Fusion-based Recommender System for Improving Serendipity

Kenta Oku
College of Information Science and Engineering,
Ritsumeikan University,
1-1-1 Nojihigashi, Kusatsu-city,
Shiga, Japan
oku@fc.ritsumei.ac.jp

Fumio Hattori
College of Information Science and Engineering,
Ritsumeikan University,
1-1-1 Nojihigashi, Kusatsu-city,
Shiga, Japan
fhattori@is.ritsumei.ac.jp

ABSTRACT

Recent work has focused on new measures that are beyond the accuracy of recommender systems. Serendipity, which is one of these measures, is defined as a measure that indicates how the recommender system can find unexpected and useful items for users. In this paper, we propose a Fusion-based Recommender System that aims to improve the serendipity of recommender systems. The system is based on the novel notion that the system finds new items, which have the mixed features of two user-input items, produced by mixing the two items together. The system consists of item-fusion methods and scoring methods. The item-fusion methods generate a recommendation list based on mixed features of two user-input items. Scoring methods are used to rank the recommendation list. This paper describes these methods and gives experimental results.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering

General Terms

Algorithms, Experimentation

Keywords

Recommender system, Fusion-based recommender system, Serendipity

1. INTRODUCTION

Various recommender systems have been proposed and developed since collaborative filtering was first introduced in the mid-1990s [8][6][1]. In the early years, most recommender systems focused on recommendation accuracy, based on the notion that providing items suitable for users' preferences contributes to an improvement in user satisfaction [8][9]. In contrast, in recent years, several researchers have indicated that recommender systems with high accuracy do not always satisfy users [4][7][5]. They say that recommender systems should be evaluated not only by accuracy, but also by various other metrics such as diversity, novelty, and serendipity.

Suppose that Alice likes "Harry Potter Part I." To recommend "Harry Potter Part II" or "Harry Potter Part III" to her is obvious and not surprising. Although, from the viewpoint of accuracy, this recommendation is good, it is hard to say that the recommendation satisfies her. Recommender systems should surprise users by providing them with unexpected and useful items.

We focus on serendipity, which is one of the measures beyond accuracy. Although the definition of serendipity has not yet been fixed, Herlocker et al. [4] define serendipity as a measure of the degree to which recommendations are both attractive and surprising to users.

Ge et al. [3] also mention two aspects related to serendipity. The first one is that a serendipitous item should not yet have been discovered by the user and should not be expected by the user. The second one is that the item should also be interesting, relevant, and useful to the user. Although several researchers have tried to improve serendipity, fixing the definition of serendipity and designing recommender systems that improve serendipity are still open problems.

In order to improve serendipity, we believe that recommender systems should have a mechanism that enables users to accidentally discover novel values from unexpected results caused by the user's active actions. "The Three Princes of Serendip" (by Horace Walpole),which is the origin of the term serendipity, tells the story of three princes. They discovered a series of novel things from various and unexpected events on their journeys. Then, they connected these things with their luck. Serendipity is also often involved in making new discoveries. Researchers notice unexpected results in their experiments by trial and error, and then, they connect these results with new discoveries.

Based on this notion, we propose a Fusion-based Recommender System that aims to improve the serendipity of recommender systems. The system recommends new items, which have the mixed features of two user-input items, produced by mixing the two items together.

Such acts of "mixing together", for example, "mixing colors," "mixing ingredients", and "mixing sounds", is intuitive, familiar to people, and has the following characteristics.

(a) New substances are created from existing ones.

(b) We can intuitively imagine the mixed results from a combination of input substances. However, some combinations yield unexpected results.

(c) Because our curiosity may be aroused by characteristic (b), we might feel like trying to mix various combinations.

Consider the case of mixing colors. If there is no existing color that we want to use, we can create a new color by mixing the existing colors. We can easily imagine that we can create sky-blue from blue and white. On the other hand, some combinations of colors yield unexpected colors. Therefore, our curiosity may be aroused, and we may feel like mixing various combinations by trial and error, for example, "What kind of colors can we create by mixing a certain color and another one?"

The Fusion-based Recommender System adopts an item-fusion approach that produces serendipitous items. The system consists of item-fusion methods and scoring methods. The item-fusion methods generate a recommendation list based on the mixed features of two user-input items. Scoring methods are used to rank the recommendation list.

The contributions of this paper include:

- providing a novel Fusion-based Recommender System that adopts a fusion-based approach to improving serendipity;
- providing three item-fusion methods depending on item representation, and several scoring methods for each item-fusion method;
- a proposed system that can be applied to any dataset that consists of at least an item table, a user table, and a rating table, which are traditional structures in the area of recommendation research;
- an evaluation of the recommender system from the viewpoint of unexpectedness and serendipity.

This paper is organized as follows. In Section 2, we discuss related work that mentions serendipity. In Section 3, we present our proposed system, i.e.,a Fusion-based Recommender System. Specifically, we describe item-fusion methods and scoring methods. In Section 4, we evaluate the system from the viewpoint of unexpectedness and serendipity. Finally, we conclude the paper and show future directions in Section 5.

