

 Open access • Book Chapter • DOI:10.1007/3-540-44751-2\_31

## Yoda: An Accurate and Scalable Web-Based Recommendation System

— [Source link](#) 

Cyrus Shahabi, Farnoush Banaei Kashani, Yi-Shin Chen, Dennis McLeod





**Institutions:** University of Southern California

**Published on:** 05 Sep 2001 - Cooperative Information Systems

**Topics:** Collaborative filtering, Yoda, Recommender system and Locality-sensitive hashing

Related papers:

- [Implicit interest indicators](#)
- [Extending a Web Browser with client-side mining](#)
- [System and method for behavioral model clustering in television usage, targeted advertising via model clustering, and preference programming based on behavioral model clusters](#)
- [Personalized ontology for web search personalization](#)
- [User-profile based web page recommendation system and user-profile based web page recommendation method](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/yoda-an-accurate-and-scalable-web-based-recommendation-2k13hkl9yl>

# Yoda: An Accurate and Scalable Web-based Recommendation System<sup>\*</sup>

Cyrus Shahabi, Farnoush Banaei-Kashani, Yi-Shin Chen, and Dennis McLeod

Department of Computer Science, Integrated Media Systems Center, University of  
Southern California, Los Angeles, CA 90089-2561, USA  
[shahabi, banaeika, yishinc, mcLeod]@usc.edu

**Abstract.** Recommendation systems are applied to personalize and customize the Web environment. We have developed a recommendation system, termed *Yoda*, that is designed to support large-scale Web-based applications requiring highly accurate recommendations in real-time. With *Yoda*, we introduce a hybrid approach that combines collaborative filtering (CF) and content-based querying to achieve higher accuracy. *Yoda* is structured as a tunable model that is trained off-line and employed for real-time recommendation on-line. The on-line process benefits from an optimized aggregation function with low complexity that allows real-time weighted aggregation of the soft classification of active users to pre-defined recommendation sets. Leveraging on *localized* distribution of the recommendable items, the same aggregation function is further optimized for the off-line process to reduce the time complexity of constructing the pre-defined recommendation sets of the model. To make the off-line process scalable furthermore, we also propose a filtering mechanism, *FLSH*, that extends the *Locality Sensitive Hashing* technique by incorporating a novel distance measure that satisfies specific requirements of our application. Our end-to-end experiments show while *Yoda*'s complexity is low and remains constant as the number of users and/or items grow, its accuracy surpasses that of the basic nearest-neighbor method by a wide margin (in most cases more than 100%).

## 1 Introduction

The World Wide Web is emerging as an appropriate environment for business transactions and user-organization interactions, because it is convenient, fast, and cheap to use. Witness the enormous popularity of e-Commerce and e-Government applications. However, since the Web is a collection of semi-structured and structured information resources, Web users often suffer from information overload. To remedy this problem, recommendation systems are appropriate to personalize and customize the Web environment. Mass customization of the Web facilitates access to the Web-based information resources and

---

<sup>\*</sup> In the Proceedings of the Sixth International Conference on Cooperative Information Systems , Trento, Italy , September 2001

realizes the full economic potential of the Web. Recommendation systems have achieved widespread success in e-Commerce and e-Government applications.

Various statistical and knowledge discovery techniques have been proposed and applied for recommendation systems. To date, *collaborative filtering (CF)* is the most successful approach employed in recommendation systems [5, 9]; it is used in many of the real recommendation systems on the Web. Collaborative filtering works based on the assumption that if user  $x$  interests are similar to user(s)  $y$  interests, the items preferred by  $y$  can be recommended to  $x$ . With collaborative filtering, the system strives to predict unknown interests of the user based on similarity of the user interests with those of other users.

There are two fundamental challenges for CF-based recommendation systems. The first challenge is to improve the scalability of the system. For modern e-Commerce applications such as eBay.com<sup>TM</sup> and Amazon.com<sup>TM</sup>, a recommendation system should be able to provide recommendations in real-time while the number of both users and items exceed millions. There are two families of collaborative filtering algorithms: *memory-based* algorithms [1, 5], and *model-based* algorithms [8, 9]. With memory-based algorithms, the entire recommendation process is generally an on-line process. On the other hand, with model-based algorithms recommendation is performed using an aggregated model. Thus, the time-consuming part of the recommendation process is performed off-line, leaving the on-line process with reasonably low time complexity. Hence, model-based approaches are preferred in large-scale applications, where millions of recommendations are to be generated in real-time. The second challenge for CF-based recommendation systems is to improve the accuracy of the recommendations to be as efficacious as possible. False recommendations, such as those that miss recommendable items (termed *false negatives*), or those that include non-recommendable items (termed *false positives*), seriously affect efficacy of the recommendation systems and must be avoided. Problems such as *sparsity* and *synonymy* further complicate enhancement of the accuracy (see Section 2). Accuracy is usually sacrificed to achieve lower space and time complexity.

