MAUT Based Recommendation

A Project Report

submitted by

BHARATH REDDY A

in partial fulfilment of the requirements
for the award of the degrees of

MASTER OF TECHNOLOGY
&
BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

April 2014

ABBREVIATIONS

CBR	Case Based Reasoning
CDR	Compormise Driven Retrieval
CF	Collaborative Filtering
IR	Information Retrieval
LIB	Less Is Better
MIB	More Is Better
MLT	More Like This
MAUT	Multi Attribute Utility Theory
PBF	Preference Based Feedback
pMLT	Partial More Like This
wMLT	Weighted More Like This

CHAPTER 1

Introduction

"Which digital camera should I buy? Which movie should I rent? Which book should I buy for my next vacation?" These are some situations where people have to make decisions about how they are going to spend money, or in a broader level, about their future. Traditionally, people have used a variety of strategies to solve such decision making problems: conversations with friends, obtaining information from a trusted third party, hiring an expert team or simply follow the crowd. In the present age where e-commerce is flourishing, most e-commerce sites have very large number of products(often in millions) in their databases. For a user wanting to purchase a product, examining all the products (eg: mobile apps) present in the catalog one after another in the hope of a finding an product that is of interest to him is impractical. Keyword search is not going to be greatly helpful because an average user does not have a clear understanding of the product space and is often unclear with what kind of products he exactly wants.

We would like to have systems that assist the user to find products of his interest and enable him to efficiently navigate through the complex product space. *Recommender systems* are constructed for this purpose - assisting a user in his/her (online) decision-making. Recommender systems play an extremely important role in matching users to products or items that they might find interesting. They filter out huge amounts of information to give personalized suggestions that its users might be interested in. This reduces the cognitive effort on the users who are spared of the need to examine a large number of irrelevant items before reaching their desired product.

Recommender Systems are broadly classified into three categories: Collaborative, Content Based and Knowledge Based.

1.1 Collaborative Recommender Systems

The main idea in these systems is that if users share the same interests in the past - if they bought or similarly rated the same items - they will also have similar tastes in the future. This technique is also called as *Collaborative Filtering*(CF). Pure CF based approaches require only rating data and do not require the additional knowledge about underlying users/items. Hence, the algorithms are usually domain independent. Most commercial recommender sytems use collaborative filtering for recommending items. There are two main approaches to perform CF: Memory Based Approaches and Model Based Approaches

1.1.1 Memory Based Approaches

In this approach, the original rating matrix is held in memory and directly used to generate predicted ratings and recommendations. There are two popular memory based approaches:

User based Nearest Neighbor(NN) Recommendation: Given a user u, the system computes top K similar users to u according to a pre-defined similarity measure. It recommends those items to user that haven't been rated/purchased by u but liked by the top K similar users.

Item based NN Recommendation: Given a user u, the system recommends items that have received similar ratings to the ones that u had previously liked.

1.1.2 Model Based Approaches

As opposed to memory based approaches that use the ratings matrix to directly generate predictions, model based approaches learn models corresponding to each item and each user from the ratings matrix. The learned models are used to make predictions at run time. Model based approaches perform well in practice for large datasets. *Matrix factorization* is a popular model based approach. The superiority of matrix factorization techniques over traditional CF in improving prediciton accuracy was clearly seen during *'The Netflix prize'* competition(Koren *et al.* (2009)). Broadly speaking, matrix factorization methods derive

a set of latent(hidden) factors from the rating patterns and characterize each item and user as vectors of these factors. In the movie domain, such latent factors can correspond to some aspects of a movie like genre, but most of them are completely uninterpretable (Koren *et al.* (2009))

1.1.3 Limitations of CF

Cold Start Problem: To provide recommendations for a user u, pure CF techniques rely on u's ratings. This means that for a new user who has not yet rated a single item, there is no way of generating personalized recommendations (*new-user problem*). Similarly, for a new item that has been recently added to the catalog and has not been rated by a single user, there is no way for it to be recommended to a user (*new-item problem*).
Sparsity: The relavance and accuracy of CF recommender's predictions is high when the user-item ratings matrix is dense. But in real-world systems, the rating matrices are typically very sparse and thus, the quality of recommendations of pure CF approaches may not be good. As an example, consider a user u whose rating pattern is very different from most of the other users. He would find it difficult to receive useful recommendations because the number of similar users to u is very less(Balabanović and Shoham (1997)).

1.2 Content based recommender systems

Collaborative Filtering Systems do not require any knowledge about underlying users/items to make recommendations. As opposed to this, content based recommender systems rely on item descriptions and explicit/learned user profiles to recommend items. For example, if the recommender system knows that *Harry Potter* is a fantasy novel and the user Alice has always like fantasy novels, the system can recommend the new *Harry Potter* book right away. Content-based recommender systems need not rely on the existence of a large user base to generate recommendations. It overcomes the cold-start problem described in Section 1.1.3. However, item characteristics are hard to acquire normally

and hence, they have to be entered manually into the system, which can be potentially expensive for some domains (Eg: music).

Having its roots in *Information Retrieval*(IR), content based recommendation most often focuses on textual products - items which can be described in terms of textual features (Eg: documents, news articles and websites). Most news recommendation systems use content based recommendation to recommend relevant news articles to the users. A news recommendation system typically recommends news articles by comparing the main keywords of a news article with the keywords in articles that the user has rated highly in the past. There are two ways in which content based systems can create user-profiles- by explicitly asking the user to rate a set of items/topics/categories when the user is new to the system or by ”learning” the user profile automatically by examining the user's past behavior/ratings. Learning user profiles from user's past behavior can be expensive and sometimes not be accurate because of time-effects(user's interests changing over time) and sparsity of user ratings. But it has the advantage that it requires no effort from the user.

1.2.1 Limitations of Content-based Recommendation

Limited Content Analysis: Content-based recommender systems perform a shallow content analysis which might not be sufficient in many scenarios. Particularly, for recommending resources such as web pages, aspects other than the keywords like aesthetics, usability and correctness of hyperlinks play a part in establishing the quality of recommendations (Jannach *et al.* (2010)). Also, content based recommender systems using limited content analysis based on just keywords, have no way to distinguish between well written and poorly written articles, both of which use the same set of keywords. Also, feature extraction techniques for text documents is relatively mature, but the same cannot be said about many multimedia objects like images and videos. Hence, the usability of content-based recommender systems is limited in multimedia domain. (Adomavicius and Tuzhilin (2005))

Overspecialization: Another drawback of content based recommender sys-

tems is that they often tend to recommend items that the user might have already seen/rated. A general goal therefore is to increase the serendipity of the recommendation lists - that is, to include "unexpected" items in which the user might be interested in. The system described by Billsus and Pazzani (1999) therefore defines a threshold to filter out not only items that are too different from the user profile but also those that are too similar.

New user problem: The new user problem discussed in Section 1.1.3 also exists for content based recommender systems. Although content-based techniques do not require a large user community, they require at least an initial set of ratings from the user, typically a set of explicit *like* and *dislike* statements. Content-based recommenders do not have the new item problem. Given a new item, the recommender system matches the new item's features to existing user profiles and recommends it directly to relevant users.

1.3 Knowledge based recommender systems

Collaborative filtering systems, discussed in Section 1.1 suggest items to user based on user's past ratings. Content based recommender systems, discussed in Section 1.2 recommend items whose features match the user's profile. Consider some high-risk product domains like houses, cars, computers etc. Typically, users do not buy a house, a car or a computer very frequently. In such a scenario, both collaborative-filtering or content-based recommender systems may not be able to generate relavant recommendations because of the low number of available ratings. User-profiles learnt by content-based systems may not be useful for making recommendations due to a heavy influence of time effects(change in user's interests and product catalogs with time). For example, if a user has given a high rating to a 'Pentium III' computer four years ago, the recommender system cannot rely on that rating for generating relevant recommendations for the current day. In complex and high-risk product domains such as computers, customers often want to define their requirements explicitly - for example, "the maximum price of the computer should be x and the hard-disk capacity should be atleast *500GB*". Knowledge based systems are used to

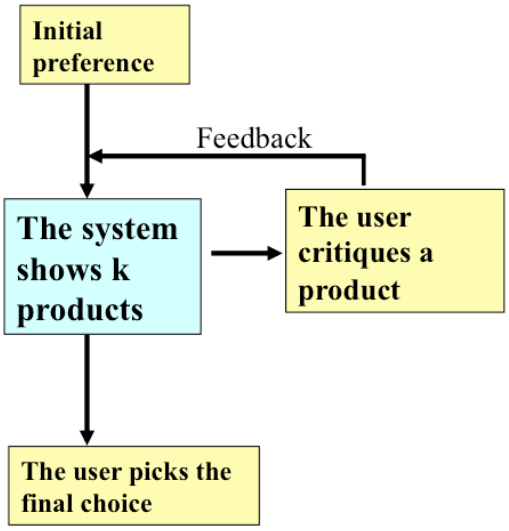

Figure 1.1: Critiquing

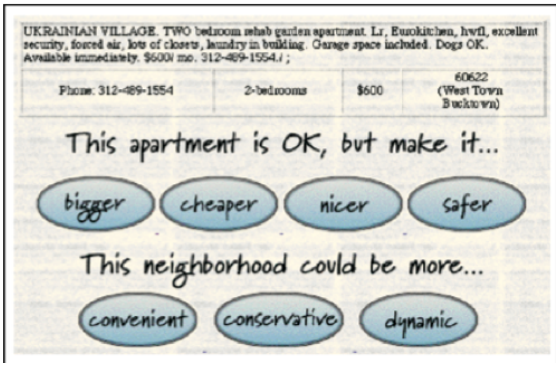

Figure 1.2: RentMe Recommender System: Burke (2000)

provide recommendations in such scenarios.

Knowledge-based recommender systems can be divided into two categories: constraint-based and case-based. Constraint-based recommender systems rely on explicit recommendation rules and case-based recommender systems use similarity/utility measures to generate recommendations. A case-based recommendation is typically generated based on similarity of the items in the casebase with the user defined query. Case-based recommendation and the algorithm for MAUT based recommendation are discussed in detail in Chapter 2.