2. RELATED WORK

Herlocker et al. [4] suggest that recommender systems with high accuracy do not always satisfy users. They say that recommender systems should be evaluated not only by their accuracy, but also by various other metrics such as diversity, novelty, and serendipity.

Several researchers mention serendipity in the context of recommendation. Ziegler et al.[11][12] assume that diversifying recommendation lists improves user satisfaction. They proposed topic diversification, which diversifies recommendation lists, based on an intra-list similarity metric. Sarwar et al. [10] mention that serendipity might be improved by removing obvious items from recommendation lists. Berkovsky et al. [2] proposed group-based recipe recommendations. They suggest that recipes loved by a group member are likely to be recommended to others, which may increase serendipity.

Hijikata et al. [5] and Murakami et al. [7] proposed recommendation methods that predict novelty or unexpectedness. Hijikata et al. [5] proposed collaborative filtering, which aims to improve novelty. Collaborative filtering predicts unknown items for a target user, based on known/unknown profiles explicitly acquired from the user. They showed that such filtering can improve novelty by providing unknown items to the user. Murakami et al. [7] proposed a method that implicitly predicts unexpectedness based on a user's action history. They introduced a preference model, which predicts items the user likes, and a habit model, which predicts items habitually selected by the user. The method estimates the unexpectedness of recommended items by considering differences between the models. They need to accumulate models or profiles for an individual user, but our proposed system does not need these. Our system can instantly recommend serendipitous items based on items the user has just selected.

Murakami et al. [7] and Ge et al. [3] introduced measures for evaluating the unexpectedness and serendipity of recommender systems.

Murakami et al. [7] assume that unexpectedness is the distance between the results produced by the system to be evaluated and those produced by primitive prediction methods. Here, primitive prediction methods mean naive methods such as recommendation methods based on user profiles or action histories. Based on this notion, they proposed *unexpectedness* for measuring the unexpectedness of recommendation lists. They also proposed *unexpectedness_r*, which takes into account the rankings in the lists.

Ge et al. [3] mention two aspects related to serendipity. The first one is that a serendipitous item should not yet have been discovered and should not be expected by the user. The second one is that the item should also be interesting, relevant, and useful to the user. Although several researchers have tried to improve serendipity, fixing the definition of serendipity and designing recommender systems that improve serendipity are still an open problem. With respect to unexpectedness, they follow the notion of Murakami et al. [7]. They defined an unexpected set of recommendations as follows:

$$UNEXP = RS \backslash PM \tag{1}$$

Here, PM denotes a set of recommendations generated by primitive prediction models and RS denotes the recommendations generated by a recommender system to be evaluated. In addition, by using $u(RS\backslash PM)$, which denotes the usefulness of the unexpected recommendations, they defined serendipity as follows:

$$SRDP = \frac{\sum_i u(UNEXP_i)}{|UNEXP|} \tag{2}$$

Here, $UNEXP_i$ denotes an element of $UNEXP$. When $u(UNEXP_i) = 1$, $UNEXP_i$ is useful to the user,and when $u(UNEXP_i) = 0$, $UNEXP_i$ is useless to the user. The usefulness of $UNEXP_i$ is given by the user. In Section 4, we evaluate our system using Ge's measures.

3. FUSION-BASED RECOMMENDER SYSTEM

In this section, we describe our proposed system, a Fusion-based Recommender System.

First of all, a user of this system selects two arbitrary items as input items to the system. Then the system finds new items that have the mixed features of both items, using the item-fusion methods described in Section 3.3. After that, the system makes a recommendation list from the item set, and ranks the list by scoring methods described in Section 3.4. Finally, the system provides the top-N items to the user. The user can then repeatedly use the system by using items in the ranking results in order to find more satisfactory items.

This section is organized as follows. In Section 3.1, we describe the database structure that the system assumes. In addition, we define item similarity and supporting user, as used in this paper. In Section 3.2, we explain the feature representations of items used to apply the system. In Section 3.3, we describe item-fusion methods for generating a recommendation list, and in Section 3.4, we describe scoring methods for ranking the list.

3.1 Preliminary

3.1.1 *Database structure*

First of all, in this section, we describe the database structure that the system assumes.

The system assumes that a database consists of the following tables:

(a) Item table (Item ID, Feature 1, Feature 2, ...)

(b) User table (User ID, Profile 1, Profile 2, ...)

(c) Rating table (User ID, Item ID, Rating)

Public datasets such as MovieLens Data Sets and Book-Crossing Data Sets[1] already include the above tables. Other datasets can also be applied to the system by relating them to the above tables.

3.1.2 *Item similarity*

The system calculates item similarity by measuring the similarity between items. This paper defines the following two types of item similarity:

(a) content-based similarity,

(b) collaborative-based similarity

Consider two items a and b.

