for i = 1 : numItems,
   if ((diff(i) == minValue) & (posBegin == 0)),
      posBegin = i;
   end;
   if ((diff(i) ~= minValue) & (posBegin ~= 0) & (posEnd == 0)),
      posEnd = i - 1;
   end;
end;

if (posEnd == 0),
   posEnd = numItems;
end;

posMid = posBegin + floor((posEnd - posBegin) / 2);
userStart = posMid - ((spread - 1) / 2);
userEnd = posMid + ((spread - 1) / 2);

L = zeros(1,spread);

j = 0;
for i = userStart : userEnd,
   j = j + 1;
   L(j) = userIndex(i);
end;
```