AWESOME – A Data Warehouse-based System for Adaptive Website Recommendations

Andreas Thor Nick Golovin Erhard Rahm

University of Leipzig, Germany
{thor, golovin, rahm}@informatik.uni-leipzig.de

Abstract

Recommendations are crucial for the success of large websites. While there are many ways to determine recommendations, the relative quality of these recommenders depends on many factors and is largely unknown. We propose a new classification of recommenders and comparatively evaluate their relative quality for a sample website. The evaluation is performed with AWESOME (Adaptive website recommendations), a new data warehouse-based recommendation system capturing and evaluating user feedback on presented recommendations. Moreover, we show how AWESOME performs an automatic and adaptive closed-loop website optimization based on continuously measured recommendation feedback. We propose and evaluate several rule-based schemes for dynamically selecting the most promising recommendations. In particular, we investigate two-step selection approaches which first determine the most promising recommenders and then apply their recommendations for the current situation. We also evaluate one-step schemes which try to directly determine the most promising recommendations.

1 Introduction

Recommendations are crucial for the success of large web sites to effectively guide users to relevant information. E-commerce sites offering thousands of products cannot solely rely on standard navigation and search features but need to apply recommendations to help users quickly find "interesting" products or services. With many users and products manual generation of recommendations is much too laborious and ineffective. Hence a key question becomes how should recommendations be generated automatically to optimally serve the users of a website.

There are many ways to automatically generate recommendations taking into account different types of information (e.g. product characteristics, user characteristics, or buying history) and applying different statistical or data mining approaches ([JKR02], [KDA02]). Sample approaches include recommendations of top-selling products (overall or per product category), new products, similar products, products bought together by customers, products viewed together in the same web session, or products bought by similar customers. Obviously, the relative utility of these recommendation approaches (*recommenders* for short) depends on the website, its users and other factors so that there cannot be a single best approach. Website developers thus have to decide about which approaches they should support and where and when they should be applied. Surprisingly, little information is available in the open literature on the relative quality of different recommenders. Hence, one focus of our work is an approach for comparative quantitative evaluations of different recommenders.

Advanced websites, such as Amazon [LSY03], support many recommenders but apparently are unable to select the most effective approach per user or product. They overwhelm the user with many different types of recommendations leading to huge web pages and reduced usability. While commercial websites often consider the buying behaviour for generating recommendations, the usage (navigation) behaviour on the website remains largely unexploited. We believe this a major shortcoming since the navigation behaviour contains detailed information on the users' interests not reflected in the purchase data. Moreover, the web usage behaviour contains valuable user feedback not only on products or other content but also on the presented recommendations. The utilization of this feedback to automatically and adaptively improve recommendation quality is a major goal of our work.

AWESOME (Adaptive website recommendations) is a new data warehouse-based website evaluation and recommendation system under development at the University of Leipzig. It contains an extensible library of recommender algorithms that can be comparatively evaluated for real websites based on user feedback. Moreover, AWESOME can perform an automatic closed-loop website optimization by dynamically selecting the most promising recommenders / recommendations for a website access. This selection is based on the continuously meas-

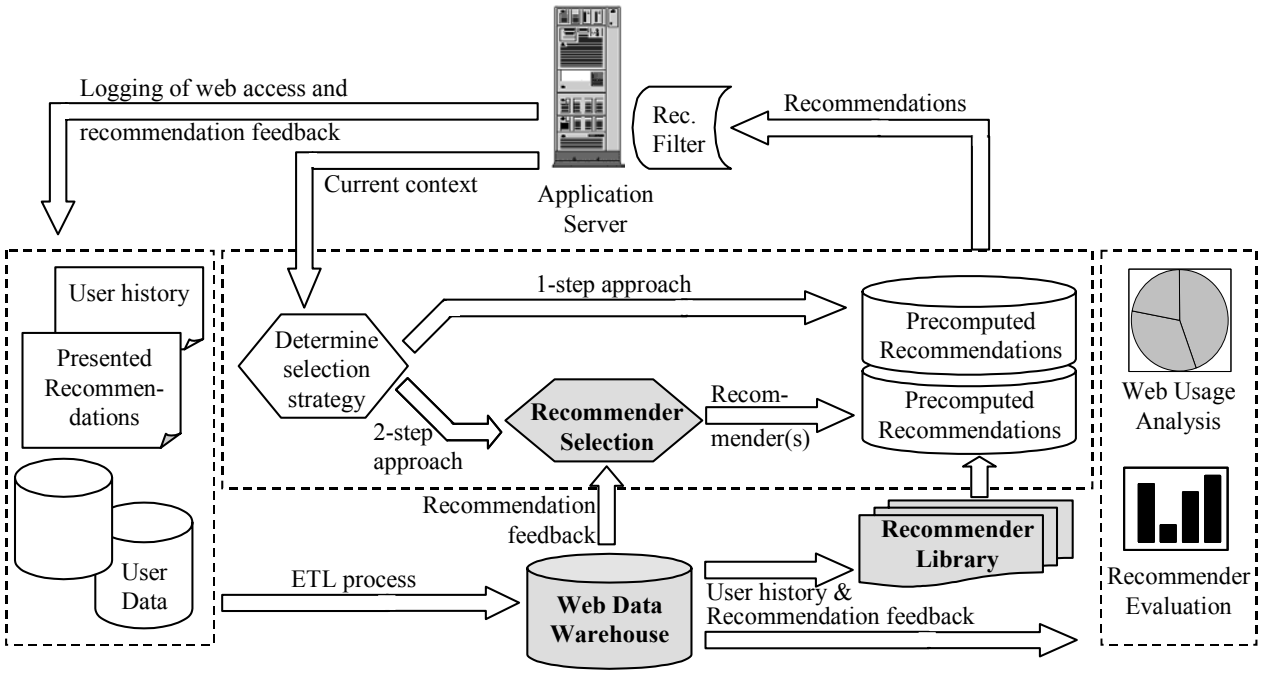

Figure 1: AWESOME architecture

ured recommendation quality of the different recommenders and recommendations so that AWESOME automatically adapts to changing user interests and changing content. To support high performance and scalability, quality characteristics of recommenders and recommendations are largely precomputed. AWESOME is fully operational and in continuous use at a sample website; adoption to further sites is in preparation.

The main contributions of this paper are as follows:

- The AWESOME architecture for warehouse-based recommender evaluation and for scaleable adaptive website recommendations
- Details on the data warehouse implementation to illustrate how good data quality is achieved and how recommendation feedback is maintained
- A new classification of recommenders for websites supporting a comparison of different approaches. We show how sample approaches fit the classification and propose a new recommender for users coming from search engines.
- A comparative quantitative evaluation of several recommenders for a sample website. The considered recommenders cover a large part of our classification's design space.
- Differentiation between two-step and one-step approaches for dynamically determining recommendations based on different types of selection rules. One-step approaches try to directly determine the most suitable recommendations while two-step approaches first determine the most promising recommenders and then apply their recommendations for the current situation.
- Description of several rule-based schemes for two-step and one-step recommendation selection including machine learning approaches for taking recommendation feedback into account.
- Comparative evaluation of the approaches for dynamic recommender and recommendation selection for a sample website.

In the next two sections we present the AWESOME architecture and the underlying data warehouse approach. We then outline our recommender classification and sample recommenders (Section 4). Section 5 contains the comparative evaluation of several recommenders for a non-commercial website. In Section 6 we describe the feedback based selection of recommenders and recommendations. A comparative evaluation of both selection approaches is given in Section 7. Related work is briefly reviewed in Section 8 before we conclude.

2 Architecture

2.1 Overview

Fig. 1 illustrates the overall architecture of AWESOME which is closely integrated with the application server running the website. AWESOME is invoked for every website access, specified by a so-called *context* including information from the current HTTP request such as URL, timestamp and user-related data. For such a context, AWESOME dynamically generates a list of recommendations which are displayed by the application server together with the requested website content. Recommendations are automatically determined by a variety

of algorithms from an extensible *recommender library*. The recommenders use information on the usage history of the website and additional information maintained in a *web data warehouse*. For performance reasons, recommendations are pre-computed and periodically refreshed and can thus quickly be looked up at runtime. The recommendations are subject to a final filter step to avoid the presentation of unsuitable or irrelevant recommendations, e.g., recommendation of the current page or the homepage.