(a) Content-based similarity is calculated based on features of items in the item table. Although the features used for the calculation of similarity depend on the datasets, for each item, the system generates a feature vector whose elements correspond to feature values. Then the system calculates item similarity by the cosine similarity between the feature vectors. Consider items a and b represented as follows:

$$\boldsymbol{a} = (a_1, a_2, \ldots, a_n) \tag{3}$$
$$\boldsymbol{b} = (b_1, b_2, \ldots, b_n) \tag{4}$$

Here, n is the number of dimensions of the vector. Then the similarity between the items $\text{sim}(\boldsymbol{a}, \boldsymbol{b})$ is calculated by the following equation:

$$\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2}\sqrt{\sum_i b_i^2}} \tag{5}$$

(b) Collaborative-based similarity is calculated based on ratings given to items in the rating table. The system finds a common set of users, $U = \{u_1, u_2, \ldots, u_m\}$ (m is the number of common users), who gave ratings to both items a and b. Let $\text{rating}(u_i, j)$ be a rating given by a user u_i to an item j. Consider items a and b, represented as follows:

$$\boldsymbol{a} = (\text{rating}(u_1, a), \text{rating}(u_2, a), \ldots, \text{rating}(u_m, a)) \tag{6}$$
$$\boldsymbol{b} = (\text{rating}(u_1, b), \text{rating}(u_2, b), \ldots, \text{rating}(u_m, b)) \tag{7}$$

Then the similarity between the items $\text{sim}(\boldsymbol{a}, \boldsymbol{b})$ is calculated by Equation (5) in the same way.

We define a similar-item set S_a for an item a as an item set that consists of items whose similarity to item a is greater than or equal to a threshold θ, i.e., the similar-item set S_a is expressed by the following equation:

$$S_a = \{x | \text{sim}(x, a) \geq \theta\} \tag{8}$$

3.1.3 *Supporting user*

We define a supporting-user set V_a for an item a as a user set that gave ratings equal to or greater than a threshold τ to the item a, i.e., the supporting-user set V_a is expressed by the following equation:

$$V_a = \{x | \text{rating}(x, a) \geq \tau\} \tag{9}$$

3.2 Feature representation of an item

We define the feature representation of items on the basis of the database described in Section 3.1. In this study, we define the following naive representation.

(1) Bit-string representation.

This representation represents an item as a bit string. Suppose that five elements, {"Action," "Adventure," "Comedy," "Horror," "Romance"}, are defined as attributes that denote item genres. If an item a corresponds to the genres {"Action," "Horror"}, the item a is represented as $\text{bit}(a) = [10010]$.

(2) Set representation.

This representation represents an item as a set of related elements. In this paper, we define the following two types of representation, depending on the types of elements:

(2a) representation by a similar-item set,

(2b) representation by a supporting-user set.

Consider an item a.

In case (a), we extract a similar-item set S_a for item a, based on item similarity as defined in Section 3.1.2. If the similar-item set is given as $S_a = \{b, c, d, e, f\}$, the item a is expressed by $\text{set}(a) = S_a = \{b, c, d, e, f\}$.

In case (b), we extract a supporting-user set V_a for item a, based on the rating table. If the supporting-user set is given as V_a ={"Alice," "Bob," "Carol"}, item a is expressed by $\text{set}(a) = V_a$ ={"Alice," "Bob," "Carol"}.

3.3 Item-fusion method for generating recommendation list

A user can find novel items by mixing two items a and b that the user selected at will. The system defines criteria for searching novel items, related to mixing features of items a and b. Then the system finds an item set that matches the

[1]http://www.grouplens.org/node/74

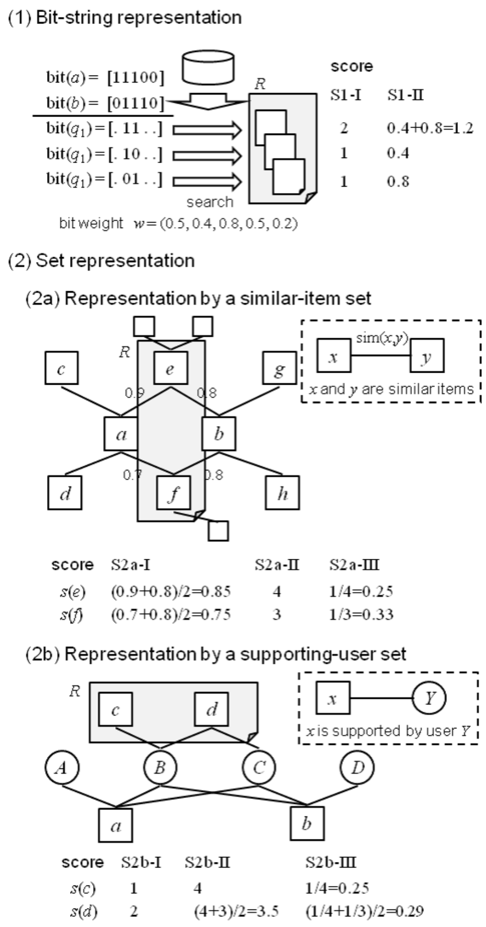
(1) Bit-string representation
bit(a)= [11100]
bit(b)= [01110]
bit(q1)=[. 11 . .]
bit(q1)=[. 10 . .]
bit(q1)=[. 01 . .]
search
R
score
S1-I S1-II
2 0.4+0.8=1.2
1 0.4
1 0.8
bit weight w=(0.5, 0.4, 0.8, 0.5, 0.2)
(2) Set representation
(2a) Representation by a similar-item set
R
c e g a b d f h
0.9 0.8 0.7 0.8
sim(x,y)
x y
x and y are similar items
score S2a-I S2a-II S2a-III
s(e) (0.9+0.8)/2=0.85 4 1/4=0.25
s(f) (0.7+0.8)/2=0.75 3 1/3=0.33
(2b) Representation by a supporting-user set
R
c d
A B C D
a b
x Y
x is supported by user Y
score S2b-I S2b-II S2b-III
s(c) 1 4 1/4=0.25
s(d) 2 (4+3)/2=3.5 (1/4+1/3)/2=0.29

Figure 1: Item-fusion methods and scoring methods in case that items a and b have common features

criteria from the database and adds them to a recommendation list. Here, we consider the following assumptions, depending on the features of the input items a and b.