In this paper, we present our scalable and accurate CF-based recommendation system, termed *Yoda*, which is designed to support large-scale Web-based applications requiring highly accurate recommendations in real-time. With *Yoda*, we introduce a tunable model-based approach that can strike a compromise between scalability and accuracy based on the specific application requirements. With our model, not only we optimize the *on-line* recommendation process, but also we propose techniques to reduce time complexity of generating the model during the *off-line* process. During the online process, observing the probable multi-nature behavior of each single user, first we *softly* classify an active user based on typical patterns of users behaviors. Subsequently, we perform a weighted aggregation on *pre-defined* recommendations associated to each class of behaviors to generate the *soft* recommendations for the user. Applying an accurate and tunable aggregated model, *FM*, and a customized similarity measure, *PPED*, accurate classification of users is performed in real-time [13]. To expedite the aggregation step, we propose an optimized fuzzy aggregation function that

reduces time complexity of the aggregation from  $O(\|I\| \times \|P\|)$  to  $O(\|I\|)$  (where  $I$  and  $P$  are the size of the item set and size of the property set associated with each item, respectively).

On the other hand, with large-scale applications since both items presented within the Web-site and user behaviors are changing rapidly, the model itself must be updated/regenerated frequently, e.g. once a day. We apply the same aggregation function to generate the pre-defined recommendations, *cluster wish-lists*, for each class of users during the off-line process. In this case, we view each cluster wish-list as a sub-system. Hence, we can incorporate an optimized aggregation algorithm proposed by Fagin [2] to further reduce the time complexity of the aggregation function to  $O(\|N\|)$ , where  $N$  is a small constant parameter selected during system tuning. Furthermore, we take advantage of the localized distribution of the items and propose an extended version of the *LSH* technique [15], termed *FLSH*, to avoid considering the items that do not show sufficient proximity to the required recommendations, while generating the cluster wish-lists. The original LSH is appropriate for fast item retrieval (sublinear to  $\|I\|$ ) in high-dimensional spaces (large  $\|P\|$ ), which makes it the optimum choice for cluster wish-list generation. However, the distance measure used in LSH, Hamming distance, does not satisfy requirements of our distance space. With FLSH, we introduce and incorporate our own distance measure to customize LSH for our application.

In sum, the major contribution of this paper is a novel content-based ranking method for CF-based models that captures associations between items. Consequently, our model considers both association between users - with collaborative filtering and items - with content-based filtering; this is to address the synonym problem of the pure CF-based approaches, and achieve higher accuracy even very large-scale applications.

## 2 Related Work

Recommendation systems are designed either based on *content-based filtering* or *collaborative filtering*. Content-based filtering approaches are derived from the concepts introduced by the Information Retrieval (IR) community. Content-based filtering systems are usually criticized for two weaknesses, **content limitation** (e.g., IR methods can only be applied to a few kinds of content) and **over-specialization**. The collaborative filtering (CF) approach remedies these two problems. Typically, CF-based recommendation systems do not use the actual content of the items to evaluate them for recommendation. Moreover, since other user profiles are also considered, user can explore new items which are in terms with avoiding the over-specialization problem. The nearest-neighbor algorithm is the earliest CF-based algorithm used in recommendation systems [5]. With this algorithm, the similarity between users is evaluated based on their rating data, and the recommendation is generated considering the items visited by nearest neighbors of the user. In its original form, CF-base recommendations

suffer from the problems of, **scalability**, **sparsity**, and **synonymy** (i.e., latent association between items is not considered for recommendations.)

In order to alleviate or even eliminate these problems, more recently, researchers introduced a variety of different techniques into collaborative filtering systems, such as *content analysis* [4] for avoiding the synonymy and sparsity problems, *categorization* [7] to alleviate the synonymy and sparsity problems, *bayesian network* [9, 8] for lightening the scalability problem, *clustering* [9] to lessen sparsity and scalability problems, and Singular Value Decomposition (SVD) [6, 10] to ease all three problems to a certain limit.

With Yoda, we introduce an integrated model which brings together the advantages of *model based*, *clustering*, *content analysis*, and *CF* approaches. As a result, we reduce the time complexity through model based and clustering approaches, alleviate the synonymy problem with content analysis method, and address the sparsity problem by implicit identification of the users interests [11, 12].

### 3 Overview

The objective of a Web-based recommendation system can be stated as follows:

**Problem Statement** Suppose  $I = \{i | \text{"i" is an item}\}$  is the set of items presented in a Web-site, termed the *item-set* of the Web-site, and  $x$  is a user interactively navigating the Web-site. The recommendation problem is defined to find a ranked list of the items  $I_x$ , termed *wish-list*, in which items in  $I_x$  are ranked based on  $x$  interests.

To provide a wish-list for a user, generally a CF-based recommendation system goes through 2 steps/phases:

1. User Classification: During this phase, data about user interests are acquired and employed to classify the user.
2. Ranking the Items: In this phase, the predicted user interests are applied to rank and order the items in the item-set to provide the final wish-list for the user.