Recommendation process of knowledge-based recommender applications is highly interactive, a foundational property that is a reason for their characterization as ***conversational systems***. The recommender system that we consider in this project is a conversational recommender system. Conversational systems assume that a user's initial query is merely a starting point for search, perhaps even an unreliable starting point. The job of a conversational system is to help the user refine his initial preference query as the interactions proceed.

1.3.1 Critiquing

Critiquing is one of the most popular forms of feedback in conversational recommender systems. In each interaction cycle, the user is presented with a list

of products. User selects a product and expresses directional preference(s) over one or more item feature values. For example, one might indicate that he is looking for a less expensive restaurant or a more formal setting(Figure 1.2). These are two individual critiques, first critique being on the *price* attribute and the second critique on the *setting* attribute. The recommender updates it's user model according to this feedback provides another set of products and proceeds to the next recommendation cycle. This continues till the user finally chooses a product. (Figure 1.1)

Unit critiques allow users to express their preference over one attribute in each interaction cycle. ***Compound critiques*** enable users to input their preferences on several attributes at a time. This can potentially shorten the number of interaction cycles in finding a target product. The early FindMe Systems Burke *et al.* (1996) had ***static critiques***. The critiques wouldn't change when users selected a particular critique. This can lead to some serious limitations. For example, the critique 'cheaper' would continue to be visible, even if there are no cheaper apartments available and when user clicks on 'cheaper', there would be no results displayed at all. Static critiques also do not represent the best set of tweaks that a user will want to make given his preference model. The notion of ***dynamic critiquing*** was first proposed by McCarthy *et al.* (2004) to overcome the limitations of static critiques. Compound critiques are generated on-the-fly for each recommendation cycle. Dynamic critiquing has been shown to improve user-experience and lower the average number of interaction cycles it takes for a customer to find his desired product.

There are two popular approaches to dynamic critiquing: Apriori algorithm based generation of compound critiques (McCarthy *et al.* (2004)) and MAUT based generation of compound critiques (Zhang and Pu (2006)). The algorithm for MAUT based recommendation is discussed in Section 2.5.

1.3.2 Limitations of Knowledge based recommendation

Knowledge based recommenders require a high level of knowledge engineering effort. One needs to define local similarity functions, utility measures, as-

sign weights to features etc.

1.4 Our contribution: Improvements to MAUT based recommendation

1.5 Organization of the Thesis

CHAPTER 2

Background & Related Work

2.1 Case-based Recommendation

Case-based recommendation traces its roots to case-based reasoning (Aamodt and Plaza (1994)). Case-based reasoning is a problem solving methodology which makes use of a case-base (database) of past problem solving experiences as its source of knowledge. A typical case in a case-base consists of a *problem specification* outlining the problem and a *solution part* which describes the solution used to solve the corresponding problem. Given a new problem at hand with a problem specification P_s, a case S is retrieved from the case-base whose problem specification is similar to P_s and then the solution of S is adapted to come up with a solution for P_s(Smyth (2007)). Case-based recommender systems are particularly suitable for generating recommendations when we are dealing with structured representations of items and there are similarity measures that can be defined across features in the particular domain. Many e-commerce websites deal with products such as cameras, computers etc. which are usually represented in terms of their features in a structured way. Once suitable similarity measures are devised, case-based recommender systems are ready to be taken to the field.

Consider a user who specificies the following query to the system: "I need a camera having a resolution of 8 Megapixels, manufactured by Canon, with a price less than $500". A case-based recommender might retrieve all cameras that have 'Canon' as their manufacturer and are similar in terms of 'Price' and 'Resolution' as mentioned in user's query and display them as recommendations. The similarity between a query q and a camera C, similarity is estimated according to weighted similarity model as:

$$Similarity(q, C) = \frac{\sum_{i=1}^{n} w_i \times sim_i(q_i, C_i)}{\sum_{i=1}^{n} w_i} \tag{2.1}$$

According to equation 2.1, the similarity between a user query q and camera C is estimated as a weighted sum of individual similarities between the corresponding features of q and C. n is the total number of features. The weights associated with each feature reflect the importance of that feature in the overall similarity calculation process. Individual feature level similarities are calculated according to the similarity function corresponding to the particular feature i which is denoted by $sim_i(q_i, C_i)$. These feature level similarities, also referred to as local similarities, are defined by domain experts at the time of system design. The similarity between two price values p_i and p_j can be defined as follows:

$$sim_{price}(p_i, p_j) = 1 - \frac{|p_i - p_j|}{max(P) - min(P)} \tag{2.2}$$

$max(P)$ and $min(P)$ refer to the maximum and minimum values of $price$ feature of all products in the case-base respectively. As we can see from Equation 2.2, greater the difference between p_i and p_j, lesser is the similarity between them. Similarity measures for other numeric attributes of a product can be defined similar to Equation 2.2. Defining similarity measures for nominal attributes is challenging. Specialized domain knowledge is required to estimate the similarities between the nominal feature (Eg:"manufacturer") values of two cameras. Case-based recommenders can be classified into two categories - 'Single-shot systems' and 'conversational systems'.

2.2 Single-shot Systems

Single shot systems are reactive systems which respond to user's query by showing him a single list of k items in a single interaction(Smyth (2007)). An example of this system is the analog device recommender described in (Wilke *et al.* (1998)) The user expresses his initial preferences to the system in terms of attribute values. The system recommends those op-amps which are most similar to his preferences.

It is often desirable to have diverse products in the recommendation list returned by the single-shot systems. Having diverse items in recommendation list helps the user to develop a better understanding of different parts of the

Algorithm 1: BoundedGreedySelection

$R \leftarrow \{\}$
$S \leftarrow$ top bk items similar to user query q
for $i \leftarrow 0$ **to** k **do**
    Sort S by $Quality(q, P, R)$ for each case P in S;
    $R \leftarrow R + First(S)$;
    $S \leftarrow S - First(S)$;
return R;

product space. This will also enable him to understand the trade-offs that exist between different product features. There have been several attempts done to achieve diversity in recommendation lists. *Bounded Greedy Selection* procedure described in Smyth and McClave (2001) has been shown to be giving the best results in many recommendation scenarios. The general algorithm for this method is illustrated in Algorithm 1. In this method, the retrieval set R is iteratively constructed till it contains k (length of recommendation list) items. The set S containing top bk items is considered in the beginning of recommendation procedure. In each iteration, quality scores of all items in set S are computed using Equation 3.1. Item that has the highest $quality$ score amongst all items in set S is added to the set R and removed from S. This step is repeated till the size of set R is equal to k. This continues till there are k items in the set R.

$$Quality(q, P, R) = similarity(q, P) * RelDiversity(p, R) \tag{2.3}$$

$$RelDiversity(p, R) = \frac{\sum_{i=1}^{|R|}(1.0 - similarity(p, r_i))}{|R|} \tag{2.4}$$

In this way, diversity in introduced into the retrieved list without significantly compromising on the similarity to the query. Another method for improving diversity in a recommendation set was proposed in Shimazu (2001), where a set of where a set of 3 recommendations $p1$, $p2$ and $p3$ is presented to the user. $p1$ is the most similar product to the user query, $p2$ is maximally dissimilar to $p1$ and $p3$ is the most dissimilar product to both $p1$ and $p2$. As we can see, the products $p2$ and $p3$ may not be very similar to the user query.

The two methods of introducing diversity into the recommendation process

discussed till now were based on the explicit characterization of diversity in being the opposite of similarity. However, there are other approaches where diversity gets introduced as a by-product. Compromise driven retrieval (CDR) (McSherry (2003)) is a method where diversity gets introduced without its explicit characterization. The central idea in CDR is to estimate the utility of a product for a user by taking into account the compromises the product might be making with respect to the user query along with the weighted similarity model. Compromises can be defined as the preferences of the user that the recommender system failed to satisfy. To identify if a particular feature has been compromised or not, dominance criteria like MIB (More is Better) and LIB (Less is Better) are used. For example, let us consider $price$ of a camera to be LIB feature and assume that a user specifies X as his preferred $price$. If a product P in the database has $price$ greater than X, then we say that P involves a compromise across the $price$ feature with respect to the query. CDR operates according to the following utility estimation principle - a given product is more acceptable than another product if it is more similar to the user query and involves a subset of the compromises that the other item involves.

2.3 Conversational Recommender Systems

Single shot retrieval works well in situations where the user is quite sure of his preferences and can frame a query appropriately. In situations when the user only has a vague idea of what he wants, he might not be able to frame his information need for the system to retrieve interesting products in a single iteration.

So in situations where user has a poor domain knowledge and is not clear about his preferences, it might be useful for the system to engage the user in an extended interaction taking his feedback in each cycle. The system can use user's feedback in each interaction cycle, revise its model of the user preferences and propose relevant products. Conversational recommender systems are classified into two categories based on the type of feedback they solicit from the user. The recommendation process of conversational systems is classified to be

Table 2.1: Navigation by Asking. Reference:Thompson *et al.* (2004)

Inquirer:	Where do you think I should eat tonight?
Advisor:	What type of food would you like?
Inquirer:	What types are there?
Advisor:	You can say things like Chinese, Indian, and Mediterranean.
Inquirer:	Oh, maybe a cheap Indian place.
Advisor:	What kind of parking would you like?
Inquirer:	I don't care, as long as it's in Palo Alto.
Advisor:	I'm sorry, I don't know of any Indian restaurants like that.
Inquirer:	Then I think I'd like Chinese instead.
Advisor:	How does Mandarin Gourmet on 420 Ramona sound?
Inquirer:	No, what else do you have?
Advisor:	How about Jing-Jing Szechwan Hunan Gourmet on 443 Emerson Street?
Inquirer:	Sure, that sounds fine.

either as *navigation by proposing* or *navigation by asking* (Simazu *et al.* (2001)).

2.3.1 Navigation by Asking

In this mode of feedback, the user is asked questions about his preference for various feature values during interaction. Thus, it proceeds in a question answer interaction where the user may question the response of the system and answer questions that are posed to him. A sample example is shown in Table 2.1 As we can see in Table 2.1, this mode of getting user feedback requires a lot of cognitive effort on the part of the user.

