Temporal Collaborative Filtering With Adaptive Neighbourhoods

Neal Lathia
Dept. of Computer Science
University College London
London, WC1E 6BT, UK
n.lathia@cs.ucl.ac.uk

Stephen Hailes
Dept. of Computer Science
University College London
London, WC1E 6BT, UK
s.hailes@cs.ucl.ac.uk

Licia Capra
Dept. of Computer Science
University College London
London, WC1E 6BT, UK
l.capra@cs.ucl.ac.uk

ABSTRACT

Recommender Systems, based on collaborative filtering (CF), aim to accurately predict user tastes, by minimising the mean error achieved on hidden test sets of user ratings, after learning from a training set. However, deployed recommender systems do not operate on, and should not be optimised to predict, a static set of user ratings because the underlying dataset is continuously growing and changing. The aim of a recommender system is therefore to *iteratively* predict users' preferences over a dynamic dataset, and system administrators are confronted with the problem of having to continuously tune the parameters calibrating their CF algorithm for best performance.

In this work, we first formalise CF as a time-dependent, iterative prediction problem. We then perform a temporal analysis of the Netflix dataset, and evaluate the temporal performance of a baseline model and the k-Nearest Neighbour algorithm. We show that, due to the dynamic nature of the data, certain prediction methods that improve prediction accuracy on the Netflix probe set do not show similar improvements over a set of iterative train-test experiments with growing data. We then address the problem of parameter selection and update, and propose a method to automatically assign and update per-user neighbourhood sizes that (on the temporal scale) outperforms setting global parameters.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Temporal Collaborative Filtering, Time-Averaged Error

1. INTRODUCTION

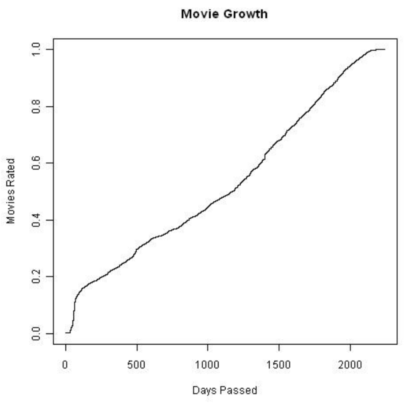
Recommender systems, based on Collaborative Filtering (CF) [1], are becoming important portals via which users access, browse, and interact with online web sites. They have successfully been applied to e-commerce, movie rental, and music recommendation sites, and are built upon the assumption that users who have been like-minded in the past can provide insight into each other's future tastes.

The first step of any recommender system is to collect a set of (implicit or explicit) ratings by each user for the content. There are then a wide range of techniques that can be applied to *predict* the ratings users have yet to input [2]; based on these predictions, recommendations can be offered. The core problem of collaborative filtering is therefore reduced to one of *prediction*, and research in this field often aims to optimise performance based on a mean error metric [3].

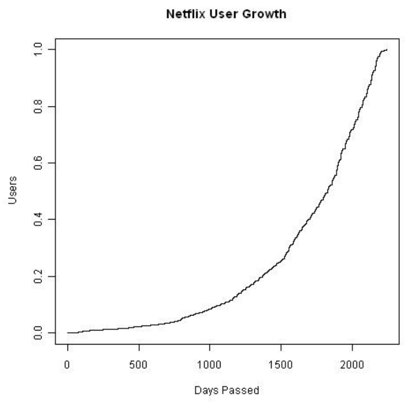
Experiments using CF data are often based on splitting a set of ratings into appropriate training and test sets, and predicting the ratings in the test set using the training set. In particular, the parameters required to calibrate the CF algorithm (for example, the k value in k-Nearest Neighbour - kNN - approaches) are found via cross-validated experiments, and set to values that are dependent on the datasets under investigation.

However, deployed implementations of these algorithms operate in a rather different setting, for a number of reasons. The *dataset* is dynamic: the user base and number of ratings grow over time. The system will therefore be iteratively updated, in order to incorporate the latest ratings provided by each user and provide up-to-date recommendations for all. A growing dataset leads to changing *features* of the ratings; both global and user statistics derived from CF data (and often used to predict ratings) change over time. Finally, *parameters* set according to cross-validated experiments on subsets of the user-rating data may no longer be optimal, and system administrators are required to continuously tune their algorithms for best performance. The aim of a recommender system is thus to continuously anticipate the behaviour of its users with a *changing* dataset; however, the algorithms applied in this context are not designed to adapt to meet the changes they will experience.