To illustrate the processing flow of Yoda, consider Figure 1. Suppose music CDs are the items presented by a given web-site. For a music CD, for instance proximity to various styles of music such as Classic, Rock, Pop, etc. can be considered as different properties of the item. Yoda is to provide each active user of the site with purchase recommendations that are compatible with the style(s) of music the user likes. To generate the recommendations, during an off-line process Yoda uses fuzzy terms such as *Low*, *Medium*, and *High* to evaluate correspondence of each CD with a defined set of properties, e.g.  $\{Rock, Pop, Indie, Heavy-Metal, Jazz\}$ . For example, a CD can be labeled with  $Rock = High$ ,  $Pop = Low$ ,  $Indie = Low$ ,  $Heavy-Metal = Medium$ , and  $Jazz = Low$ . Also,

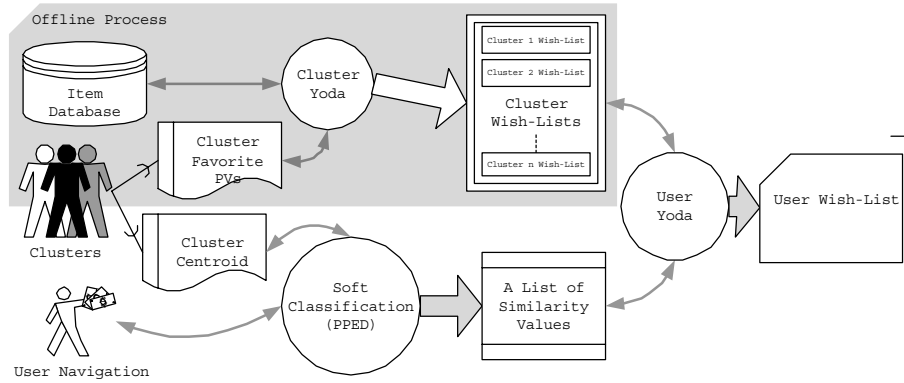


Fig. 1. Architecture of Yoda

during the off-line process, Yoda identifies similar groups of users by clustering user sessions from a training set, and learns typical pattern of users interests in each cluster, termed *favorite property values (favorite PVs)* of the cluster, by taking vote among items browsed by users belonging to the cluster. For instance, favorite PVs of a cluster can be *Rock = High, Pop = Low, Indie = Medium, Heavy-Metal = High, and Jazz = Low*. Then, Yoda applies favorite PVs of each cluster as a measure to rank items in the item-set and predict a list of recommendations, termed *wish-list*, for each cluster. Later, during the on-line recommendation process first an active user is softly classified with clusters of users based on his/her partial navigation pattern. Thereafter, Yoda uses the classification factors to generate the final wish-list for the active user by weighted aggregation of the cluster wish-lists. Items in the user wish-list are now ranked based on preferences of the corresponding user.

## 4 System Design

In this section, we provide a detailed description of Yoda's components. Since phase I is performed based on our previous work [12, 13], here we elaborate more on phase II of Yoda.

### 4.1 Phase I - User Classification

Yoda uses the client-side tracking mechanism we proposed in [12] to capture view-time, hit-count, and sequence of visiting the web-pages that particularly provide information about the items presented within a web-site. These features are applied to infer users interests in items. To analyze these features and infer the user interests, Yoda employs a flexible and accurate model we introduced in [13], the *Feature Matrices (FM)* model. FM is a set of variable-order hyper-cube data structures that can represent various aggregated access features with

any required precision. With FM, we can model access patterns of both a single user, and a cluster of users. Here, Yoda uses FM to model the access patterns of the active users individually and the aggregated access pattern of each cluster of users from the training data. Yoda also applies a version of the similarity measure we proposed in [13], *Projected Pure Euclidean Distance (PPED)*, to evaluate the similarity of a user access/interest pattern to a cluster access/interest pattern modeled by FM. PPED allows highly accurate classification of the users' partial interest patterns. Since essentially PPED is a *dissimilarity* measure, here we use a slightly different version of PPED to quantify *similarity* of user  $u$  to cluster  $k$ ,  $S_{uk}$ :

$$S_{uk} = \text{MaxDistance} - \text{TPPED}(u, k) \quad 0 \leq S_{uk} \leq \text{MaxDistance}$$

where  $\text{MaxDistance}$  is a constant parameter that is selected as the upper bound for distance, and Truncated PPED (TPPED) is defined as follows:

$$\text{TPPED}(u, k) = \begin{cases} \text{PPED}(u, k) & \text{if } \text{PPED}(u, k) \leq \text{MaxDistance} \\ \text{MaxDistance} & \text{if } \text{PPED}(u, k) > \text{MaxDistance} \end{cases}$$

Thus, by computing  $S_{uk_i}$ , Yoda can quantify the similarity of user  $u$  interests to the interest pattern of each cluster  $k_i$  and *softly* classify user interests to typical interest patterns within the web-site. During the item ranking process (see Section 4.2), Yoda uses  $S_{uk_i}$  as the weight factor for aggregating wish-list of cluster  $k_i$  into the final user wish-list.

## 4.2 Phase II - Ranking the Items

In this section, first we formally define some terms. Second, we explain the off-line process through which the cluster wish-lists are constructed. Third, we describe how user wish-list is generated during the on-line process. Finally, we explain a filtering mechanism to optimize the time complexity of generating cluster wish-lists.

