Reinforcement Learning Architecture for Web Recommendations

Nick Golovin, Erhard Rahm
Univ. of Leipzig, Germany
http://dbs.uni-leipzig.de

Abstract

A large number of websites use online recommendations to make web users interested in their products or content. Since no single recommendation approach is always best it is necessary to effectively combine different recommendation algorithms. This paper describes the architecture of a rule-based recommendation system which combines recommendations from different algorithms in a single recommendation database. Reinforcement learning is applied to continuously evaluate the users' acceptance of presented recommendations and to adapt the recommendations to reflect the users' interests. We describe the general architecture of the system, the database structure, the learning algorithm and the test setting for assessing the quality of the approach.

1. Introduction

Web recommendations have become an inseparable part of many websites. Large, medium-size and small companies use web recommendations to increase the usability of their websites and customer satisfaction. Many techniques have been developed to solve the task of generating recommendations, taking into account various types of information (e.g. product or user characteristics, buying history, …) and using different statistical or data mining approaches ([BU02],[JKR02]). Since no single recommendation technique is always best it has become apparent that several techniques need to be effectively combined to increase t he quality of recommender systems [BU02].

We present a new approach for combining different recommendation techniques by utilizing a central database for storing all recommendations and applying a reinforcement learning algorithm to dynamically adapt and optimize the recommendations based on implicit user feedback. The main incentives for designing such an architecture are:

1) To allow reactive adaptation of the system to changes in user behavior and website content.
2) To provide a platform for flexible selection of recommendations taking into account that effective websites should restrict the number and types of presented recommendations.
3) To make the recommendation system generic and extendable. The system should be applicable to a wide range of website types and allow addition of new recommendation algorithms.

The presented approach has been implemented in a prototype which is deployed at a commercial online retail store.

In the next section, we present an overview of our recommendation system architecture. We then detail the structure of recommendation rules and the rule database (Section 3), our current set of recommendation algorithms (Section 4), the strategy for dynamically selecting recommendations (Section 5) and the feedback-based learning strategy (Section 6). In Section 7, we describe the prototype and the test setting for assessing the quality of our recommendations. A brief overview of related work is given in Section 8.

2. Architecture overview

To achieve the above objectives, we use the recommendation system architecture shown in Fig. 1. The main components of the system are:

- The *website* – interacts with the web user, presents recommendations and gathers the feedback.
- *Web warehouse* – stores information about the content of the website (e.g., products and product catalog), users, and the usage logs generated by the web server or the application server.
- Set of *recommender algorithms* – the recommender algorithms generate recommendations using data from the web warehouse. Recommendations can also be created by a *human editor*.
- *Recommendation rule database* – stores the recommendations.
- *Learning module* – refines the recommendation database based on the feedback obtained from the website.

As indicated in the figure we use two feedback loops for making recommendations. The first loop is periodically executed and involves calculating recommendation candidates by several recommendation algorithms utilizing more static information on the content as well as recent usage information from the web warehouse. The output of the algorithms is combined in one recommendation database which is used to dynamically select recommendations. In the second feedback loop we continuously gather and evaluate user reactions on presented recommendations. The learning module uses this information to refine the recommendations in the database and thus to immediately impact the selection of future recommendations.