AWESOME supports a two-step and one-step selection of recommendations at runtime. The former approach first selects the most appropriate recommender(s) for the current context and then applies their recommendations. The one-step approach directly determines the most promising recommendations for the current context. Both approaches are controlled by selection rules which may manually or automatically be determined. The automatic approaches utilize user feedback on previously presented recommendations which is recorded in the web data warehouse.

Two-step selection approaches typically use a small or moderate number of *recommender rules* to determine the most promising recommenders per context. Feedback is associated to recommenders, which is simpler and more stable than trying to use feedback for individual recommendations, e.g. specific web pages or products. A completely manual specification of recommender rules is also facilitated. Adding and deleting web pages or products does not directly influence the aggregated feedback and can thus easily be supported. Moreover, adapting the rules can be performed less frequently than in one-step approaches since significant changes in the selection rules can only be expected when there is a certain amount of new feedback for most recommenders.

On the other hand, one-step approaches use so-called *recommendation rules* to directly determine the most promising recommendations per context. This approach is thus more fine-grained and allows selection of good recommendations irrespective of by which algorithm they were determined. One problem with the one-step approach is that individual pages/products may be frequently added and that there is no feedback available for such new content. Conversely, removing content results in a loss of the associated recommendation feedback.

Since there are typically many more possible recommendations than recommenders, the number of recommendation rules needs to be much larger than the number of recommender rules. This may make it more difficult not only to achieve fast recommendation selection but also to ensure that all potential recommendations are actually presented and thus obtain a fair chance to collect recommendation feedback. On the other hand, associating feedback to individual recommendations may enable a more responsive approach by adapting the selection rules more frequently than with the two-step scheme. We have thus implemented approaches which not only re-compute recommendations periodically (offline) but also use short-term feedback on presented recommendations for an online adaptation of recommendation rules.

Strategy type	2-step	1-step
Rule type / granularity	Recommender	Recommendation
No. of rules	Small to moderate	Large
Rule adaptation	Offline	Offline and Online

Table 1: Strategies for dynamic selection of recommendations

Table 1 summarizes some of the characteristics of the two strategy types for dynamic selection of recommendations. AWESOME implements several two-step and one-step selection strategies including machine learning approaches for taking recommendation feedback into account. We will present the approaches and evaluate their effectiveness in sections 6 and 7, respectively.

AWESOME is based on a comprehensive web data warehouse integrating information on the website structure and content (e.g., product catalog), website users and customers, the website usage history and recommendation feedback. The application server continuously records the users' web accesses and which presented recommendations have been and which ones have NOT been followed. During an extensive ETL (extract, transform, load) process (including data cleaning, session and user identification) the usage data and recommendation feedback is added to the warehouse.

The warehouse serves several purposes. Most importantly it is the common data platform for all recommenders and keeps feedback for the dynamic recommender/recommendation selection thus enabling an automatic closed-loop website optimization. However, it can also be used for extensive offline evaluations, e.g., using OLAP tools, not only for web usage analysis but also for a comparative evaluation of different recommenders and of different selection strategies.

This functionality of AWESOME allows us to systematically evaluate the various approaches under a large range of conditions. It is also an important feature for website designers to fine-tune the recommendation system, e.g. to deactivate or improve less effective recommenders. Therefore, the recommender library is strictly separated from the selection process. New recommenders can be added to AWESOME without changing other parts of the system. Furthermore, the functionality of all recommenders is encapsulated so that further development of recommenders is not restricted.

2.2 Prototype implementation

The current AWESOME implementation runs on different servers. The recommendation engine is largely implemented by PHP programs on a Unix-based application server running the Apache web server. The precomputed recommendations and selection rules are maintained in a MySQL database. During the 2-step selection process, for

a given context and a set of selected recommenders a dynamic SQL statement is generated to select recommendations at runtime.
The warehouse is on a dedicated machine running MS SQL server. Most recommenders are implemented in SQL, but also Java and Perl programs are utilized. All recommenders are periodically (currently, once a day) executed to recompute the recommendations.

3 Web Data Warehouse

To illustrate the value of a data warehouse approach for generating web recommendations we provide some details on its implementation and design. We present the log formats of the application server and explain some ETL functions important for achieving good data quality. Furthermore, we sketch the warehouse schema used for web accesses and recommendation feedback.

3.1. Web log files

The standard log files of web servers are not sufficient for our purposes because to obtain sufficient recommendation feedback we need to record all presented recommendations and whether or not they have been followed. We thus decided to use tailored application server logging to record this information. Application server logging also enables us to apply effective approaches for session and user identification and early elimination of crawler accesses, thus supporting high data quality (see below).

We use two log files: a web usage and a recommendation log file with the formats shown in Table 2. The recommendation log file records all presented recommendations and is required for our recommender evaluation. It allows us to determine positive and negative user feedback, i.e. whether or not a presented recommendation was clicked. For each presented recommendation, we also record the relative position of the recommendation on the page, the generating recommender and the used selection strategy.

Page	requested page
Date, Time	date and time of the request
Client	IP address of the user's computer
Referrer	referring URL
Session ID	session identifier
User ID	user identifier

a) Web usage log

Pageview ID	page view where recommendation has been presented
Recommendation	recommended content
Position	position of this recommendation inside a recommendation list
Recommender	recommender that generated the recommendation (only for 2-step selection)
Strategy	name and type (1-step or 2-step) of selection strategy

b) Recommendation log

Table 2: Log file formats

The web usage log file adds two elements to the standard Common Log Format (CLF) of common web servers: user ID and session ID. User IDs are stored inside permanent cookies and are used for user identification. If the user does not accept permanent cookies, user recognition is not done. The session ID is generated by the application server and stored inside a temporary cookie on the user's computer (if enabled). It is used for data preparation inside the ETL process, which is described in the following section.

3.2. ETL process

The ETL process periodically processes the log files of the application server and other data sources to refresh the data warehouse. Like in other data warehouse projects the main ETL tasks are data cleaning and transformation to ensure good data quality for the warehouse. The specifics of web usage data require tailored data preparation steps which have been investigated thoroughly in recent years; [CMS99] gives a good overview. We illustrate the AWESOME approaches for two data preparation problems: crawler detection and session reconstruction.

Crawler detection deals with the identification of page requests not originating form human users but programs such as search engine crawlers or website copiers. Obviously, we do not want to generate recommendations for such programs and only consider human navigation behavior. We use a combination of several heuristics to detect crawler accesses and avoid their logging or eliminate them from the log. First, we utilize a list of IP addresses of known crawlers (e.g., crawl1.googlebot.com as a crawler from the Google search engine). Similarly, we provide a list of known agent names of website copiers (e.g., HTTrack[1]) to identify and eliminate their requests. In addition, on every delivered page the application server integrates a link that is invisible to humans. Sessions containing a request to this linked page are deleted. Finally we analyze the navigation behavior by measuring attrib-

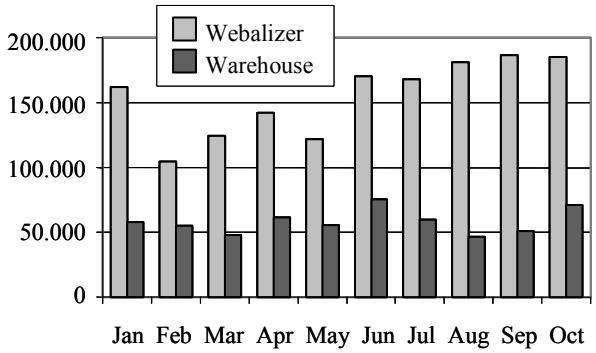

Figure 2: Number of monthly page views (2004)

[1] http://www.httrack.com/

Session ID	Host	Time	Page	Referrer	Session
-	cw12.H1.srv.t-online.de	01:15:02	/DBS1/ch4-1.html	www.google.com/search?q=sql+statements	A
-	pD9E1621B.dip.t-dialin.net	01:15:10	/ADS1/ch3-5.html	www.google.com/search?q=linked+list	B
1	vc05cop7001.cpmc.columbia.edu	01:15:12	/People/rahm.html	research.microsoft.com/~philbe/	C
-	cw12.H1.srv.t-online.de	01:15:20	/DBS1/ch4-2.html	/DBS1/ch4-1.html	A
1	vc05cop7001.cpmc.columbia.edu	01:15:21	/index.html	/People/rahm.html	C
-	cw08.H1.srv.t-online.de	01:15:23	/DBS1/ch4-3.html	/DBS1/ch4-2.html	A
-	pD9E1621B.dip.t-dialin.net	01:15:43	/ADS1/ch3-4.html	/ADS1/ch3-5.html	B

Table 3: Fraction of web usage log file containing three sessions

utes like session length, average time between two requests and number of requests with blank referrer to distinguish between human user sessions and web crawler sessions [TK00]. To illustrate the effectiveness of our crawler detection we performed a comparison with a freely available web log file analysis tool[2] which currently does not eliminate crawler accesses. Figure 2 compares the number of page views determined with this tool compared with the number of page views determined by AWESOME after elimination of crawler accesses for the same website and time period. The comparison illustrates that about 70% of all page views are identified as non human requests for the considered website and that this share may actually vary significantly. This demonstrates the need and importance of an exhaustive crawler detection and that not performing this kind of data cleaning leads to irrelevant data and analysis results.