**Defining Terms** Here, we define the notions of property, item, and user/cluster wish-list. An *item* is an instance of product, service, etc. that is presented in a web-site. The set of items presented in a web-site comprise the *item-set*,  $I$ , of the web-site. Items are described by their *properties*, which are abstract perceptual features. For example, for a music CD as an item, “styles” of the music (such as Classic, Rock and Pop) and “ratings” of this CD by different critics can be considered as properties of the item. Since properties are perceptual we use fuzzy-sets to evaluate properties [14].

**Definition** If  $\varphi = \{f_l \mid f_l \text{ is a fuzzy set, and } \forall l \in N - \{1\} \quad f_l > f_{l-1}\}$  and  $P = \{p \mid p \text{ is a property}\}$ , then an item  $i \in I$  is defined as:

$$i = \{(p, \tilde{p}_i) \mid p \in P, \tilde{p}_i \in \varphi\}$$

**Example** Suppose item  $K$  is defined as:  $\{ (\text{Pop}, \text{high}), (\text{Rock}, \text{low}), (\text{Critic-A}, \text{good}), (\text{Critic-C}, \text{neutral}) \}$ . It represents that the style of this item is very similar to “Pop” and only a little similar to “Rock”. Moreover, it also describes the opinions of two critics.

**Definition** A wish-list,  $I_x$ , for user/cluster  $x$  is defined as:

$$I_x = \{(i, v_x(i)) | i \in I, v_x(i) \in [0, 1]\}$$

where the *preference* value  $v_x(i)$  measures the probability of  $x$  being interested in the item  $i$ .

**Generating Cluster Wish-lists (off-line process)** Yoda represents the aggregated interests of the users in each cluster by a set of property values (PVs), termed *favorite PVs* of the cluster. Each favorite PV identifies likelihood of the cluster being interested in a specific property of the items. Favorite PVs of each cluster are extracted by applying a *voting procedure* to the set of items visited by the users of a cluster, termed the *browse-list* of the cluster, as follows:

**Definition** *User browse-list*,  $b_u$ , and *cluster browse-list*,  $B_k$ , are defined as:

$$b_u = \{i \mid i \in I, \text{“}i\text{” is visited by } u \in U\}$$

$$B_k = \bigcup_{u \in k} b_u$$

where  $U$  is the training set of users. The voting procedure extracts the favorite PV,  $F_p(k)$ , corresponding to property  $p$  for cluster  $k$  as follows<sup>1</sup>:

$$C_{p,f}(k) = \|\{i \mid i \in B_k, \tilde{p}_i = f\}\|$$

$$F_p(k) = \text{fmax}\{f \mid f \in \varphi, C_{p,f}(k) = \max_{f' \in \varphi} \{C_{p,f'}(k)\}\}$$

**Example** Suppose the browse-list of cluster  $K$  is  $\{\text{CD-A}, \text{CD-B}, \text{CD-G}, \text{CD-K}, \text{CD-Y}, \text{CD-Z}\}$ , and the values of property “Rock” for the corresponding CDs are  $\{ (\text{CD-A}, \text{high}), (\text{CD-B}, \text{high}), (\text{CD-G}, \text{low}), (\text{CD-K}, \text{medium}), (\text{CD-Y}, \text{high}), (\text{CD-Z}, \text{high}) \}$ . Because “high” has the maximum vote, the favorite PV of cluster  $K$ ,  $F_{\text{Rock}}(K)$ , is “high”.

*Cluster-Yoda* is the module that evaluates  $v_k(i)$ , preference value of an item  $i$  for cluster  $k$ . To compute  $v_k(i)$ , cluster-Yoda employs a fuzzy aggregation function to measure and quantify the similarity between favorite PVs of the cluster  $k$  and specific property values associated with the item  $i$ . We use an optimized aggregation function with a triangular norm, which satisfies *conservation*, *monotonicity*, *commutativity*, and *associativity* requirements for data aggregation [2]. Here, we formally define the aggregation function used to compute  $v_k(i)$ :

<sup>1</sup> “fmax” is the fuzzy *max* function.

**Definition** First, properties are grouped based on their corresponding values in favorite PVs of the cluster  $k$ :

$$G_f(k) = \{p \mid f \in \varphi, p \in P, F_p(k) = f\}$$

then, the preference value  $v_k(i)$  for item  $i$  is computed as:

$$\begin{aligned} E_{k,f}(i) &= f \times \max\{\check{p}_i \mid p \in G_f(k)\} \\ v_k(i) &= \max\{E_{k,f}(i) \mid \forall f \in \varphi\} \end{aligned}$$

**Example** Suppose properties are grouped as  $G_{medium}(K) = \{\text{Vocal, Soundtrack}\}$ ,  $G_{high}(K) = \{\text{Rock, Pop}\}$ , and  $G_{low}(K) = \{\text{Classic}\}$ , and the item  $i$  is defined as  $\{(\text{Rock, low}), (\text{Pop, low}), (\text{Vocal, low}), (\text{Soundtrack, high}), (\text{Classic, medium})\}$ . According to the equations above, the preference value  $v_K(i) = \max\{(\text{high} \times \text{low}), (\text{medium} \times \text{high}), (\text{low} \times \text{low})\} = (\text{medium} \times \text{high}) = 0.75$ .