It is generally accepted that the users are not particularly fond of long question answer sessions. Additionally, it is possible that the user might not be able to answer the question that the system might ask him. Oftentimes they will not know the answers to questions that demand a high-level of domain knowledge or they may reject questions that ask for sensitive or personal information. Finally, there are a lot of interfacing issues in implementing text based question answer system. For example, expecting the users to answer specific questions in a text based format might not be appropriate in the context of recommenda-

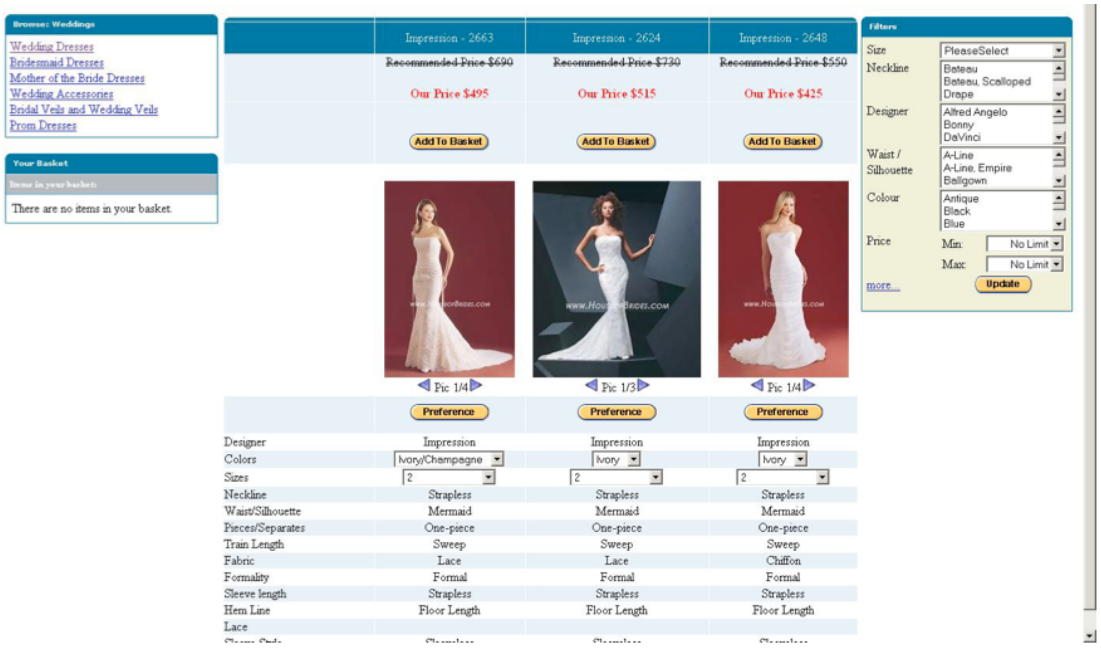

Figure 2.1: Recommender System that uses preference based feedback (Smyth (2007)

tions over devices such as mobile phones (Smyth (2007))

2.3.2 Navigation by Proposing

The key feature of navigation by proposing is that the user is presented with one of more recommendation alternatives, rather than a question, during each recommendation cycle. The user is asked to offer feedback in relation to these alternatives. There are three important kinds of feedback: Ratings-based feedback, Preference-based feedback and Critique-based feedback.

Ratings-based feedback: In this kind of feedback, the recommender system provides a list of recommendations and asks the user to provide an explicit rating for each item in the list. One of the most famous systems that used this kind of feedback was the PTV system (Smyth and Cotter (1999)). This system allowed the user the grade the T.V. programs recommended to them into two categories - positive and negative. These ratings then were made part of the user profile which was used to generate further recommendations.

Preference-based feedback: This is the simplest form of feedback which requires the user to indicate a preference for one recommendation over another. It

is also particularly well suited to domains where users have very little domain knowledge. Consider the example of the bridal dresses recommender system shown in Figure 2.1. The system shows 3 different items along with the feature values of each item in each cycle. The user can select one of the items as his preference. The system takes user preference into account and generates 3 new recommendations in the next cycle. Bridal wedding dresses is a good example of a domain where average shopper is likely to have a limited knowledge, in terms of the technical feature values of the items (Smyth (2007)) However, most users will be able to select one dress as preference over the other. Unfortunately, while this approach carries very little feedback overhead from a users perspective, it is ultimately limited in its ability to guide the recommendation process. For example, a user might have chosen a particular item because he likes certain features of the item, but we can never be sure about which features compelled the user to select that item. The system will have to then make an assumption that the user likes all of the features of the selected product and consequently, the quality of recommendations may not be very good.

Mc Ginty and Smyth (2002) propose several query revision strategies to address the above problem. The straightforward strategy(*More Like This(MLT)*) simply adopts the preferred case as the new query and proceeds to retrieve the k most similar cases to it for the next cycle. This approach does not generate very effecient recommendations because it doesn't try to infer user's true preferences. An alternative approach(*Partial More Like This(pMLT)*) transfers features from the preferred case only if these features are absent from all of the rejected cases, thus allowing the recommender to focus on those aspects of the preferred cases that are unique in the current cycle. An another strategy(*Weighted More Like This(wMLT)*) attempts to give weights to each of the features in the updated query according to how confident the recommender can be that these features are responsible for user's preference. For example, in a Personal Computer(PC) recommender system, if the preferred PC has the manufacturer "Apple" and the other $k - 1$ rejected PCs have different manufacturers, the system can give a high weight to 'manufacturer' attribute, since we can be confident that the particular user under consideration is genuinely interested in "Apple" computers. On the other hand, if the preferred PC has

Figure 2.2: Entree Restaurant Recommender System

a screen-size of 15 inches and all other $k-1$ rejected PCs also have the same screen-size, we can give a low weight to 'screen-size' attribute, since we cannot really infer whether the user is particularly interested in PCs with screen-size 15 inches.

Critique-based feedback: As already discussed in Section 1.3.1, critiquing based recommenders allow users to choose a product and provide directional preference(s) over one or more feature values of the product. In many domains, we cannot assume that users will be able to express their preferences at the beginning of interaction. Most users will not have an idea of the trade-offs/compromises that exist between different features of a product. Instead, as users become more familiar with the domain and the product options available, their preferences often change and become more rigid. Critique-based conversational recommenders offer support as users navigate the product space and help them to better understand their preference requirements. Instead of requiring users to specify their preferences from the outset, user preferences are built up over a series of *recommendation cycles*. Feature critiques typically take the form of *directional* or *replacement* critiques. Through replacement critiques, user can request for the substitution of any value (i.e., aside from critiqued value) for a non-numeric feature (e.g., different manufacturer implies [! = man-

facturer]). Through directional critiques user can express a request to increase or decrease over one or more numeric attribute values (e.g., cheaper implies [< price]).

The FindMe systems developed by Burke *et al.* (1996) were the first to employ critiquing in web-based recommenders recognising the need to focus on educating the user about the options space. The Entree recommender (Figure 2.2) suggests restaurants in Chicago and each recommendation allows the user to select from seven different critiques. When a user selects a critique such as *cheaper*, Entree eliminates cases (restaurants) that do not satisfy the critique and selects the case that is most similar to the current recommendation among the remaining cases; thus each critique acts as a filter over the cases. Originally, FindMe systems were developed as *browsing assistants* that helped users browse through large information-spaces, such as restaurants (*Entree*), automobiles (*Car Navigator*), apartments (*RentMe*), and movie rentals (*Video Navigator*) using critiquing. Other examples of early critiquing based systems include: *Apt Decision*(Shearin and Lieberman (2001)) and *Automated Travel Assistant (ATA)* (Linden *et al.* (1997)). More recent research has highlighted that there is a tension between the need for a user to explore the space of items to understand the options and desire for short recommendation dialog (McGinty and Reilly (2011)).

Critiquing provides users with a straight-forward mechanism to provide feedback, one that requires limited domain knowledge on the part of the user and it is easy to implement with even the simplest of interfaces. Over the past decade, a variety of critique-based recommendation methodologies have been proposed. Researchers have demonstrated the benefits of critiquing over other forms of feedback in conversational recommender systems. The primary reason why critiquing has become so popular is that it strikes an acceptable balance between the effort that a user must expend when providing feedback and the information value it provides(McGinty and Reilly (2011)). In comparison to the standard value elicitation approach, critiquing is a very low-cost form of feedback (in terms of user effort) that provides a relatively unambiguous indication of the user's current requirement(as compared to preference-based feedback where the feedback can be ambiguous). Critiquing is also well-suited to even

the most basic interfaces and to users with only a rudimentary understanding of certain recommendation domains.

Most early systems that implemented critiquing for recommendation, used *unit critiques*. Unit critiques operate only on a single feature in a recommendation cycle. This ultimately limits the ability of the recommender to narrow its focus which can result in unnecessarily long recommendation dialogues. Furthermore, a user may not understand the feature trade-offs that exist within a particular domain. An alternative strategy is to use *compound critiques*, which operate on multiple features in a single recommendation cycle. The application of compound critiques enables the user to take larger steps in the recommendation space and hence they can arrive very quickly towards their target product. As seen in Section 1.3.1, *static* critiques do not change as the recommendation session progresses. On the other hand, *dynamic* critiques change in each interaction cycle. In Sections 2.4 and 2.5, we discuss the two most popular approaches to generating of *dynamic, compound* critiques, namely Apriori Algorithm based and Multi-Attribute Utility Theory (MAUT) based generation of compound critiques.

2.4 Apriori Algorithm Based Generation of Dynamic Critiques

The screenshot of a camera recommender system that uses Apriori algorithm to generate compound critiques is shown in Figure 2.3. Each recommendation session will be commenced by an initial user query and this will result in the retrieval of the most similar case available for the first recommendation cycle. In each recommendation cycle, only one product is displayed to the user. The user can choose to either select this product and end the recommendation session, or critique the product. In each cycle, the system presents unit critiques for each product feature and a set of compound critiques(Figure 2.3). There are two steps involved in the generation of compound critiques: *Generating Critique Patterns* and *Mining Compound Critiques*.

