
Evolving Good Recommendations

Supiya Ujjin

Department of Computer Science
University College London
Gower Street
London
S.Ujjin@cs.ucl.ac.uk
+44 (0)20 7679 2878

Peter J. Bentley

Department of Computer Science
University College London
Gower Street
London
P.Bentley@cs.ucl.ac.uk
+44 (0)20 7679 1329

Abstract

Recommender systems are new types of internet-based software tools, designed to help users find their way through today's complex on-line shops and entertainment websites. This paper describes a new recommender system, which employs a genetic algorithm to learn personal preferences of users and provide tailored suggestions.

1 INTRODUCTION

The rapid expansion of the Internet has brought about a new market for trading. Electronic commerce or e-commerce has enabled businesses to open up their products and services to a massive client base that was once available only to the largest multinational companies. As the competition between businesses becomes increasingly fierce, consumers are faced with a myriad of choices. Although this might seem to be nothing but beneficial to the consumer, the sheer wealth of information relating to the various choices can be overwhelming. One would normally rely on the opinions and advice of friends or family members but unfortunately even they have limited knowledge.

Recommender systems provide one way of circumventing this problem. As the name suggests, their task is to recommend or suggest items or products to the customer based on his/her preferences. These systems are often used by E-commerce websites as marketing tools to increase revenue by presenting products that the customer is likely to buy. An internet site using a recommender system can exploit knowledge of customers' likes and dislikes to build an understanding of their individual needs and thereby increase customer loyalty [1, 2].

This paper focuses on the use of evolutionary search to fine-tune a profile-matching algorithm within a recommender system, tailoring it to the preferences of individual users. This enables the recommender system to make more accurate predictions of users' likes and dislikes, and hence better recommendations to users.

The paper is organised as follows: section 2 outlines related work, and section 3 describes the recommender system and genetic algorithm. Section 4 provides

experimental results and analysis. Finally section 5 concludes.

2 BACKGROUND

From the literature, it seems that the definition of the term "recommender system" varies depending on the author. Some researchers use the concepts: "recommender system", "collaborative filtering" and "social filtering" interchangeably [3,4]. Conversely, others regard "recommender system" as a generic descriptor that represents various recommendation/prediction techniques including collaborative, social, and content based filtering, Bayesian networks and association rules [5,6]. In this paper, we adopt the latter definition when referring to recommender systems.

MovieLens (<http://www.movielens.umn.edu>), a well-known research movie recommendation website, makes use of collaborative filtering technology to make its suggestions. This technology captures user preferences to build a profile by asking the user to rate movies. It searches for similar profiles (i.e., users that share the same or similar taste) and uses them to generate new suggestions. One shortcoming that most websites using collaborative filtering suffer from is that they do not have any facility to provide explanations of how recommendations are derived. This is addressed in [7] which proposes explanation facilities for recommender systems in order to increase users' faith in the suggestions. (The dataset collected through the MovieLens website has been made available for research purposes and shall be used to test the lifestyle recommender system.)

By contrast, LIBRA (<http://www.cs.utexas.edu/users/libra>) combines a content-based approach with machine learning to make book recommendations. The content-based approach differs from collaborative filtering in that it analyses the contents of the items being recommended. Furthermore, each user is treated individually - there is no sense of "community" which forms the basis of collaborative filtering. It also uses Bayesian text-categorisation machine learning techniques to build a model of each user's preferences relative to the content of the items. Although the model is very small and fast and as accurate as a collaborative filtering technique [3], it is

often built off-line which can take anything from hours up to days. This method is therefore not suitable in a dynamic environment where user preference model needs to be updated frequently. The key advantage of this approach is that explanations can be very easily produced. However a content-based approach is inappropriate when the items being considered are in a non-textual form such as images, and video or music clips.

Dooyoo (<http://www.dooyoo.co.uk>) operates in a slightly different way. It too is a useful resource that provides recommendations to those seeking advice, but it focuses mainly on gathering qualitative opinions from its users, and then making them available to others. Visitors will often submit reviews on items or services ranging from health spas to mobile phones. These items are categorised in a similar fashion to the layout on a structured search engine, such as Yahoo! While Dooyoo is not a recommender system in the normal sense, it tries to encourage the concept of community by allowing users to rate things, evaluate the “usefulness” of other people’s reviews, and establish “circles of friends” - people who share similar opinions on various issues. In focussing on opinions in the form of short reviews, Dooyoo can already provide “explanations” that are not present in most other recommender systems.