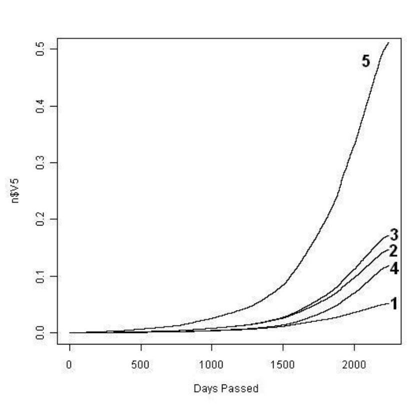
In this work, we extend CF evaluation to gain insight into the temporal performance of these algorithms as they are iteratively applied. We first explore the temporal characteristics of a large-scale dataset of user-movie ratings, in Section 2, and evaluate two CF algorithms on a static dataset (in Section 3). We then formalise CF as a time-dependent prediction problem and evaluate the same algorithms on a



(a) Movie Growth

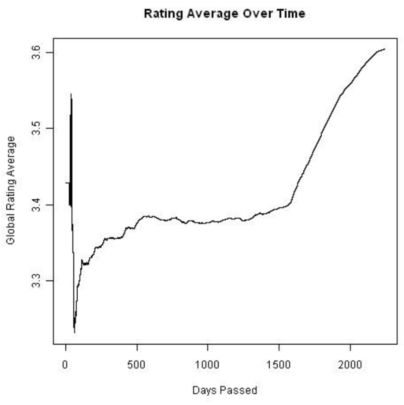


(b) User Growth

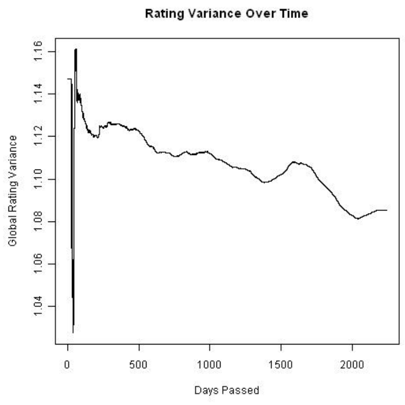


(c) Ratings Over Time

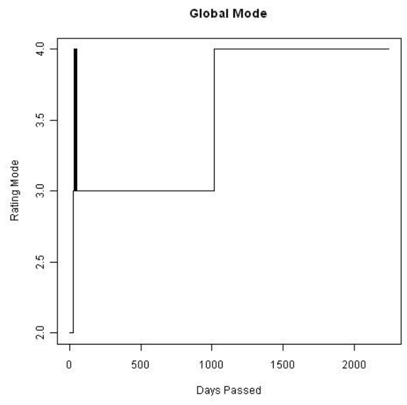
Figure 1: Netflix Data: Growth of Number of Movies/Users and Ratings (By Group)



(a) Mean Rating Over Time



(b) Variance Over Time



(c) Mode Over Time

Figure 2: Netflix Data: Global Rating Mean, Variance, and Mode Over Time

series of iterative experiments in Section 4. Based on this evaluation, we propose and evaluate *temporal-adaptive* CF, a method of temporally adapting the size of user kNN neighbourhoods based on the performance measured up to the current time, in Section 5, and conclude in Sections 6 and 7 with a discussion on the context of our current and future work.

2. TEMPORAL ANALYSIS

In this work, we examine the Netflix prize dataset[1]: this sparse dataset contains over 100 million ratings from 480 thousand users on $17,770$ movies. Each entry in the dataset can be viewed as a 4-tuple, consisting of a user id u, a movie id i, a date d, and value $r \in [1, 5]$. The data ranges from the 11^{th} of November 1999 to December 31^{st} 2005, spanning a total of $2,243$ days.