(i) If there are common features between items a and b, the user requests items that have the common features.

(ii) Otherwise, the user requests diverse items that relate to either item a or item b.

Based on the above assumptions, we propose item-fusion methods for each feature representation defined in Section 3.2. Figure 1 illustrates examples of the item-fusion methods and scoring methods, which are described in Section 3.4, in case that items a and b have common features.

(1) Bit-string representation.

Based on the notion of a bitwise AND and a bitwise OR, which are the primitive bitwise operations, the system generates a recommendation list R.

Suppose that a user inputs two items a and b, represented as $\text{bit}(a) = [11100], \text{bit}(b) = [01110]$, respectively. Here, since the values of the second and the third bit are all 1 in both items a and b, the items a and b have common features. In order to emphasize these common features, the system generates a representative query bit-string $\text{bit}(q_1)$, based on the notion of the bitwise AND, i.e., in the case of bits whose values are 1 in both items a and b, let the values of the query bits be 1. In the other case, let the values of the query bits be ".". In this example, the query is expressed by $\text{bit}(q_1) = [.11..]$. Here, "." matches either of $\{0, 1\}$ while searching.

However, if the number of bits is large, there is little possibility of finding items that match the query. In order to avoid such cases, we consider a query set that considers all combinations of the values $\{0, 1\}$ except the bits whose values are ".", i.e., in this example, the generated query set is expressed by $Q = \{[.11..], [.10..], [.01..]\}$ (see Figure 1 (1)).

Now consider two items a and b that are represented as $\text{bit}(a) = [10000], \text{bit}(b) = [00110]$, respectively. In this case, the items a and b have no common feature. In order to diversify the recommendation list, the system generates a representative query bit-string $\text{bit}(q_1)$, based on the notion of the bitwise OR, i.e., in the case of bits whose values are 1 in either item a or b, let the values of the query bits be 1. In the other case, let the values of the query bits be ".". In this example, the query is expressed by $\text{bit}(q_1) = [1.11.]$.

In the same way, in this example, the generated query set is expressed by $Q = \{[1.11.], [1.10.], [1.01.], [1.00.], [0.11.], [0.10.], [0.01.]\}$.

Finally, the system finds an item set that matches each query bit-string from the item table and then adds the item set to the recommendation list R.

Now, we generalize the above examples. Consider items a and b represented by n-digit bit-strings. Let $\text{bit}(a)_i$ be the ith bit of item a. We define items a and b as having common features if there is at least one i that satisfies $\text{bit}(a)_i = \text{bit}(b)_i = 1$ $(i = 1, 2, \ldots, n)$.

(i) If items a and b have common features, each bit of the representative query bit-string $\text{bit}(q_1)$ is as follows:

$$\text{bit}(q_1)_i = \begin{cases} 1 & \text{if } \text{bit}(a)_i = 1 \wedge \text{bit}(b)_i = 1 \\ \text{"."} & \text{otherwise} \end{cases} \tag{10}$$

(ii) Otherwise, each bit of the string is as follows:

$$\text{bit}(q_1)_i = \begin{cases} 1 & \text{if } \text{bit}(a)i = 1 \vee \text{bit}(b)i = 1 \\ \text{"."} & \text{otherwise} \end{cases} \tag{11}$$

As we stated in the above examples, the system generates a query set Q, which considers all combinations of query bit-strings. Finally, the system generates a recommendation list R, based on the query set Q.

(2) Set representation.

Based on the notions of intersection and union, which are the primitive set operations, the system generates a recommendation list R.

Consider items a and b represented as $\text{set}(a) = \{a_1, a_2, \ldots, a_n\}$, $\text{set}(b) = \{b_1, b_2, \ldots, b_m\}$, respectively. We define items a and b as having common features if $\text{set}(a) \cap \text{set}(b) \neq \phi$. We explain how to generate the recommendation list R in cases of representation by (2a) a similar-item set, (2b) a supporting-user set.

(2a) Representation by a similar-item set.