Researchers at the University of the West Of England have also been working on a movie Recommender System [9]. Their idea is to use the immune system to tackle the problem of preference matching and recommendation. User preferences are treated as a pool of antibodies and the active user is the antigen. The difference in their approach and the other existing methods is that they are not interested in finding the one best match but a diverse set of antibodies that are a close match.

3 SYSTEM OVERVIEW

The system described in this paper is based around a collaborative filtering approach, building up profiles of users and then using an algorithm to find profiles similar to the current user. (In this paper, we refer to the current user as the *active user*, A). Selected data from those profiles are then used to build recommendations. Because profiles contain many attributes, many of which have sparse or incomplete data [7], the task of finding appropriate similarities is often difficult. To overcome these problems, current systems (such as MovieLens) use stochastic and heuristic-based models to speed up and improve the quality of profile matching. This work takes such ideas one step further, by applying an evolutionary algorithm to the problem of profile matching.

In this research, the MovieLens dataset was used for initial experiments. It contains details of 943 users, each with many parameters or *features*: demographic information such as age, gender and occupation is collected when a new user registers on the system. Every time a vote is submitted by a user, it is recorded in the

database with a timestamp. The movie information in the dataset includes genres, and theatre and video release dates. The evolutionary recommender system uses 22 features from this data set: movie rating, age, gender, occupation and 18 movie genre frequencies: action, adventure, animation, children, comedy, crime, documentary, drama, fantasy, film-noir, horror, musical, mystery, romance, sci-fi, thriller, war, western.

3.1 PROFILE GENERATOR

Before recommendations can be made, the movie data must first be processed into separate profiles, one for each person, defining that person’s movie preferences.

A profile for user j , denoted

$profile(j)$

is represented as an array of 22 values for the 22 features considered. The profile has two parts: a variable part (the rating value, which changes according to the movie item being considered at the time), and a fixed part (the other 21 values, which are only retrieved once at the beginning of the program). Because user j may have rated many different movies, we define

$profile(j,i)$

to mean the profile for user j on movie item i , see fig. 1.

| 1 | 2 | 3 | 4 | ..22 |
|--------|-----|--------|------------|----------------------|
| Rating | Age | Gender | Occupation | 18 Genre frequencies |
| 5 | 23 | 0 | 45 | 000000100010000000 |

Figure 1: $profile(j,i)$ - profile for user j with rating on movie item i , if i has a rating of 5.

Once profiles are built, the process of recommendation can begin. Given an active user A , a set of profiles similar to $profile(A)$ must be found.

3.2 NEIGHBOURHOOD SELECTION

The success of a collaborative filtering system is highly dependent upon the effectiveness of the algorithm in finding the set or neighbourhood of profiles that are most similar to that of the active user. It is vital that, for a particular neighbourhood method, only the best or closest profiles are chosen and used to generate new recommendations for the user. There is little tolerance for inaccurate or irrelevant predictions.

The neighbourhood selection algorithm consists of three main tasks:

1. profile selection
2. profile matching
3. best profile collection.

3.2.1 Profile Selection

In an ideal world, the entire database of profiles would be used to select the best possible profiles. However this is not always a feasible option, especially when the dataset is very large or if resources are not available. As a result, most systems opt for random sampling and this process is the responsibility of the profile selection part of the algorithm.

This work investigates two methods of profile selection:

1. Fixed: the first n users from the database are always used in every experiment
2. Random: n users are picked randomly from the database,

where $n = 10$ or 50 in our experiments.

3.2.2 Profile Matching

After profile selection, the profile matching process then computes the distance or similarity between the selected profiles and the active user's profile using a distance function. This research focuses on this profile matching task, i.e., the evolutionary algorithm is used to fine-tune profile matching for each active user.