Basically, this aggregation function partitions the properties into  $\|\varphi\|$  different subgroups according to the favorite PVs of the cluster  $k$ . Subsequently, the system maintains a list of maximum property values for all subgroups. Finally, the system computes the preferences of all items in the cluster wish-list by iterating through all subgroups. As compared to a naive weighted aggregation function with time complexity  $O(\|P\| \times \|I\|)$ , the complexity of the proposed aggregation function is  $O(\|\varphi\| \times \|I\|) = O(\|I\|)$ , where  $\|\varphi\|$  is a small constant number.

To reduce the time complexity of generating the cluster wish-lists further, we apply a cut-off point to the cluster wish-lists. Each cut wish-list includes the  $N$  best-ranked items according to their preference values for the corresponding cluster. In [2], Fagin has proposed an optimized algorithm, the  $A_0$  algorithm, to retrieve  $N$  best items from a collection of subsets of items with time complexity proportional to  $N$  rather than total number of items. Here, by taking the subgroups of items (as described above) as the subsets, the  $A_0$  algorithm can be incorporated into the cluster-Yoda<sup>2</sup>. Applying the  $A_0$  algorithm to generate a cluster wish-list with cut-off point  $N$ , we reduce the time complexity to  $O(\|\varphi\| \times \|N\|) = O(\|N\|)$ , where  $\|N\| \ll \|I\|$ .

**Generating User Wish-lists (on-line process)** During the on-line recommendation process, *user-Yoda*, which is an aggregation module similar to the cluster-Yoda, aggregates the cluster wish-lists to generate the final resolved user wish-list for the active user  $u$ . User-Yoda applies a fuzzy aggregation function to compute the preference value  $v_u(i)$  of each item  $i$  ( $i \in \bigcup_{\forall k} I_k$ ) for the user  $u$  based on similarity  $S_{uk}$  of user  $u$  to clusters  $k$  (where  $i \in I_k$ ).

<sup>2</sup> Since our aggregation function is in triangular norm form, it satisfies the requirements of the  $A_0$  algorithm.

**Definition** First, clusters are grouped based on their similarity to the user  $u$  as follows<sup>3</sup>:

$$G_f(u) = \{k \mid f \in \varphi, S_{uk} = f\}$$

then, the preference value  $v_u(i)$  for item  $i$  is computed as:

$$\begin{aligned} E_{u,f}(i) &= f \times \max\{v_i(k) \mid k \in G_f(u)\} \\ v_i(u) &= \max\{E_{u,f}(i) \mid \forall f \in \varphi\} \end{aligned}$$

**Filtering Mechanism** Cluster-Yoda generates a cluster wish-list based on the favorite PVs of the cluster. The large size of the item-set renders our approach to off-line ranking practically time complex. Therefore, we have to incorporate a mechanism into the cluster-Yoda to target the set of items that more probably contribute towards higher preference values.

The item-set in large-scale applications is a high-dimensional database. In [16], it is demonstrated that all index structures degrade to a linear search for sufficiently high dimensions. On the other hand, we observe that items comprising the item-set are *locally* distributed; a property that can be exploited to reduce time complexity of the item selection for the cut cluster wish-lists. We extend the *Locality Sensitive Hashing (LSH)* algorithm proposed in [15] to hash items into hash buckets preserving the locality in storage. This algorithm has sublinear dependency on the item-set size, even when the item-set is a high-dimensional database. However, a fuzzy feature space bears the property that the higher fuzzy terms (i.e.  $f_{l_1} > f_{l_2}$ , then  $f_{l_1}$  is *higher*) show more proximity to the solution space as opposed to the lower fuzzy terms. Thus, we have developed a novel fuzzy distance measure as opposed to the Hamming distance measure used by [15] to consider this property. We term the locality sensitive hashing algorithm customized for our application as *Fuzzy Locality Sensitive Hashing (FLSH)*. FLSH reduces the *potential* solution space of the problem stated below, so that the aggregation function of the cluster. Yoda achieves the ideal solution with lower time complexity:

**Problem Statement** Find the  $N$  nearest neighbors for favorite PVs of cluster  $k$ ,  $V_k$ :

$$V_k = \{(p, F_p(k)) \mid p \in P\}$$

from the item-set  $I$ , where  $\|I\| = M \gg N$ .

