Latent Factor Models and Aggregation Operators for Collaborative Filtering in Reciprocal Recommender Systems

James Neve
james.neve@bristol.ac.uk
University of Bristol
Bristol, England

Ivan Palomares
i.palomares@bristol.ac.uk
University of Bristol & The Alan Turing Institute
Bristol, England

ABSTRACT

Online dating platforms help to connect people who might potentially be a good match for each other. They have exerted a significant societal impact over the last decade, such that about one third of new relationships in the US are now started online, for instance. Recommender Systems are widely utilized in online platforms that connect people to people in e.g. online dating and recruitment sites. These recommender approaches are fundamentally different from traditional user-item approaches (such as those operating on movie and shopping sites), in that they must consider the interests of both parties jointly. Latent factor models have been notably successful in the area of user-item recommendation, however they have not been investigated within user-to-user domains as of yet. In this study, we present a novel method for reciprocal recommendation using latent factor models. We also provide a first analysis of the use of different preference aggregation strategies, thereby demonstrating that the aggregation function used to combine user preference scores has a significant impact on the outcome of the recommender system. Our evaluation results report significant improvements over previous nearest-neighbour and content-based methods for reciprocal recommendation, and show that the latent factor model can be used effectively on much larger datasets than previous state-of-the-art reciprocal recommender systems.

CCS CONCEPTS

• **Information systems → Social recommendation**; *Decision support systems*; *Information retrieval*; Social networks.

KEYWORDS

Reciprocal Recommender Systems; Latent Factor Models; Online Dating; Preference Aggregation

ACM Reference Format:
James Neve and Ivan Palomares. 2019. Latent Factor Models and Aggregation Operators for Collaborative Filtering in Reciprocal Recommender Systems. In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347026

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347026

1 INTRODUCTION

Reciprocal Recommender Systems (RRS) connect users with other users as opposed to recommending items to users [1]. They have a variety of applications in areas with a significant impact on society, such as online dating, recruitment and social networking platforms.

RRSs are fundamentally different from conventional recommender systems, and these differences go beyond the scope of what is being recommended. Where recommender systems consider a unidirectional preference relationship - how much a user is likely to prefer one item to another - an RRS must consider bidirectional preference relationships. A successful recommendation in this setting is one where both the recommended user appeals to the target user, and the target user appeals to the recommended user. An RRS must also consider additional factors that aren't present in conventional recommender systems, such as the consequences of recommending a popular user to an unpopular user, or the consequences of recommending a passive user to another passive user [2]. They are therefore inherently more complex to design and build. Most RRSs in the literature are aimed at online dating services, but there are also examples applications in recruitment [5] and social networks [6].

There are a number of implementations of RRSs in the literature, including content-based filtering based approaches on user profiles [3] and nearest-neighbor reciprocal collaborative filtering [4]. However, content-based filtering is limited in online dating, where physical appearance is often the main factor taken into consideration by users deciding who to interact with [7]. Nearest neighbour-based collaborative filtering also has a number of well-known limitations in terms of the computational complexity of calculating similarities between all pairs of users [8].

Many modern recommender systems take advantage of Latent Factor (LF) models to make recommendations with successful results (e.g. [9], [10]). LF algorithms use matrix factorization to generate vectors of factors that describe correlations in the original preference matrix, thereby condensing correlations between preferences into a much smaller number of factors. This often results in better predictions on very large, sparse matrices, which is often the case in online dating, where most users only indicate a preference for an unduly small number of other users. To the extent of our knowledge, LF-based solutions have never been investigated and applied to RRSs, in spite of the clear potential these models have as evidenced by their prevalence in the landscape of traditional recommender systems. Therefore, how LF models can influence the process of making effective reciprocal recommendations is an unaddressed research question in the field.

Previous approaches to reciprocal recommendation, including RECON and RCF, generate two preference scores and combine them

by using the harmonic mean (e.g. [3], [4]). The choice to use the harmonic mean seems arbitrary, and existing studies do not provide a clear motivation for this choice being made. Besides averaging functions, there are many other aggregation functions, such as uninorms [11], that could be used effectively to combine the two preference scores, but their potential has not been explored in this setting as of yet.

To overcome the aforementioned limitations inherent in previous RRS solutions, in this paper we firstly present a novel reciprocal recommender system based on latent factor models. We consider, without loss of generality, the application of our proposed model, named Latent Factor Reciprocal Recommender (LFRR) herein, to the domain of online dating. This approach requires two latent factor models to be generated - one that determines the preference of individual male users for female users, and one that determines the preference of individual female users for male users. Many previous approaches used datasets of fewer than ten thousand instances ([4], [12]), which is not representative of the size of many real-life applications for RRSs. Based on real data provided by *Eureka Inc.*[1], we demonstrate that our method is effective on datasets containing hundreds of thousands of users and millions of instances. We also test a number of different aggregation functions to combine the two unidirectional preference scores into a single reciprocal score and investigate their effect on the resulting recommendations.

The evaluation of this system was conducted on a popular Japanese online dating platform, *Pairs*[2]. Pairs users must *Like* each other before they can communicate using messages, and we used this as our main indicator of preference. On a dataset containing approximately 1M users and 10M interactions, we found that the latent factor-based recommender system outperformed a successful content-based RRS and a successful nearest neighbour-based RRS. We also tested training times on datasets of various sizes, and found that LFRR gave similar performance to RCF and, unlike RCF, was able to generate recommendations on a large dataset in real time.