From the analysis of Breese et. al [3], it seems that most current recommender systems use standard algorithms that consider only "voting information" as the feature on which the comparison between two profiles is made. However in real life, the way in which two people are said to be similar is not based solely on whether they have complimentary opinions on a specific subject, e.g., movie ratings, but also on other factors, such as their background and personal details. If we apply this to the profile matcher, issues such as demographic and lifestyle information which include user's age, gender and movie genres must also be taken into account. Every user places a different importance or priority on each feature. These priorities can be quantified or enumerated. Here we refer to these as *feature weights*. For example, if a male user prefers to be given recommendations based on the opinions of other men, then his feature weight for gender would be higher than other features. In order to implement a truly personalised recommender system, these weights need to be captured and fine-tuned to reflect each user's preference. Our approach shows how such weights can be evolved by a genetic algorithm.

A potential solution to the problem of evolving feature weights, $w(A)$, for the active user, A is represented as a set of weights as shown below in Figure 2.

| | | | | |
|-------|-------|-------|---------|----------|
| w_1 | w_2 | w_3 | \dots | w_{22} |
|-------|-------|-------|---------|----------|

Figure 2: Phenotype of an individual in the population.

where w_f is the weight associated with feature f whose genotype is a string of binary values. Each individual contains 22 genes, which are evolved by an elitist genetic algorithm (described in section 3.4).

The comparison between two profiles can now be conducted using a modified Euclidean distance function, which takes into account multiple features. $Euclidean(A,j)$ is the similarity between active user A and user j :

$$euclidean(A, j) = \sqrt{\sum_{i=1}^z \sum_{f=1}^{22} w_f * diff_{i, f}(A, j)^2}$$

where: A is the active user

j is a user provided by the profile selection process, where $j \neq A$

z is the number of common movies that users A and j have rated.

w_f is the active user's weight for feature f

i is a common movie item, where $profile(A,i)$ and $profile(j,i)$ exists.

$diff_{i, f}(A, j)$ is the difference in profile value for feature f between users A and j on movie item i .

Note that before this calculation is made, the profile values are normalised to ensure they lie between 0 and 1. When the weight for any feature is zero, that feature is ignored. This way we enable feature selection to be adaptive to each user's preferences.

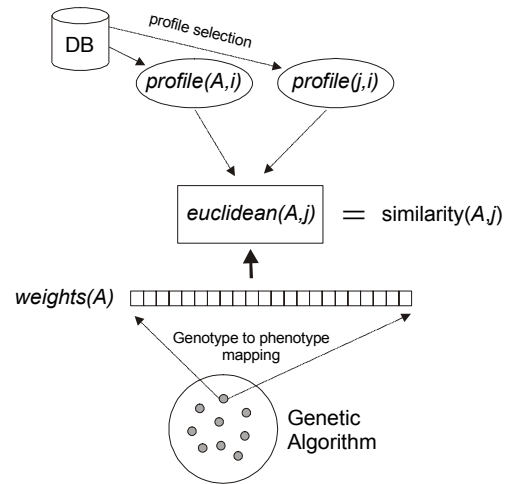


Figure 3 Calculating the similarity between A and j .

3.2.3 Best Profile Collection

Once the Euclidean distances, $euclidean(A,j)$, have been found between $profile(A)$ and $profile(j)$ for all values of j picked by the profile selection process, the "best profile collection" algorithm is called. This ranks every $profile(j)$ according to its similarity to $profile(A)$. The system then simply uses half of the users that are most similar to the active user as the neighbourhood of A (although this value is a system constant that can be changed).

3.3 MAKING A RECOMMENDATION

To make a recommendation, given an active user A and a neighbourhood set of similar profiles to A , it is necessary to find movie items seen (and liked) by the users in the neighbourhood set that the active user has not seen. These are then presented to the active user through a user interface. Because the neighbourhood set contains those users who are most similar to A (using in our case the specific preferences of A through evolved weighting values), movies that these users like have a reasonable probability of being liked by A .

3.4 GENETIC ALGORITHM

As described in section 3.2.2, a genetic algorithm has been used to evolve feature weights for the active user, and hence help tailor the matching function to the user's specific personality and tastes.

An elitist genetic algorithm was chosen for this task, where a quarter of the best individuals in the population are kept for the next generation. When creating a new generation, individuals are selected randomly out of the top 40% of the whole population to be parents. Two offspring are produced from every pair of parents, using single-point crossover with probability 1.0. Mutation is applied to each locus in genotype with probability 0.01. A simple unsigned binary genetic encoding is used in the implementation, using 8 bits for each of the 22 genes. The GA begins with random genotypes.