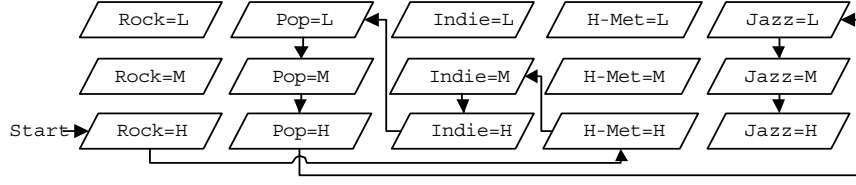
First, Yoda hashes the items in the item-set  $I$  using the LSH algorithm. Each item  $i \in I$  is embedded into a Hamming cube  $H^{d'}$ , where  $d' = f_{max} \times d$  represents the number of dimensions for the Hamming cube,  $f_{max}$  is the highest

---

<sup>3</sup> Numerical  $S_{uk}$  values are converted to fuzzy equivalents. See [14] for details of the conversion procedure.

value of the fuzzy terms in  $\varphi$ , and  $d = \|P\|$ . This procedure transforms each item  $i$  into a binary vector  $z_i$ . The LSH algorithm is described as follows [15]: choose  $l$  subsets  $Q_1, \dots, Q_j, \dots, Q_h$  of  $\{1, \dots, d'\}$ ; let  $z|_{Q_j}$  denote the projection of vector  $z$  on the coordinate set  $Q_j$ ; the hash function is defined as  $g_j(z) = z|_{Q_j}$ . The locality sensitive hash function maps the binary representation of the item  $i$  into the bucket  $g_j(z_i)$ . This pre-processing achieves a localized distribution of the item-set  $I$  into the hash buckets.

After inserting the items into the hash buckets, Yoda can retrieve the  $N$  nearest items to  $V_k$  by visiting the hash buckets one-by-one until the  $N$  required items are found. We define a fuzzy distance measure to determine the optimum order of visiting the hash buckets based on their distance from  $V_k$ , starting from the closest bucket. Here, we formally define our fuzzy distance measure.



**Fig. 2.** Filtering Mechanism - Example Schematic Diagram

**Definition** Set  $L$  of all pairs of the properties and their corresponding fuzzy values that need to be searched is defined as:

$$L = \{(p, \tilde{p}) \mid p \in P, \tilde{p} \geq F_p(k)\}$$

There is a one-to-one relationship between elements of  $L$ ,  $e_i$ , and the hash buckets. To select the next bucket in each step, distance of the bucket centroid  $\beta_j$  from  $V_k$ ,  $D(\beta_j, V_k)$ , is measured as:

$$S_k(p) = \begin{cases} \text{fmin}\{\tilde{p} \mid \forall (p, \tilde{p}) \in L\} & \exists (p, \tilde{p}) \in L \\ 0 & \text{otherwise} \end{cases}$$

$$D(\beta_j, V_k) = \sum_{p \in P} [w(S_k(p)) \times S_k(p) \times \beta_j(p)]$$

where the weight  $w(f_i)$  for fuzzy term  $f_i$  is defined so that  $w(f_i) > d \times w(f_{i-1})$ , and  $\beta_j(p)$  is the value for property  $p$  in bucket centroid  $\beta_j$ . The closest bucket is selected for the next step. During transition between bucket  $i$  and bucket  $i+1$ ,  $L$  is updated according to the following rule:

$$L \leftarrow L - \{e_i\} \quad (\forall i > 0)$$

where

$$e_i = \begin{cases} (p, f_{i+1}) & e_{i-1} = (p, f_i < \text{fmax}(\varphi)) \text{ and } i > 1 \\ (p, \text{fmax}\{\tilde{p} \mid \forall (p, \tilde{p}) \in L\}) & \text{otherwise} \end{cases}$$

Figure 2, illustrates an example search path according to FLSH retrieval algorithm. The transition diagram is generated based on the favorite PVs of the cluster  $k$  defined as:  $F_{Rock}(k) = High$ ,  $F_{Pop}(k) = Low$ ,  $F_{Indie}(k) = Medium$ ,  $F_{Heavy-Metal}(k) = High$ ,  $F_{Jazz}(k) = Low$ , where  $P = \{Rock, Pop, Indie, Heavy-Metal, Jazz\}$ , and  $\varphi = \{Low, Medium, High\}$ .

## 5 Performance Evaluation

We performed an end-to-end simulation to compare accuracy and scalability of Yoda with the basic nearest-neighbor (BNN) method [10]. In this simulation framework, both Yoda and BNN are implemented in Java<sup>TM</sup>, on top of Microsoft<sup>TM</sup> Access 2000. We used a Microsoft<sup>TM</sup> NT 4.0 personal computer with a Pentium<sup>TM</sup> II 233MHz processor to run our experiments.

### 5.1 Experimental Methodology

In order to generate synthetic data for evaluation purposes, we propose a parametric algorithm to simulate various benchmarks (see Table 1). First, the benchmark method generates  $K$  clusters. Each cluster comprises a browse-list, a list of favorite PVs, and a pattern of navigation as the cluster centroid. Every item in the cluster browse-list is assigned a rating value to be used by BNN. For each cluster, the algorithm then randomly generates  $S/K$  users. Each user has a current browse-list,  $b_u$ , and an expected browse-list,  $e_u$ , that are both constructed around the centroid of cluster  $k$  with 30% noise.

| Parameter | Definition                                  |
|-----------|---|
| $d$       | Number of properties ( $\ P\ $ )            |
| $M$       | Number of items in the item-set ( $\ I\ $ ) |
| $S$       | Number of user sessions used for analysis   |
| $L_{min}$ | Minimum size of user browse-lists           |
| $L_{max}$ | Maximum size of user browse-lists           |
| $\Psi$    | Number of fuzzy terms ( $\ \varphi\ $ )     |
| $K$       | Number of clusters                          |
| $N$       | The cut-off point                           |

