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

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(often in millions) of products in their databases. For a user wanting to purchase a product, examining all the products (eg: books) present in the catalog one after another in the hope of a finding an product that is of interest to him is impractical. 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 viewed or bought the same books - 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 approaches to do 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 ratings matrix and 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. 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, a new item that has been recently added to the catalog and has not been rated by a single user, has no possibility of being 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. For example, a user u whose rating pattern is very different from most of the other users 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.

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 that appeared in other 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 (Jannach *et al.* (2010)). 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. 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 systems 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 profile but also those that are too similar.

New user problem: The cold-start 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. The prediction accuracy of these systems improves with increase in the number of ratings.

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 those items whose features match the user's profile. However there are some domains in which both collaborative and content based methods have limitations. Typically, we 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, we cannot rely on that rating for generating relevant recommendations. 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 provide recommendations in such scenarios. 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.

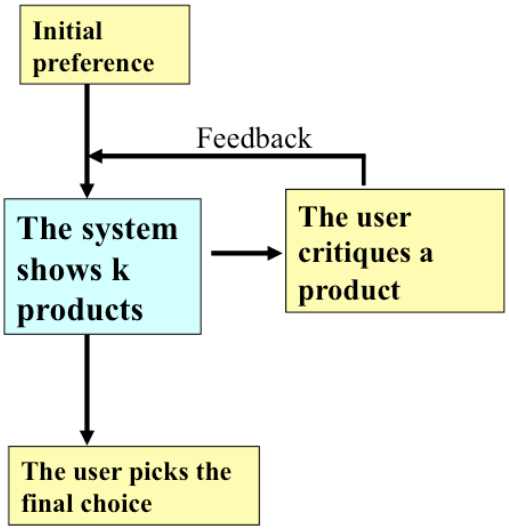

Figure 1.1: Critiquing

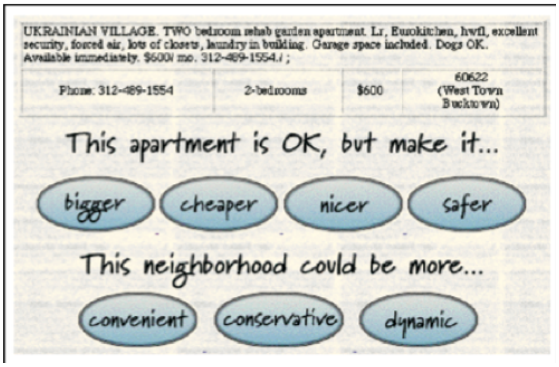

Figure 1.2: RentMe Recommender System: Burke (2000)

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. Knowledge-based recommender systems can be divided into two classes: constraint-based and case-based recommender systems. Constraint-based recommender systems rely on explicit recommendation rules and case-based recommender systems use similarity/utility measures to generate recommendations. Specifically, constraint-based recommendation is generated by looking at a collection of items satisfying the recommendation rules; case-based recommendation is typically generated based on similarity of the items in the database with the user defined query. MAUT based recommendation falls in the category of case-based recommendation. We discuss case-based recommendation in detail in Chapter 2

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/she 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 Chapter 2.

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 and individual feature level similarities are calculated according to the similarity function pertaining to the particular feature i which is denoted by $sim_i(q_i, C_i)$. These feature level similarities are referred to as local similarities, which are defined by domain experts. 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 in the database respectively. As we can see from Equation 2.2, greater the difference between p_i and p_j, lesser is the similarity between them. To estimate the similarities between the nominal feature (Eg:"manufacturer") values of two cameras, specialized domain knowledge is required. 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 analog device recommender recommends those op-amps to the user which are most similar to his query according to a pre-defined similarity measure(Wilke *et al.* (1998)). It is often desirable to have diverse products in recommendation lists. Having diverse items in recommendation list helps the user to develop a better understanding of different parts of the 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. 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 are considered in the beginning of recommendation procedure. In the first step, the top item in S is added to R and removed from S. In the next step, quality scores of all items in set S are computed using Equation 2.3. 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.

All the methods of introducing diversity into the recommendation process 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. In Order based retrieval (OBR) proposed by (Bridge and Ferguson (2002)) constructs an ordering relation based on user preferences revealed through user query. The ordering relation makes use of some basic operators to construct partial orders on the features constituting the user query Compromise driven retrieval (CDR) (McSherry (2003)) is a method where diversity gets introduced without its explicit characterization. The central idea put forth 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 her 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 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 users feedback at each interaction with the user to 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 either as *navigation by proposing* or *navigation by asking* (Simazu *et al.* (2001))

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.