This paper's contributions are threefold. 1) We present a novel method for reciprocal recommendation based on latent factor models, which have been relatively successful compared to state-of-the-art nearest neighbour-based RRS models in a wide variety of domains. 2) We demonstrate that our model has similar performance to current solutions, and is able to generate recommendations much more efficiently. 3) We evaluate a number of different aggregation functions and demonstrate that the choice of aggregation function has a significant impact on the performance of the RRS.

2 RELATED WORKS

This section overviews some related works on collaborative filtering (and in particular LF models), RRSs and aggregation functions.

2.1 Latent Factor Models in Collaborative Filtering

Collaborative filtering uses similarities between the actions or preferences of users on items to determine likely preferences and therefore generate recommendations. The most elementary form of collaborative filtering is nearest neighbour algorithms, which compute the similarity between pairs of users or items [13] and recommend items based on the preferences of very similar users. Despite their promising results on smaller datasets (e.g. [14]), these methods become increasingly intractable as the number of users grows into the millions [15], as is the case with many internet services.

Latent factor models for conventional recommender systems have been frequently and thoroughly investigated in the literature, and have been extensively studied. Koren, Bell and Volinsky provide an overview of the various modern methods of computing latent factor models [16]. Matrix factorization techniques such as Singular Value Decomposition (SVD) can be used to extract vectors that describe the user-item matrix from correlations between similar user interactions, and similar items being interacted with. Mathematically, these techniques can only be used on complete matrices. However, it is unrealistic to assume there is complete information about every user's opinion on every item. Previous works used various methods to estimate the missing values in the user-item matrix (e.g. [17]), but modern methods initialize latent factor vectors with random values from a distribution, and then they learn the correct values via optimisation techniques such as gradient descent (e.g. [18], [9]). Latent factor models have been relatively successful, and competitions such as the Netflix Prize have demonstrated that on conventional recommender systems, they are superior to nearest neighbour implementations [19]. However, their potential in the paradigm of reciprocal user-to-user recommendation is still unexplored.

2.2 Reciprocal Recommender Systems

The objective of RRSs is to create a good match between two users, not only recommending one user to another [1]. Because of the bidirectional nature of RRSs, they benefit from separate study to conventional recommender systems.

Historically, RRSs have received comparatively little attention. A widely cited work by Pizzato et al. proposes RECON, a content-based filtering algorithm for reciprocal recommendation [3]. RECON extracts implicit preferences by looking at attributes in common amongst those users whom a given user has sent messages to. It then calculates preference scores from a given user u_i to another user u_j and vice-versa using implicit preferences as a base, and these two users'preference scores are then combined into a reciprocal score by calculating the harmonic mean of both. RECON's evaluation demonstrates that it performs better than explicit search in creating satisfactory matches for the two end users involved.

Reciprocal Collaborative Filtering (RCF), a collaborative filtering based algorithm, was able to significantly improve on RECON's results [4]. RCF also calculates scores for multiple users, but unlike RECON, it uses a nearest-neighbor collaborative filtering approach, where user a's likelihood to like user b is estimated from their similarity to other users who liked user b. RCF's evaluation proved the algorithm to significantly outperform RECON, hence it is deemed as the current benchmark against which new reciprocal recommender systems are compared.

RCF is commonly used as a baseline for testing collaborative filtering-based RRSs on small datasets (e.g. [2], [12]). It is a nearest neighbour-based collaborative filtering algorithm that retrieves nearest neighbours based on other users who have liked the user

[1] https://eure.jp/
[2] https://pairs.lv

in question. For example, to calculate the likelihood that a user a will like a user b, RCF calculates the similarity between a and other users n who have previously liked b. A high degree of similarity implies that a will probably like b, and vice-versa. The similarity between users a and n, based on the well-known Jaccard Coefficient, is defined as:

$$Similarity_{a,n} = \frac{IFrom_a \cap IFrom_n}{IFrom_a \cup IFrom_n} \tag{1}$$

Where

$$IFrom_a = \{b : b \text{ has received a Like from } a\} \tag{2}$$

$$ITo_b = \{a : a \text{ has sent a Like to } b\} \tag{3}$$

In the following algorithm, let a be the current user. Then:

$$Candidates = \{b : b \text{ is a possible recommendation for } a\} \tag{4}$$

In the online dating domain, this may be the entire set of opposite-sex users, or there may be restrictions (for instance, opposite-sex users within a certain geographical distance from the current user). *Recs* is the output of the algorithm, a set of pairs such that:

$$Recs = \{(b, reciprocalScore_{a,b})\} \tag{5}$$

Where $reciprocalScore_{a,b} \in [0, 1]$ is the aggregation of two unidirectional preference scores between current user a and recommendation candidate b.

The full algorithm is described in Algorithm 1. Note that the normalising factor in lines 11 and 12 is necessary because the number of neighbours of a will depend on the number of users who have liked b, and will therefore vary from user to user.