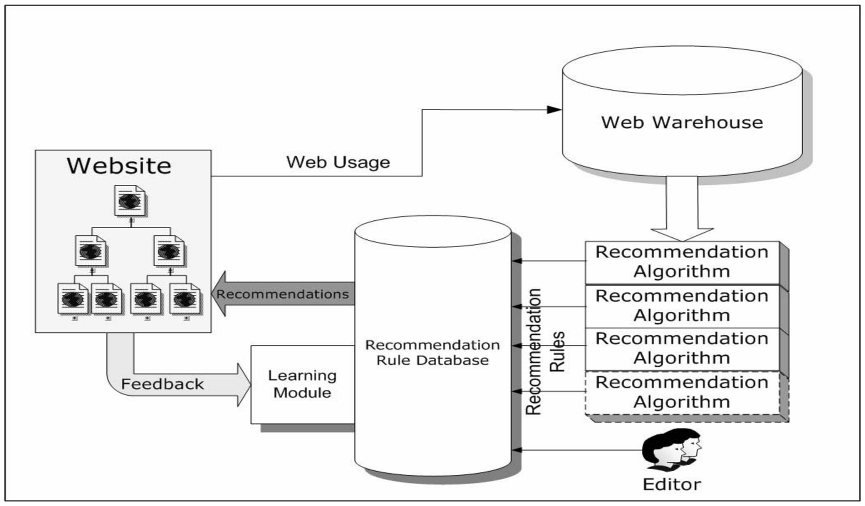

Figure 1: System architecture

3. Recommendation rules

3.1 Rule structure

Web recommendations are usually chosen depending on the current state of interaction between the user and the website as well as on the state of the environment. We use *recommendation rules* to specify which recommendations should be presented in which situation. Recommendation rules can be generated automatically by a recommendation algorithm (also called *recommender*) or come from a human editor. Since different recommenders produce recommendations which depend on different parameters, combining them into one data structure is not a straightforward task.
Recommendation rules are of the following basic structure:

Context{a1,a2,a3...} -> RecommendedContent.

Here, *context* is a vector of values a1,a2,a3, … from different *dimensions* describing the current state of interaction between the user and the website. Relevant dimensions include content (e.g. to indicate the current URL or product) and user. *RecommendedContent* is the pointer to the content being recommended (e.g. recommended product or URL). Recommended content can be either a part of our website or external to it.

In our approach we use refined rules of the following format:
CurrentContent, CurrentUser, CurrentTime -> RecomendedContent, Weight.

That is we describe the current context to which a recommendation rule applies by three dimensions: content, user and time. The *weight* metric is used for selecting recommendations especially when several rules apply for a given context. It is initially set by a recommendation algorithm and can be adjusted by the learning module.

Examples of recommendation rules for product content are:

ProductID="ECD00345"-> "ECD000884", 0.8
ProductID="ECD00345", UserCountry="DE"-> "ECD000345D", 0.95
ProductCategoryID=32867-> "ECD000890", 0.3
UserDomain LIKE '%.edu' OR UserDomain LIKE '%uni-%' -> "ECD000320", 0.7

Note that the context may only be partially specified. In the examples, the time dimension is unspecified so that the rules apply irrespective of when a user accesses the website. A rule with empty context is always applicable.

A main flexibility of our rule model is that we allow each context dimension be specified by a variety of attributes for which the current value can be determined at the time of a user interaction, for example

Content
- ProductID
- Product category (1st level, 2nd level, …)
- Author (Books)
- Platform (Software),…

User
- User ID
- User country
- User browser, operating system, language
- User group,…

Time
- Day of week
- Time of the day
- Season of the year
- Specific date or time interval,…

Obviously, the choice of suitable attributes and their granularity depends on the type of website (for example news, ecommerce, educational) and on the content of the website itself. We especially support an ontological structuring of the content, user and time dimensions, e.g. in the form of a product catalog or a hierarchy of user groups. To

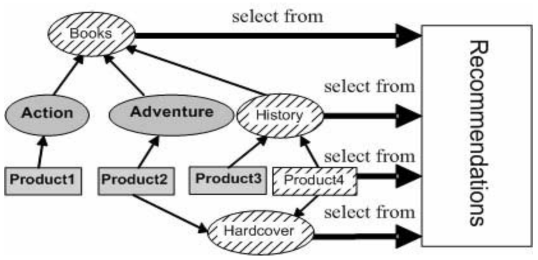

Figure 2. Example of a concept graph for Content dimension

be generic we represent such ontological structures in the form of acyclic *concept graphs,* i.e. we are not limited to hierarchical relationships. We use directed edges to point from more specific concepts to more general concepts, from subcomponents to aggregated components, etc. Fig. 2 shows an example of such a graph for the content dimension. Recommendations can be assigned to any node in such a graph. When selecting recommendations, the ontology is traversed from less general to more general nodes, and for each node the corresponding recommendations are selected from the database (see Section 5).