2.3.1 Navigation by Asking

In this mode of feedback, 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 recommendations over devices such as mobile phones (Smyth (2007))

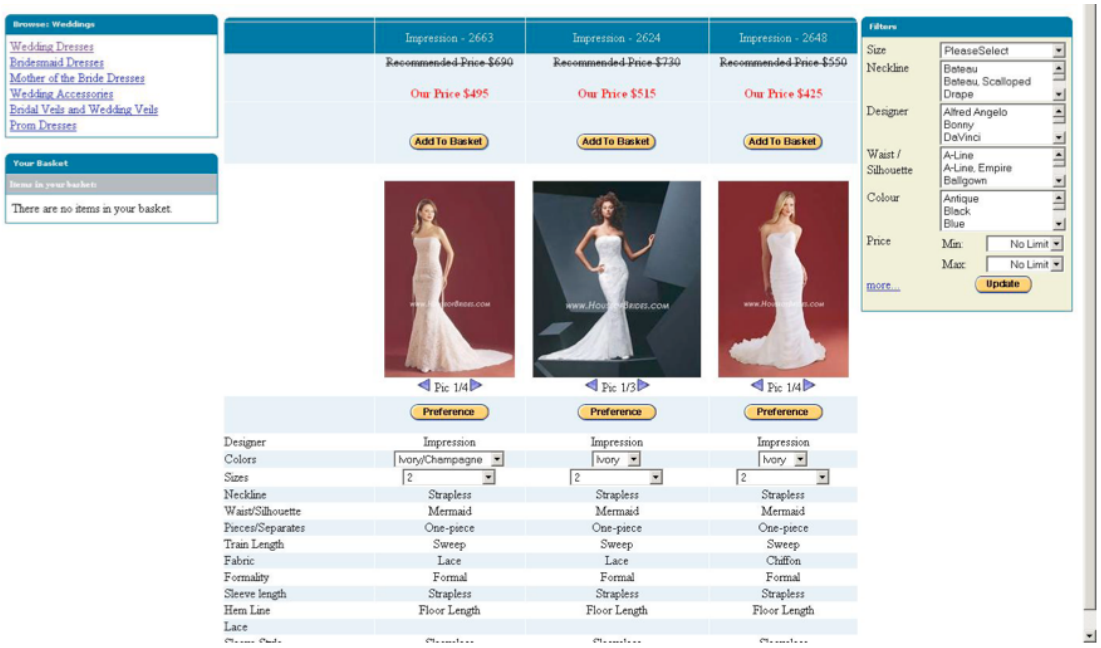

Figure 2.1: Recommender System that uses preference based feedback

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 recommender system shown in

Figure 2.1. The system shows 3 different item along with the feature values of each item in each cycle and 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 it is always not clear 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 is does not generate very effecient recommendations because it doesn't try to infer user's true preferences. There may be some features of the selected product that the user doesn't like. 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 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 a screen-size of 15 inches and all other $k-1$ rejected PCs also have the same screen-size, we can give a low

Figure 2.2: Entree Restaurant Recommender System

weight to 'screen-size' attribute, since we can't 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. Instead, as users become more familiar with the domain and the product options available, their preferences often change, becoming 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 [! = manfacturer]). Through directional critiques users 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 from consideration in the next cycle, and selects that case which is most similar to the current recommendation from those remaining; 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 by providing critiques on presented examples, such as restaurants (*Entree*), automobiles (*Car Navigator*), apartments (*RentMe*), and movie rentals (*Video Navigator*). 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.