Another important data preparation task is session reconstruction to group page views from the same user during a website visit. Knowledge about sessions is obviously needed to analyze the navigation behavior of users as well as to determine whether presented recommendations are accepted. In the terminology of [SMB+03], AWESOME supports both a proactive and reactive session reconstruction. Proactive session reconstruction is done by a session ID generated by the application server and stored within temporary cookies. If temporary cookies are enabled (which is the case for about 85% of the users of our prototype website) they provide a safe method to session construction. The reactive approaches are based on heuristics and come into play when temporary cookies are disabled. In a first heuristic, we use the combination of the user's host name (actually only the last four parts of it to deal with varying host names, e.g., from proxy servers) and agent name as a temporary identifier. All accesses with the same identifier and a temporal distance below a threshold are considered to belong to the same session. Additionally, we use the referrer information to reconstruct sequences of page views. Given the high share of proactive session construction and the typically high accuracy of the heuristics we expect that the vast majority of sessions can be correctly constructed.

3.3 Data warehouse design

The web data warehouse is a relational database with a "galaxy" schema consisting of several fact tables sharing several dimensions. Like in previous approaches on web usage analysis [KM00] we use separate fact tables for page views, sessions, and – for commercial sites – purchases. In addition we use a recommendation fact table. The simplified schema in Fig. 3 shows the page view and recommendation fact tables and their dimension tables which are referenced by the respective ID attributes. The details of these tables depend on the website, e.g. on how the content (e.g., products), users or customers are categorized. In the example of Fig. 3 there are two content dimensions for different hierarchical categorizations. Such hierarchies may represent different kinds of product catalogs, e.g., organized by topic or by media type. Other dimensions such as user, customer, region and date are also hierarchically organized and thus allow evaluations and recommendation decisions at different levels of detail.

The page view fact table represents the web usage history and contains a record for each requested page. The duration until the next page view of the same session is stored in the view time attribute. The recommendation fact table represents the positive and negative user feedback on recommendations. Each record in this table refers to one presented recommendation. The recommended content is characterized by the content dimension tables, whereas the utilized recommender and the applied selection strategy are described using additional dimension tables. Three Boolean measures are used to derive recommendation quality metrics (see Section 5). *Accepted* indicates whether or not the recommendation was directly accepted (clicked) by the user, while *Viewed* specifies whether the recommended content was viewed later during the respective session (i.e., the recommendation was a useful hint). *Purchased* is only used for e-commerce websites to indicate whether or not the recommended product was purchased during the current session.

[2] http://www.mrunix.net/webalizer/

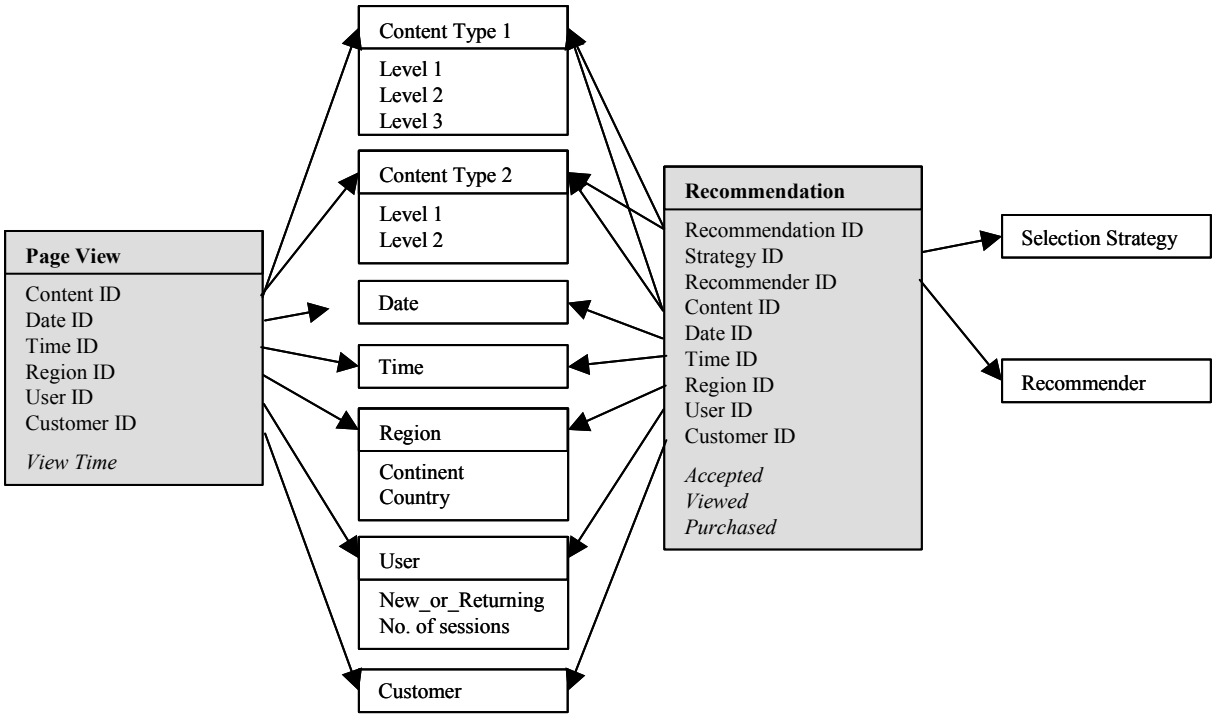

Figure 3: Simplified data warehouse galaxy schema

4 Recommenders

A recommender generates for a given web page request, specified by a context, an ordered list of recommendations. Such recommendations link to current website content, e.g. pages describing a product or providing other information or services.

To calculate recommendations, recommenders can make use of the information available in the context as well as additional input, e.g. recorded purchase and web usage data. We distinguish between three types of context information relevant for determining recommendations:

- Current content, i.e. the currently viewed content (page view, product, …) and its related information such as content categories
- Current user, e.g. identified by a cookie, and associated information, e.g. her previous purchases, previous web usage, interest preferences, or current session
- Additional information available from the HTTP request (current date and time, user's referrer, …)

4.1 Recommender classification

Given the many possibilities to determine recommendations, there have been several attempts to classify recommenders ([Bu02], [KDA02], [SKR01], [TH01]). These classifications typically started from a given set of recommenders and tried to come up with a set of criteria covering all considered recommenders. This led to rather complex and specialized classifications with criteria that are only relevant for a subset of recommenders. Moreover, new recommenders can easily require additional criteria to keep the classification complete. For example, [SKR01] introduce a large number of specialized criteria for e-commerce recommenders such as input from target customers, community inputs, degree of personalization, etc.

To avoid these problems we propose a general top-level classification of website recommenders focusing on the usable input data, in particular the context information. This classification may be refined by taking additional aspects into account, but already leads to a distinction of major recommender types thus illustrating the design space. Moreover, the classification helps to compare different recommenders and guides us in the evaluation of different approaches.

Fig. 4 illustrates our recommender classification and indicates where sample approaches fit in. We classify recommenders based on three binary criteria, namely whether or not they use information on the current *content*, the current *user*, and recorded usage (or purchase) *history* of users. This leads to a distinction of eight types of recommenders (Fig. 4). We specify each recommender type by a three-character-code describing whether (+) or not (–) each of the three types of information is used. For instance, type [+,+,–] holds for recommenders that use information on the current content and current user, but do not take into account user history.