A genotype is mapped to a phenotype (a set of feature weights) by converting the alleles of the binary genes to decimal. The feature weights can then be calculated from these real values. First, the importance of the 18 genre frequencies are reduced by a given factor, the *weight reduction size*. This is done because the 18 genres can be considered different categories of a single larger feature, Genre. Reducing the effect of these weights is therefore intended to give the other unrelated features (movie rating, age, gender, occupation) a more equal chance of being used. Second, the total value of phenotype is then calculated by summing the real values for all 22 features. Finally, the weighting value for each feature can be found by dividing the real value by the total value. The sum of all the weights will then add up to unity.

3.4.1 Fitness function

Calculating the fitness for this application is not trivial. Every set of weights in the GA population must be employed by the profile matching processes within the recommender system. So the recommender system must be re-run on the MovieLens dataset for each new set of weights, in order to calculate its fitness.

But running a recommender system only produces recommendations (or predictions), not fitnesses. A poor set of weights might result in a poor neighbourhood set of profiles for the active user, and hence poor recommendations. A good set of weights should result in

a good neighbourhood set, and good recommendations. So a method of calculating the quality of the recommendations is required, in order that a fitness score can be assigned to the corresponding weights.

One solution would be to employ the active user as a fitness function. This would involve obtaining feedback from the user by asking him to judge the quality of recommendations [8]. His input could be used to help derive fitness scores for the current set of feature weights. This fitness score would give a highly accurate view of the user's preferences. However, it is unlikely that every user will be willing to participate in every recommendation – the time needed would be too great.

Instead, it was decided to reformulate the problem as a supervised learning task. As described previously, given the active user A and a set of neighbouring profiles, recommendations for A can be made. In addition to these recommendations, it is possible to predict what A might think of them. For example, if a certain movie is suggested because similar users saw it, but those users only thought the movie was "average", then it is likely that the active user might also think the movie was "average". Hence, for the MovieLens dataset, it was possible for the system to both recommend new movies and to predict how the active user would rate each movie, should he go and see it.

The predicted vote computation used in this paper has been taken from [3] and modified such that the Euclidean distance function (section 3.2.2) now replaces the weight in the original equation. The predicted vote, $predict_vote(A,i)$, for A on item i , can be defined as:

$$predict_vote(A,i) = mean_A + k \sum_{j=1}^n euclidean(A,j)(vote(j,i) - mean_j)$$

where: $mean_j$ is the mean vote for user j

k is a normalising factor such that the sum of the euclidean distances is equal to 1.

$vote(j,i)$ is the actual vote that user j has given on item i

n is the size of the neighbourhood.

To calculate a fitness measure for an evolved set of weights, the recommender system finds a set of neighbourhood profiles for the active user, as described in section 3.2. Three movie items that the active user has seen (i.e. $profile(A,i)$ exists for all 3 values of i) are then selected, where those movie items that have more ratings in the user database have a higher probability of being picked.

This results in three movie items that the active user has seen, and that are quite likely to have been seen by the users in the neighbourhood set. The ratings of these users are then employed to compute the predicted rating for the active user on each movie item. Because the active user has already rated the movie items, it is possible to compare the actual rating with the predicted rating. So, the average of the differences between the three actual

and predicted votes are used as fitness score to guide future generations of weight evolution.

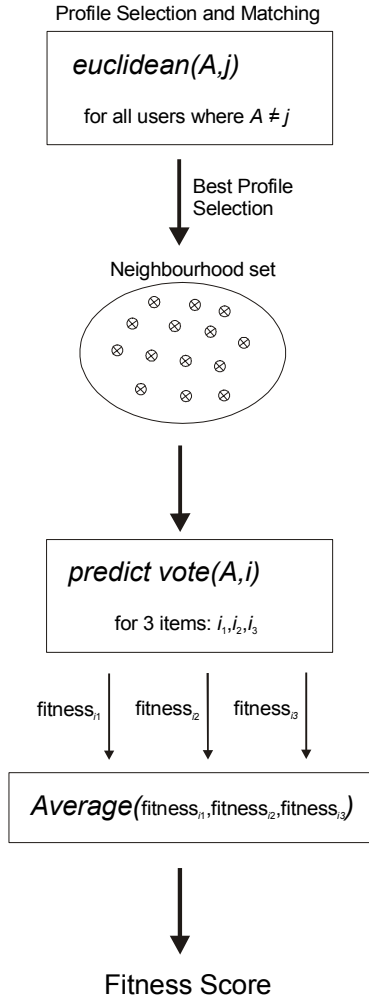


Figure 4: finding the fitness score of an individual (the active user's feature weights).