Algorithm 1 RCF Algorithm for target user a

1: $Recs \leftarrow \emptyset$
2: **for all** $b \in Candidates$ **do**
3: $score_{a,b} \leftarrow 0$
4: $score_{b,a} \leftarrow 0$
5: **for all** $n \in ITo_b$ **do**
6: $score_{a,b} \leftarrow score_{a,b} + Similarity_{a,n}$
7: **end for**
8: **for all** $n \in ITo_a$ **do**
9: $score_{b,a} \leftarrow score_{b,a} + Similarity_{b,n}$
10: **end for**
11: $score_{a,b} \leftarrow \frac{score_{a,b}}{|ITo_b|}$
12: $score_{b,a} \leftarrow \frac{score_{b,a}}{|ITo_a|}$
13: $reciprocalScore_{a,b} \leftarrow \mathbf{Agg}(score_{a,b}, score_{b,a})$
14: $Recs \leftarrow Recs + (b, reciprocalScore_{a,b})$
15: **end for**
16: **return** *Recs*

The complexity of RCF depends on the computation of the similarity of user a with the n users who have liked user b. If all users have an average of k interactions with other users, the process will require approximately nk computations. However, if by chance the n users have significantly more than an average number of interactions, the computation could take much longer than this, which is deeply undesirable in an online service. The unpredictability in the efficiency of RCF is a weakness that we hoped to address with LFRR: latent factor model preference score computations are always constant time.

The **Agg** function on line 13 is used to combine two unidirectional preference scores into a single reciprocal preference score, and represents one of the four aggregation functions described in section **2.3**.

2.3 Aggregation Operators

Classical means are the most widely-used types of aggregation functions. Examples of these include the arithmetic mean, harmonic mean and geometric mean, all of which are widely used in different fields such as multi-criteria and group decision making (e.g. [20]). Aggregation functions have also been applied successfully to group recommendation in the past [21]. In the domain of user-to-user recommendation based on combining two preference scores, only the two-input version of the function is required. Without loss of generality, we focus our discussion on aggregation operators in the unit interval.

We use the three *Pythagorean means* as our basic aggregation operators. The *arithmetic mean* M, the *geometric mean* G and the *harmonic mean* H for two values x_1 and $x_2 \in [0, 1]$ are defined as follows

$$M(x_1, x_2) = \frac{x_1 + x_2}{2} \tag{6}$$

$$G(x_1, x_2) = \sqrt{x_1 x_2} \tag{7}$$

$$H(x_1, x_2) = \frac{2x_1 x_2}{x_1 + x_2} \tag{8}$$

The AM is the most commonly used aggregation function in all fields. In the case of two values, unlike other aggregation functions, it is not biased towards the higher or lower value. Intuitively, therefore, the AM may not be appropriate for an RRS: the presence of a single very high score does not compensate for a low score, as both parties need to accept the recommendation for it to be deemed as *reciprocal*. In contrast, the output of the geometric mean is significantly closer to the smaller number than the larger one. This is often desirable in the case of an RRS for the reasons outlined above: two very similar, moderate scores are more likley to produce a successful match than one very high score and one very low score. The result of the HM is also closer to the smaller of the two values, and is often used in situations involving rates and ratios. To the extent of our knowledge, it is the only aggregation function to have been used in RRSs [3][4].

We also use the cross-ratio uninorm as an example of aggregation operator with mixed behaviour that might prove useful in the case of aggregation for RSSs. Uninorms [11] represent generalisations of t-norm and t-conorms. A t-norm is a mapping $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$, and a t-conorm is a mapping $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$. Both are associative and symmetric. However, the t-norm has a neutral element of 1, and the t-conorm has a neutral element of 0. Uninorms have a neutral element that can lie in the interval $]0, 1[$. The cross-ratio uninorm [22] is defined as:

$$U(x_1, x_2) = \frac{x_1 x_2}{x_1 x_2 + (1 - x_1)(1 - x_2)} \tag{9}$$

In the case of the classical means applied to two values, the result of aggregation always lies somewhere between the two values being aggregated. However, uninorms are different in the respect that: (i) two low values are aggregated to produce a lower value (*conjunctive behaviour*); (ii) two high values are aggregated to produce a higher value (*disjunctive behaviour*); (iii) a high and a low value are aggregated to a value that lies in between both (*averaging behaviour*).

This is known as the *full reinforcement property*. We hypothesise that the full reinforcement property in uninorms might be useful for RRSs because intuitively, two relatively high positive unidirectional preference scores are more likely to produce a successful match than one extremely high preference score and one lower one. In this case, the uninorm will produce a higher positive aggregated score than the classical means.

3 METHODOLOGY

In this section, we present LFRR, a novel reciprocal recommender system approach based on latent factor models. In order to account for the bidirectional nature of reciprocal recommendation, we learn two latent factor models, one to indicate male users' preferences for female users, and one to indicate female users' preferences for male users. We then combine these two preference metrics using aggregation operators to indicate which are likely to be a positive match, and therefore which recommendations to display.

LFRR is designed for use with online dating, and in particular, it has been evaluated for its use on *Pairs*, a popular Japanese online dating platform. *Pairs* users must send each other a *Like* before they are able to communicate with each other by messages. Much of the message data is ambiguous, as users often quickly exchange contact details and move their communications off the service. However, users who succeed in achieving a large number of mutual *Likes* - *Matches* - are more likely to subsequently find a relationship. Our objective is therefore to maximise the number of *Matches* as a proxy for helping the largest possible number of users to succeed.

3.1 Latent Factor Model for Reciprocal Recommendation

In this section, we describe the implementation of a new algorithm, *Latent Factor Reciprocal Recommender* (LFRR), which uses latent factor models to compute two unidirectional preference scores, that are subsequently aggregated into a single reciprocal preference score.

Pairs has over 10 million users, and most users will only see a small fraction of other users, and give an opinion on an even smaller fraction. We therefore decided that imputing unknown data was likely to introduce a great deal of inaccuracy into the model, and that using a learning algorithm to generate latent factors using only the known data was likely to produce much more accurate results.