(i) If the items a and b have common features, the system regards the intersection of similar-item sets of items a and b as the recommendation list R. (ii) Otherwise, the system regards the union of the sets as the recommendation list R, i.e., the recommendation list R in each case is as follows:

$$R = \begin{cases} \mathrm{set}(a) \cap \mathrm{set}(b) & \text{if } \mathrm{set}(a) \cap \mathrm{set}(b) \neq \phi \\ \mathrm{set}(a) \cup \mathrm{set}(b) & \text{otherwise} \end{cases} \tag{12}$$

For example, given an item $a = \{c, d, e, f\}$ and an item $b = \{e, f, g, h\}$, the recommendation list is $R = \mathrm{set}(a) \cap \mathrm{set}(b) = \{e, f\}$ (see Figure 1 (2a)). On the other hand, given an item $a = \{c, d\}$ and an item $b = \{e, f\}$, the recommendation list is $R = \mathrm{set}(a) \cup \mathrm{set}(b) = \{c, d, e, f\}$.

(2b) Representation by a supporting-user set.

(i) If the items a and b have common features, consider the intersection V of the supporting-user sets of items a and b. (ii) Otherwise, consider their union V, i.e., the user set in each case is as follows:

$$V = \begin{cases} \mathrm{set}(a) \cap \mathrm{set}(b) & \text{if } \mathrm{set}(a) \cap \mathrm{set}(b) \neq \phi \\ \mathrm{set}(a) \cup \mathrm{set}(b) & \text{otherwise} \end{cases} \tag{13}$$

Then the system regards an item set supported by each user $v_i \in V$ (who gave ratings equal to or greater than a threshold τ to the item), i.e., the recommendation list is as follows:

$$R = \bigcup_{v_i \in V} \{x | \mathrm{rating}(v_i, x) \geq \tau\} \tag{14}$$

For example, given an item set(a) ={“Alice,” “Bob,” “Carol”} and an item set(b) ={“Bob,” “Carol,” “Dave”}, the user set is $V = \mathrm{set}(a) \cap \mathrm{set}(b)$ ={“Bob,” “Carol”}. Furthermore, given an item set $\{c, d\}$ supported by “Bob”, and an item set $\{d\}$ supported by “Carol,” the recommendation list is $R = \{c, d\}$ (see Figure 1 (2b)).

On the other hand, given an item set(a) = {“Alice,” “Bob”} and an item set(b) ={“Carol”}, the user set is $V = \mathrm{set}(a) \cup \mathrm{set}(b)$ ={“Alice,” “Bob,” “Carol”}. Furthermore, given an item set $\{a\}$ supported by “Alice,” an item set $\{c, d\}$ supported by “Bob”, and an item set $\{d, e\}$ supported by “Carol,” the recommendation list is $R = \{a, c, d, e\}$.

3.4 Scoring method for ranking recommendation list

Some combinations of items produce recommendation lists that consist of an enormous number of items. In such cases, the recommendation list should be ranked according to some criteria in order to narrow the list of recommended items shown to the user. In this section, we define scoring methods for each item-fusion method provided in Section 3.3.

(1) Bit-string representation.

We define the following two scoring methods, S1-I and S1-II, for bit-string representation. Examples of these scoring methods are illustrated in Figure 1 (1).

(S1-I) Score based on the number of common bits.

As we stated in Section 3.3, the system generates a query set Q, which consists of all combinations of query bit-strings. We also explained that the value of each bit $\mathrm{bit}(q_i)_j$ can take $\{1, 0, “.”\}$. We assume that the larger the number of bits that satisfy $\mathrm{bit}(q_i)_j = 1$, the more strongly the query bit-string reflects common features of items a and b. Therefore, we define the following score $s(r_k)$ for a recommended item r_k:

$$s(r_k) = |\{x | \mathrm{bit}(q_k)_x = 1\}| \tag{15}$$

Here, q_k denotes a query used for searching the item r_k.

(S1-II) Weighted score based on the number of common bits.

Some datasets have different weights for each bit. If the weight w_j for each bit j is assigned in advance, we define the following weighted score $s(r_k)$:

$$s(r_k) = \sum_{j \in \{x | \mathrm{bit}(q_k)_x = 1\}} w_j \tag{16}$$

How the weight for each bit is calculated depends on the datasets, but, for a simple example, we can employ the bit variance in the dataset.

(2) Set representation.

(2a) Representation by a similar-item set.

We define the following three scoring methods, S2a-I, S2a-II, and S2a-III, for representation by a similar-item set. Examples of these scoring methods are illustrated in Figure 1 (2a).

(S2a-I) Score based on item similarity to input items.

We define the following score $s(r_k)$, based on the collaborative-based similarities $\mathrm{sim}(r_k, a)$ and $\mathrm{sim}(r_k, b)$ between the recommended item r_k and the input items a and b:

$$s(r_k) = \begin{cases} \frac{1}{2}(\mathrm{sim}(r_k, a) + \mathrm{sim}(r_k, b)) & \text{if } \mathrm{set}(a) \cap \mathrm{set}(b) \neq \phi \\ \max(\mathrm{sim}(r_k, a), \mathrm{sim}(r_k, b)) & \text{otherwise} \end{cases} \tag{17}$$

(S2a-II) Score based on the number of items similar to the recommended item.

We define the following score $s(r_k)$, based on the number of items similar to the recommended item r_k:

$$s(r_k) = |\{x | \mathrm{sim}(r_k, x) \geq \theta\}| \tag{18}$$

Here, θ denotes the similarity threshold.

(S2a-III) Score based on the reciprocal of the number of items similar to the recommended item.