The first classification criteria considers whether or not a recommender uses the current content, i.e. the currently requested page or product. A sample content-based approach (type [+,–,–]) is to recommend content that is

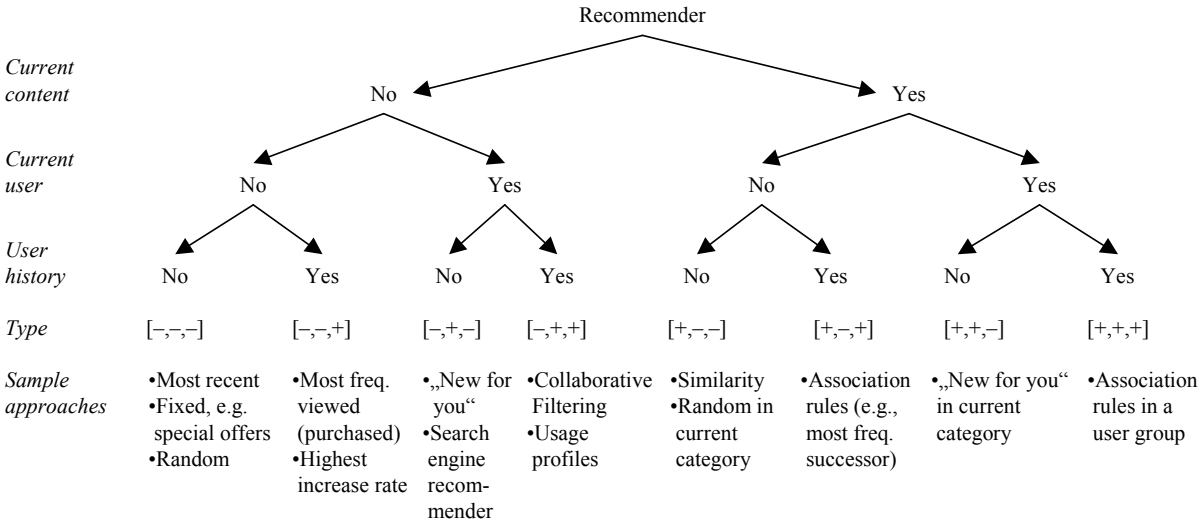

Figure 4: Top-level classification of recommenders

most *similar* to the current content, e.g. based on text-based similarity metrics such as TF/IDF. Content-based recommenders may also use generalized information on the content category (e.g., to recommend products within the current content category). Sample content-insensitive recommenders (type [–,–,–]) are to recommend the *most recent* content, e.g. added within the last week, or to give a fixed recommendation at each page, e.g. for a special offer.

At the second level we consider whether or not a recommender utilizes information on the current user. User-based approaches could thus provide special recommendations for specific user subsets, e.g. returning users or customers, or based on personal interest profiles. Recommenders could also recommend content for individuals, e.g. new additions since a user's last visit (*"New for you"*). We developed a new recommender of type [–,+,–] for users coming from a search engine such as Google. This *search engine recommender (SER)* utilizes that the HTTP referrer information typically contains the search terms (keywords) of the user [KMT00]. SER recommends the website content (different from the current page that was reached from the search engine) that best matches these keywords. The SER implementation in AWESOME utilizes a predetermined search index of the website to quickly provide the recommendations at runtime.

With the third classification criteria we differentiate recommenders by their use of *user history* information. For commercial sites, recommenders can consider information on previous product purchases of customers. Another example is the evaluation of the previous navigation patterns of website users. Simple recommenders of type [–,–,+] recommend the *most frequently* purchased/viewed content (top-seller) or the content with the *highest* recent *increase* of interest.

While not made explicit in the classification, recommenders can utilize additional information than on current content, current user or history, e.g. the current date or time. Furthermore, additional classification criteria could be considered, such as metrics used for ranking recommendations (e.g. similarity metrics, relative or absolute access/purchase frequencies, recency, monetary metrics, etc.) or the type of analysis algorithm (simple statistics, association rules, clustering, etc.).

4.2 Additional approaches

Interesting recommenders often consider more than one of the three main types of user input. We briefly describe some examples to further illustrate the power and flexibility of our classification and to introduce approaches that are considered in our evaluation.

[+,–,+]: *Association rule* based recommenders such as "Users who bought this item also bought …", made famous by Amazon [LSY03], consider the current content (item) and purchase history but are independent of the current user (i.e. every user sees the same recommendations for an item). Association rules can also be applied on web usage history to recommend content which is frequently viewed together within a session.

[–,+,+] Information on navigation/purchase history can be used to determine *usage profiles* [MDL+02] or groups of similar users, e.g. by *collaborative filtering* approaches. Recommenders can assign the current user to a user group (either based on previous sessions or the current session) and recommend content most popular for this group.

In our evaluation we test a *personal interests* recommender, which is applicable to returning users. It determines the most frequently accessed content categories per user as an indication of her personal interests. When the

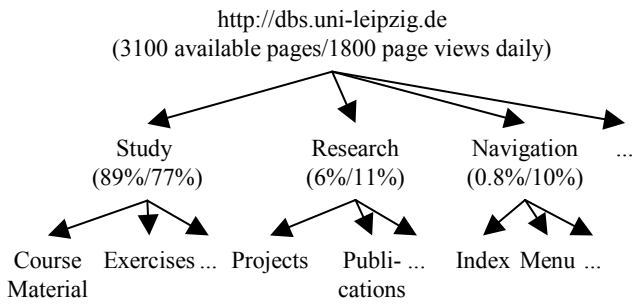

Figure 5: Example of content hierarchy

user returns to the website, the most frequently accessed content of the respective categories is recommended.

[+,+,+] A recommender of this type could use both user groups (as discussed for [–,+,+]) and association rules to recommend the current user those items that were frequently accessed (purchased) by similar users in addition to the current content.

5 Recommender evaluation

The AWESOME prototype presented in Section 2 allows us to systematically evaluate recommenders for a given website. In Section 5.2, we demonstrate this for a sample non-commercial website. Before that, we introduce several metrics for measuring recommendation quality which are needed for our evaluation of recommenders and selection strategies.

5.1 Evaluation metrics

To evaluate the quality of presented recommendations we utilize the Accepted, Viewed, and Purchased measures recorded in the recommendation fact table (Section 3.3). The first two are always applicable, while the last one only applies for commercial websites. We further differentiate between metrics at two levels of granularity, namely with respect to page views and with respect to user sessions.

Figure 6: Recommendation screenshot

Acceptance rate is a straightforward, domain-independent metric for recommendation quality. It indicates the share of page views for which at least one presented recommendation was accepted, i.e. clicked. The definition thus is

$AcceptanceRate = |P_A| / |P|$

where P is the set of all page views containing a recommendation and P_A the subset of page views with an accepted recommendation.

Analogously we define a session-oriented quality metric

$SessionAcceptanceRate = |S_A| / |S|$

where S is the set of all user sessions and S_A the set of sessions for which at least one of the presented recommendations was accepted.

Recommendations can also be considered of good quality if the user does not directly click them but reaches the associated content later in the session (hence, the recommendation was a correct prediction of user interests). Let P_V be the set of all page views for which any of the presented recommendations was reached later in the user session. We define

$ViewRate = |P_V| / |P|$

The corresponding metric at the session level is

$SessionViewRate = |S_V| / |S|$

where S_V is the set of all user sessions with at least one pageview in P_V. Obviously, every accepted recommendation is also a viewed recommendation, i.e. $P_A \subseteq P_V \subseteq P$ and $S_A \subseteq S_V \subseteq S$, so that view rates are always larger than or equal to the acceptance rates.

In commercial sites, product purchases are of primary interest. Note that purchase metrics should be session-oriented because the number of page views needed to finally purchase a product is of minor interest. A useful metric for recommendation quality is the share of sessions S_{AP} containing a purchase that followed an accepted recommendation of the product. Hence, we define the following metric:

$RecommendedPurchaseRate = |S_{AP}| / |S|$

Obviously, it holds $S_{AP} \subseteq S_A \subseteq S$.

5.2 Sample evaluation

We implemented and tested the AWESOME approach for recommender evaluation for a sample website, namely the website of our database group (http://dbs.uni-leipzig.de). We use two content hierarchies and Fig. 5 shows a fragment of one of them together with some numbers on the relative size and access frequencies. The website contains more than 3100 pages and receives about 1800 human page views per day (excluding accesses from members of our database group and from crawlers). As indicated in Fig. 5, about 89% of the content is educational study material, which receives about 77% of the page views.