Machine learning-driven latent factor models initialize latent factor vectors to random values from a distribution, and then aim to minimise the error on the known ratings. Specifically, the likelihood of user a liking user b is calculated by the dot product between user a's feature vector p_{u_a} and user b's feature vector q_{u_b}. Using the training set of known ratings $R = (r_{u_a,u_b})_{rows \times columns}$, the error can be calculated with regularization parameter λ.

$$\min_{q,p} \sum_{(u_a,u_b) \in R} (r_{u_a u_b} - q_{u_b}^T p_{u_a})^2 + \lambda(||q_{u_b}||^2 + ||p_{u_a}||^2) \tag{10}$$

It can then be minimised by gradient descent. Because the Pairs dataset contains sparse, explicit data, Stochastic Gradient Descent was used for the minimisation. This tends to perform better on explicit data than the other common method, Alternating Least Squares, which tends to give better results with dense, implicit data [8]. Stochastic gradient descent calculates the error for each individual datapoint.

$$e_{u_a u_b} = r_{u_a u_b} - q_{u_b}^T p_{u_a} \tag{11}$$

It then modifies the relevant feature vectors in the negative direction of the gradient proportional to the learning rate, γ

$$q_{u_b} \leftarrow q_{u_b} + \gamma(e_{u_a u_b} p_{u_a} - \lambda q_{u_b}) \tag{12}$$

$$p_{u_a} \leftarrow p_{u_a} + \gamma(e_{u_a u_b} q_{u_b} - \lambda p_{u_a}) \tag{13}$$

The datapoint order is randomised, and this process is repeated for a number of epochs until the feature vectors are stable. They can then be used to make predictions about how likely user a is to like user b by the dot product of their feature vectors.

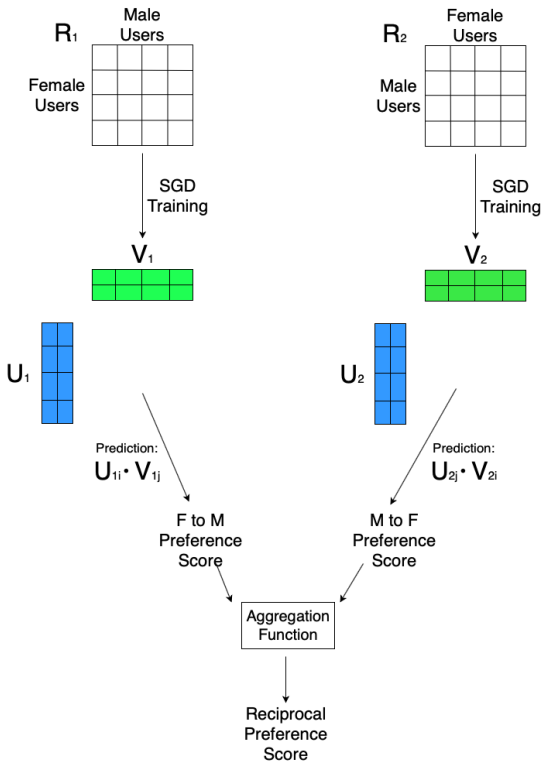

Figure 1: LFRR Visualisation

Given matrices for a trained female-male preference latent factor model U_1 and V_1, and the same matrices vice-versa for male-female

preferences U_2 and V_2, let the vector representing the row for a user a in matrix U_1 be denoted by $U_{1,a}$. Then the prediction algorithm for LFRR is described in Algorithm 2. (The other variables follow the same conventions as in Algorithm 1.)

Algorithm 2 LFRR Predictions for two users a and b

INPUT: Trained feature matrices U_1, U_2, V_1, V_2

$Recs \leftarrow \emptyset$
for all $b \in Candidates$ **do**
  if a is female **then**
    $score_{a,b} \leftarrow U_{1,a} \cdot V_{2,b}^T$
    $score_{b,a} \leftarrow U_{2,b} \cdot V_{1,a}^T$
  else
    $score_{a,b} \leftarrow U_{2,a} \cdot V_{1,b}^T$
    $score_{b,a} \leftarrow U_{1,b} \cdot V_{2,a}^T$
  end if
  $reciprocalScore_{a,b} \leftarrow \mathbf{Agg}(score_{a,b}, score_{b,a})$
  $Recs \leftarrow Recs + (b, reciprocalScore_{a,b})$
end for
return $Recs$

As is evident from the algorithm description, computing a reciprocal score requires only a single dot product, and is therefore guaranteed to be linear time, regardless of the number of interactions from that particular user.

4 EVALUATION

In this section, we first describe the dataset used in the study. We then describe the experiments performed and the metrics used to evaluate them. We display the results of these experiments, and compare the evaluations of RCF and LFRR with four different aggregation functions.

4.1 Dataset Description

Pairs is an online dating service primarily based in Japan, with branches in Korea and Taiwan. For this paper, our experimental study focuses on the Japanese service, where the vast majority of users are. By default, Pairs displays users of a similar age and living in a similar area to the active user. Users can search for other users using attributes such as body type and smoker or non-smoker. The dataset we used for our experimental evaluation contained only interactions between members of opposite sexes, and we use this assumption in designing our algorithm. This is consistent with previous RRS research [3][4][2].

After searching, users can view other users' profiles, which have both selectable attributes such as age and income, and a free text intoduction. Users can also provide pictures of themselves. When a user finds another user they want to communicate with, they can send a *Like*. The receiving user sees a notification, and can choose to either return the *Like* or send a *Nope*, indicating that they are not interested. Once two users have *Liked* each other, they can exchange messages and arrange to meet. This process is visualised in Figure 2.