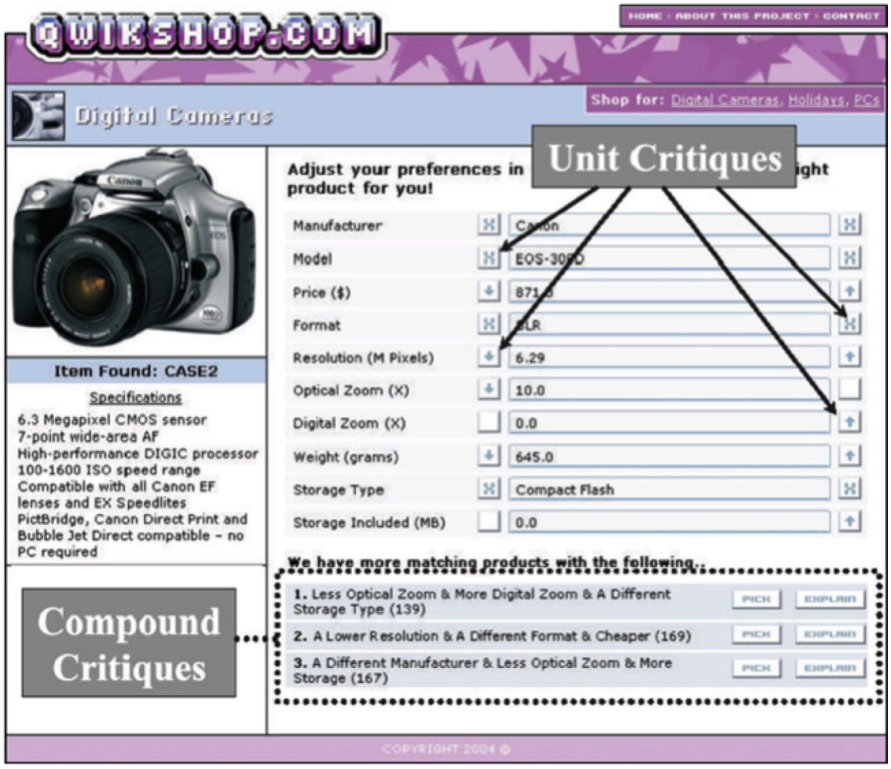

Figure 2.3: Screenshot of digital camera recommender system that uses Apriori algorithm to generate compound critiques

2.4.1 Generating Critique Patterns

In any given interaction cycle, when the user selects a critique, remaining cases in the case base are filtered using that critique and the product that is most similar to previous recommended case is shown as the next recommendation. We call this product as the *current product*. Each remaining case in the case-base is compared to the *current product* to generate a so-called *critique pattern*. Figure 2.4 illustrates how critique pattern is generated. As we can see in Figure 2.4, individual feature values of both the *current product* and product p are com-

	Current Product	Product p	Critique Pattern
Manufacturer	Apple	Sony	!=
Price (Euro)	2450	2039	<
Screen-Size (inches)	17	13.3	<
Operating System	Mac OS X	Windows XP Home	!=
RAM (MB)	2048	1024	<
HardDisk (GB)	100	120	>
Processor Type	Intel Core Duo	Intel Core Duo	=
Speed (GHz)	2.16	1.83	<
Weight (Kgs)	2.5	1.9	<
Battery-Life (Hours)	5.6	6	>

Figure 2.4: Critique Pattern for the product p(Reilly *et al.* (2004))

pared. An attribute is *numeric* if it can take numbers as its values. An attribute is *non-numeric* or *nominal* if it can take only a fixed number of possible values. '<', '>' and '=' are the possible values for numeric attributes in the critique pattern. '! =' and '=' are the possible values for non-numeric attributes (Eg: manufacturer). The critique pattern includes the critique [Price <] because the comparison product p is less expensive compared to the *current product*. These patterns serve as the source of compound critiques.

2.4.2 Mining Compound Critiques

In this step, we would like to recognize useful recurring subsets of critiques within the large collection of critique patterns. For example, if we find 40% of the remaining cases have a lower price and lower screen size; and if user is actually looking for PCs that are cheaper and have smaller screen size, the application of this critique will immediately filter out 60% of the remaining cases enabling the user to arrive at his product very quickly. Apriori Algorithm (Agrawal *et al.* (1996)) is used to generate recurring item subsets as association rules in the form of $A \rightarrow B$. Apriori measures the importance of a rule in terms of *support* and *confidence*. The support of a rule $A \rightarrow B$ is the percentage of patterns for which the rule is correct. Confidence is the ratio of the number of patterns that contain both A and B divided by the total number of patterns that contain A. For example, the rule [Monitor <] $\rightarrow$ [Price <] has support 0.6 if there are 100 critique patterns but only 60 of them contain both [Monitor <] and [Price <]. The confidence of the rule would be 0.75 if exactly 80 critique patterns contain [Monitor <]. During any given cycle, there will be a large number of association rules/ compound critiques mined by the Apriori algorithm. It is not feasible to present all of the compound critiques to the user. McCarthy *et al.* (2004) have experimentally observed that when compound critiques with low support values are presented in each cycle, the average number of interaction cycles in a recommendation session is minimum. A compound critique with a low support value means that it is present in a small proportion of critique patterns and thus it is only applicable to a few remaining cases. If applied, the critique will therefore eliminate many cases from consideration. Thus we

can intuitively understand why application of compound critiques with low support reduces the number of interaction cycles.

2.5 Multi Attribute Utility Theory (MAUT) based Generation of Dynamic Critiques

Apriori algorithm based generation of compound critiques described in Section 2.4 was one of the first proposed approaches to dynamic critiquing. It was proven to offer significant benefits in terms of reduction of number of interaction cycles and educating the user about the domain through live-user studies. However, there is a serious limitation to the above approach - it does not take user's preferences into account while generating compound critiques. There is no guarantee that the compound critiques generated by this methodology will be relevant to the user, since only product domain knowledge is utilitized in generation of compound critiques. Zhang and Pu (2006) propose an alternative methodology that relies on user preference knowledge. They use Multi Attribute Utility Theory (MAUT) for dynamically generating compound critiques.

MAUT is a well-known and powerful method in decision theory for ranking a list of multi-attribute products according to their utilities. Zhang and Pu (2006) use a very simple additive form to calculate the utility of a product $O = \langle x_1, \ldots, x_n \rangle$ as described below:

$$U(\langle x_1, \ldots, x_n \rangle) = \sum_{i=1}^{n} w_i V_i(x_i) \tag{2.5}$$

In Equation 2.5, n is the number of attributes that each of the products has, w_i is the weight/importance associated with the attribute i. V_i is the value function associated with the attribute i and it is given by experts during design time. The algorithm for MAUT based compound critiquing is illustrated in Figure 2.6. The authors use a preference model which contains the weights and preferred values of product attributes to represent user's preferences. The weights of all numeric attributes are initialized to $1/n$. At the beginning of the interac-

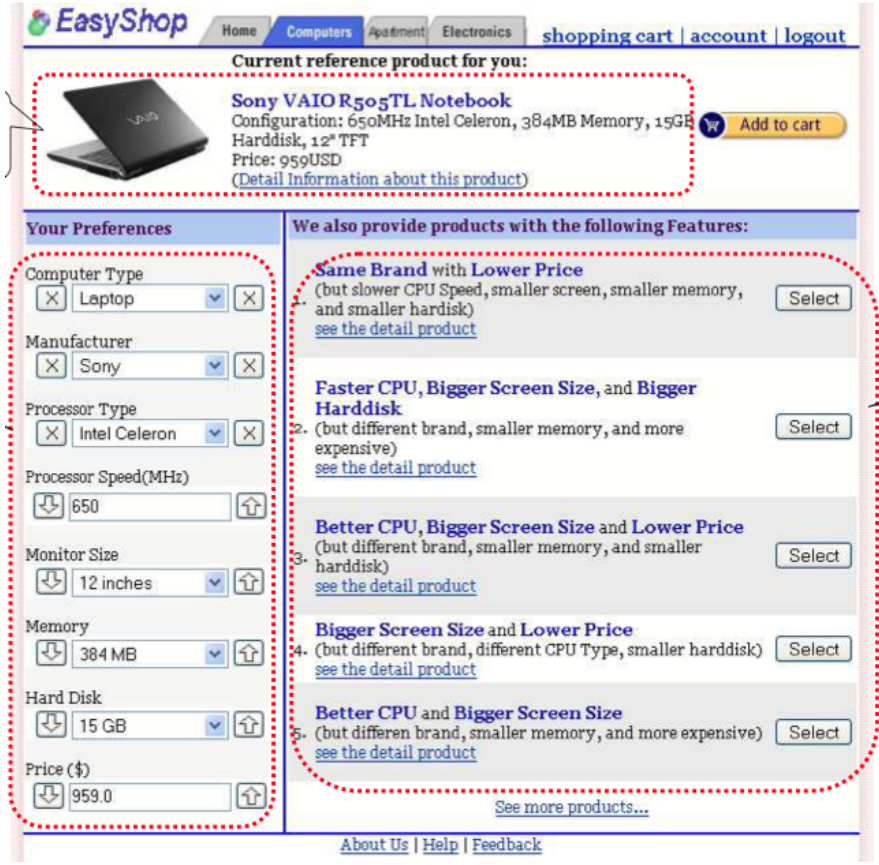

Figure 2.5: Screenshot of user interface of the recommender system that uses MAUT to generate compound critiques(Zhang and Pu (2006)

PM— user's preference model; *ref*— the current reference product;
IS— item set; *CI*— critique items; *CS*— critique strings; *U*— utility value;
β— the weight adaptive factor

```
   //The main procedure
 1. procedure Critique_MAUT ( )
 2.   PM = GetUserInitialPreferences ( )
 3.   ref = GenInitialItem (PM)
 4.   IS ⟵ all available products − ref
 5.   while not UserAccept (ref)
 6.     CI = GenCritiqueItems (pm, IS)
 7.     CS = GenCritiqueStrings (ref, CI)
 8.     ShowCritiqueInterface (CS)
 9.     id = UserSelect (CS)
10.     ref' = CI_id
11.     ref ⟵ ref'
12.     IS ⟵ IS − CI
13.     PM = UpdateModel (PM, ref)
14.   end while
15.   return

   //user select the critique string
16. function UserSelect (CS)
17.   cs = the critique string user selects
18.   id = index of cs in CS
19.   return id

   //select the critique items by utilities
20. function GenCritiqueItems (PM, IS)
21.   CI = {}
22.   for each item O_i in IS do
23.     U(O_i) = CalcUtility(PM, O_i)
24.   end for
25.   IS' = Sort_By_Utility (IS, U)
26.   CI = Top_K (IS')
27.   return CI

   //Update user's preferences model
28. function UpdateModel(PM, ref)
29.   for each attribute x_i in ref do
30.     [pv_i, pw_i] ⟵ PM on x_i
31.     if (V(x_i) ≥ pv_i)
32.       pw'_i = pw_i × β
33.     else
34.       pw'_i = pw_i / β
35.     end if
36.     PM ⟵ [V(x_i), pw'_i]
37.   end for
38.   return PM
```