We changed the existing website to show two recommendations on each page so that approx. 3600 recommendations are presented every day. For each page view

Recommender		User type		
Type	Name	New users	Returning users	Σ
[–,–,–]	Most recent	(0.72%)	(2.25%)	*(0.93%)*
[–,–,+]	Most frequent	1.05%	1.19%	*1.09%*
[–,+,–]	SER	2.83%	1.42%	*2.77%*
[–,+,+]	Personal Interests	–	1.43%	*1.43%*
[+,–,–]	Similarity	1.86%	0.76%	*1.73%*
[+,–,+]	Association Rules	1.61%	1.49%	*1.59%*
	Σ	*1.85%*	*1.29%*	*1.74%*

Table 4: Acceptance rate vs. user type

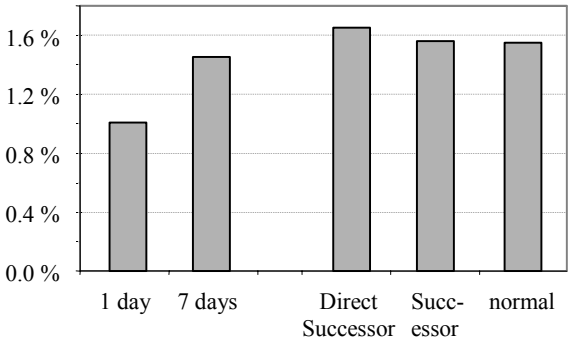

Figure 7: Page view acceptance rate for different types of most frequent and association rule recommenders

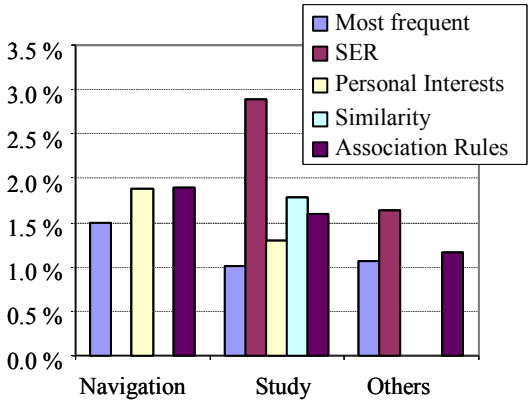

Figure 8 : Acceptance rate vs. page type

AWESOME dynamically selects one recommender and presents its two top recommendations (see example in Fig. 6) for the respective context as described in Section 2. We implemented and included more than 100 recommenders in our recommender library. Many of them are variations of other approaches, e.g. considering different user categories or utilizing history data for different periods of time. Due to space constraints we only present results for the six representative recommenders of different types listed in Table 4, which were already introduced in Section 4. The presented results refer to the period from December 1st, 2003 until April 30th, 2004.

The AWESOME warehouse infrastructure allows us to aggregate and evaluate recommendation quality metrics for a huge variety of constellations (combination of dimension attributes), in particular for different recommenders. For our evaluation we primarily use (page view) acceptance rates as the most precise metric[3]. The average acceptance rate for all considered recommenders was 1.44%; the average view rate was 15.89%. For sessions containing more than one page view the average session acceptance rate was 9.03%, and the session view rate was 27.83%. These rather low acceptance rates are influenced by the fact that every single web page contains a full navigation menu with 78 links (partly nested in sub menus) and that we consciously do not highlight recommendations using big fonts or the like. Note however, that reported "click-tru" metrics are in a comparable range than our acceptance rates [CLP02]. Furthermore, the absolute values are less relevant for our evaluation than the relative differences between recommenders.

Table 4 shows the observed acceptance rates for the six recommenders differentiating between new and returning users. Fig. 8 compares the recommenders w.r.t. the current page type. As expected there are significant differences between recommenders. For our website the *search engine* recommender (SER) achieved the best average acceptance rates (2.77%), followed by the *similarity* and *association rules* recommenders. On the other hand, simple approaches such as recommending the *most frequently* accessed or *most recent* content achieved only poor average results.

To more closely analyze the differences for different user and content types, one has to take into account that some recommenders are not always applicable. For instance, the *personal interests* recommender is only applicable for returning users (about 15% for our website). Similarly, SER can only be applied for users coming from a search engine, 95% of which turned out to be new users of the website. In Fig. 8 we only show results for recommenders with a minimum support of 5% i.e. they were applied for at least 5% of all page views of the respective page type. In Table 4 the results for the *most recent* recommender are shown in parentheses because the minimal support could not be achieved due to relatively few content additions during the considered time period.

Table 4 shows that new users are more likely to accept recommendations than returning users. An obvious explanation is that returning users (e.g., students for our website) often know where to find relevant information on the website. We also observed that the first page view of a session has a much higher acceptance rate (5.77 %) than later page views in a session (1.35 %). In the latter value, the last page view of a session is not considered, because its acceptance rate obviously equals 0[4].

Although we proposed a general top level classification of recommenders, we can consider additional (recommender type specific) attributes to characterize recommenders. Such attributes are helpful for recommender fine-tuning, i.e. manual optimization of single recommenders regarding their functionality. Fig. 7 illustrates

[3] The recommendations presented during a session typically come from different recommenders making the session-oriented quality metrics unsuitable for evaluating recommenders. Session acceptance rates will be used in Section 7. The Recommended-PurchaseRate does not apply for non-commercial sites.

[4] Layout aspects and other factors also influence acceptance rates. For instance, from the two recommendations shown per page the acceptance rate of the first one was about 50% higher compared to the second recommendation.

{ Usertype='new user' AND ContentCategory1='Navigation' }	➠	*'Most frequent'*	*[0.6]*
{ Referrer='search engine' }	➠	*'SER'*	*[0.8]*
{ Clienttype='university' AND Usertype='returning user' }	➠	*'Personal interest'*	*[0.4]*

Figure 9: Examples of recommender rules

fine-tuning experiments for the *most frequent* and *association rules* recommenders. For the *most frequent* recommender we varied the time window of the considered user history from 24 hours to 7 days. The results obtained indicate that for our website a user history of the last 7 days is to be preferred. For the *association rule* recommender we considered the influence of the relative page view position. For a given page *(direct) successor* only takes into account the (directly) following page views, whereas *normal* utilizes all page views inside the same session. The results shown in Fig. 7 indicate that the differences between the approaches are small for our website and that it suffices to only consider the direct successors by the association rule recommender.

Fig. 8 illustrates that the relative quality of recommenders differs for different contexts. While the SER recommender achieved the best average results for Study pages, the *association rules* and *personal interests* recommenders received the best user feedback on navigation pages. For study pages and non search engine users (when SER is not applicable), either the *similarity* or *association rules* promise the best recommendation quality.

While these observations are site-specific they illustrate that the best recommender (and the best recommendation respectively) depends on context attributes such as the current content or current user. A careful OLAP analysis may help to determine manually which recommender should be selected in which situation. However, for larger and highly dynamic websites this is difficult and labor-intensive so that recommender and recommendation selection should be automatically optimized.

6 Adaptive website recommendations

AWESOME supports a dynamic selection of recommendations for every website access. As discussed in Section 2, this selection is either a two-step or one-step process controlled by recommender or recommendation rules, respectively. Such rules may either be manually defined or automatically generated. We first present the structure and use of these two types of selection rules. In 6.2, we propose two approaches to automatically generate recommender rules which utilize recommendation feedback to adapt to changing conditions. In 6.3, we present an approach to automatically generate and adapt recommendation rules.

6.1 Rule-based selection of recommendations

Determination of adaptive website recommendations entails the dynamic selection of the most promising recommendations. Since AWESOME supports two ways of recommendation selection, it utilizes two types of selection rules of the following structure:

1. Recommender rule:
   ContextPattern ➠ *recommender* [*weight*]

2. Recommendation rule:
   ContextPattern ➠ *recommendation* [*weight*]

Here context pattern is a sequence of values from different context attributes which are represented as dimension attributes in our warehouse. Typically, only a subset of attributes is specified implying that there is no value restriction for the unspecified attributes. The examples in Fig. 9 illustrate this for recommender rules.

On the right hand side of recommender rules, *recommender* uniquely identifies an algorithm of the recommender library. Conversely, recommendation rules directly point to single recommendations of the set of precomputed recommendations. *Weight* is a real number specifying the relative importance of the rule.

In AWESOME, we maintain all selection rules in a *Recommender-Rules* and *Recommendation-Rules* table respectively. In fact, for the 1-step selection the selection rules and precomputed recommendations fall together. Hence, the recommendation-rule table corresponds to the set of precomputed recommendations extended with the weight attribute which may be dynamically adapted. To simultaneously allow 1-step and 2-step recommendation selection, we keep the recommendation-rule table separate from the precomputed recommender-specific recommendations.