3.2 Recommendation rule database structure

The recommendation rules are stored in a relational database. The schema is shown in Fig.3. The rules are maintained in the central table *Rules*. Some additional information is stored together with the rules, such as the number of times the recommendation was presented (Npresented), number of times the recommendation was clicked (Nclicked), the recommendation type (the recommender which generated the rule), and the creation time of the rule. The attributes ContentNode, UserNode and TimeNode are foreign keys to uniquely identify values for the respective context dimension (null values are allowed to cover partially specified context information). RecomNode identifies the recommended content and thus also refers to the ContentNodes table.

The context dimensions are stored in pairs of tables ContentNodes/ContentArcs, UserNodes/UserArcs, TimeNodes/TimeArcs to allow the representation of concept graphs. The node tables contain information on all relevant content items (products or URLs), users and time events that may occur in the context or recommendation part of a rule. Each such item is described by a so-called match rule, i.e. a predicate on attributes of the dimension (e.g., ProductID=”ECD00345” for the content dimension, UserDomain LIKE ‘%.edu’ for the user dimension). Most match rules are automatically determined and simply use equality conditions. Manually specified match rules may be more complex. The fields MatchPref can be used to specify the order (priority) in which the matching rules are applied. The recommended content is represented by the fields RecomLink and RecomDescription in ContentNodes.

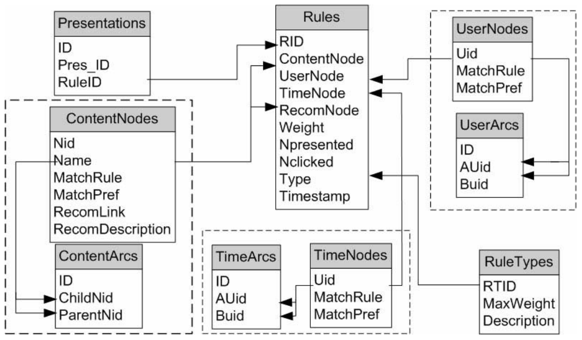

Figure 3. Recommendation rule database structure

The table *RuleTypes* provides some information on the used recommenders (description and their maximal value for the initial weight of a rule). The recommendations presented to users are logged in the table *Presentations*.

4. Recommender algorithms

Our architecture does not limit the number of recommender algorithms. Every recommender produces recommendations for the rule database with an initial weight from the interval [0 .. MaxWeight$_i$], where MaxWeight$_i$ is a recommender-specific value. The initial weights are set according to recommender-specific heuristics. As shown in [SB98] the specific settings should not adversely affect the convergence of reinforcement learning.

In the prototype implementation, we determine product recommendations with the following recommenders:

1. *Content similarity.* This recommender determines for each products (content node) the N most similar products. We determine the similarity by using the TF/IDF score on product descriptions. The initial weight for each rule is the similarity score, normalized to [0..MaxWeight].
2. *Top-N.* The N products which received the most clicks over a certain period of time are recommended. The context remains unspecified, i.e. these recommendations apply to all products and users. The initial weight for each rule is the relative frequency of clicks.
3. *Top-N for category.* For each product category the N most frequently clicked products from the same category are recommended. The initial weight for each rule is the relative frequency.
4. *Sequence patterns.* Products most often succeeding other products in the same user session are recommended to them. Scaled confidence of such association rules is used as the initial weight.
5. *Item-to-Item collaborative filtering.* Products, which most often appear together in a user’s basket, are recommended to each other. The initial weight is set to the normalized frequency of joint appearance.

These algorithms generate recommendations either based on the past user behavior (top-N, sequence patterns, collaborative filtering) or on website content analysis (content similarity).