4 EXPERIMENTS

Four sets of experiments were designed to observe the difference in performance between the evolutionary recommender system and a standard, non-adaptive recommender system based on the Pearson algorithm [3]. In each set of experiments, the predicted votes of all the movie items that the active user has rated were computed using the final feature weights for that run. These votes were then compared against those produced from the simple Pearson algorithm.

The Pearson algorithm used in the experiments is based on the k Nearest Neighbour algorithm. A correlation coefficient, shown below, is used as the matching function for selecting the k users that are most similar to the active user to give predictions. This replaces the

Euclidean function described earlier; all other details remain the same.

$$correlation(A, j) = \frac{\sum_{i=1}^n (vote(A, i) - mean_A)(vote(j, i) - mean_j)}{\sqrt{\sum_{i=1}^n (vote(A, i) - mean_A)^2 (vote(j, i) - mean_j)^2}}$$

The four experiments also evaluated two system variables to assess their effect on system performance: the *profile selection* task (the way in which profiles were selected from the database), and the size of the neighbourhood.

The following parameter values were kept the same in all four experiments:

1. *population size* = 75. The number of individuals in the population at each generation.
2. *termination threshold* = 0.06. When the fitness score of the best individual (set of feature weights) is below the threshold, a good solution is found and this set of weights is used as the final result for the current run.
3. *maximum number of generations for each run* = 300. If the number of generations reaches this value and the solution has not been found, the best individual for that generation is used as the final result.
4. *weight reduction size* = 4. The scaling factor for the 18 genre frequencies.
5. *number of runs* = 30. The number of times the system was run for each active user.

The four sets of experiments are as follows:

Experiment 1: Each of the first 10 users was picked as the active user in turn, and the first 10 users (fixed) were used to provide recommendations.

Experiment 2: Each of the first 50 users was picked as the active user in turn, and the first 50 users (fixed) were used to provide recommendations.

Experiment 3: Each of the first 10 users was picked as the active user in turn, and 10 users were picked randomly and used to provide recommendations (the same 10 used per run).

Experiment 4: Each of the first 50 users was picked as the active user in turn, and 50 users were picked randomly and used to provide recommendations (the same 50 used per run).

Real-world Applications

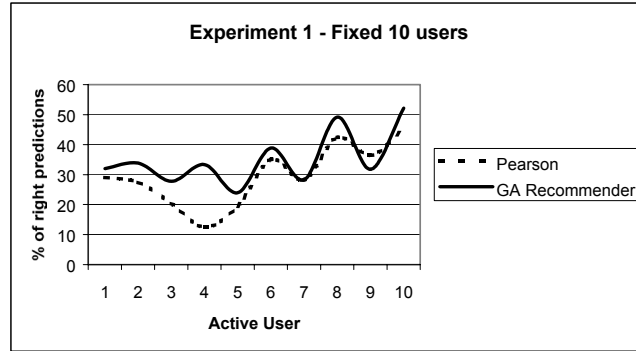


Figure 5. Results for experiment 1.

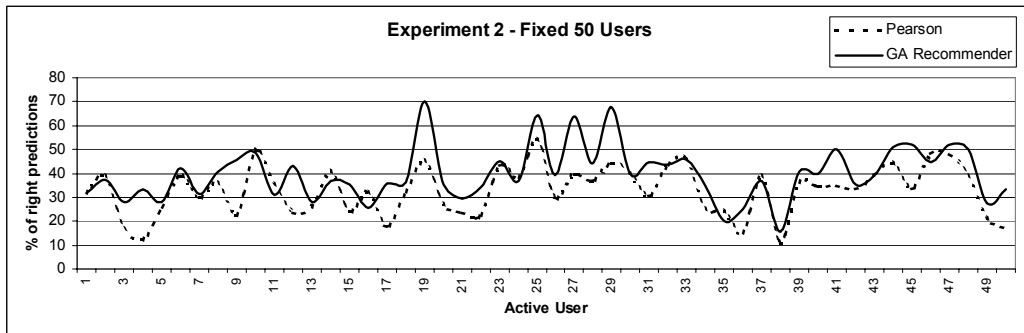


Figure 6. Results for experiment 2.