Selection rules allow a straight-forward and efficient implementation of the selection process. It entails a *match* step to find all rules with a context pattern matching the current context. The rules with the highest weights then indicate the recommenders/recommendations to be applied. The number of recommenders/recommendations to choose is typically fixed, say *k*, i.e., we choose the *k* rules with the highest weight. Note, that *k* is usually greater for

```
SELECT TOP k Recommender, MAX (Weight)
FROM RecommenderRules
WHERE ((RuleContextAttribute1=CurrentContextAttribute1)
        OR (RuleContextAttribute1 IS NULL))
AND     ((RuleContextAttribute2=CurrentContextAttribute2)
        OR (RuleContextAttribute2 IS NULL))
AND ...
GROUP BY Recommender
ORDER BY MAX(Weight) DESC
```

Figure 10: Simplified SQL query for recommender selection strategy execution

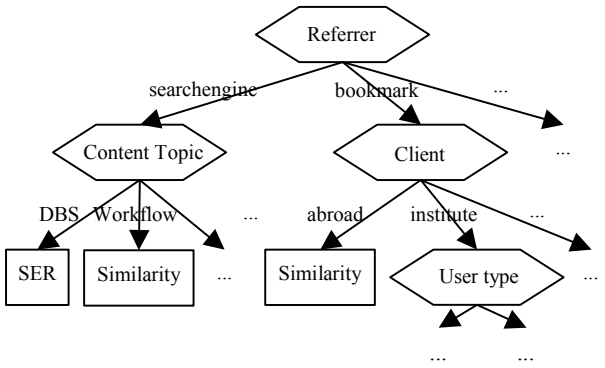

Figure 11: Fraction of decision tree constructed by the machine learning approach to generate recommender rules

recommendation rules than for recommender rules since each recommender can provide multiple recommendations per context.

The SQL query of Fig. 10 illustrates how the sketched selection process can be done for the case of recommender rules. Since there may be several matching rules per recommender, the ranking could also be based on the average instead of the maximal weight per recommender.

Example: Consider a new user who reaches the website from a search engine. If her current page belongs to the navigation category, only the first two rules in Fig. 9 match. For k=1, we select the recommender with the highest weight – SER.

The rule-based selection is highly flexible. Selection rules allow the dynamic consideration of different parts of the current context, and the weights can be used to indicate different degrees of certainty. Rules can easily be added, deleted or modified independently from other rules. Moreover, rules can be specified manually, e.g. by website editors, or be generated automatically. Another option is a hybrid strategy with automatically generated rules that are subsequently modified or extended manually, e.g. to enforce specific considerations.

6.2 Generating recommender rules

We present two approaches to automatically generate recommender rules, which have been implemented in AWESOME. Both approaches use the positive and negative feedback on previously presented recommendations. The first approach uses the aggregation and query functionality of the data warehouse to determine selection rules. The second approach is more complex and uses a machine learning algorithm to learn the most promising recommender for different context constellations.

6.2.1 Query-based top recommender

This approach takes advantage of the data warehouse query functionality. It generates recommender rules as follows:

1. Find all relevant context patterns in the recommendation fact table, i.e. context patterns exceeding a minimal support
2. For every such context pattern P do
   a) Find recommender R with highest acceptance rate A
   b) Add recommender rule P -> R [A]
3. Delete inapplicable rules

The first step ensures that only context constellations with a minimal number of occurrences are considered. This is important to avoid generalization of very rare and special situations (overfitting problem). Note that step 1 checks all possible context patterns, i.e. any of the content attributes may be unspecified, which is efficiently supported by the CUBE operator (SQL extension: GROUP BY CUBE) [GBL+95]. AWESOME is based on a commercial RDBMS providing this operator. For every such context pattern, we run a query to determine the recommender with the highest acceptance rate and produce a corresponding selection rule.

Finally, we perform a rule pruning taking into account that we only want to determine the top recommender per context. We observe that for a rule A with a more general context pattern and a higher weight than rule B, the latter will never be applied (every context that matches rule B also matches rule A, but A will be selected due to its higher weight). Hence, we eliminate all such inapplicable rules in step 3 to limit the total number of rules.

6.2.2 Machine-learning approach

Recommender selection can be interpreted as a classifier selecting one recommender from a predefined set of recommenders. Hence, machine learning (classification) algorithms can be applied to generate recommender rules. Our approach utilizes a well-known classification algorithm constructing a decision tree based on training instances (Weka J48 algorithm [WF00]). To apply this approach, we thus have to transform recommendation feedback into training instances. An important requirement is that the generation of training data must be completely automatic so that the periodic re-calculation of recommender rules to incorporate new recommendation feedback is not delayed by the need of human intervention.

The stored recommendation feedback indicates for each presented recommendation, its associated context attributes, and the used recommender whether or not the recommendation was accepted. A naïve approach to generate training instances would simply select a random sample from the recommendation fact table (Fig. 3), e.g. in the format (*context*, *recommender*, *accepted*). However, classifiers using such training instances would rarely predict a successful recommendation since the vast majority of the instances may represent negative feedback (> 98% for the sample website). Ignoring negative feedback is also no solution since the number of accepted recom-

mendations is heavily influenced by the different applicability of recommenders and not only by their recommendation quality. Therefore, we propose a more sophisticated approach that determines the number of training instances according to the acceptance rates:

1. Find all relevant feedback combinations *(context, recommender)*
2. For every combination *c* do
   a) Determine acceptance rate for c. Scale and round it to compute integer weight n_c
   b) Add instance *(context, recommender)* n_c times to training data
3. Apply decision tree algorithm
4. Rewrite decision tree into recommender rules

In Step 1, we do not evaluate context patterns (as in the previous approach), which may leave some context attributes unspecified. We only consider fully specified context attributes and select those combinations exceeding a minimal number of recommendation presentations. For each such relevant combination c *(context, recommender),* we use its acceptance rate to determine the number of training instances n_c. To determine n_c, we linearly scale the respective acceptance rate from the 0 to 1 range by multiplying it with a constant *f* and rounding to an integer value. For example, assume 50 page views for the combination of context ("returning user", "search engine", "Navigation", …) and recommender "Most frequent". If there are 7 accepted recommendations for this combination (i.e. acceptance rate 0,14) and *f*=100, we add n_c =14 identical instances of the combination to the training data. This procedure ensures that recommenders with a high acceptance rate produce more training instances than less effective recommenders and therefore have a higher chance to be predicted.

The resulting set of training instances is the input for the classification algorithm producing a decision tree. With the help of cross-validation, all trainings instances are simultaneously used as test instances. The final decision tree (see Fig. 11 for an example) can easily be rewritten into recommender rules. Every path from the root to a leaf defines a context pattern where all unspecified context attributes are set to NULL. Each leaf specifies a recommender and the rule weight is set to the relative fraction of correctly classified instances provided by the classification algorithm.

6.3 Generating and adapting recommendation rules

AWESOME also supports a dynamic one-step selection of recommendations, independent of the recommenders. This process is controlled by recommendation rules which may be continuously adapted. We introduced the underlying approach already in [GR04] but without evaluating its effectiveness. We integrated the one-step with the two-step approach in one system to allow their comparative evaluation.

The recommendation rules are created by the same set of recommenders from the recommender library and are stored in a *recommendation rule table*. When several recommenders make the same recommendation for the same context, the recommendation rule is stored in the recommendation rule table only once. In contrast to the recommender rules, we support both an offline and online adaptation of recommendation rules. During the periodic offline computation of recommendations we do not completely re-generate all recommendation rules but only add new recommendations to preserve the dynamically accumulated recommendation feedback. We also delete recommendations for deleted content and with low weights. The online adaptation of the recommendation rules dynamically adjusts the rule weights according to the recommendation feedback. The adaptation technique is described below.

We have explored two possibilities of setting the *initial weights* of newly generated recommendation rules. In the first approach, we simply set all initial weights to zero (ZeroStart). The second approach uses normalized recommender-specific weights or relative priorities for the respective contexts. When several recommenders generate the same recommendation we use the maximum of their weights. The initial weights are expected to be relevant primarily for new recommendations since the weights for presented recommendations are continuously adapted.