Figure 1 shows three distinct features of the growing dataset: the number of rated movies, the number of users, and the number of ratings (per rating) over time. Each follows a different pattern; while the user base and ratings grow exponentially, the number of movies available grows linearly. The number of 5 star ratings continuously dominates the dataset, followed by 3 and 2 star ratings. The global mean is shown in Figure 2(a). It consistently falls between 3 and 4 stars, but varies quite largely within this range. The most notable time segments are before the first 500 days, where the global mean sharply rises, falls, and then once again rises, and after the first 1,500 days have passed, where the mean begins to grow again. The global rating variance, Figure 2(b), also suffers from high fluctuation in the initial days of the dataset, and then decreases from 1.14 to 1.08 in a near-linear fashion, highlighting the tendency of users to avoid the extremely negative ratings in these contexts. To understand why the mean and variance display such change, consider Figure 2(c), which shows the rating mode (i.e. the most frequent rating value in the dataset) over time. The initial fluctuation in the mean is mirrored by a change of the rating mode from 2 to 4 stars. The mode then reverts and stabilises at 3 stars, until it again changes, and remains, at 4 stars - accounting for the rise of the rating average. The rating median (not included due to space constraints) behaves very similarly to the mode: days after the mode jumps to 4 stars, the median increases from 3 stars to 4 stars, reflecting the surge in the 4 and 5 star ratings that are input in this

[1]http://www.netflixprize.com

time, and accounting for the changes observed in both the mean and mode. A median of 4 tells us that *half* of the ratings in the system are 4 and 5 stars; however, more importantly, the *change* the median displays over time reflects that the distribution of ratings does not remain consistent. It is impossible to deduce from the data why the global behaviour changed as we see here; changes to the Netflix interface, recommendation algorithm, user base, or combinations of these may be, but cannot be confirmed to be, the cause. However, we note that a temporal analysis of the MovieLens dataset [4] exhibits similar growth and change: the changes that we observe here are echoed in other datasets as well.

While it is possible to explore CF datasets from a global perspective, it is important to remember that the datasets represent a *collection* of individuals' profiles, and that the global state of the dataset can mask the state and changes that single profiles undergo. For example, as shown in Figure 3, if we first split the users into groups according to profile size, we can then see how the group sizes fluctuate over time. The continuing sparsity of the data is highlighted by the fact that majority of the users are in group a, the group with profiles of fewer than 10 ratings. This group quickly diminishes in size towards the end of the dataset, and the group of users who have more than 100 ratings sharply rises. The full dataset contains only 3% of users with less than ten ratings, while the temporal perspective shows that this group is the dominant proportion. In the following sections we introduce and evaluate algorithms whose performance is dependent on the underlying data structure.

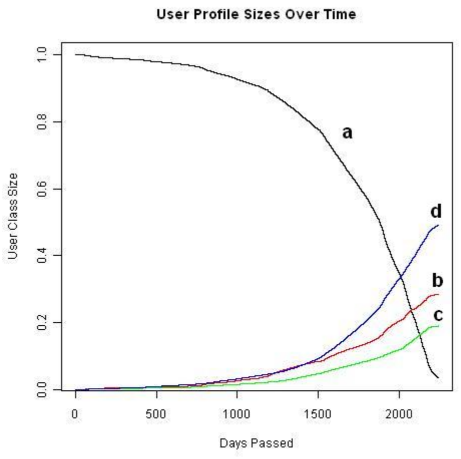


Figure 3: Proportions of Total Users With Different #Ratings Over Time: $a \in [0, 10), b \in [10, 50), c \in [50, 100), d \in [100, \infty)$

3. ALGORITHMS

We focus on two prediction methods that form part of state of the art approaches. The first is a baseline *bias model*, as described by Potter [5], which predicts user u's rating for an item i using u's mean rating (bias, b_u) and the average deviation from each user's mean for item i (preference, p_i). Overall, this method is highly dependent on each user's mean to predict the ratings; we selected to investigate it since user means will change over time. For a set of users U and items I, we define R_u as the ratings input by user u, and R_i as the ratings input for item i. The biases and preferences are initialised as zeroes and then computed by iterating over the following:

$$\forall i \in I : p_i = |R_i|^{-1} \times \sum_{u \in R_i} r_{u,i} - b_u$$

$$\forall u \in U : b_u = |R_u|^{-1} \times \sum_{i \in R_u} r_{u,i} - p_i$$

The iteration continues until the difference in the Root Mean Squared Error (RMSE) achieved on the training set is less than a pre-defined value δ. The predicted rating $\hat{r}_{u,i}$ for a user-item pair is computed as:

$$\hat{r}_{u,i} = b_u + p_i$$

Similar baseline methods have been used to normalise data prior to applying other classifiers to the ratings [6]. A further amendment to the method scales each rating by the user's variance, if that variance is non-zero [5].