When several recommenders produce an identical recommendation rule, only the rule with the maximal recommender-specific initial weight is stored in the database. If a rule is already present in the database, even with a lower weight, the new rule is discarded.

5. Selecting recommendations

Currently, to present recommendations for a given context we use a simple selection strategy where the three context dimensions (content, user and time) are considered to be equally important. In order to avoid overwhelming the user with too many uninteresting recommendations we only select a specific number, *n*, of promising recommendations.

Recommendation selection comprises the following three steps:

1. For the current context we determine the best matching CurrentContentNode, CurrentUserNode and CurrentTimeNode values using matching rules. The matching rules are selected from the dimension tables and are applied to the current context in the order determined by the matching preference.

2. The following database (SQL) query is used to select all recommendations which either fully or partially match the three context components:

```
SELECT TOP N RecomNode From Rules  WHERE
        (ContentNode=CurrentContentNode OR
        ContentNode is NULL)  AND
        (UserNode=CurrentUserNode OR
        UserNode is NULL)  AND
        (TimeNode=CurrentTimeNode OR
        TimeNode is NULL)
WHERE Weight>=Threshold
ORDER BY Weight DESC
```

Threshold is a lower limit for the weight of recommendations to avoid presenting recommendations of insufficient quality. The threshold value is chosen for each website individually, based on the owner's preferences.

3. If the query does not produce the needed number of recommendations, then the current context is iteratively extended using the concept graphs of the dimensions, as illustrated in Fig 2. In each step, one of the dimension nodes - CurrentContentNode, CurrentUserNode or CurrentTimeNode, is changed to its parents and the selection process (step 2) is repeated, until the needed number of recommendations is reached. For the context switching strategy in our prototype, ContentNode has the highest preference and TimeNode the lowest. In the example of Fig. 2, if we would not obtain enough recommendations for CurrentContentNode "Product4" we would in the next iteration search recommendations for categories "History" or "Hardcover", etc.

6. Learn rules from feedback

We use a learning approach to continuously evaluate feedback from presented recommendations. The goal is to adaptively increase the weights of successful recommendations and to decrease the weight of unsuccessful recommendations. This should be done in a way so that the approach can quickly react to significant changes in user interests but without overreacting to short-term fluctuations. To avoid the high effort and difficulty of a training phase we need an unsupervised learning approach that learns automatically from the users' reactions. This also allows adding new recommendations to the rule database at any time.

The learning process evaluates all recommendation trials, called presentations. In a presentation, the user is shown several recommendations selected from the recommendation rule database. The presented recommendations are also logged in a temporary table.

A naïve approach to determine the weight of recommendations would be to just use the number of clicks divided by the number of presentations. However this approach is too greedy and subject to self-amplification – already learned recommendations would not allow other recommendations to be presented.

To better adapt the weight of recommendations, we use a simple reinforcement learning algorithm of [SB98] called exponential, recency-weighted average. It allows us to dynamically balance the need to show the best recommendations (exploitation) and the need to learn how good they are (exploration). After a presentation, the system determines which recommendations r participated in it and updates their weights Q(r) according to the formula:

$$Q(r) = (1-1/T)*Q(r) + Feedback(r) \;/ T.$$

In this formula, T is the size of the sampling window used for learning. The latest trials have the most impact on the resulting weight value while the contribution of past trials' decreases exponentially (as the method's name exponential, recency-weighted average suggests). The feedback in the formula is determined as follows. When some recommendation r in a presentation is clicked, r receives positive feedback, all other recommendations receive negative feedback. When no recommendation is clicked, after a predefined timeout all participating recommendations receive negative feedback.

The specific values of the feedback should be chosen according to the goals and specifics of the website. For our prototype, we use the following feedback values:

1	if r is a clicked (successful) recommendation

-1/(*n*-1)	if some other recommendation is clicked (*n* is the number of recommendations shown simultaneously in a presentation).
-*p*	if no recommendation is clicked (*p* is the overall probability for a website that a web page is reached because of a recommendation).