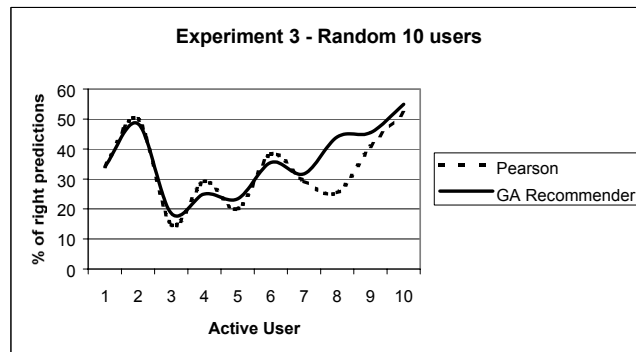


Figure 7. Results for experiment 3.

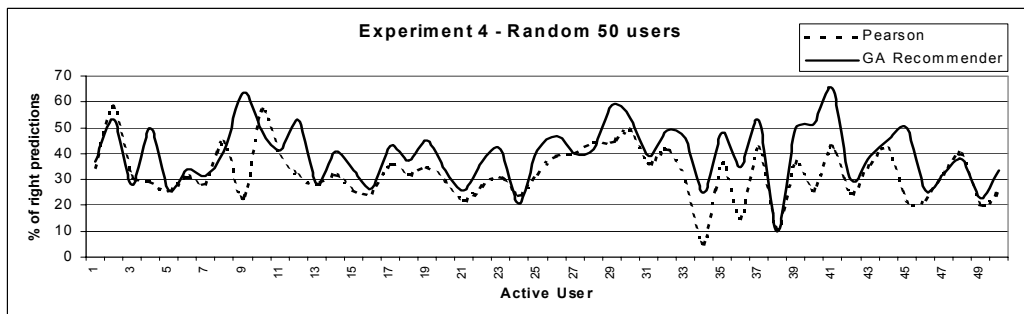


Figure 8. Results for experiment 4.

4.1 RESULTS

Figures 5 to 8 show the results for experiments 1 to 4, respectively. Each graph shows the percentage of the number of ratings that the system predicted correctly out of the total number of available ratings by the current active user. Whilst the predictions computed with the Pearson algorithm always remain the same given the same parameter values, those obtained from the GA vary according to the feature weights of that run. Out of the 30 runs for each active user in each experiment, the run with the best feature weights (that gave the highest percentage of right predictions) was chosen and plotted against the result from the Pearson algorithm.¹

Figure 5 shows that in the first experiment, the GA recommender performed equally well (or better) than the Pearson algorithm on all active users but one, user 9.

Figure 6 shows that in the second experiment the accuracy for the GA recommender fell below that of the Pearson algorithm only for 2 active users, 16 and 35. On the rest of the active users, the accuracy for the GA recommender was found to be better – in some cases the difference was as great as 25%.

The random sampling for experiment 3 did not produce a great deal of improvement on the prediction accuracy for the GA recommender, see figure 7. However, it could be argued that the number of users, 10, is too small to find a good set of profiles that matched the active users.

The results for the last experiment show that the accuracy for the GA recommender was better for all but 3 active users, see figure 8. This suggests that random sampling can work very well and could be used as a good solution whenever the database is too large for all users to be examined.

4.2 ANALYSIS OF RESULTS

The patterns for the active users 1 to 10 look very similar in both experiments 1 and 2. Examining them in more detail: figure 5 indicates that the prediction accuracy for the active user 9 on the GA recommender was worse than that obtained from using the Pearson algorithm. But when the number of users was increased to 50 in experiment 2, the accuracy also improved greatly. This was expected – as the number of users goes up, the probability of finding a better matched profile should be higher and hence, the accuracy of the predictions should increase as well.