Since unit critiques operate only on a single feature in a recommendation

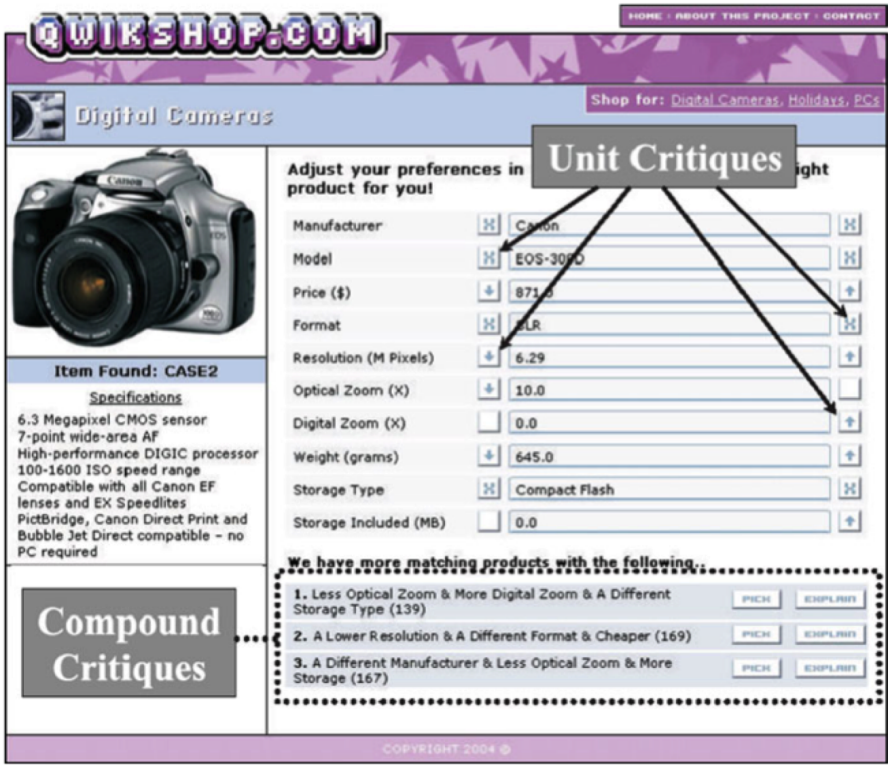

Figure 2.3: Screenshot of digital camera recommender system that uses Apriori algorithm to generate compound critiques

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. In Sections 2.4 and 2.5, we discuss the two most popular approaches to generation of dynamic 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

	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))

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. User is presented with unit critiques that correspond to each product feature and a set of compound critiques in each cycle. There are two steps involved in the generation of compound critiques: *Generating Critique Patterns* and *Mining 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 this new case to generate a so-called *critique pattern*. Figure 2.4 illustrates how critique pattern is generatd. As we can see in Figure 2.4, individual feature values of both the *current product* and product p are compared. $' <'$, $' >'$ and $' ='$ are 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)) can be 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 0.9 if there are 100 critique patterns but only 90 of them contain [Monitor $<$] and [Price $<$]. The confidence of the rule would be 1.0 if exactly 90 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. It has been observed experimentally 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. (McCarthy *et al.* (2004)) 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 that application of compound critiques with low support reduces the number of interaction cycles.

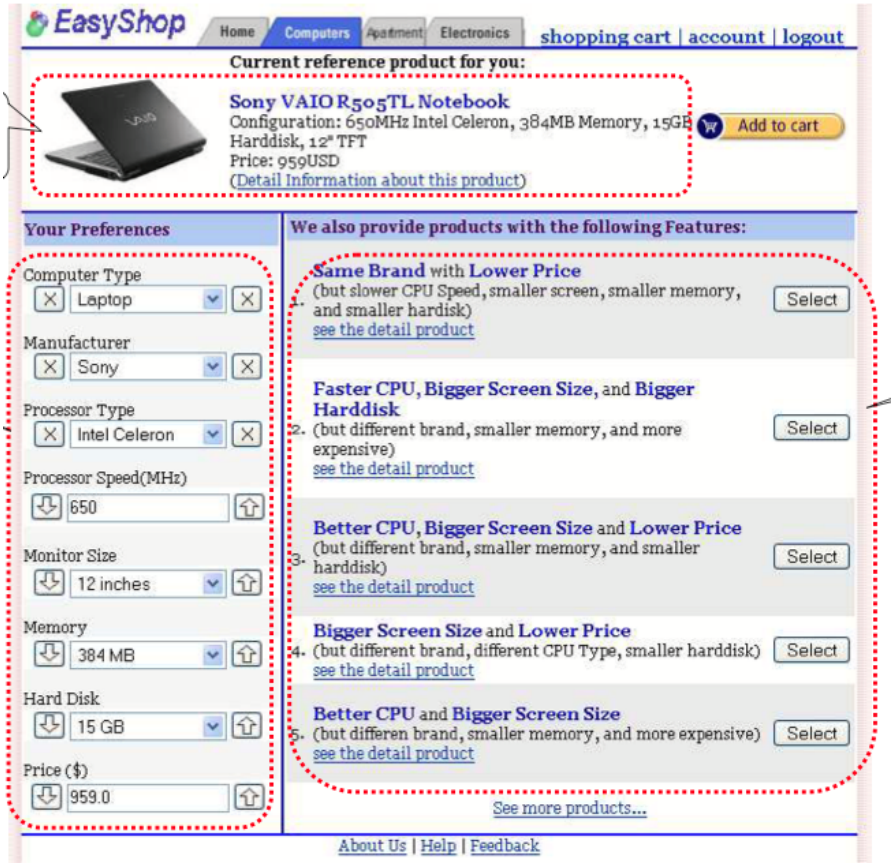

Figure 2.5: Screenshot of user interface of the recommender system that uses MAUT to generate compound critiques