The second method we explore is the k-Nearest Neighbours (kNN) approach, since it has been widely applied in CF contexts [1, 6]. This method can been implemented using a user-based or item-based approach, which only differ in terms of the perspective taken on the CF data, i.e. whether items are a collection of user ratings or users are a collection of item ratings. Due to the fact that there are significantly less items than users in the Netflix dataset, we adopt the item-based approach (although the method we describe can be equally applied to users). The algorithm can be decomposed into a number of steps; the first step is to find a *neighbourhood* for each item in the system based on a notion of distance between items. Distance can be measured in a number of ways; in fact, [6] states that the distance function used to select neighbours is arbitrary. In this work, we select neighbours whose profiles minimise the mean error on the current profile. To do so, we define the distance d between two ratings $(r_{a,i}, r_{b,i})$ by users a and b as the absolute difference between the residuals of the ratings, once the respective profile mean $(\bar{r}_a, \bar{r}_b)$ has been subtracted:

$$d(r_{a,i}, r_{b,i}) = |(r_{a,i} - \bar{r}_a) - (r_{b,i} - \bar{r}_b)|$$

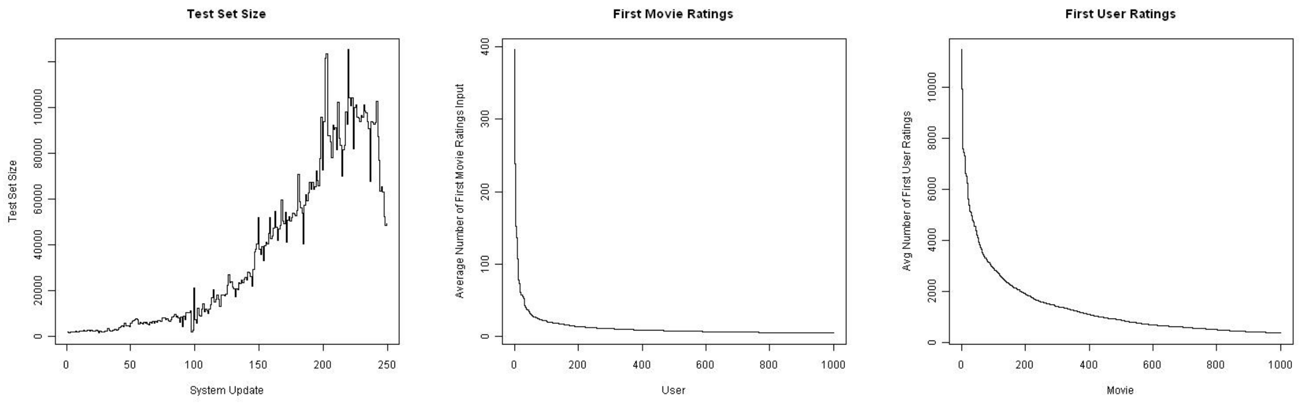
The similarity between profile a and b is then computed as the inverse mean difference from a to b:

$$w_{a,b} = |R_a|^{-1} \times \sum_{i \in R_a} 1 - \beta d(r_{a,i}, r_{b,i})$$

The idea is to award high similarity to neighbours who have low difference between their profiles: we thus invert the difference by subtracting it from one, after it has been scaled by a *penalising* factor $\beta \in [0, 1]$. The role of the penalising factor is to moderate the extent to which large differences between input ratings are punished; even though the two ratings may diverge, they share the common feature of having been input to the system, which is nevertheless relevant in sparse environments such as CF. A low penalising factor will therefore have the effect of populating neighbourhoods with profiles that are very similar in terms of *what* was rated, whereas a high penalising factor places the emphasis on *how* items are rated.

