LEARNING USER PREFERENCES USING EVOLUTION

Supiya Ujjin and Peter J. Bentley

Department of Computer Science
University College London, Gower Street, London WC1E 6BT
S.Ujjin@cs.ucl.ac.uk P.Bentley@cs.ucl.ac.uk

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

As the name suggests, recommender systems' 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. 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 [4], 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 (http://www.movielens.umn.edu), was used for initial experiments. 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.

2.1. Profile Generator

Before recommendations can be made, the movie data is processed into separate profiles, one for each person, defining that person's movie preferences. We define *profile(j,i)* to mean the profile for user *j* on movie item *i*, see fig. 1. The profile of *j*, *profile(j)* is therefore a collection of *profile(j,i)* for all the items *i* that *j* has seen.

1 Rating	**2** Age	**3** Gender	**4** Occupation	**..22** 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 or neighbourhood of profiles similar to *profile(A)* must be found.

2.2. 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: *profile selection, profile matching and best profile collection.*

2.2.1. 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.

2.2.2. 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. 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 preferences of movie genres must also be taken into account. Every user places a different importance or priority on each feature. Our approach shows how weights defining user's priorities 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	…	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 2.3).

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. The difference in profile values for occupation is either 0, if the two users have the same occupation, or 1 otherwise.

2.2.3. 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 selects the users whose Euclidean distance is above a certain threshold value (considered most similar to the active user) as the neighbourhood of A. This value is a system constant that can be changed. 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.

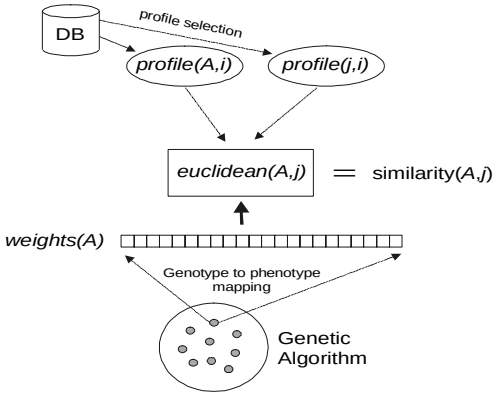

Figure 3: Calculating the similarity between A and j.

2.3. Genetic Algorithm

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.

2.3.1. 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.

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 actual vote of user *j* for item *i*
n is the size of the neighbourhood.

All the movie items that the active user has seen are randomly partitioned into two datasets: a training set (1/3) and a test set (2/3). 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 2.2. The ratings of the users in the neighbourhood set are then employed to compute the predicted rating for the active user on each movie item in the training set. 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 actual and predicted votes of all items in the training set are used as fitness score to guide future generations of weight evolution, see figure 4.

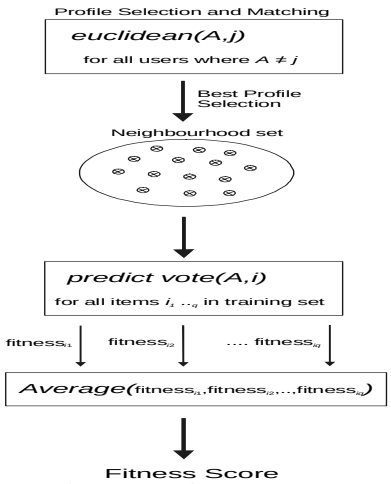

Figure 4: finding the fitness score of an individual (the active user's feature weights).

3. 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 in the test set (the items that the active user has rated but were not used in weights evolution) were computed using the final feature weights for that run. These votes were then compared against those produced from the simple Pearson algorithm.

The four sets of experiments were as follows:

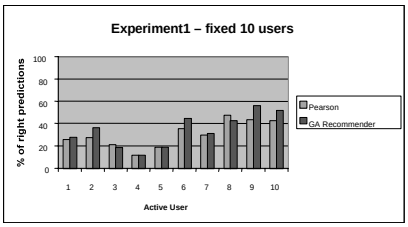

Figure 5: Results for experiment 1

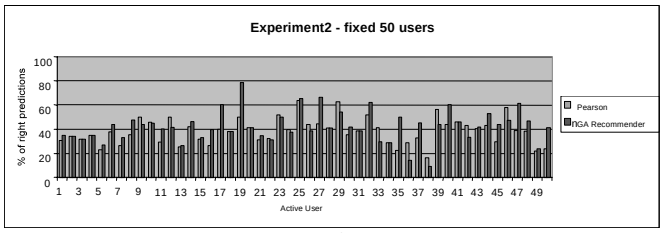

Figure 6: Results for experiment 2

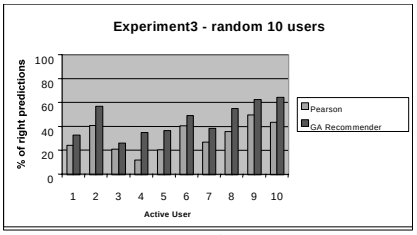

Figure 7: Results for experiment 3

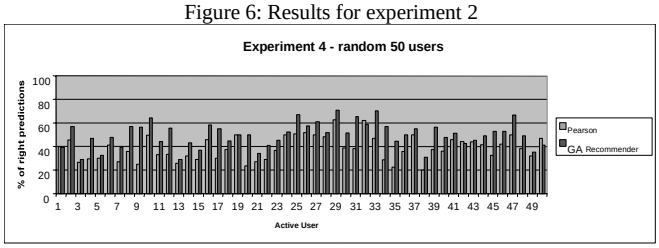

Figure 8: Results for experiment 4

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 (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 (same 50 used per run).