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 outperformed several critiquing recommenders that used static critiquing. 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.

```
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

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.

2.6 Offline Evaluation

Ideally, we would like to conduct live user studies to test the effectiveness of system generated compound critiques, but live user studies are very expensive to conduct. Instead we conduct offline evaluations to test the effectiveness of a

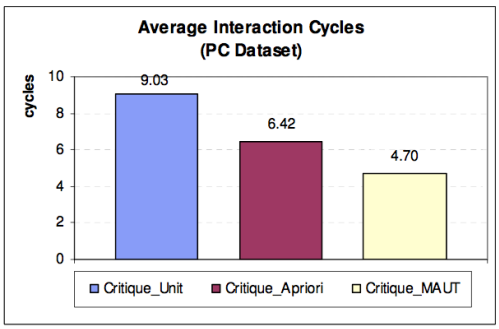

Figure 2.7: PC Dataset(Zhang and Pu (2006))

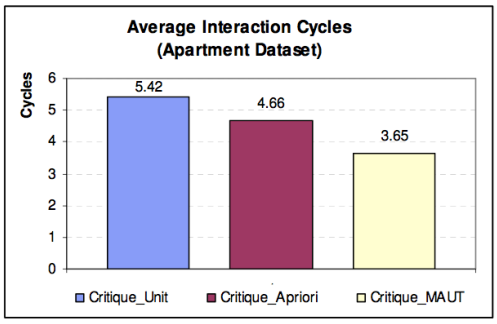

Figure 2.8: Apartment Dataset(Zhang and Pu (2006))

recommendation algorithm. 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 5 compound critiques which correspond to the 5 products with the highest utility. The compatiblity of each compound critique with respect to the target product is then calculated. 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 5 products. Each product in the dataset is appointed as the target product 10

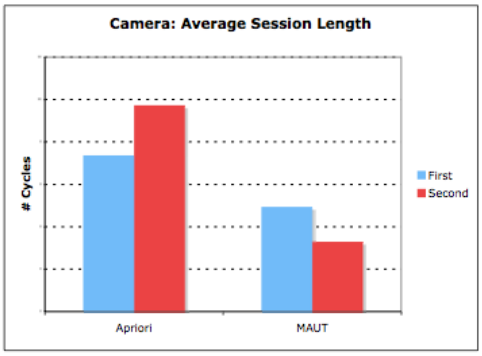

Figure 2.9: Avg session lengths on camera dataset (Reilly *et al.* (2007)

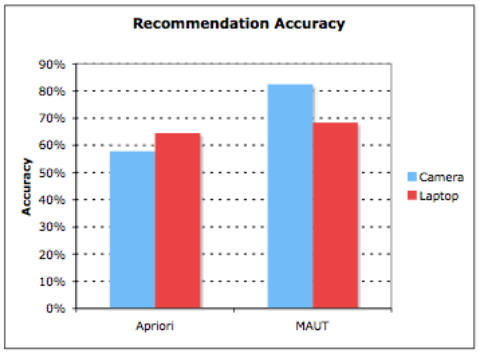

Figure 2.10: Avg recommendation accuracy (Reilly *et al.* (2007))

times and the average number of interaction cycles for all the experiments is calculated.

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 containing 103 digital cameras. A total of 83 users separately evaluated both systems by using each system to find a laptop that they would be willing to purchase The order in which the different systems were presented was randomized and at the start of the trial they were provided with a brief description of the basic recommender interface to explain the use of unit and compound critiques and basic system operation. 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 efficiently guide a user through a product-space and, in gen- eral, short recommendation sessions are to be preferred. Sequencing bias was eliminated by randomizing the presentation order in terms of critiquing technique and dataset. For the PC dataset, Apriori performed better (7.0 cycles) compared to MAUT (10.1 cycles). But for the camera dataset, MAUT performed better (4.1

cycles) compared to Apriori (8.5 cycles). The average session length for two trials is graphically shown in Figure 2.9. A possible explanation 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 users has interacted with the recommender sytem, he is asked to review their final selection with reference to the full set of products in the dataset. The accuracy of the recommender is evaluated in terms of percentage of times that the user chooses to stick with their selected product. Higher the number of times a user sticks with the product, 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 appears to lead to higher quality recom-