Figure 2.6: Algorithm for critiquing based on MAUT (Zhang and Pu (2006)

tion process, user gives some initial preferences to the system (Eg: *HardDisk* = 500GB, *Price* $<$ \$400 etc.). In each interaction cycle, Critique_MAUT approach will determine the top K (=5) products with maximum utilities. Each of the K products is converted into a critique string by comparing it with the current reference product. Determining critique string from a product is similar to generating critique patterns described in Section 2.4.1

In a given interaction cycle, when the user selects one of the critique strings, the product corresponding to the selected string is made as the reference product in the next cycle and user's preference model is updated based on this selection. For each attribute, the attribute value of the new reference product is assigned as the preference value. For LIB attributes(Eg: Price), if the new preference value is lesser than the old preference value, we want to favor lesser priced products in the next cycle and hence the weight of attribute 'Price' is multiplied by a factor β. On the other hand, if the new preference value is greater than the old preference value, the weight of 'Price' attribute is divided by β. For MIB attributes(Eg: HardDisk capacity), if the new preference value is higher than the old preference value, weight of the attribute is multiplied by β. Else, it is divided by β. The authors set $\beta = 2.0$ in practice. All the top K products are removed from the case-base before the beginning of the next cycle. Based on the new reference product and new preference model, the system recommends another set of critiques in the next cycle. This process continues till the user find his target product and terminates the recommendation session. Figure 2.5 shows a screenshot of PC recommender system developed based on the above approach.

2.5.1 Value functions of numeric attributes

In Zhang and Pu (2006), the value functions that were used in the implementation are not mentioned. In our implementation, we first classify numeric attributes into two categoires - LIB (Less is Better) and MIB (More is Better). For the Camera dataset, the attribute *Price* is classified into LIB, since given two cameras which have all other features the same, users would prefer the camera which has lower *Price*. Similarly, the attribute *Weight* is classified into LIB. The

attributes *Optical Zoom*, *Digital Zoom*, *Storage Included* and *Resolution* are classified as MIB attributes. The value function for LIB attributes and MIB attributes is given in Equations 2.6 and 2.7 respectively. i refers to the attribute, x_i is the value of product's i^{th} attribute, max_i and min_i are the maximum and minimum values of the attribute among all the products in the casebase.

$$V_i(x_i) = \frac{max_i - x_i}{max_i - min_i} \tag{2.6}$$

$$V_i(x_i) = \frac{x_i - min_i}{max_i - min_i} \tag{2.7}$$

2.5.2 Updating value functions of nominal attributes

For nominal attributes (Eg:manufacturer), if we assume that all 'manufacturers' have equal value in the beginning of recommmendation session, updating only the weights of nominal attributes will not affect rankings because the $w_i * V_i(x_i)$ term for all cases in case-base will be the same. Therefore, we update the value functions of nominal attributes instead of their weights. The weights of all nominal attributes at the beginning of a recommendation session are initialized to 1. Let $v_1, v_2, v_3, \ldots v_k$ be k possible values that a nominal attribute N(say manufacturer) can take. Value associated with each of $v_1, v_2, v_3, \ldots, v_k$ is initialized to $1/k$ at the beginning of interaction. In the first cycle, if selected product's manufacturer is v_1, value associated with v_1 is changed to $(1+\gamma)/(k+\gamma)$; and the value associated with $v_2, v_3, \ldots, v_k$ will be $1/(1+\gamma)$ in the next cycle. For example, if there are 4 different possible manufacturers for a PC -Apple, Toshiba, Compaq and HP, the values associated with each of the manufacturers at the beginning of the recommendation session will be 1/4. Let γ = 1. If a HP manufactured PC is selected by the user in the first cycle, the value associated with HP after the first cycle will be 2/5 and the values associated with Apple, Toshiba and Compaq are 1/5. In general, in cycle t, each of $v_1, v_2, v_3, \ldots, v_k$ is multiplied by $(1 + t\gamma)$; if v_p is the value of attribute N in the selected product, then value associated with v_p is increased by γ and all the value functions associated with attribute N are then normalized (so they all sum up to 1) by dividing them all

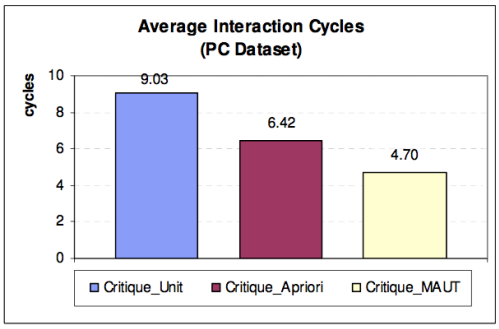

Figure 2.7: Average interaction cycles for PC Dataset(Zhang and Pu (2006))

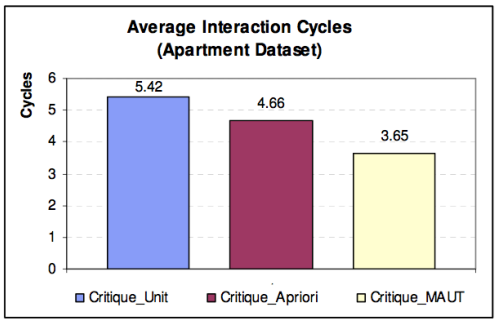

Figure 2.8: Average interaction cycles for Apartment dataset(Zhang and Pu (2006))

by $(1 + (t + 1)\gamma)$.

2.5.3 Function to calculate utility of products

In addition to the functions defined in Figure 2.6 the function $CalcUtility(PM, u)$ is defined as follows:

Algorithm 2: CalcUtility(PM, u)

```
1 retVal ← 0;
2 for each attribute x_i in u do
3     retVal += pw_i × V(x_i)
4 return retVal
```

2.6 Offline Evaluation

Zhang and Pu (2006) conducted a set of simulation experiments to compare the performance of the basic unit critiquing approach (Critique_Unit), the apriori algorithm (Critique_Apriori) and MAUT based generation of compound critiques (Critique_MAUT). The evaluation has been done based using a modified version of *leave-one-out* method, a well-known offline evaluation mechanism for knowledge based systems.

For each experiment, the number of preferences(N) in user's initial query are fixed at a constant (Eg: 1,3,5) at the beginning of evaluation. A product C is selected from the dataset and N randomly selected attributes of the product are

used to formulate user's initial query. Product C is the target product for the recommender system. In each cycle of Critique_MAUT, the simulated user is presented with five compound critiques that correspond to the five highest utility products. The simulator calculates the compatiblity of each compound critique with respect to the target product. For example, considering PC domain, if a compound critique string is "Lesser Price, Larger screen size and Lower Hard-disk capacity", and the target product has "Higher Price, Larger screen size and Higher hard-disk capacity" with respect to current reference product, the compatibility of the given compound critique with the target product is 2/3 (2 out of 3 attributes have same direction as that of the target). The simulated user selects the compound critique that is most compatible with the target product. In case of a tie, he selects the critique whose corresponding product has a higher utility. Based on the critique that the simulated user selects, the system changes the reference product and provides another set of critiques in the next cycle. This interaction continues till the target product C appears in the top-five products. Each product in the dataset is appointed as the target product 10 times and the average number of interaction cycles for all the experiments is calculated.

Two different datasets were used in the experiments: apartment data-set containing 50 apartments with 6 attributes: *type, price, size, bathroom, kitchen,* and *distance*. The PC dataset contains contains 120 PCs with 8 different attributes. This data set is available at `http://www.cs.ucd.ie/staff/lmcginty/PCdataset.zip`.

Figures 2.7 and 2.8 show the average interaction cycles for PC, apartment datasets for different approaches. Critique_MAUT approach reduces the number of interaction cycles by 20% compared to Critique_Apriori approach.

2.7 Live User Studies

Reilly *et al.* (2007) conducted a live-user study to compare the performances of Apriori-based and MAUT based approaches. For these experiments they have used two datasets- PC dataset containing 403 computers and camera dataset

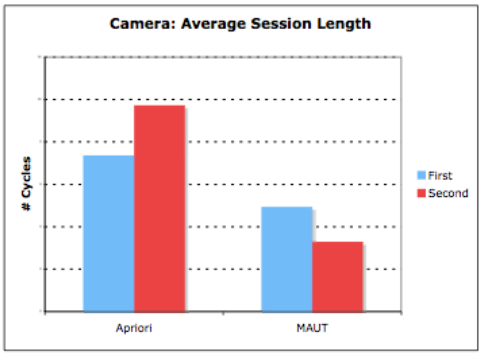

Figure 2.9: Avg session lengths on camera dataset (Reilly *et al.* (2007)

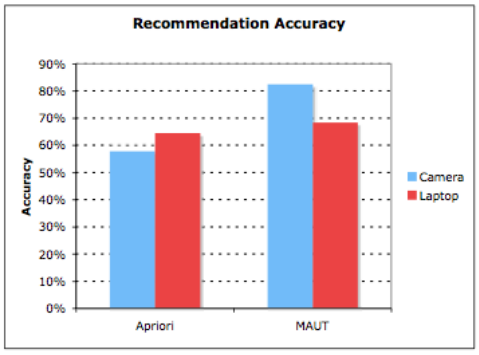

Figure 2.10: Avg recommendation accuracy (Reilly *et al.* (2007))

1: Which system did you **prefer**?

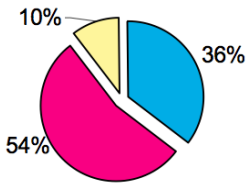

Figure 2.11: Final Questionnaire - Question 1
(Blue - Apriori, Pink - MAUT, Yellow - No difference)(Reilly *et al.* (2007)

2: Which system did you find more **informative**?