The service therefore has various indicators of preference that are saved to a database. In increasing strength of interest: 1) viewing a profile; 2) sending a *Like*; 3) sending an initial message; 4) a message exchange; 5) arranging to meet. In most previous studies on reciprocal recommender systems, there was no system of *Likes* and messages were used as an indicator of preference, with a reply being used as a indicator of mutual preference (e.g. [3]). There are two reasons why a *Like* might be a more useful preference indicator for recommender system design. Firstly, *Likes* have a binary value - it was either sent or not. A message may have positive, strongly positive or even negative value, and extracting this value from free text might be a daunting task. Secondly, a *Like* and a response takes only the effort required to press the button, whereas users may wish to express preference by sending or replying to a message but decide they don't have the time.

Note that the number of *Likes* a user can send is limited by their subscription level, which also adds to the overall sparse nature of the user-user data. It is not possible for users to negatively impact the recommender system by sending *Likes* indiscriminately to users they have no interest in.

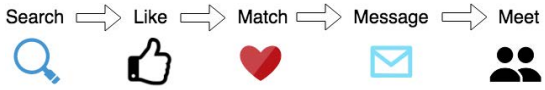

Figure 2: The Pairs usage flow

4.2 Experimental Setup

The data for our evaluation was provided by *Pairs* online dating site, surpassing 10M users at the time of this writing. *Pairs* has over 8 years of data. As described in Section **1**, users express preference for each other by sending a *Like*. As of the time of this writing, users send a total of approximately 9 million *Likes* per week.

Most datasets that have been used to test collaborative filtering-based reciprocal recommender systems in the past have been relatively small in comparison, comprising only a few thousand interactions at most. To the best of our knowledge, there is no example in the literature of a reciprocal recommender system being tested on a dataset of the size of the *Pairs* dataset.

We chose to sample data from varying time periods, from one day to 3 months of interactions. Previous marketing analysis of Pairs user data indicates that the vast majority of users either find a relationship or cease using the site during this time, and that long-term users often change their preferences. 3 months was therefore the longest period of time we consider useful for generating recommendations for data in the online dating field.

We also made several other limitations on the data we selected for our experiments, which are outlined below:

- Users who live in Tokyo and the surrounding areas. These users represent a significant majority of Pairs users.
- Users between 18 and 30 years of age, for the same reason as above. Users outside this age range are outliers in the user base.
- Users who have sent at least 10 *Likes*.

Limitation **3** is because both LFRR and RCF suffer from the *Cold Start Problem* [23]. To the best of our knowledge, there are not effective solutions to the Cold Start Problem for RCF. There are

some general solutions to this problem for latent factor models (e.g. [24]) but these are outside the scope of the work on this paper. We therefore focus on users with enough data to make effective recommendations using both algorithms.

4.3 Evaluation Metrics

Our experiments evaluate the algorithms run on two different factors: effectiveness and efficiency.

4.3.1 Effectiveness Evaluation. We chose to use offline evaluation to measure the effectiveness of our algorithm. There were two reasons for this. The first is that we wanted to test two algorithms with four different aggregation functions each - a total of eight different combinations. Offline evaluation allowed us to quickly demonstrate that there was a difference in effectiveness between these different combinations without a protracted online test. The second reason is that offline evaluation allowed us to test the efficiency of the different algorithms without having to expose real users to unnecessarily slow algorithms. However, studies have demonstrated that offline evaluation is not always representative of real recommendation results [28], so in a future work, it is necessary to confirm these results by online evaluation.

We ran cross-validation on 2000 pairs of users. 1000 of these pairs had *Matched* - that is to say, they had both sent each other a *Like*, thereby expressing preference for each other. In 1000 of these pairs, at least one of the two users had *Noped* one of the other users - that is to say, explicitly expressed a negative preference.

Recall that the output of each of the two algorithms - RCF and LFRR - is a reciprocal preference score representing the likelihood that the two users will both express a preference for each other. The offline evaluation was performed by running the algorithms with a threshold representing the preference score required for two users to like each other. A preference score at or above the threshold for two users who have *Matched* is considered a true positive. A preference score below the threshold for two users where one has *Noped* the other is considered a true negative. The two algorithms were evaluated in this way across a number of thresholds so an ROC curve could be drawn based on the number of true positives and false positives.

Studies have shown that accuracy is sometimes not a representative metric for offline evaluation of the results of recommender systems [25]. We therefore focus on precision and recall as effectiveness metrics. Precision is the proportion of the results that are true positive as compared to the number of false positives. Where RL is defined as the set of users who were recommended each other and sent a *Like* to each other, and RN is the set of users who were recommended to each other but at least one of them sent a *Nope*, it is defined as:

$$Precision = \frac{|RL|}{|RL| + |RN|} \tag{14}$$

Precision is considered particularly important for recommender systems because users lose trust in systems where a high number of unsatisfactory recommendations are displayed.

Recall is defined as the proportion of total positive results that are returned by the algorithm. A low recall indicates a small total number of recommendations, and therefore a high chance that a returning user will see the same recommendations repeatedly. Where R is the set of total recommended users, recall is defined as:

$$Recall = \frac{|RL|}{|R|} \tag{15}$$

F1 Score is a combination of precision and recall commonly used in machine learning as a measure of the overall effectiveness of the system. It is defined as:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{16}$$

4.3.2 Efficiency Evaluation. To the best of our knowledge, all RRSs in the literature work by calculating a reciprocal preference score between pairs of users (e.g. [4]). They generate recommendations by calculating reciprocal preference scores between a user a and every candidate user x, ordering users in descending order of score, and then taking the top N users as recommendations. There are therefore two important metrics to consider where efficiency is concerned:

(1) The time taken to calculate a single reciprocal preference score between two users
(2) The time taken to generate a list of N recommendations by calculating the reciprocal preference scores between a given user A and every other user in the dataset

We consider these two metrics separately, because it is easy to imagine situations where an RRS which could perform (1) efficiently might still be useful (for example, by displaying to the active user the likelihood that another user might be a good match), even in the absence of the ability to perform (2) in reasonable time.