To adapt the shown recommendations to the users' interests, we adjust the weights of the recommendation rules in such a way, that the more useful recommendations are shown more often than less useful. This is achieved by a machine learning approach based on *reinforcement learning* [SB98]. The adaptation process evaluates whether or not presented recommendations have been accepted and adjusts the weights of the participating recommendation rules according to the obtained feedback. When some presented recommendation r is clicked, r receives positive feedback and all other recommendations shown together with r receive negative feedback. When no recommendation is clicked, all presented recommendations receive negative feedback. To prevent the weights from sliding into extreme values, the feedback values should be chosen in such a way, that an approximate equilibrium is maintained throughout the process:

$$\Sigma(\text{positive feedback}) \approx -\Sigma(\text{negative feedback})$$

For this purpose we set for a presented recommendation r

Feedback(r) = 1 if r was clicked

Feedback(r) = -*p* if r was not clicked

where *p* is the overall acceptance rate of the website, i.e. the probability that a recommendation is clicked (default p=0,01).

Name	Description
Top-Rec	Automatic 2-step strategy of section 6.2.1 (query-based)
Decision Tree	Automatic 2-step strategy of section 6.2.2 (machine learning)
Reinf. Learning	Automatic 1-step strategy of section 6.3 (reinforcement learning)
Reinf. Learning Zero	Automatic 1-step strategy of section 6.3 (reinforcement learning , with initial weights set to zero)
Manual	For search engine users, the search engine recommender is applied. Otherwise the content similarity recommender (for course material pages) or association rule recommender (for other pages) is selected.
Random	Random selection of a recommender

Table 5: Tested selection strategies

After each presentation, for every presented recommendation r we adapt its weight W(r) by the following formula [GR04]:

$$W(r) = (1-1/T)*W(r) + Feedback(r) / T \,.$$

The integer-valued parameter T (>= 1) is an aging parameter reducing the weight of the old value of the weight compared to the more recent feedback (default T=500).

7 Evalation of selection strategies

To evaluate the effectiveness of the presented selection strategies we tested them with AWESOME on the sample website introduced in Section 5.2. For comparison purposes we also evaluated one set of manually specified rules and a random recommender selection giving a total of six approaches (see Table 5). For every user session AWESOME randomly selected one of the strategies; the chosen strategy is additionally recorded in the recommendation log file for evaluation purposes. We applied the recommender selection strategies from January 1st until June 30th, 2004 and the recommendation selection strategies from April 1st until September 30th, 2004.

Table 6 shows the average number of rules per selection strategy. For the automatic recommender selection approaches the number of rules is moderate (250 – 2000) whereas the recommendation selection approaches generate significantly more rules. Nevertheless, all tested selection strategies resulted in fast execution times between 10ms – 100ms (depending on the overall utilization of the server) and thus the delays are unnoticeable.

Table 6 also shows the *average* (page view) acceptance rates and session acceptance rates for the six selection strategies. All automatic feedback-based strategies showed significantly better average quality than random.

Strategy	No. of rules	Acceptance rate	Session acceptance rate
Top-Rec	~ 2000	1.35 %	10.27 %
Decision Tree	~ 250	1.74 %	11.13 %
Reinf. Learning	~ 60000	1.92 %	11.22 %
Reinf. Learning Zero	~ 60000	1.87 %	10.35 %
Manual	5	1.97 %	12.54 %
Random	137	0.96 %	6.98 %

Table 6: Comparison of selection strategies

The decision tree strategy and both reinforcement learning strategies nearly obtain the acceptance rates of the Manual strategy. Note that the very effective strategy Manual utilizes background knowledge about the website structure and typical user groups (students, researchers) as well as evaluation results obtained after an extensive manual OLAP analysis (partially presented in Section 5.2), such as the effectiveness of the search engine recommender.

The fact that the completely automatic machine learning algorithms achieves comparable effectiveness is thus a very positive result. It indicates the feasibility of the automatic closed-loop optimization for generating recommendations and the high value of using feedback to significantly improve recommendation quality without manual effort. In our experiments the 1–step approaches based on reinforcement learning slightly outperformed the 2-step recommender-based strategies. Even the simple approach starting with weights 0 achieved surprisingly good results indicating that the initial weights are not crucial for the success of the adaptation algorithm. The effectiveness of the 1-step approach is mainly due to the online consideration of recommendation feedback providing a high responsiveness to current user behaviour.

The comparison of the two automatic recommender selection strategies shows that the machine learning approach performs much better than the query-based top recommender scheme. The decision tree approach uses significantly fewer rules and was able to order the context attributes according to their relevance. The most significant attributes appear in the upper part of the decision tree and therefore have a big influence on the selection proc-

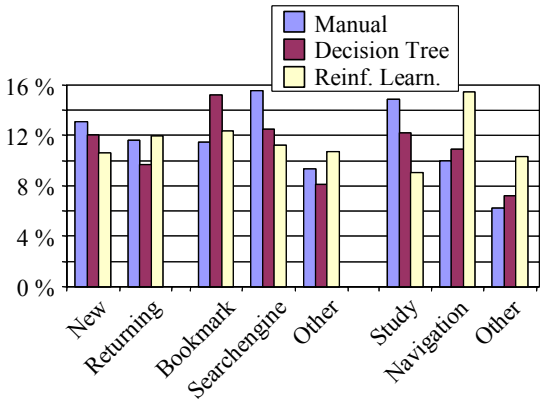

Figure 12: Session acceptance rate w.r.t. user type, referrer, and session entry page type.

ess. On the other hand, Top-Rec handles all context attributes equally and uses many more rules. So recommender selection was frequently based on less relevant attributes resulting in poorer acceptance rates.
The warehouse infrastructure of AWESOME allows us to analyze the recommendation quality of selection strategies for many conditions, similar to the evaluation of individual recommenders (Section 5.2). Figure 12 shows the session acceptance rates of three selection strategies w.r.t. user type, referrer, and entry page type, i.e. the page type of the first session page view. We observe that the manual strategy is more effective for search engine users by always applying the SER recommender to them. This helped to also get slightly better results for new users and sessions starting with an access to study material. On the other hand, both automatic approaches were more effective for users using a bookmark to reach the website. These results indicate that the automatically generated selection rules help generate good recommendations in many cases without the need of extensive manual evaluations, e.g. using OLAP tools.

The two automatic approaches show significant differences for sessions starting with a study page and a navigation page. As indicated in Fig. 5, our prototype website contains a huge number of study pages, whereas the number of navigation pages is rather small. However, navigations pages receive almost 15 times more feedback per page than study pages. Therefore the recommendation based selection approach can easily identify the best recommendations for navigation pages, but not for study pages. On the other hand, the feedback aggregation of the recommender based approach can better handle this lack of feedback, but loses information to generate better recommendations for navigation pages.

The evaluation also illustrates general capabilities of the 1-step and 2-step approaches. The usage of recommender rules makes it possible to completely define a selection strategy by a few manually specified rules. Due to the many possible recommendations this is not feasible for the 1-step approach. Still, recommendation rules can be added manually to directly influence the recommendations for single products or pages. This can be done to promote certain products or to incorporate recommendations that are not detected by any of the available recommenders.

8 Related work

An overview of previous recommendation systems and the applied techniques can be found in [JKR02], [KDA02] and [SCD+00]. [LSY03] describes the Amazon recommendation algorithms, which are primarily content (item)-based and also heavily use precomputation to achieve scalability to many users. [Bu02] surveys and classifies so-called hybrid recommendation systems which combine several recommenders. To improve hybrid recommendation systems, [SKR02] proposes to manually assign weights to recommenders to influence recommendations. [MN03] presents a hybrid recommendation system switching between different recommenders based on the current page's position within a website. The Yoda system [SC03] uses information on the current session of a user to dynamically select recommendations from several predefined recommendation lists. In contrast to AWESOME, these previous hybrid recommendation systems do not evaluate or use recommendation feedback.

[LK01] sketches a simple hybrid recommendation system using recommendation feedback to a limited extent. They measure which recommendations produced by three different recommenders are clicked to determine a weight per recommender (with a metric corresponding to our view rate). These weights are used to combine and rank recommendations from the individual recommenders. In contrast to AWESOME negative recommendation feedback and the current context are not considered for recommender evaluation. Moreover, there is no automatic closed-loop adaptation but the recommender weights are determined by an offline evaluation.