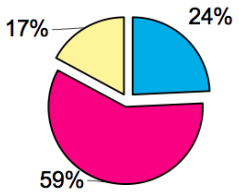

Figure 2.12: Final Questionnaire - Question 2 (Reilly *et al.* (2007)

containing 103 digital cameras. A total of 83 users separately evaluated both systems by using each system to find a PC or camera that they would be willing to purchase. The order in which both the systems were presented to the user was randomized and at the start of the trial they were explained about the use of unit and compound critiques. The performance of both the approaches was compared using metrics like Recommendation Efficiency, Recommendation Accuracy and User Experience

Recommendation Efficiency: To be successful, recommender systems must be able to quickly and efficiently guide a user through a product-space. Shorter the length of recommendation session, better is the recommendation efficiency. Sequencing bias was eliminated by randomizing the presentation order in terms of critiquing technique and dataset. For the camera dataset, MAUT performed

better (4.1 cycles) compared to Apriori (8.5 cycles). The average session length for two trials for the camera dataset is graphically shown in Figure 2.9. But for the PC dataset, Apriori performed better (7.0 cycles) compared to MAUT (10.1 cycles). A possible explanation as to why Apriori performed better in PC domain and MAUT performed better in camera domain is that Apriori is likely to perform on larger and more complex datasets. Overall, both recommenders are quite efficient.

Recommendation Accuracy: Recommenders should also be measured by the *quality* or *accuracy* of the recommendations made to users over the course of a session. After a user has interacted with the recommender sytem and chosen a product p, he is asked to view the entire catalog of products. The user is then asked whether he is willing to retain p or change it for another better product in the catalog. The accuracy of the recommender is evaluated in terms of percentage of times that the user chooses to retain his selected product(p). Higher the number of times a user retains the selected product(p), higher is the accuracy of the recommender system. Accuracy results for both approaches are shown in Figure 2.10. MAUT achieves 68.4% accuracy on PC dataset and 82.5% accuracy on camera dataset. Apriori achieves 57.9% and 64.6% accuracy on PC and camera datasets respectively. Therefore, MAUT seems to be recommending optimal products to the users.

User Experience: After users have interacted with both the recommender systems, they were asked to fill out a questionnaire in order to guage their level of satisfaction with the system. Some of the questions were: ”I found the compound critiques easy to understand”, ”The compound critiques were relevant to my preferences”, ”I would use this recommender in the future to buy other products” etc. Users were asked to give a score of -2 to +2 for each of these statements, where -2 is strongly disagree and +2 is strongly agree. Both systems received a positive feedback from the questionnaires, but most results were strongly in favor of MAUT-based approach. The final questionnaire simply asked the user to vote which system (Apriori or MAUT) performed better in terms of various criteria such as overall performance, informativeness etc.

The results for the first two questions are shown in Figures 2.11 and 2.12

The conclusion of this study is that the Apriori-based approach has an advantage when it comes to producing more efficient sessions in complex product spaces but the MAUT-based approach tends to produce higher quality recommendations.

CHAPTER 3

Improvements to MAUT Based Generation of Compound Critiques

3.1 Limitations of MAUT Based Critiquing:

We have seen in Sections 2.6 and 2.7 that MAUT based recommendation performs slightly better than Apriori algorithm in offline experiments and live user studies. The reason for the improved performance of MAUT based recommendation is that compound critiques are generated through utilities given by MAUT and these critiques tend to be more relevant to the user than the ones generated by Apriori algorithm. However, there are several limitations to MAUT based recommendation. Some of the limitations which we have addressed in this project are given below:

- Critique strings in a given iteration can be very similar to each other. This doesn't give the user too many options to choose from.
- Preference model is updated only based on the most recently selected product. History of user selected products/critique strings is not considered while updating the preference model.
- Top five products shown in the first cycle is same for all user-queries
- In a particular recommendation cycle, if all the critique strings have "Higher Price", user is forced to select a critique string with "Higher Price". We would ideally not want to update the weight of the attribute $price$, since we cannot infer whether user is willing to compromise on $price$ attribute. But MAUT based recommendation algorithm will actually decrease the weight of $price$ attribute by a factor of β.
- As an extension to the above point, Consider the case when there are 4 critique strings with "Lower Resolution" and 1 critique string with "Higher Resolution". If the user selects the critique string that has "Higher Resolution", we can update the weight of 'Resolution' attribute by a higher factor.
- The fact that a particular product has been preferred over the remaining (k-1) rejected products is not exploited.

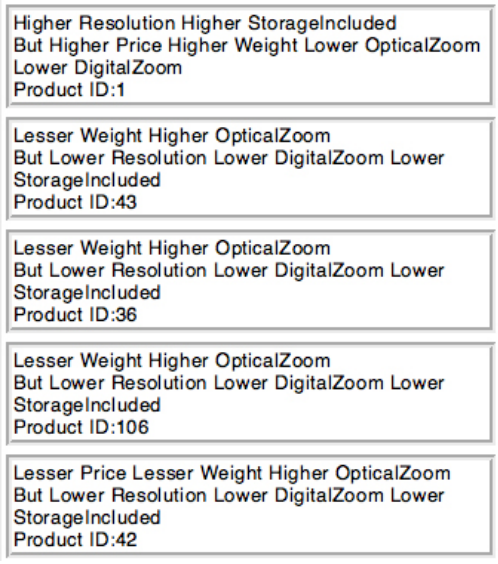

Figure 3.1: Before diversifying critique strings

Figure 3.2: After diversifying critique strings

We address each of the above limitations and also propose some additional improvements to MAUT based recommendation in Sections 3.2 to 3.10.

3.2 Diversity in Critiques

In a live user-study conducted by McCarthy *et al.* (2005) to compare Apriori based critiquing with unit critiques, a number of trialists complained that their the compound critiques were often too similar to each other and hence they didn't have many options in each cycle. As shown in Figure 3.1 the same problem exists in MAUT based critiquing also. In the first few cycles, an average user does not have a good understanding of the product space and also different trade-offs that exist between the product features. Diverse critiques help educate a user about the product space relatively quickly and hence can lead to efficient recommendation sessions. But in an attempt to make the critiques diverse, the recommender will have to choose a few sub-optimal products as the top K products in every cycle. So, there is a risk of having prolonged sessions because of this.

The default strategy for selecting the short-list of K products is to select K products that have highest utility. Instead, we use a variant of ***Bounded Greedy Selection*** algorithm described in Smyth and McClave (2001) to select

the top K products. The main idea of this approach is that we select products with a high utility while minimizing the product's average similarity to the cases/compound critiques selected so far. A metric that is similar to Equation 3.1 is used to compute $Quality$ scores of products in each cycle.

$$Quality(c, P) = \alpha * utility(c) + (1 - \alpha) * critiqueSim(c, P) \tag{3.1}$$

The modified *GenCritiqueItems(PM, IS)* function is given in Algorithm 3.

Algorithm 3: GenCritiqueItems(PM, IS)

```
R ← {};
CB' ← IS;
for i ← 0 to k do
    Sort CB' by Quality(p, R, PM) for each case p in IS;
    R ← R + First(CB');
    CB' ← CB' − First(CB');
return R;
```

Algorithm 4: Quality(p, R, PM)

```
ret ← α × utility(p, PM);
if R == {} then
    return ret;
else
    disSim ← (Σ_{r_j ∈ R} (1 − critiqueSim(p, r_j))) / |R|;
    ret+ = (1 − α) × disSim;
return ret;
```

critiqueSim(a, b) returns the extent of overlap between the individual attribute directions of products a and b. We get the best results when $\alpha = 0.5$. Introducing diverse critiques in every cycle results in significant improvement in the number of interaction cycles and it also improves user experience.

3.3 Selectively updating value functions of nominal attributes

As seen in Section 2.5.2, the value of γ used to update the value functions of nominal attributes is constant in each cycle. Instead, we propose an alternative

approach in which the value of γ varies according to the number of alternatives the user rejected while choosing a particular attribute value. Higher the number of rejected alternatives, higher is the value of γ. The intuition for varying the value of γ is as follows: Consider the case when there are five PCs displayed to the user and the manufacturers of the five PCs are "Compaq", "HP", "Apple", "Dell" and "Toshiba". If the user selects the PC with "Compaq" as the manufacturer, we can see that he has accepted "Compaq" and rejected four other manufacturers. Thus, we infer that the user has a strong preference for "Compaq" PCs. On the other hand, if all the five PCs have a screen size of 15 inches and the user selects the PC with screen-size of 15 inches, we cannot really infer whether the user has a strong preference for 15 inch PCs. Generalizing the examples above, if k is the value of the attribute N selected in a cycle and R is the list of values of attribute N of the top-K products other than the selected product, we define

$$\gamma = \frac{\#\ of\ alternatives\ to\ k\ in\ R}{|R|} \tag{3.2}$$

If attribute N = "Manufacturer"; γ = 1 when manufacturer of all the remaining products is different from the selected product's manufacturer(M) and also different from each other; meaning that the user has a strong preference for M. γ = 0 when manufacturer of all remaining products is same as the selected product's manufacturer(M). This is the case when we cannot infer whether the user has a strong preference for M. This is similar to weighted MLT described in Mc Ginty and Smyth (2002). The value of γ for each attribute when the product *P3* is selected, is illustrated in Table 3.1. Varying the value of γ according to the other products' attribute values results in a significant improvement in the number of interaction cycles.