Predicted ratings are computed as the mean of similarity-weighted residuals:

$$\hat{r}_{u,i} = \bar{r}_i + \frac{\sum_{n \in T_i} (r_{n,i} - \bar{r}_i) \times w_{n,i}}{\sum w_{n,i}} \quad (1)$$



(a) Test Set Size Over Time (b) Users Ratings Removed (c) Items Ratings Removed

Figure 4: Temporal Experiment Test Sets' Characteristics: Size, and Distribution of Users Who Rate Items First and Items that Are Rated First

The performance of the kNN algorithm depends on an appropriate k value being selected. Previous work reports values between 20 and 50 [6], although other work has pushed this parameter up to 100 [1]. In this work we investigate 3 k-values, assuming that this is sufficient to explore the temporal behaviour of kNN.

To get insight into the performance of the bias model and kNN (with varying k values used in Equation 1), we tested their predictive power on the Netflix probe set. The probe set contains $1,408,395$ ratings by $462,858$ users on $16,938$ movies. Table 1 reports the RMSE obtained on the probe; from this we see that incorporating the user variance into the bias model results in a slight improvement. None of the kNN results are as accurate as the bias model. This is due to the fact that it is implemented over the raw Netflix data; improvements are possible by first normalising the ratings [6]. However, we do note an improvement in accuracy as the k parameter is increased.

It is important to note that there are a number of other methods that can be used to predict user ratings, ranging from Singular Value Decomposition (SVD) to Bayesian networks [2]. In particular, a number of state of the art approaches combine classifiers to produce hybrid algorithms that aim to reap the benefits of each method while addressing the weaknesses suffered by each one alone. However, in this work we aim to evaluate the changes in performance of the above methods as the recommender system is iteratively updated, and address how parameters can be automatically tuned during this process. The analysis we conduct can be repeated for each classifier available for CF; moreover, we expect that different methods would also exhibit similar results. We therefore limit our work to the above two algorithms.

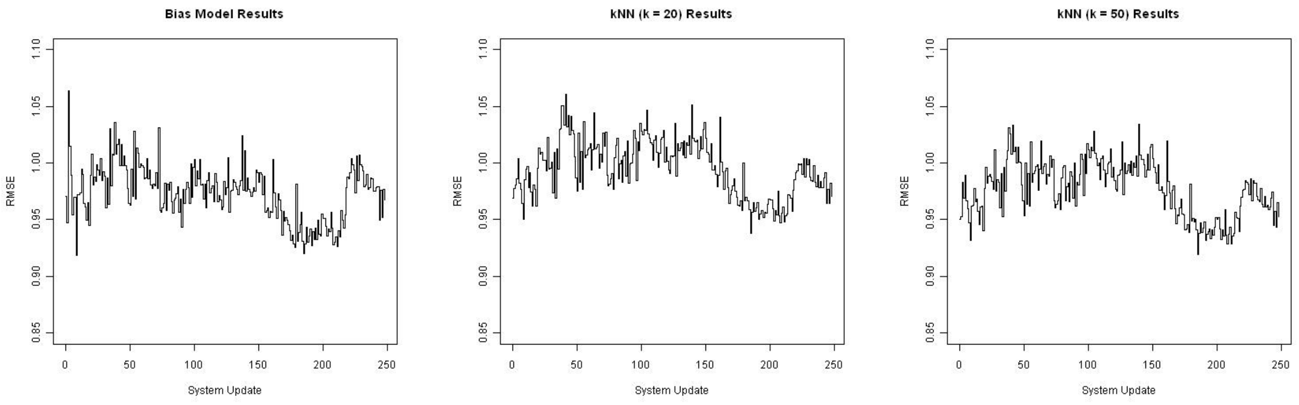
4. TEMPORAL EXPERIMENTS

In the previous section, we described algorithms for predicting user ratings, and evaluated their performance on the Netflix probe. We now evaluate the same algorithms *over time*, as they are iteratively applied to a growing dataset of ratings. In order to cross-validate our results, we subsampled the Netflix dataset. To do so, we split the users into 50 bins (according to profile size) and randomly selected $1,000$ users from each bin; by repeating this process, we produced five subsets of $50,000$ users. We then selected all ratings belonging to these users and any rating input before a predetermined "edge" time ϵ. Including ratings before this time aims to remove the system-wide cold-start problem that recommender systems face, i.e. a state where predictions can not be made as there are little to no historical ratings available [7]. In this work, we set $\epsilon = 500$ days from the first rating in the dataset, and our final subsets have about $60,000$ users: setting the ϵ value as we did is equivalent to bootstrapping a recommender system with $10,000$ users.