A significant difference between RCF and LFRR is that LFRR requires training of the latent factor matrices U and V, whereas RCF does not rely on any pre-training. We therefore also list the training times for LFRR. These results are presented with the caveat that training times generally do not significantly affect the usefulness of the model unless training takes more than a few hours.

4.4 Effectiveness Results

We show graphs based on cross-validation from one week of data. Figure 3 displays results from various aggregation functions used to aggregate results from the RCF algorithm. Figure 4 shows results from the same aggregation functions used on the LFRR algorithm.

In the case of the RCF algorithm, as is evident from the area under the ROC curve, the HM significantly outperforms the AM and GM. More interestingly, we found that the cross-ratio uninorm outperforms the state-of-the-art RCF based on harmonic mean. This is consistent with our hypothesis that, because the HM and uninorm penalise situations where the two aggregation inputs differ strongly from each other, they are likely to perform better in the online dating domain, where two relatively high scores are more likely to result in a match than a very high and very low score.

The difference between the performance of the aggregation functions is less evident on the LFRR model, where the AM, GM and HM are much closer together. The uninorm performance is significantly different. The shape of the cross-ratio uninorm on the LFRR algorithm can be explained by the mixed behaviour of uninorm functions. The threshold passing the neutral element of the

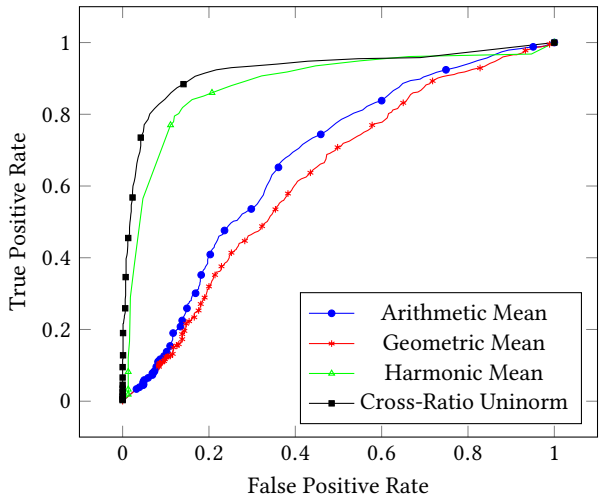

Figure 3: ROC curve obtained for each aggregation function considered in the RCF model.

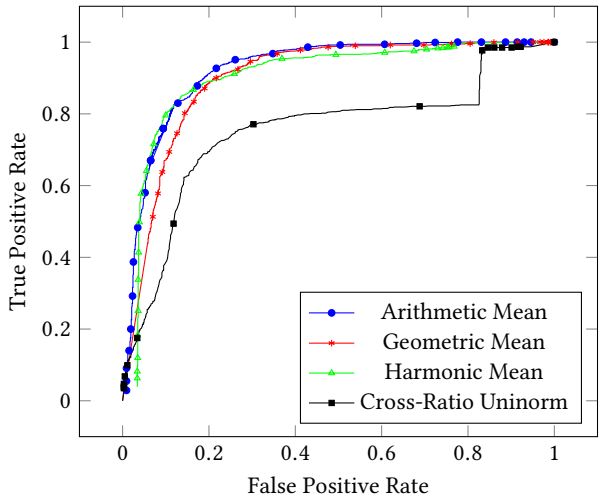

Figure 4: ROC curve obtained for each aggregation function considered in the LFRR model.

aggregation function causes it to move from averaging behaviour to disjunctive behaviour, and can cause a sharp increase in the number datapoints classified as positive.

The best F1 scores with their corresponding precision and recall from each aggregation function are displayed in table 1 for both RCF and LFRR. Based on the best F1 score, the harmonic mean consistently gives high precision, which is ideal for recommender systems in online dating: users are more likely to trust a system that gives them a high proportion of very good matches. However, in the case of RCF, the cross-ratio uninorm gives both a higher precision and higher F1 score. The offline evaluation shows that compared to RCF, LFRR is more consistent across different aggregation functions. However, using the optimal aggregation function for each of algorithm (the cross-ratio uninorm in the case of RCF and the harmonic mean in the case of LFRR), there is no significant

Algorithm	Precision	Recall	Best F1 Score
RCF (Arithmetic Mean)	0.58	0.86	0.69
RCF (Geometric Mean)	0.55	0.90	0.68
RCF (Harmonic Mean)	0.83	0.84	0.84
RCF (Uninorm)	0.84	0.90	0.87
LFRR (Arithmetic Mean)	0.81	0.92	0.87
LFRR (Geometric Mean)	0.83	0.86	0.85
LFRR (Harmonic Mean)	0.86	0.85	0.86
LFRR (Uninorm)	0.54	0.98	0.70

Table 1: Results based on best F1 score from each aggregation function tested applied to RCF and LFRR

difference between the performance of the two algorithms based on offline evaluation. However, the evaluation does demonstrate a significant difference between the different aggregation functions for each algorithm. In the domain of online dating, depending on the algorithm used, the harmonic mean or (in the case of RCF) consistently gives the best performance. However, in situations where optimising recall is desirable (such as a social network where recommending a large number of possible friends might not diminish trust in the system), the arithmetic mean might be desirable.