The second value is motivated by the fact that when *n* recommendations are simultaneously shown to the user, each recommendation has *n*-1 times larger probability of getting negative than positive feedback. Hence, negative feedback is divided equally between all not clicked recommendations. The last value ensures that all non-clicked recommendations are equally decreased according to the average probability that recommendations are used at all.

7. Prototype

The prototype of the system is implemented on a small commercial online software store (http://www.softunity.com, approximately 5000 page views per day). Our approach is used to automatically select and present 5 recommendations (*n*=5) on each product detail page. Some facts concerning the prototype are shown in the table below.

Number of products:	~2182
Number of topics:	~ 207
Number of ContentNodes:	~ 8300
Number of Rules:	~ 35970

The prototype uses a mysql installation for the recommendation rule database and the web warehouse. All recommenders, the learning module as well as the website itself, are implemented using the PHP scripting language.

To assess the effectiveness of the proposed approach, we plan to compare our learning approach for determining recommendations with the use of randomly generated recommendations. To do so we use either method for 50% of the presentations by changing the selection strategy for recommendations for every presentation between intelligent (weight-based) and random. The latter mode is achieved by a slight modification of the SQL selection query (Section 5) by using the clause "ORDER BY RAND()") instead of "ORDER BY Weight DESC". Comparing the recommendation usage for the two cases should indicate whether the proposed approach brings an increase in user acceptance.

8. Related work

A recent survey provides a detailed overview of the research on web recommendation algorithms and their combination [Bu02]. According to their classification, the presented system belongs to the category of so-called "mixed" hybrid recommendation systems, i.e. systems where recommendations from different algorithms are presented together. [NM03] propose to choose recommender algorithms based on the degree of connectivity for a given web page. [MB97] also use a central repository for recommendations but rank the recommendations based on explicit user feedback. One of the authors is investigating another feedback-oriented approach for making dynamic web recommendations which focuses on the automatic selection of recommenders to use (instead of selecting individual recommendations) [TR03]. The distinguishing feature in this work compared to previous approaches is the use of reinforcement learning as a flexible and reactive technique for learning and combining recommendations from several recommendation algorithms.

9. Summary and outlook

This paper describes the architecture of a novel hybrid recommendation system. Our approach uses multiple techniques to generate recommendations and employs reinforcement learning to refine their quality.

We plan to comprehensively evaluate the approach and use the results for improvement, e.g. with respect to the selection and learning strategy or to extend our library of recommenders. For instance, the ontological knowledge represented in concept graphs should also be useful for recommenders to determine the similarity between products or users.

10. References

[AG03] S. Acharyya, J. Ghosh: *Context-Sensitive Modeling of Web-Surfing Behavior using Concept Trees.* Proc. WebKDD, 2003

[Ba97] M. Balabanovic: *An Adaptive Web Page Recommendation Service.* CACM, 1997

[BS03] S. Baron, M. Spiliopoulou: *Monitoring the Evolution of Web Usage Patterns*. Proc. ECML/PKDD, 2003

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

[JKR02] A. Jameson, J. Konstan, J. Riedl: *AI Techniques for Personalized Recommendation*. Tutorial presented at AAAI, 2002

[LSY03] G. Linden, B. Smith, and J. York: *Amazon.com Recommendations: Item-to-Item Collaborative Filtering.* IEEE Internet Computing. Jan. 2003

[NM03] M. Nakagawa, B. Mobasher: *A Hybrid Web Personalization Model Based on Site Connectivity.* Proc. 5th WEBKDD workshop, Washington, DC, USA, Aug. 2003

[SKKR00] B. Sarwar, G. Karypis, J. Konstan, J. Riedl: *Analysis of Recommendation Algorithms for E-Commerce*. Proc. of ACM E-Commerce, 2000

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

[TR03] A. Thor, E. Rahm. *Data-Warehouse-basierte Architektur für adaptive Online-Recommendations* (in German). Proc. LIT2003, Leipzig, 2003