This score is in contrast to that in S2a-II. It is based on the assumption that the more the recommended item r_k is restricted to only items similar to the input items a and b, the more strongly the recommended item r_k is related to the input items a and b. Thus, we define the following score $s(r_k)$:

$$s(r_k) = \frac{1}{|\{x | \mathrm{sim}(r_k, x) \geq \theta\}|} \tag{19}$$

Here, θ denotes the similarity threshold.

(2b) Representation by a supporting-user set.

We define the following three scoring methods, S2b-I, S2b-II, and S2b-III, for representation by a supporting-user set. Examples of these scoring methods are illustrated in Figure 1 (2b).

(S2b-I) Score based on the number of common users.

Among users who support the recommended item r_k, we assume that the larger the number of users who support both input items a and b, the more strongly the recommended item r_k is related to the input items a and b. Therefore, we define the following score $s(r_k)$:

$$s(r_k) = |V_{r_k} \cap (V_a \cup V_b)| \tag{20}$$

Here, V_{r_k}, V_a, and V_b denote the supporting-user set for item r_k, and the input items a and b, respectively.

(S2b-II) Score based on the mean of the number of items supported by supporting users.

We define a score based on the mean of the number of items supported by the supporting-user set V_{rk}. We assume that the larger the number of items the user supports, the more reliable the user is. Thus, we define the following score $s(r_k)$:

$$s(r_k) = \frac{1}{|V_{rk}|} \sum_{v_i \in V_{r_k}} |\{x|\text{rating}(v_i, x) \geq \tau\}| \tag{21}$$

Here, τ denotes the threshold for whether the user supports the item.

(S2b-III) Score based on the mean of the reciprocal of the number of items supported by supporting users.

It is based on the assumption that the greater the extent to which the user restricts support to only the recommended item r_k, the better the user supports item r_k. Thus, we define the following score $s(r_k)$:

$$s(r_k) = \frac{1}{|V_{rk}|} \sum_{v_i \in V_{r_k}} \frac{1}{|\{x|\text{rating}(v_i, x) \geq \tau\}|} \tag{22}$$

Based on each scoring method, the system calculates the score for each recommended item. Then the system orders the recommendation list R according to the scores.

4. EXPERIMENTS

We conducted experiments for evaluating serendipity of recommendation lists generated by the item-fusion methods and scoring methods described in Section 3. In Section 4.1, we explain the dataset used for evaluation, feature representation of items, measures and baseline methods for comparison, respectively. After we describe experimental steps in Section 4.2, we show experimental results and discuss them in Section 4.3.

4.1 Experimental setup

4.1.1 Dataset and feature representation of items

We used MovieLens Data Set in the experiments. This dataset consists of 100,000 ratings (1-5) from 943 users on 1682 movies. This dataset has the principal tables shown in Table 1.

Table 1: Contents of MovieLens Data Set

Table type	Table name	Attributes
Item table	u.item	movie id, movie title, release date, video release date, IMDb URL, unknown, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western
User table	u.user	user id, age,, gender, occupation, zip code
Rating table	u.data	user id, item id, rating, timestamp

Followed the tables described in Section 3.1.1, the u.item, u.user and u.data correspond to an item table, user table, and rating table, respectively. Attributes, “movie title” to “Western” of u.item, correspond to item features. Particularly, 18 attributes, “Action” to “Western,” represent item genres, which are given by either $\{0, 1\}$ according to contents of the movie.

According to Section 3.2, we define feature representation of items based on the Table 1 as follows.

(1) Bit-string representation.

We represent an item as a bit string based on the genres of the item. For example, since the movie whose movie id = 1, which is “Toy Story,” corresponds to the third, fourth and fifth genres, {“Animation,” “Children,” “Comedy”}, respectively, it is represented as bit(1) = [001110000000000000]. We also employed the bit variance in the dataset as the weight for each bit w_j in the scoring method S1-II in the experiments.

(2a) Representation by a similar-item set.

We obtain a similar-item set for each item based on the item similarity described in Section 3.1.2. In the experiments, we calculated the item similarity by the collaborative-based similarity. Here, we let the similarity threshold be $\theta = 0.95$.

(2b) Representation by a supporting-user set.

We obtain a supporting-user set for each item based on the calculation shown in Section 3.1.3. We let the threshold be $\tau = 3.0$.

4.1.2 Measures

Our proposed system regards serendipity as important rather than recommendation accuracy. Therefore, we evaluate our sytem from the point of view of how the system can generate serendipitous items.

In the experiments, we introduce *r-unexpectedness* and *r-serendipity* based on the notions of Murakami et al. [7] and Ge et al. [3] presented in Section 2.

First of all, we present the Equation (1) again.

$$UNEXP = RS \backslash PM \tag{23}$$

Here, RS denotes a recommendation list generated by the proposed system. We also employed the following two methods as the primitive prediction methods: PM_{mean}, a prediction method based on the mean ratings, and PM_{num}, a prediction method based on the number of ratings.

The PM_{mean} regards the top-N items with the highest mean ratings as a recommendation item set. The PM_{num} regards the top-N items with the largest number of ratings as a recommendation item set. Finally, we utilize the union of the PM_{mean} and PM_{num} as PM. Thus, the PM includes items whose total number is $2N$.