In summary, not only we found that our proposed LFRR algorithm outperforms RCF in three out of four configurations of the fusion operator being used. We also discovered a new setting for RCF - based on uninorm operators - that outperforms the harmonic mean-based approach adopted so far in existing works.

4.5 Efficiency Results

To the best of our knowledge, a collaborative filtering-based reciprocal recommender system's running time has never been measured on dataset with a large number of users. We measure two aspects for both the RCF and LFRR algorithms: the time taken to calculate a reciprocal preference score for a single pair of users, averaged over 100 pairs of users, and the time taken to generate a recommendation list by calculating the scores for all candidate users and ordering them in descending order. Because LFRR requires a training step, we also present the training time on the same datasets, with the caveat that a long training time doesn't have a significant impact on the algorithm's usefulness as compared to the time taken to generate recommendations. We measure the size of the dataset based on the number of interactions, which is the factor that has the highest impact on the time taken to generate recommendations and to train LFRR. To generate the datasets, an equal number of male and female users were sampled from the same dataset used to measure the effectiveness of the algorithms, with the restrictions detailed in section **4.1**.

Size	RCF Score	RCF List	LRFF Score	LFRR List
10^3	0.003	1.75	1×10^-5	0.0001
10^4	0.005	13.7	1×10^-5	0.001
10^5	0.008	163	1×10^-5	0.025
10^6	0.09	> 1800	1×10^-5	0.63
10^7	0.5	> 1800	1×10^-5	2.0

Table 2: Time (seconds) to calculate a user-user score, and to generate recommendations, from a dataset of N interactions

Tests were run on serial implementations written in Python on a *Google Cloud Platform n1-highcpu-16* machine[3]. Table 2 shows the time taken to generate recommendations on various dataset sizes. Where recommendations are generally expected by users in real time, we consider times to generate recommendations of over 30 minutes (1800 seconds) not to be practical, hence record them as >1800.

RCF struggles to generate recommendations in real time for even datasets containing very small numbers of interactions. On a dataset containing a million interactions (a realistic number for a very small dating service), the algorithm took over thirty minutes to produce a list of recommendations for a single user. A parallel implementation is outside the scope of this paper. However, the scaling of RCF is such that even a parallel implementation would not make the algorithm feasible for a large dating service with millions of interactions per week.

LRFF remains able to generate recommendations in real time with a serial implementation for datasets of up to ten million interactions. However, as the time taken to generate a single reciprocal preference score is constant, calculating a large number of reciprocal scores would be easy to parallelise, and generating recommendations in real time for datasets with hundreds of millions of interactions would therefore be feasible.

Size	LFRR Training Time
10^3	0.74
10^4	2.8
10^5	23.2
10^6	229
10^7	2332

Table 3: Training time in seconds for LRFF over 10 iterations of gradient descent, from a dataset of N interactions

The training times of the feature matrices for LFRR are displayed in Table 3. A dataset with a hundred million interactions (the monthly volume of an extremely popular service) takes a few hours to train, and we consider this a realistic daily investment.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we presented a novel algorithm LFRR which applies latent factor models to reciprocal recommendation, and compared it to the current baseline for reciprocal collaborative filtering, RCF. We conclude that the effectiveness of LFRR is similar to RCF based on offline evaluations. We also tested both algorithms on a much larger dataset than has previously been used for reciprocal recommendation, and conclude that LFRR has considerably better efficiency, and is therefore a more realistic algorithm for generating reciprocal recommendations in real time on a service with a large number of interactions.

Previous reciprocal recommender systems have used only the harmonic mean. We presented an analysis of four functions for aggregating preference scores. Although the harmonic mean was the most consistent aggregation function across both RCF and LFRR, the cross-ratio uninorm function gave marginally better performance in the case of RCF. We conclude that the choice of aggregation function has a significant impact on the effectiveness of the model, and that future research in this field should consider the testing of various aggregation strategies to find which one fits best with the algorithm and data in question.

We performed offline evaluation of our model in this case because *Pairs* had other immediate goals, and it was not possible to arrange online testing within a reasonable time frame. However, since in some cases offline evaluation is not always representative of user interaction with recommendations online [28], we argue that this model would benefit from a thorough online evaluation and comparison with the baseline algorithm RCF. Online A/B tests have been arranged with Pairs and constitute our most immediate future work.

There are also other methods of training latent factor models, including data imputation and factorization [16], Alternating Least Squares [26] and neural network-based methods such as Restricted Boltzmann Machines [27]. These methods have yet to be tested on reciprocal environments against stochastic gradient descent. A comparison of various latent factor models on RRSs could be therefore investigated.

With regard to the efficiency evaluation, we tested all implementations based serial code written in Python. However, there are a number of theoretical efficiency improvements that could be made to both RCF and LFRR. Both algorithms have elements that could be implemented in parallel. While this would not change the difference in time complexity between the two algorithms, it might allow RCF to be used successfully on larger datasets, and reduce the training time for LFRR.

In our experiment, we used *Matches* as a proxy for a successful interaction and *Nopes*. However, users on an online dating site often have different goals - some of them are looking for a short relationship, some for marriage. It would be useful to extract user goals, explicitly or implicitly, and use those goals as part of the recommendation process. Similar relationship goals would provide move value to the user over and above mutual attractiveness.