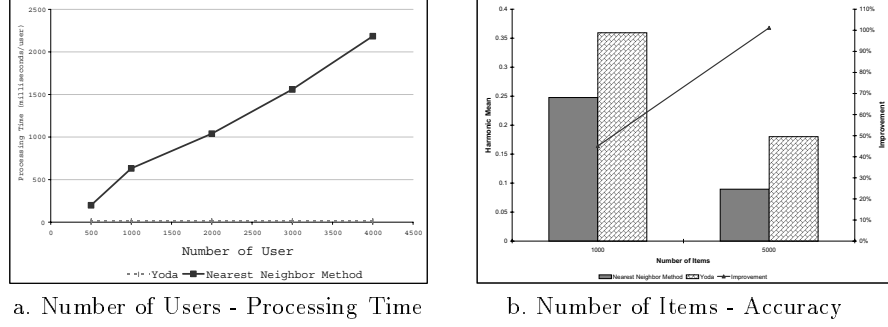
Table 1. Benchmarking Parameters

Subsequently, each item in  $b_u$  is assigned a rating value based on the original rating value in cluster  $k$ . Next, the algorithm generates PVs for each item  $i$  based on the favorite PVs of the cluster that has the highest rating for item  $i$ , say cluster  $k'$ . The higher is the rating of item  $i$  in cluster  $k'$ , the more similar are PVs of  $i$  to favorite PVs of cluster  $k'$ . The rating values and PVs are represented by fuzzy terms. We use Yoda and BNN to construct wish-list  $I_u$  for each user  $u$  and compare the wish-list with  $e_u$  to evaluate the accuracy of these systems.

### 5.2 Experimental Results

We conducted several sets of experiments to compare Yoda with BNN. In these experiments, we observed a significant margin of improvement over BNN in matching the user expectations for various settings. Moreover, it is shown that performance of Yoda is independent of the number of users.

The results shown for each set of experiments are averaged over many runs, where each run is executed with different seeds for the random generator functions. The coefficient of variance of these runs is smaller than 5%, which shows our results are independent of the specific run.



**Fig. 3.** Impacts of number of users on processing time and number of items on accuracy

In Figure 3, we compare performances of Yoda and BNN. The benchmark settings of these experiments,  $K$ ,  $L_{min}$ ,  $L_{max}$ ,  $d$ ,  $N$ , and  $\Psi$  are fixed at 18, 25, 30, 10, 150, 10, respectively.  $M$  in Figure 3.a is 5000 and  $S$  in Figure 3.b is 500. X-axis of Figure 3.a is  $S$  varying from 500 to 4000, and X-axis of Figure 3.b is  $M$ . Y-axis of Figure 3.a is the system processing time for each user, and Y-axis of Figure 3.b depicts the Harmonic mean computed by Equation 1, and improvement computed by Equation 2.

$$\text{Harmonic Mean}(\text{Precision}, \text{Recall}) = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \quad (1)$$

$$\text{Improvement}(\text{Yoda}, \text{BBN}) = \frac{(\text{Yoda} - \text{BBN})}{\text{BBN}} \quad (2)$$

Figure 3.a verifies that Yoda is scalable. As the number of users grow, the system processing time of Yoda remains unchanged while the processing time of BBN increases linearly. This is because Yoda is a model-based recommendation system. Figure 3.b indicates that Yoda always outperforms BBN in accuracy. Although performances of both systems decrease as the number of items grow, the margin of improvement between Yoda and BBN expands. We attribute this improvement to the incorporation of content-based filtering into the Yoda infrastructure.

With Figure 4, we demonstrate impacts of  $N$  in accuracy and processing time of the two systems. The benchmark settings of these experiments,  $K$ ,  $S$ ,  $M$ ,  $d$ ,  $L_{min}$ ,  $L_{max}$ , and  $\Psi$  are fixed at 18, 1000, 5000, 5, 25, 30, and 10, respectively. X-axis is  $N$  varying from 50 to 250. Y-axis of Figure 4.a depicts the Harmonic mean and improvement, and the Y-axis of Figure 4.b is the system processing time.