Feature	P1	P2	**P3**	P4	P5	γ
Manufacturer	Dell	Apple	**Compaq**	HP	Toshiba	1.0
Type	Laptop	Laptop	**Laptop**	Laptop	Laptop	0
Processor	Core-i3	AMD-E1	**Core-i7**	AMD-E1	Core-i3	0.5
Screen-size	15	13.3	**15**	15	15	0.25

Table 3.1: Values of γ when P3 is selected by the user

3.4 Additional preference to products that are similar to the most recent user selected product

As seen in Section 2.5, the weights of numeric attributes and the value functions of nominal attributes are updated in each cycle based on the critique string selected by the user. Utility of the remaining products in the case base are calculated as a weighted sum of value functions of all attributes according to Equation 2.5. Along with the weighted additive formula in Equation 2.5 , we introduce an additional term in the utility function that favors the products having a similar critique pattern as that of the most recently selected product. The method of computing critique patterns for products has been discussed in Section 2.4.1. The intuition behind this modification is that, if the user selects the critique string "Lower Price, Higher Resolution but Higher Weight", he is interested in those products where this trade-off is obeyed. Considering only three attributes, Price, Resolution and Weight, if the critique pattern of the selected product is {Price:<, Resolution:>, Weight:>} and the critique pattern of a product p in the case base is {Price:<, Resolution:>, Weight:<}, the overlap between the two critique patterns is 2/3. The function $CalcUtility(PM, u)$ (Algorithm 2), which is used to calculate utility of products at the beginning of each cycle is modified as follows:

So far, we have considered the critique overlap between a product in the case-base and the product that is most recently selected by the user. We now extend this algorithm by considering the critique overlap of a product with all the products that have been selected so far. The additional term added to the

Algorithm 5: CalcUtility(PM, u)

retVal $\leftarrow 0$;
for *each attribute* x_i *in* u **do**
    retVal += $pw_i \times V(x_i)$
$p1 \leftarrow$CritiquePattern(u);
$p2 \leftarrow$CritiquePattern(previousSelectedProduct);
retVal += $overlap(p1, p2)$;
return retVal

utility of a product, *overlapTerm* is now calculated as follows:

$$addTerm(p) = \frac{\sum_{i=1}^{n} w_i \times overlap(p, sel_i)}{\sum_{i=1}^{n} w_i} \tag{3.3}$$

In Equation 3.3, sel_i refers to the product selected by the user in i^{th} cycle. w_i is the weight/importance associated with the overlap between the product p and product selected in i^{th} cycle. In our implementation, we set $w_i = \frac{1}{2^{n-i}}$. The modified *CalcUtility(PM, u)* function is as follows:

Algorithm 6: CalcUtility(PM, u)

retVal $\leftarrow 0$;
for *each attribute* x_i *in* u **do**
    retVal += $pw_i \times V(x_i)$
for *each product* t_i *in previouslySelectedProducts* **do**
    $p1 \leftarrow$CritiquePattern(u);
    $p2 \leftarrow$CritiquePattern(t_i);
    retVal += $w_i \times overlap(p1, p2)$;
return retVal

3.5 Varying the level of diversity

In Section 3.2, we have discussed a method of introducing diversity in every cycle. We know that, at the beginning of a recommendation session, an average user does not have a good understanding of the product space and trade-offs that exist between different product attributes. But after interacting with the recommender system for a few cycles, he develops a good understanding of the product space and his preferences become more stable. Therefore, it is a good idea to introduce diversity in the beginning of a recommendation session

so that user will develop a better understanding of the product space and then show targeted recommendations after a few cycles when his preferences have stabilized.

We propose a new approach where diversity is varied adaptively according to whether user's preferences have stabilized. Consider an attribute 'Price'. If the maximum difference between prices of previous K(=3) products selected by the user is less than a pre-defined threshold, we assume that the user is satisfied with this particular product price and we promote cases that have a price closer to the average price of the K products in the next cycle. If the maximum difference between the weights of the previous K(=3) products is greater than the pre-defined threshold, we assume that user's preference for the 'weight' attribute has not stabilized yet. Hence, we try to display products that are diverse to each other in terms of 'weight' attribute. The numeric attributes for which we assume that user's preference has stabilized are classified into the set $simA$. The other numeric attributes are classified into the set $divA$. In the implementation, the function *GenCritiqueItems(PM, IS)* is the same as described in Algorithm 3. The function *Quality(p, R, PM)* is modified as follows:

Algorithm 7: Quality(p, R, PM)

$ret \leftarrow \alpha \times utility(i, PM)$;
$simA, divA$ = classifyAttributes();
avg = previousKProductAttributeAverages();
$tmp \leftarrow 0$;
for *each attribute* x_i *in* $simA$ **do**
    tmp += (sim($avg(x_i)$, $p(x_i)$));
for *each attribute* x_i *in* $divA$ **do**
    $tmp+ = \frac{\sum_{r_j \in R}(1-sim(p(x_i),r_j(x_i)))}{|R|}$;
$tmp = (1-\alpha) \times \frac{tmp}{|simA|+|divA|}$
return $ret + tmp$;

3.6 Initializing the value functions of nominal attributes with unequal values

The default strategy for initializing value functions of nominal attributes as described in Section 2.5 is to initialize them with equal values. For example, in the PC dataset, if there are 8 different manufacturers (Eg: 'Apple', 'HP', 'Compaq' etc.), the value associated with each of the manufacturers at the beginning of the recommendation session is initialized to $1/8$. In this section, we will look at a different way to initialize the value functions that can lead to better performance.

A product p is called as a *dominator* of product q, if all the attribute values of p are 'better than' (or dominate) that of q. We can also say that the product q is *dominated* by the product p. For LIB ('Less Is Better') attributes (Eg: Price), lower price is 'better than' higher price. For MIB ('Less Is Better') attributes (Eg: Disk Storage), higher storage is 'better than' lower storage. For nominal attributes (Eg: Manufacturer), it is challenging to define an ordering among the attribute values. Consider two PCs a and b with 4 attributes. Product a: {Manufacturer: Apple, Price: $1400, Weight: 2.1kg, Storage: 320GB} Product b: {Manufacturer: Acer, Price: $1000, Weight: 2.0kg, Storage: 500GB} We can say that Product b dominates Product a with respect to all numeric attributes. Intuitively, we can see that any given product p should not have a dominator q with respect to all attributes. If there exists a product q which is better than product p in all attributes, there would be no demand for product p in the market because people would just purchase product q instead of p.

Using the intuition above, we can say that a product which has many dominators with respect to all the numeric attributes should have nominal attributes of high value. For example, in the PC dataset containing 120 PCs, an "Apple" computer has 21.6 dominators on an average (w.r.t. numeric attributes). So we can infer that the manufacturer "Apple" should have a higher value than other manufacturers. If the manufacturer Apple did not have a higher value compared to other manufacturers, then Apple computers would cease to exist in the market.

Manufacturer	Average number of dominators w.r.t. numeric attributes	initialization value
Fujitsu	34.4	0.38
Apple	21.6	0.25
HP	9.6	0.11
Compaq	7.6	0.08
Gateway	7.0	0.07
Dell	6	0.06
Toshiba	3.14	0.03
Sony	1.5	0.02

Table 3.2: Values with which different manufacturers are initialized

We first compute the average number of dominators for products of each 'manufacturer' and initialize their values in the ratio of the number of dominators. The number of dominators for products of each manufacturer and the values with which each of them is initialized is given in Table 3.2. In the experiments, in both PC and Camera datasets, we have initialized only the "Manufacturer" attribute with unequal values. Other nominal attributes were initialized to equal values. This led to a decent performance improvement. We have consider initializing other nominal attributes with unequal values, but that performed worser than the actual MAUT algorithm.

3.7 Selectively updating the weights of numeric attributes

As seen in Section 2.5, the weight of a numeric attribute is either multiplied or divided by a constant factor $\beta(= 2.0)$ depending on whether the new preference value is better than the old preference value or not at the beginning of each cycle. But this approach has some limitations as discussed in Section 3.1. In a particular recommendation cycle, if all the critique strings have "Higher Price" as their sub-critique, the user is forced to select a critique string with "Higher Price". The original MAUT based recommendation algorithm will divide the weight of $price$ attribute by a factor of β, promoting higher priced products in

the next cycle. To avoid this, we modify the implementation of MAUT based recommendation algorithm such that the weight of the $price$ attribute in such cases is not changed. As an extension to the above limitation, we consider the case when there are four critique strings with "Lower Disk Storage" and one critique string with "Higher Disk Storage". If the user selects critique string that has "Higher Disk Storage", we can infer that the user has strong preference for PCs with higher disk storage and hence multiply the weight of the attribute 'disk storage' by a factor bigger than β.

3.8 Considering history of user selected products

As seen in Section 2.5, after the user selects a critique string, preference model is updated only based on the product corresponding to this critiquing string. Instead, we now maintain a list of products corresponding to the critique strings selected by the user in previous interaction cycles and update the model according to the weighted average of these products' attributes. For example, if the prices of the products selected by the user in the first four cycles are \$250, \$200, \$200 and \$400. If he chooses a product with a price \$800 in the fifth cycle, the standard MAUT algorithm will make \$800 as the preferred value of 'price' attribute in it's user model and divide by weight of the price attribute by β, because the price of the reference product is \$350. However, the user might have most likely selected the product with price \$800 because he likes other features of the product. When this product becomes the reference product in the next cycle, the user will most likely choose "Lower Price" critique because all his previous preferrred products have a price around \$400. Instead of making \$800 as the preferred price, we compute the weighted average (wa) of all previous product prices and make that as the preferred price and update the weight of 'price' attribute based on the value of wa. In our implementation, we considered the weight associated with i^{th} product is $\frac{1}{n}$. Hence the preferred price for the given example will be:

$$wa = \frac{250 + 200 + 200 + 400 + 800}{5} = 370 \tag{3.4}$$

The value of new preferred price($370) is less than the old preferred price ($400). Hence, the weight of the price attribute is multiplied by a factor of β and lower priced products are promoted in the next cycle instead of higher priced ones. Note that this strategy is applicable only to numeric attributes. The value functions of nominal attributes are updated in the standard way as described in Section 2.5.2. The function UpdateModel(PM, ref) is updated as follows:

Algorithm 8: UpdateModel(PM, refL)

```
1  R ← {}
2  CB' ← IS
3  for each attribute x_i do
4      ref[x_i] = 0;
5      ws = 0;
6      for each product p ∈ refL do
7          ref[x_i] += hw_i × p[x_i];
8          ws += hw_i;
9      ref[x_i] = ref[x_i]/ws;
10 for each attribute x_i in ref do
11     [pv_i, pw_i] ← PM on x_i;
12     if V(x_i) ≥ pv_i then
13         pw'_i = pw_i × β;
14     else
15         pw'_i = pw_i/β;
16     PM' ← [V(x_i), pw'_i];
17 return PM;
```

3.9 Similar Products in the First Cycle

As seen in Section 2.5, weights of numeric attributes and the value functions of nominal attributes are initialized to default values at the beginning of every recommendation session. Therefore, the top K utility products presented to the user in the first cycle is the same irrespective of initial preferences given by the user. Instead of presenting the top K utility products in the first cycle, we present products that are most similar to the user's query. When the user selects one of those similar products, the preference model is updated in the proper direction such that the target product shows up quickly in top K products.