Method	RMSE
Bias Model	0.9824
Bias Model + Variance	0.9808
kNN, k = 20	1.0302
kNN, k = 35	1.0134
kNN, k = 50	1.0057

Table 1: RMSE of the Bias Model and kNN Algorithm on the Netflix Probe

In our experiments, starting at time ϵ, the recommender system is updated at regular intervals μ; in this work we set $\mu = 7$ days (the recommender system is updated once a week); our data thus allows for 250 temporal updates. At time t, based on all ratings input prior to t, we aim to predict any ratings that will be input before time $t + \mu$. This set up has two implications, due to the temporal structure of the dataset (as seen in Section 2): on the one hand, the number of historical ratings (or training set) will grow as t increases. On the other hand, the number of ratings in $t + \mu$ (the test set) will also increase, as plotted in Figure 4. We also pruned each test set of any user-item pairs that have a history less than $h = 1$ (whether it be that the user has rated fewer than h movies or the movie has been rated fewer than h times). As above, this was done to avoid the extreme cold-start problem, and from having to modify the algorithms in Section 3 to return predicted values for items with no history. It is interesting to note that pruning the test



(a) Bias Model (b) kNN, $k = 20$ (c) kNN, $k = 50$

Figure 5: Sequential RMSE Results for: kNN Algorithm With $k \in [20, 50]$ and the User Bias Model

sets in this way tends to exclude some users more than others. Figure 4 includes two plots that highlight this feature. Figure 4(b) shows the average number of ratings pruned per user; these ratings are pruned from the test set since the user is rating movies that have no historical ratings. Figure 4(c) shows the equivalent distribution for the movies; these show ratings excluded from the test set because they are the first ratings input by each user. This highlights an important characteristic of the data set: there are certain users who are consistently rating items that have never been rated before (items that have no data available for them to be recommended), and seem to be exhibiting behaviour that extends beyond merely responding to recommendations [3]. There are also movies that consistently appear as users' first rating, which may give insight into what recommendations Netflix was offering to new users.

Sequential experiments differ from individual static experiments since they test the CF algorithms on a wide range of dataset sizes, and can also highlight the temporal performance of these algorithms. There are two ways we can visualise the results. The first is a *sequential* view, where we compute the RMSE on each week's worth of predictions separately (since $\mu = 7$). The alternative is the *continuous* time-averaged RMSE metric. If we define R_t as the set of predictions made up to time t, then the time-averaged error is simply the RMSE achieved on the predictions made so far:

$$RMSE_t = \sqrt{\frac{\sum_{\hat{r}_{u,i} \in R_t}^{N} (\hat{r}_{u,i} - r_{u,i})^2}{|R_t|}} \qquad (2)$$

It is important to note that, at time t, predictions are *only* made for ratings in $t + \mu$; no predictions are recomputed or updated for ratings that the users have already input or will input after $t + \mu$. In other words, predictions are only made once. This may differ from deployed recommender systems, that do not know *when* users will rate items, and will therefore be able to update predictions until the user inputs a rating. However, in our case we assume that the rating that has the greatest impact (in terms of the recommendations generated for each user) is the last one made before the rating is input, and we thus limit ourselves to only exploring these values.