Equally, for active user 4, it can be seen that in the first experiment (figure 5), the difference between successful Pearson and GA predictions was already large – and it

became even better in experiment 2 (figure 6). It seems likely that the most similar users to active user 4 were amongst the first 10 users, so even though the number of users was increased, the same users (plus another 15 equally good or better matches) were picked to give recommendations.

The accuracy for the active users 4 and 6 in experiment 3 was below that of the Pearson algorithm (figure 7). Again, this was perhaps due to the fact that 10 users were not enough to find a good set of profiles to match the active users. Experiment 4 (figure 8) seems to corroborate this view. When the number of users increased to 50, the accuracy for the two mentioned active users rose and outperformed the other algorithm.

As shown in experiment 4, the results suggest that random sampling is a good choice for the *profile selection* task of retrieving profiles from the database. Random sampling was expected to be better than fixing which users to select because it allowed the search to consider a greater variety of profiles and hence find a better set of well matched profiles. In addition, as the size of neighbourhood increases, the prediction accuracy also increases. This, again, was expected as more users are examined, the probability of finding good matches should rise.

This approach has been shown to work well, but there are problems. As fitness scores are computed by getting the differences between the actual and predicted votes, this is only achievable if the user has already actively voted on movies, otherwise the intersection between recommended items and those already voted for by the active user would return very few titles or even none. In this case, this approach will fail, as a fitness score cannot be determined.

In earlier experiment with only 4 features: Rating, Gender, Gender and Occupation, it was noticed that many solutions were found for items which are sometimes associated with gender (inferred by gender). For example, when the active user's feature weights showed that the user preferred to be recommended by people of the same gender, solutions were often found for items that belonged to the Action genre. Because of this, it would be interesting to see if results can be improved if we make use of association rules.

5 CONCLUSIONS

This work has shown how evolutionary search can be employed to fine-tune a profile-matching algorithm within a recommender system, tailoring it to the preferences of individual users. This was achieved by reformulating the problem of making recommendations into a supervised learning task, enabling fitness scores to be computed by comparing predicted votes with actual votes. Experiments demonstrated that, compared to a non-adaptive approach, the evolutionary recommender system was able to successfully fine-tune the profile matching algorithm. This enabled the recommender system to make

¹ The best rather than average was plotted since this is closest to the real world scenario where this system could be run off-line and the current best set of feature weights would be set as the initial preference of the active user. Following this, the evolved weights could be stored on the user's local machine. A local copy of the system would then be responsible for fine-tuning the weights to suit that user's preferences further. This way the processing load on the server would be reduced and parallelism can be achieved.

more accurate predictions, and hence better recommendations to users.

Acknowledgments

Thanks to Philip Treleaven for his guidance. This work is funded by a scholarship provided by the Thai Government.

References

- [1] Schafer, J.B., Konstan, J. A. and Riedl, J. January 2001. E-Commerce Recommendation Applications. *Journal of Data Mining and Knowledge Discovery*.
- [2] Schafer, J.B., Konstan, J. and Riedl, J. 1999. Recommender Systems in E-Commerce. *Proceedings of the ACM 1999 Conference on Electronic Commerce*.
- [3] Breese, J.S., Heckerman, D. and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43-52.
- [4] Goldberg, K., Roeder, T., Gupta, D. and Perkins, C. August 2000. *Eigentaste: A Constant Time Collaborative Filtering Algorithm*. UCB ERL Technical Report M00/41
- [5] Terveen, L. and Hill, W. 2001. Beyond Recommender Systems: Helping People Help Each Other. In *HCI In The New Millenium*, Carroll, J. ed. Addison-Wesley.
- [6] Delgado, J.A. February 2000. *Agent-Based Information Filtering and Recommender Systems on the Internet*. PhD thesis, Nagoya Institute of Technology.
- [7] Herlocker, J.L., Konstan, J. A. and Riedl, J. 2000. Explaining Collaborative Filtering Recommendations. *Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work*.
- [8] Bentley, P. J. and Corne, D. W. 2001. *Creative Evolutionary Systems*. Morgan Kaufman Pub.
- [9] Cayzer, S. and Aickelin U. 2001. A Recommender System based on the Immune Network. Submitted and under review.