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

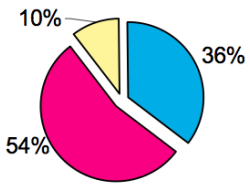

Figure 2.11: Final Questionnaire - Question 1
(Blue - Apriori, Red - MAUT, Yellow - No difference)

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

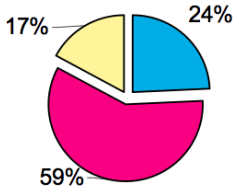

Figure 2.12: Final Questionnaire - Question 2

mendations. Overall, users responded equally well to both systems in terms of the recommendation performance, accuracy and interface style.

CHAPTER 3

Modifications 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. However, there are several limitations to MAUT based recommendation, which when addressed can improve it's performance significantly. The limitations to MAUT based recommendation are as follows:

- 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

We try to address the above each of the above limitations in Sections (Numbers) and also propose some additional improvements that improve the performance of MAUT based recommendation.

3.2 Diversity in Critiques

A variant of ***Bounded Greedy Selection*** algorithm described in Smyth and McClave (2001) was used to generate diverse critiques in every cycle. The function *GenCritiqueItems(PM, IS)* is modified as follows:

Algorithm 1: GenCritiqueItems(PM, IS)

$R \leftarrow \{\}$
$CB' \leftarrow IS$
for $i \leftarrow 0$ **to** k **do**
    Sort CB' by $Quality(i, R, PM)$ for each case in IS;
    $R \leftarrow R + First(CB')$;
    $CB' \leftarrow CB' - First(CB')$;
return R

Algorithm 2: Quality(i, R, PM)

if $R == \{\}$ **then**
    return 0;
else
    $retVal \leftarrow \alpha \times utility(i, PM)$;
    $disSim \leftarrow \frac{\sum_{r_j \in R}(1 - critiqueSim(i, r_j))}{|R|}$;
    $retVal+ = (1 - \alpha) \times disSim$;
    $returnretVal$;
return retVal

$critiqueSim(a, b)$ 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 (Users don't prefer critiques being very similar to each other).

3.3 Adaptive Selection

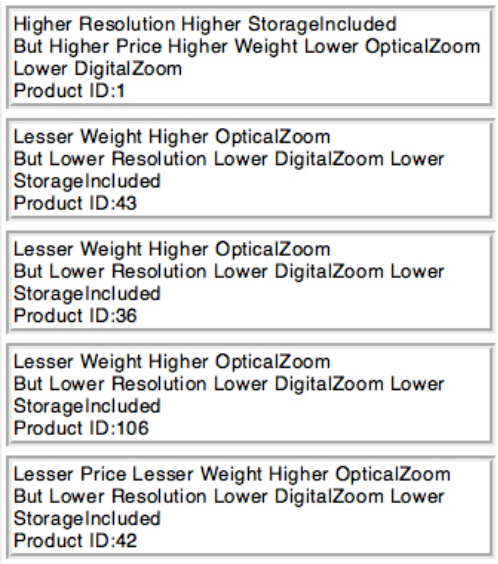

Figure 3.1: Before diversifying critique strings

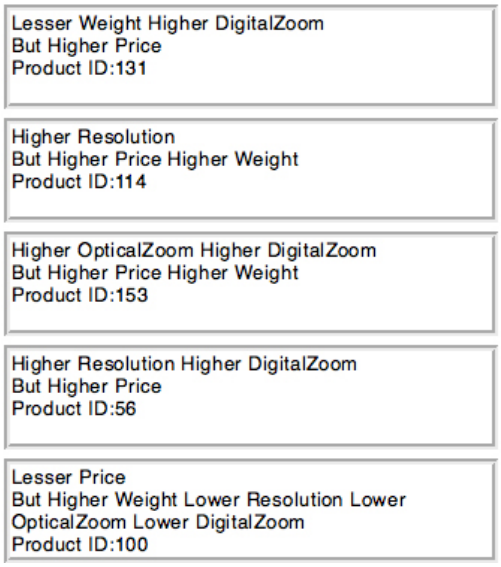

Figure 3.2: After diversification

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

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

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

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

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

19. **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.

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

21. **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.

22. **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.

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

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

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

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

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

28. **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.