We introduce *r-unexpectedness* that denotes a ratio of the number of the unexpected items of the top-r ranked recommendation list, and represent it as follows:

$$r\textit{-unexpectedness} = \frac{|UNEXP(r)|}{r} \tag{24}$$

Here, $UNEXP(r)$ denotes $UNEXP$ when the top-r items are provided by the RS.

We also introduce serendipitous item set as follows (note that this is different from the Equation (1)):

$$SERENDIP = UNEXP \cap USEFUL \tag{25}$$

Here, $USEFUL$ denotes useful item set given separately. In the experiments, we gave the usefulness of items based on their mean ratings. Then we regard items equal to or greater than a threshold v as useful items. The $USEFUL$ consists of the useful item set.

In the same way, we introduce *r-serendipity* that denotes a ratio of the number of the serendipitous items of the top-r ranked recommendation list, and represent it as follows:

$$r\text{-}serendipity = \frac{|SERENDIP(r)|}{r} \tag{26}$$

Here, $SERENDIP(r)$ denotes $SERENDIP$ when the top-r items are provided by the RS.

4.1.3 Baseline methods

In order to evaluate serendipity of the proposed system, we compare the system with the following three types of baseline methods: CB_a and CB_b, content-based filtering for input items a and b, CF_a and CF_b, collaborative filtering for input items a and b, and $RAND$, random method.

Here, the CB_a and CB_b provide items in order of content-based similarity to items a and b, respectively. In the experiments, in order to calculate the similarity, we used cosine similarity between feature vectors whose elements denote 18 item genres. The CF_a and CF_b provide items in order of collaborative-based similarity to items a and b, respectively. The $RAND$ provides items selected and ordered at random from the item table.

4.2 Experimental steps

We conducted the experiments by using 1000 pairs of items selected from the item table at random. Given an item pair (a, b), the experimental steps are as follows:

step 1 Generate a recommendation list R by each item-fusion method (see Section 3.3) for the item pair (a, b).

step 2 Make a ranking list R' for the recommendation list R by each scoring method, i.e., S1-I, S1-II, S2a-I, S2a-II, S2a-III, S2b-I, S2b-II, and S2b-III (see Section 3.4), and by each baseline method.

step 3 Obtain *r-unexpectedness* and *r-serendipity* for each R'.

4.3 Results and discussion

In this section, we show experimental results and discuss them. In the experiments, we used $N = 50$, which is the number of items provided by each primitive prediction method PM, and $r = 20$ for *r-unexpectedness* and *r-serendipity*. Before the experiments, we conducted preliminary experiments under conditions of $N = \{10, 20, \ldots, 100\}$ and $r = \{10, 20, \ldots, 100\}$. Note that we found that relative relationship between scoring methods did not significantly depend on the conditions. We also used $v = 3.0$ that is the threshold for whether the item is useful.

4.3.1 Comparison with baseline methods

Figure 2 shows the mean *r-unexpectedness* and *r-serendipity* by the scoring methods and baseline methods. The horizontal axis denotes *r-unexpectedness* and the vertical axis denotes *r-serendipity*. Note that, on the baseline

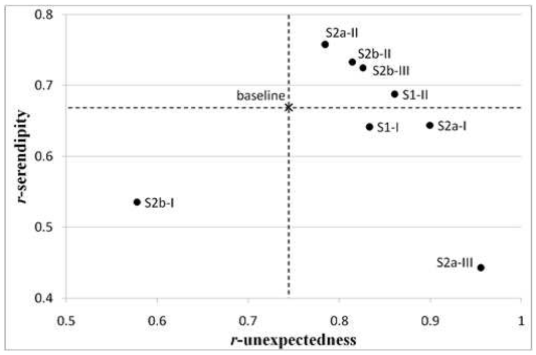

Figure 2: Mean r-unexpectedness and r-serendipity by scoring methods and baseline methods

methods, this figure shows the mean of them obtained by three types of baseline methods described in Section 4.1.3.

We found that S1-II, S2a-II, S2b-II, and S2b-III produced significantly higher *r-unexpectedness* and *r-serendipity* than the baseline methods produced ($p < 0.01$). Notably, since S1-II, S2a-II, S2b-II, or S2b-III can yield high serendipity for each item-fusion method, (1), (2a), and (2b), we believe that the proposed system works effectively by using any feature representation of items. We discuss these cases in detail in the next section.

On the other hand, *r-unexpectedness* by S1-I, S2a-I, and S2a-III are less than ones by the baseline methods while *r-serendipity* by them are higher than ones by the baseline methods. Particularly, the *r-unexpectedness* by S2a-III is much less than one by the baseline methods. In addition, both *r-unexpectedness* and *r-serendipity* by S2b-I are less than ones by the baseline methods.

As we described in Section 3.4 (2b), the S2b-I calculates scores based on the number of users who support both the recommended item r_k and input item a or b. In the experiments, we employed the PM_{num}, which is based on the number of ratings, as one of the primitive prediction methods. Since the items rated by many users can be easy to be predicted, the S2b-I yields low *r-unexpectedness*.