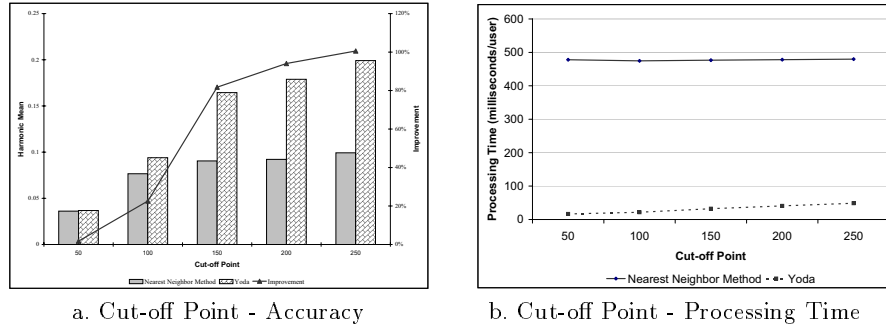


Fig. 4. Impacts of cut-off point on accuracy and processing time

In Figure 4.a, accuracy of both Yoda and BNN improves as  $N$  grows because more items are considered in the wish-list. However, the margin of improvement between Yoda and BNN grows as the cut-off point is increased. Again, applying content-based filtering to retrieve the recommendable items enables Yoda to identify more items similar to the items in the user browse-list. In Figure 4.b, as the cut-off point increases processing time of Yoda grows while processing time of BNN remains unaffected. This observation shows that based on the size of the required wish-list, Yoda searches only a subset of the item-set to generate the wish-lists while with BNN, regardless of the size of the wish-lists, the entire item-set is processed.

## 6 Conclusions and Future Work

In this paper, we described a recommendation system, termed *Yoda*, that is designed to support large-scale web-based applications requiring highly accurate recommendations in real-time. Our end-to-end experiments show while Yoda scales as the number of users and/or items grow, it achieves up to 120% higher accuracy as compared to the basic nearest-neighbor method. We intend to extend this study in several ways. First, we would like to run more experiments with real data to verify our results and to compare with other approaches. Second, we want to incorporate the content-based filtering mechanism into the *user classification* phase. Finally, our aggregation function is defined in the domain of the original fuzzy logic theory, fuzzy type-I. However, recently Karnik et al. [17] introduced fuzzy type-II to incorporate uncertainty in computation.

## Acknowledgments

We are grateful for the assistance given by Javed Faruque in conducting some experiments and for Amol Ghoting's helpful suggestions in improving the *FLSH* filtering mechanism. This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and ITR-0082826, NASA/JPL contract nr. 961518, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

## References

1. Sarwar B., G. Karypis, J. Konstan, and J. Riedl: Analysis of Recommendation Algorithms for E-Commerce, Proceedings of E-Commerce Conference, 17-20 October, Minneapolis, Minnesota, 2000.
2. Fagin R.: Combining Fuzzy Information from Multiple Systems, Proceedings of Fifteenth ACM Symposium on Principles of Database Systems, Montreal, pp. 216-226, 1996.
3. Shahabi C., and Y. Chen: Efficient Support of Soft Query in Image Retrieval Systems, Proceedings of SSGRR 2000 Computer and eBusiness Conference, Rome, Italy, August, 2000.
4. Balabanovi M., and Y. Shoham: Fab, content-based, collaborative recommendation, Communications of the ACM, Vol 40(3), pp. 66-72, 1997.
5. Resnick P., N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl: GroupLens, An Open Architecture for Collaborative Filtering of Netnews, Proceedings of ACM conference on Computer-Supported Cooperative Work, Chapel Hill, NC, pp. 175-186, 1994.
6. Sarwar B., G. Karypis, J. Konstan, and J. Riedl: Analysis of Recommendation Algorithms for e-Commerce, Proceedings of ACM e-Commerce 2000 Conference, Minneapolis, MN, 2000.
7. Good N., J. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl: Combining Collaborative Filtering with Personal Agents for Better Recommendations, Proceedings of the 1999 Conference of the American Association of Artificial Intelligence, pp. 439-446, 1999.
8. Kitts B., David Freed, and Martin Vrieze: Cross-sell, a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities, Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, pp. 437-446, August, 2000.
9. Breese J., D. Heckerman, and C. Kadie: Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, pp. 43-52, July, 1998.
10. Sarwar B., G. Karypis, J. Konstan, and J. Riedl: Application of Dimensionality Reduction in Recommender System – A Case Study, ACM WebKDD 2000 Web Mining for e-Commerce Workshop, 2000.
11. Konstan, J., B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl: Applying Collaborative Filtering to Usenet News, Communications of the ACM (40) 3, 1997.
12. Shahabi C., A.M. Zarkesh, J. Adibi, and V. Shah: Knowledge Discovery from Users Web Page Navigation, Proceedings of the IEEE RIDE97 Workshop, April, 1997.
13. Shahabi C., F. Banaei-Kashani, J. Faruque, and A. Faisal: Feature Matrices: A Model for Efficient and Anonymous Web Usage Mining, EC-Web 2001, Germany, September 2001
14. Shahabi C., and Y. Chen: A Unified Framework to Incorporate Soft Query into Image Retrieval Systems, International Conference on Enterprise Information Systems, Setubal, Portugal, July 2001
15. Gionis A., P. Indyk, and R. Motwani: Similarity search in high dimensions via hashing, Proceedings of the 25th International Conference on Very Large Databases, Edinburgh, Scotland, 1999.
16. Weber R., H. Schek and S. Blott: A quantitative analysis and performance study for Similarity Search Methods in High Dimensional Spaces, Proceedings of the 24th International Conference on Very Large Data bases, 1999.
17. Karnik N., and J. Mendel: Operations on Type-2 Fuzzy Sets, International Journal on Fuzzy Sets and Systems, 2000.