The evaluation of recommendation systems and quantitative comparison of recommenders has received little attention so far. [KCR02] monitored users that were told to solve certain tasks on a website, e.g. to find specific information. By splitting users in two groups (with recommendations vs. without) the influence of the recommendation system is measured. Other studies [GH02], [HC02] asked users to explicitly rate the quality of recommendations. This approach obviously is labor-intensive and cannot be applied to compare many different recommenders.

[GH02] and [SKK+00] discuss several metrics for recommendation quality, in particular the use of the information retrieval metrics precision and recall. The studies determine recommendations based on an offline evaluation of web log or purchase data; the precision metric, for instance, indicates how many of the recommendations were reached within the same session (thus corresponding to our view rate). In contrast to our evaluation, these studies are not based on really presented recommendations and measured recommendation feedback so that the predicted recommendation quality remains unverified.

In [HMA+02] a methodology is presented for evaluating two competing recommenders. It underlines the importance of such an online evaluation and discusses different evaluation aspects. Cosley et. al. developed the REFEREE framework to compare different recommenders for the CiteSeer website [CLP02]. Click metrics (e.g., how often a user followed a link or downloaded a paper), which are similar to the acceptance rates used in our study, are used to measure recommendation quality.

9 Summary

We presented AWESOME, a new data warehouse-based website evaluation and recommendation system. It allows the coordinated use of a large number of recommenders to automatically generate website recommendations. Recommendations are dynamically determined by a flexible rule-based approach selecting the most promising recommender / recommendations for the respective context. AWESOME supports a completely automatic generation and optimization of selection rules to minimize website administration overhead and quickly adapt to changing situations. This optimization is based on a continuous measurement of user feedback on presented recommendations. To our knowledge, AWESOME is the first system enabling such a completely automatic closed-loop website optimization. The use of data warehouse technology and precomputation of recommendations support scalability, high data quality and fast web access times.

We presented a simple but general recommender classification. It distinguishes eight types of recommenders based on whether or not they consider input information on the current content, current user and users history. To evaluate the quality of recommendations and recommenders, we proposed the use of several acceptance rate metrics based on measured recommendation feedback. We used these metrics for a detailed comparative evaluation of different recommenders and different recommendation selection strategies for a sample website.

We have described and evaluated several rule-based strategies for dynamically selecting the most promising recommender or recommendations for a given context. We differentiated between 1-step and 2-step approaches depending on whether recommendations are directly selected or whether the most promising recommenders are determined first. Our results so far indicate that in both cases the use of machine learning is very effective for considering recommendation feedback. For a direct (1-step) selection of recommendations the described use of reinforcement learning effectively allows an online adaptation of individual recommendations regardless of the used recommenders. For automatic recommender selection we presented a powerful decision tree approach. The proposed policy is able to automatically determine suitable training data so that its periodic re-execution to consider new feedback does not require human intervention.

We have begun to adopt AWESOME to additional websites, in particular e-shops, to further verify and fine-tune the presented approach. We also plan to explore domain-specific recommendation opportunities such as selecting the best recommender for product bundling (cross-selling).

Acknowledgements

We thank Robert Lokaiczyk for his help with the implementation. The first two authors are funded by the German Research Foundation within the Graduiertenkolleg "Knowledge Representation".

References

[Bu02] Burke, R.: *Hybrid Recommender Systems: Survey and Experiments*. User Modeling and User-Adapted Interaction 12(4), 2002

[CMS99] Cooley, R., Mobasher, B., Srivastava, J.: *Data preparation for mining world wide web browsing patterns.* Knowledge and Information Systems. 1(1), 1999

[CLP02] Cosley, D., Lawrence, S., Pennock, D. M.: *REFEREE: An open framework for practical testing of recommender systems using ResearchIndex*. Proc. 28th VLDB conf., 2002

[GBL+95] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. *Data cube: A relational aggregation operator generalizing groupby, cross-tab, and sub-total*. Proc. of the 12th EEE International Conference on Data Engineering (ICDE), 1995

[GH02] Geyer-Schulz, A., Hahsler, M.: *Evaluation of Recommender Algorithms for an Internet Information Broker based on Simple Association rules and on the Repeat-Buying Theory*. Proc. of ACM WebKDD Workshop, 2002

[GR04] Golovin, N., Rahm, E.: *Reinforcement Learning Architecture for Web Recommendations*. Int. Conf. on Information Technology (ITCC), Las Vegas, 2004

[HC02] Heer, J., Chi, E. H.: *Separating the Swarm: Categorization Methods for User Sessions on the Web*. Prof. Conf. on Human Factors in Computing Systems, 2002

[HMA+02] Hayes, C., Massa, P., Avesani, P., Cunningham, P.: *An on-line evaluation framework for recommender systems*. Proc. of Workshop on Personalization and Recommendation in E-Commerce, 2002

[JKR02] Jameson, A., Konstan, J., Riedl, J.: *AI Techniques for Personalized Recommendation*. Tutorial at 18th National Conf. on Artificial Intelligence (AAAI), 2002

[KCR02] Kim, K., Carroll, J. M., Rosson, M. B.: *An Empirical Study of Web Personalization Assistants: Supporting End-Users in Web Information Systems*. Proc. IEEE 2002 Symp. on Human Centric Computing Languages and Environments, 2002

[KDA02] Koutri, M., Daskalaki, S., Avouris, N.: *Adaptive Interaction with Web Sites: an Overview of Methods and Techniques.* Proc. 4th Int. Workshop on Computer Science and Information Technologies (CSIT), 2002

[KM00] Kimball, R., Merz, R.: *The Data Webhouse Toolkit – Building Web-Enabled Data Warehouse*. Wiley Computer Publishing, New York, 2000

[KMT00] Kushmerick, N., McKee, J., Toolan, F.: *Toward zero-input personalization: Referrer-based page recommendation*. Proc. Int. Conf. on Adaptive Hypermedia and Adaptive Web-based Systems, 2000

[LK01] Lim, M., Kim, J.: *An Adaptive Recommendation System with a Coordinator Agent*. In Proc. 1st Asia-Pacific Conference on Web Intelligence: Research and Development, 2001

[LSY03] Linden, G., Smith, B., York, J.: *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*. IEEE Distributed Systems Online 4(1), 2003

[MDL+02] Mobasher, B., Dai, H., Luo, T., Nakagawa, M.: *Discovery and Evaluation of Aggregate Usage Profiles for Web Personalization*. Data Mining and Knowledge Discovery, Kluwer, 6 (1), 2002

[MN03] Mobasher, B., Nakagawa, M.: *A Hybrid Web Personalization Model Based on Site Connectivity*. Proc. ACM WebKDD Workshop, 2003

[SB98] R.S. Sutton, A.G. Barto: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 1998

[SC03] Shahabi, C., Chen, Y.: *An Adaptive Recommendation System without Explicit Acquisition of User Relevance Feedback*. Distributed and Parallel Databases 14(2), 2003

[SCD+00] Srivastava, J., Cooley, R., Deshpande, M., Tan, P-T.: *Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data*. SIGKDD Explorations, (1) 2, 2000

.[SKK+00] Sarwar, B., Karypis, G., Konstan, J. , Riedl, J.: *Analysis of recommendation algorithms for e-commerce*. Proc. of ACM E-Commerce, 2000

[SKR01] Schafer, J.B., Konstan, J. A., Riedl, J.: *Electronic Commerce recommender applications.* Journal of Data Mining and Knowledge Discovery, 5 (1/2), 2001

[SKR02] Schafer, J. B., Konstan, J. A., Riedl, J.: *Meta-recommendation systems: user-controlled integration of diverse recommendations*. Proc. 11th Int. Conf. on Information and Knowledge Management (CIKM), 2002

[SMB+03] Spiliopoulou, M., Mobasher, B., Berendt, B., Nakagawa, M.: *A Framework for the Evaluation of Session Reconstruction Heuristics in Web Usage Analysis*. INFORMS Journal of Computing, Special Issue on Mining Web-Based Data for E-Business Applications, 15 (2), 2003

[TH01] Terveen, L., Hill, W.: *Human-Computer Collaboration in Recommender Systems*. In: Carroll, J. (ed.): Human Computer Interaction in the New Millenium. New York: Addison-Wesley, 2001

[TK00] Tan, P., Kumar, V.: *Modeling of Web Robot Navigational Patterns*. Proc. ACM WebKDD Workshop, 2000

[WF00] Witten, I.H., Frank, E.: *Data Mining. Practical Machine Learning Tools and techniques with Java implementations*. Morgan Kaufmann. 2000