3.10 Additive model for updating weights of attributes

As discussed in Section 2.5, the weight of a numeric attribute is either multiplied or divided by a constant factor $\beta(= 2.0)$ depending on whether the new preference value is better than the old preference value or not. Instead of the standard multiplicative model, we propose a new additive model for updating the weights of attributes. In a given interaction cycle, if the new preference value for a product is better than the old preference value, the weight of the attribute is increased by a factor β. Else, the weight of the attribute is not changed. The weights of numeric attributes are normalized after they are updated so that they sum up to one. In our implementation, we have set β = 0.5 and it gave the best results.

CHAPTER 4

Experimental Results

4.1 Experimental Methodology

We use two standard datasets in our experiments - **Camera** and **PC**. Both the datasets are available for download at `http://www.cs.ucd.ie/staff/lmcginty/PCdataset.zip`. Cases with missing values have been removed from the dataset. The Camera dataset contains 173 cameras with 10 attributes and the PC dataset contains 120 PCs with 8 attributes. A typical PC in the PC dataset is shown in Table 4.1 and a typical camera is shown in Table 4.2.

To evaluate the algorithms in offline setting, we simulate an artificial user who interacts with the recommender system. We perform the evaluation of our algorithms in two scenarios described in Section **??** and **??**, where the artifical user interacts with the system in two different ways. The complete implementation for all the improvements described in Sections 3.2 to 3.10 is available at `https://github.com/abharath27/MAUTNew.git`.

Table 4.1: A typical PC in the PC dataset

Manufacturer	Apple
Processor Type	PowerPC G3
Processor Speed(MHz)	600
Monitor (Inches)	15
Type	Laptop
RAM (MB)	512
Drive Capacity(GB)	40
Price ($)	986

Table 4.2: A typical camera in the Camera dataset

Manufacturer	Sony
Model	DSC-T11
Price($)	383
Format	Ultra-compact
Resolution (MP)	5
Optical Zoom (X)	3
Digital Zoom (X)	4
Weight (grams)	230
Storage Type	Memory Stick
Storage Included (MB)	32

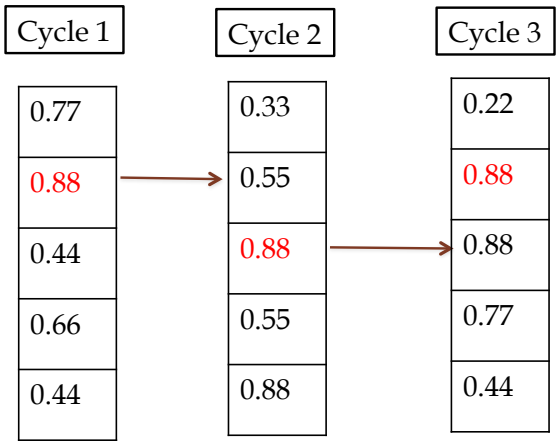

Figure 4.1: Numbers in the boxes represent the compatibilities of each of the five critique strings with the target product. Simulated user selects the most compatible critique string (red) in each cycle.

4.2 Highly Focused Recommendation Framework

Evaluation in Highly Focused Recommendation Framework is the same way of evaluation described in Section 2.6. In this scenario we assume that the user is relatively sure of his preferences and hence chooses the critique string that is maximally compatible with the target product in each cycle. The notion of selecting the most compatible critique string is shown in Figure 4.1

4.3 Noisy Framework

In this scenario, the simulated user does not select the most optimal critique string during each cycle. This kind of evaluation is similar to the evaluation

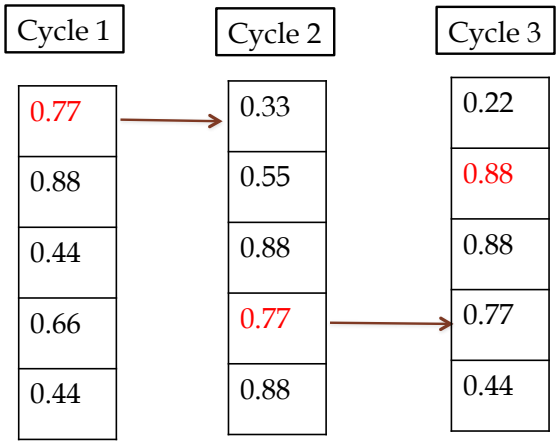

Figure 4.2: Simulated user selecting sub-optimal critique strings due to the introduction of noise

procedure described in Smyth and McGinty (2003). Noise is introduced into the process by varying the compatibility scores of the critique strings within some threshold. In our experiments, we have used a noise level of 10%, i.e, the compatiblity scores can be changed by upto +/-10% of their actual values. Due to the introduction of noise, the uesr makes sub-optimal choices in each cycle as seen in Figure **??**.

REFERENCES

1. **Aamodt, A.** and **E. Plaza** (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, **7**(1), 39–59.

2. **Adomavicius, G.** and **A. Tuzhilin** (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, **17**(6), 734–749.

3. **Agrawal, R.**, **H. Mannila**, **R. Srikant**, **H. Toivonen**, **A. I. Verkamo**, *et al.* (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining*, **12**(1), 307–328.

4. **Balabanović, M.** and **Y. Shoham** (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, **40**(3), 66–72.

5. **Billsus, D.** and **M. J. Pazzani**, A personal news agent that talks, learns and explains. *In Proceedings of the third annual conference on Autonomous Agents*. ACM, 1999.

6. **Bridge, D.** and **A. Ferguson** (2002). An expressive query language for product recommender systems. *Artificial Intelligence Review*, **18**(3-4), 269–307.

7. **Burke, R.** (2000). Knowledge-based recommender systems. *Encyclopedia of library and information systems*, **69**(Supplement 32), 175–186.

8. **Burke, R. D.**, **K. J. Hammond**, and **B. C. Young**, Knowledge-based navigation of complex information spaces. *In Proceedings of the national conference on artificial intelligence*, volume 462. 1996.

9. **Jannach, D.**, **M. Zanker**, **A. Felfernig**, and **G. Friedrich**, *Recommender systems: an introduction*. Cambridge University Press, 2010.

10. **Koren, Y.**, **R. Bell**, and **C. Volinsky** (2009). Matrix factorization techniques for recommender systems. *Computer*, **42**(8), 30–37.

11. **Linden, G.**, **S. Hanks**, **N. Lesh**, *et al.* (1997). Interactive assessment of user preference models: The automated travel assistant. *Courses and lectures-International centre for mechanical sciences*, 67–78.

12. **Mc Ginty, L.** and **B. Smyth**, Comparison-based recommendation. *In Advances in Case-Based Reasoning*. Springer, 2002, 575–589.

13. **McCarthy, K.**, **J. Reilly**, **L. McGinty**, and **B. Smyth**, On the dynamic generation of compound critiques in conversational recommender systems. *In Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2004.

14. **McCarthy, K.**, **J. Reilly**, **L. McGinty**, and **B. Smyth**, Experiments in dynamic critiquing. *In Proceedings of the 10th international conference on Intelligent user interfaces*. ACM, 2005.

15. **McGinty, L.** and **J. Reilly**, On the evolution of critiquing recommenders. *In Recommender Systems Handbook*. Springer, 2011, 419–453.

16. **McSherry, D.**, Diversity-conscious retrieval. *In Advances in Case-Based Reasoning*. Springer, 2002, 219–233.

17. **McSherry, D.**, Similarity and compromise. *In Case-Based Reasoning Research and Development*. Springer, 2003, 291–305.

18. **Reilly, J.**, **K. McCarthy**, **L. McGinty**, and **B. Smyth**, Dynamic critiquing. *In Advances in Case-Based Reasoning*. Springer, 2004, 763–777.

19. **Reilly, J.**, **K. McCarthy**, **L. McGinty**, and **B. Smyth** (2005). Incremental critiquing. *Knowledge-Based Systems*, **18**(4), 143–151.

20. **Reilly, J.**, **J. Zhang**, **L. McGinty**, **P. Pu**, and **B. Smyth**, Evaluating compound critiquing recommenders: a real-user study. *In Proceedings of the 8th ACM conference on Electronic commerce*. ACM, 2007.

21. **Shearin, S.** and **H. Lieberman**, Intelligent profiling by example. *In Proceedings of the 6th international conference on Intelligent user interfaces*. ACM, 2001.

22. **Shimazu, H.**, Expertclerk: navigating shoppers' buying process with the combination of asking and proposing. *In Proceedings of the 17th international joint*

conference on Artificial intelligence-Volume 2. Morgan Kaufmann Publishers Inc., 2001.

23. **Simazu, H.**, **A. Shibata**, and **K. Nihei** (2001). Expertguide: A conversational case-based reasoning tool for developing mentors in knowledge spaces. *Applied Intelligence*, **14**(1), 33–48.

24. **Smyth, B.**, Case-based recommendation. *In The adaptive web*. Springer, 2007, 342–376.

25. **Smyth, B.** and **P. Cotter**, Surfing the digital wave. *In Case-Based Reasoning Research and Development*. Springer, 1999, 561–571.

26. **Smyth, B.** and **P. McClave**, Similarity vs. diversity. *In Case-Based Reasoning Research and Development*. Springer, 2001, 347–361.

27. **Smyth, B.** and **L. McGinty**, The power of suggestion. *In IJCAI*. 2003.

28. **Thompson, C. A.**, **M. H. Göker**, and **P. Langley** (2004). A personalized system for conversational recommendations. *J. Artif. Intell. Res.(JAIR)*, **21**, 393–428.

29. **Wilke, W.**, **M. Lenz**, and **S. Wess**, Intelligent sales support with cbr. *In Case-based reasoning technology*. Springer, 1998, 91–113.

30. **Zhang, J.** and **P. Pu**, A comparative study of compound critique generation in conversational recommender systems. *In Adaptive Hypermedia and Adaptive Web-Based Systems*. Springer, 2006.