3.1. 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) compared to the Pearson algorithm on 8 active users out of 10.
Figure 6 shows that in the second experiment, out of the 50 users the accuracy for the GA recommender fell below that of the Pearson algorithm for 14 active users. 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 31%.

The random sampling for experiment 3 showed great improvement on the prediction accuracy for the GA recommender, see figure 7. All 10 active users performed better than the Pearson algorithm.

The results for the last experiment show that the accuracy for the GA recommender was significantly better for all but 4 active users, see figure 8.

3.2. Analysis of Results

Figure 5 indicates that the prediction accuracy for the active user 3 and 8 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 for the two mentioned active users rose and outperformed the other algorithm. This was expected – as the number of users goes up, the probability of finding a better matched profile should be higher and hence accuracy of the predictions should also increase.

The patterns in both experiments 3 and 4 for the active users 1 to 10 look very similar. Both show an improved accuracy compared to the Pearson algorithm but in experiment 4 there seems to be a greater improvement. Again, this is likely to be because of the increase in the number of users. 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 (potentially 10*30 runs = 300 users in

[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. The evolved weights could then be stored on the user's local machine. A local copy of the system would 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.

experiment 3 and 50 * 30 = 1500 users in experiment 4) and hence find a better set of well matched profiles.

As mentioned earlier, only the run(s) with the best feature weights for each active user were considered for this analysis. We now look into these runs in more detail to see how the feature weights obtained and users selected for the neighbourhood in these runs played a part in determining user preference.

Looking at experiment 1, when more than 1 run for an active user achieved the same best performance (highest number of votes being predicted correctly) results indicate that the same set of users had been selected for the neighbourhood to give recommendations. Moreover, for other runs that did not perform as well as the best run(s), different users that gave the best performance had been selected. For example, for active user 2 in experiment 1, all the runs that got the same percentage as the best, chose user 4 to be in the neighbourhood. The other active users did not select any users to give recommendations, instead the mean vote was used. Data gathered during experiment 2 corroborates this view. In addition, as the number of users was increased, the users that were originally selected for the neighbourhood in experiment 1 were still being chosen in experiment 2 as a subset of a larger neighbourhood. For example, as mentioned above, in experiment 1 user 2 picked user 4 to be in the neighbourhood, in experiment 2 this user picked users 4,13,18,22,42,43,49. This, however, only applies to the active users that performed better than the Pearson algorithm in experiment 1. The accuracy for active user 8 was worse in experiment 1, in which users 4, 5, 7 and 10 were selected. In experiment 2 when users 4 and 10 were not included in the neighbourhood, the accuracy improved tremendously as seen in figure 6. The trend described could not be observed when random sampling was used in experiments 3 and 4, as it was more difficult for the system to select the same users to examine at each run.

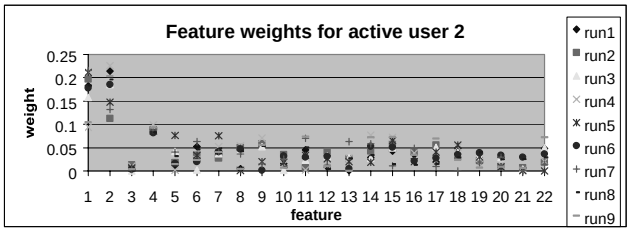

Figure 9: feature weights for active user 2, note that weights 5 to 22 are lower because of the scaling factor

Looking at the final feature weights obtained for each active user, many interesting observations have been found. Here we focus on the first 2 experiments as they have 10 common active users. Firstly, in experiment 2 when more than 1 run came up with the best performance, the feature weights seem to show very similar trends. For example, figure 9 shows the weight emphasis on the first 2 features: rating and age. It is also clear that this user does not show any interest in the 3rd feature which is gender. So as long as the people that are giving him recommendations have similar opinions and are in the same age group as him, he does not care whether they are male or female.

The feature weights obtained for active user 8 were also interesting. They show that for this user, age and gender are more significant. By looking further at the movie genres, we found that people who have similar opinions as this user on action, adventure, horror, romantic and war movies are likely to be picked for the neighbourhood set. As these genres are stereotypically related to gender and age, for example, men prefer action movies and war movies, the weights showed consistent description of the user's preference. Another example is active user 7 whose weights show strong feelings for documentary, mystery, sci-fi and thriller genres and emphasis on age. This user is a 57 year old male which may explain reduced significance of children and romance genres.

From the observations above, we can see that age is often as or more important as rating. This shows that the theory behind the original collaborative filtering does not always hold. This is hardly surprising as everyday experience suggests that most people listen to the recommendations made by their friends who are most likely to be in the same age group as them.

4. 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 more accurate predictions, and hence better recommendations to users.

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. Proc. of the ACM 1999 Conf. on Electronic Commerce.

[3] Breese, J.S., Heckerman, D. and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In Proc. of the 14th Conf. on Uncertainty in AI, pp. 43-52.

[4] Herlocker, J.L., Konstan, J. A. & Riedl, J. 2000. Explaining Collaborative Filtering Recommendations. Proc. of ACM 2000 Conf. on Computer Supported Cooperative Work.