As we described in Section 3.4 (2a), we assume that the more the recommended item r_k is restricted to only items similar to the input items a and b, the more strongly the recommended item r_k is related to the input items a and b. However, this result shows that the assumption is not correct. Since S2a-II, which is opposite to the S2a-III, showed higher *r-serendipity*, we found that we should employ S2a-II for the purpose of improving serendipity.

4.3.2 Comparison of serendipity by different relationship between input items

We conducted an additional experiment to analyze difference between S1-II, S2a-II, S2b-I, and S2b-II, which yielded higher serendipity. We analyzed the difference of *r-serendipity* depending on the relationship between input items. We focus on the relationship between input items shown in Table 2. We grouped 1000 item pairs used in the experiments by the relationship between input items. As shown in Figure 3, we obtained *r-serendipity* for each group.

We found that S2a-II could produce significant high serendipity ($p < 0.01$) in all cases except for mean-HH and num-HH. We also found that S2b-II and S2b-III in case of

Table 2: Relationship between input items

Symbol	Relationship between input items
sim-H	items with high content-based similarity
sim-L	items with low content-based similarity
mean-HH	items with high mean ratings
mean-HL	an item with high mean ratings and an item with low mean ratings
mean-LL	items with low mean ratings
num-HH	items with many ratings
num-HL	an item with many ratings and an item with few ratings
num-LL	items with few ratings

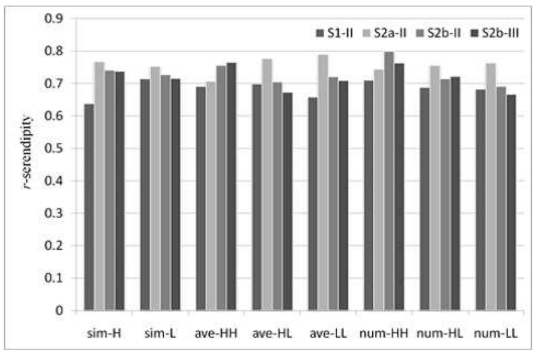

Figure 3: Mean r-serendipity by scoring methods depending on combinations of input items

mean-HH, and S2b-II in case of num-HH could produce significant high serendipity ($p < 0.01$), respectively. On the basis of this result, we can expect that effectiveness of the system can be improved by introducing a switching method that dynamically switches scoring methods depending on relationship between user-input items.

Furthermore, we are interested in that S2a-II can produce high serendipitous items by using unpopular items, i.e., items with low or few ratings, as materials for item-fusion. We believe that we can expect that system usage can be broadened by using such items effectively.

5. CONCLUSION

In this paper, we proposed a Fusion-based Recommender System that aims to improve the serendipity of recommender systems. The system is based on the novel notion that the system finds new items, which have the mixed features of two user-input items, produced by mixing the two items together. The system consists of item-fusion methods and scoring methods. We proposed three item-fusion methods and eight scoring methods on the basis of the item-fusion methods.

Experimental results showed that S1-II, S2a-II, S2b-II, and S2b-III produced higher serendipitous items than baseline methods produced. This paper describes these methods and gives experimental results. The results also showed that S2a-II could produce high serendipity in most cases. We also found that S2b-II and S2b-III in case of using input items with high mean ratings, and S2b-II in case of using input items with high ratings could produce high serendipity, respectively. These results suggest that effectiveness of the system can be improved by introducing a switching method that dynamically switches scoring methods depending on relationship between user-input items.

In the future, we would like to analyze the results qualitatively. We want to know what kind of item pair yields what kind of recommendations. Although we used static ratings for judging useful items, we will conduct experiments with real users for evaluating serendipity. We plan to implement other feature representation of items, e.g., by a tag set and feature vectors. We also plan to design user interface of the system that makes the system usage more effective.

6. ACKNOWLEDGEMENT

This work was supported by Grant-in-Aid for Young Scientists (B) (23700132).

7. REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

[2] Shlomo Berkovsky and Jill Freyne. Group-based recipe recommendations: analysis of data aggregation strategies. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 111–118. ACM, 2010.

[3] Mouzhi Ge, C. Delgado-Battenfeld, and D. Jannach. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 257–260. ACM, 2010.

[4] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[5] Y. Hijikata, T. Shimizu, and S. Nishida. Discovery-oriented collaborative filtering for improving user satisfaction. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 67–76. ACM, 2009.

[6] G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[7] Tomoko Murakami, Koichiro Mori, and Ryohei Orihara. Metrics for evaluating the serendipity of recommendation lists. *New Frontiers in Artificial Intelligence*, pages 40–46, 2008.

[8] Paul Resnick, N. Iacovou, M. Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[9] Paul Resnick and H.R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[10] Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, New York, New York, USA, 2001. ACM.

[11] Cai-Nicolas Ziegler, Georg Lausen, and Lars Schmidt-Thieme. Taxonomy-driven computation of product recommendations. In *Proceedings of the Thirteenth ACM conference on Information and knowledge management - CIKM '04*, page 406, New York, New York, USA, 2004. ACM Press.

[12] C.N. Ziegler, S.M. McNee, J.A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32, New York, New York, USA, 2005. ACM.