Finally, in our experiment, we used a relatively simple measure of user popularity to weight the recommendations against very popular users. Sophisticated models of popularity based on machine learning techniques have been used to influence recommendations [2], and could also be applied to latent factor recommendations. In addition, it is important that reciprocal recommendation systems take account of the difference between active users, who often send *Likes*, and passive users, who only respond to *Likes*, and avoids recommending passive users to other passive users. This could also be integrated into a model that would maximise the number of matches.

ACKNOWLEDGMENTS

This research has been supported by the EPSRC Doctoral Training Programme (DTP). The authors would also like to thank Eureka Inc. for providing the dataset to test our research and the resources to train the models, and in particular Mr. Shintaro Kaneko (CTO at Eureka Inc.) and Mr. Yusuke Usui (AI Team Leader at Eureka Inc.) for their support.

[3]Exact specifications at https://cloud.google.com/compute/docs/machine-types

REFERENCES

[1] Luiz Pizzato, Tomasz Rej, Joshua Akehurst, Irena Koprinska, Kalina Yacef, and Judy Kay. 2012. Recommending people to people: the nature of reciprocal recommenders with a case study in online dating. User Model User-Adap Inter (2013) 23: 447.

[2] Akiva Kleinerman, Ariel Rosenfeld, Francesco Ricci, and Sarit Kraus. 2018. Optimally balancing receiver and recommended users' importance in reciprocal recommender systems. Proceedings of the 12th ACM Conference on Recommender Systems.

[3] Luiz Pizzato, Tomek Rej, Thomas Chung, Irena Koprinska, and Judy Kay. 2010. RECON: a reciprocal recommender for online dating. Proceedings of the fourth ACM conference on Recommender systems P. 207-214.

[4] Peng Xia, Benyuan Liu, Yizhou Sun, and Cindy Chen. 2015. Reciprocal Recommendation System for Online Dating. Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining P. 234-241.

[5] Zheng Siting, Hong Wenxing, Zhang Ning, and Yang Fan. 2012. Job recommender systems: A survey. 7th International Conference on Computer Science & Education (ICCSE).

[6] Jianming He, and Wesley Chu. 2010. A Social Network-Based Recommender System (SNRS). Data Mining for Social Network Data P. 47-74.

[7] Christian Rudder. We Experiment on Human Beings!. Last accessed 2017. OK Cupid Development Blog. https://web.archive.org/web/20180326122444/https://theblog.okcupid.com/we-experiment-on-human-beings-5dd9fe280cd5

[8] Charu Aggarwal. 2016. Recommender Systems: The Textbook. Springer.

[9] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, P. 426-434.

[10] Deepak Agarwal, and Bee-Chung Chen. 2009. Regression-based latent factor models. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining P. 19-28.

[11] Ronald Yager, and Alexander Rybalov. 1996. Uninorm aggregation operators. Fuzzy Sets and Systems Volume 80, Issue 1, P. 111-120.

[12] Akiva Kleinerman, Ariel Rosenfeld, Sarit Kraus. 2018. Providing explanations for recommendations in reciprocal environments. Proceedings of the 12th ACM Conference on Recommender Systems P. 22-30.

[13] Badrul Sarwar, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th International World Wide Web Conference (WWW10).

[14] John Breese, David Heckerman, and Carl Kadie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. UAI'98 Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence P. 43-52.

[15] Aaron Defazio, and Tiberio Caetano. 2012. A Graphical Model Formulation of Collaborative Filtering Neighbourhood Methods with Fast Maximum Entropy Training. ICML12.

[16] Robert Bell, Yehuda Koren, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. Computer August 2009, pp. 30-37, vol. 42.

[17] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. Application of Dimensionality Reduction in Recommender System - A Case Study. Proc. KDD Workshop on Web Mining for e-Commerce: Challenges and Opportunities (WebKDD), ACM Press.

[18] Arkadiusz Paterek. 2007. Improving regularized singular value decomposition for collaborative filtering. Proceedings of KDD cup and workshop.

[19] Robert Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. SiGKDD Explorations.

[20] Hsi-Mei Hsu, and Chen-Tung Chen. 1996. Aggregation of fuzzy opinions under group decision making. Fuzzy Sets and Systems Volume 79, Issue 3, 13 May 1996, P. 279-285.

[21] Ivan Palomares, Fiona Browne, and Peadar Davis. 2018. Multi-view fuzzy information fusion in collaborative filtering recommender systems: Application to the urban resilience domain. Data & Knowledge Engineering Volume 113, January 2018, P. 64-80.

[22] Orestes Appel, Francisco Chiclana, Jenny Carter, and Hamido Fujita. 2017. Cross-ratio uninorms as an effective aggregation mechanism in sentiment analysis. Knowledge-Based Systems Volume 124, 15 May 2017, P. 16-22.

[23] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. 2008. Addressing cold-start problem in recommendation systems. Proceedings of the 2nd international conference on Ubiquitous information management and communication P. 208-211.

[24] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. 2013. Addressing cold-start in app recommendation: latent user models constructed from twitter followers. Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval P. 283-292.

[25] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. Proceedings of the fourth ACM conference on Recommender systems P. 39-46.

[26] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. Algorithmic Aspects in Information and Management P. 337-348.

[27] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. Proceedings of the 24th international conference on Machine learning P. 791-798.

[28] Joeran Beel, Marcel Genzmehr, Stefan Langer, Andreas NÄijrnberger, and Bela Gipp. 2013. A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation P. 7-14.