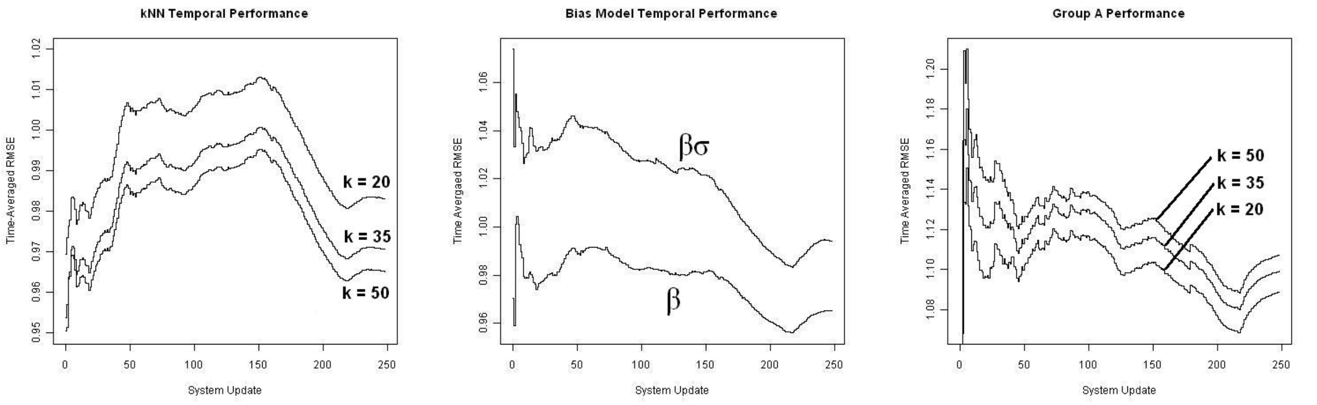
4.1 Sequential Results

The sequential results for the bias model and $k = 20, 50$ Figure 5. From these we can observe that CF algorithm performance lies on a *range* of values. The $k = 50$ results fall in $[0.9193, 1.034]$, while the range for $k = 20$ is slightly worse, $[0.9383, 1.0608]$; nevertheless, the ranges overlap significantly. However, while the bias model outperformed both kNN methods on the probe, its temporal performance is between 0.9186 and 1.0637: at best, it shows a minor improvement over $k = 50$, while in other cases is outperformed by $k = 20$. The performance across all three methods falls by 0.02 between the 218th and 219th update, highlighting a change in the *data* that results in all methods degrading in performance. While the trend between the different plots is roughly similar, the precise moments that each algorithm performs best (or worst) differs between each method. Both kNN methods achieve their lowest RMSE on the 186th update; however, $k = 50$ yields its worst performance on the 140th update, while the equivalent for $k = 20$ happens at the 42nd update. The bias model, on the other hand, achieves both the best and worst performance within the first 10 updates.

Viewing the sequence of RMSE results emphasises the difficulty of identifying which algorithm outperforms the others. However, the $k = 50$ parameter was more accurate than $k = 20$ in 248 (of the 250) iterations. The balance between $k = 50$ and the bias model is not as one-sided: the bias model is more accurate in two-thirds of the updates, while in the other cases the kNN model is more accurate. In other words, while it is possible to deduce relative performance based on a set of results, the best performing method in any *individual* time segment varies.

4.2 Time-Averaged Results

The time-averaged results of 5-fold cross validated experiments are shown in Figure 6. This visualisation provides a different perspective on the experiment results, and there are a number of observations that can be made. Recall from Section 3 that larger k values resulted in improved accuracy. Figure 6(a), the time-averaged results, show a similar ordering: $k = 50$ over time outperforms $k = 20$. We also tried experiments with $k = 0$; in this case the current item



(a) kNN $k = (20, 35, 50)$ (b) Bias Model (c) kNN For Users With $|R_u| \in [0, 10)$

Figure 6: Time-Averaged RMSE for: kNN Algorithm With $k \in [20, 35, 50]$, User Bias Model, With ($\beta\sigma$) and Without (β) User Variance, and the Group of Users With Fewer Than 10 Ratings

mean is returned. All values of $k \neq 0$ consistently outperformed the baseline; this result persists if the current *user* (rather than item) mean rating is returned. The difference in time-averaged performance of each kNN parameter setting is less than 0.02, and remains approximately constant after the 50th system update. The performance itself varies: after the $50th$ update predictive accuracy wanes. However, after the $150th$ update performance once again improves, falling sharply by 3% in the case of $k = 50$. This highlights the dependence that these methods have on the *quality* of the data they train on, a matter we plan on exploring with more detail in future work.

Figure 6(b) plots the time-averaged performance of the bias model. The bias model with variance scaling is consistently outperformed by the model that has no variance adjustment. The difference in performance are in the range $[0.03, 0.1]$: scaling user ratings with a *dynamic* variance introduces more error to the predictions. Why do the probe and temporal results differ? One indicative factor is the difference in the rating distribution over time; as Figure 3 shows, users with fewer than 10 ratings make up more than half of the dataset for most of the interval we consider. However, only 3% of the users remain in this group when considering the entire dataset. The majority of the user variance values, in the temporal case, are therefore computed with incredibly sparse data.

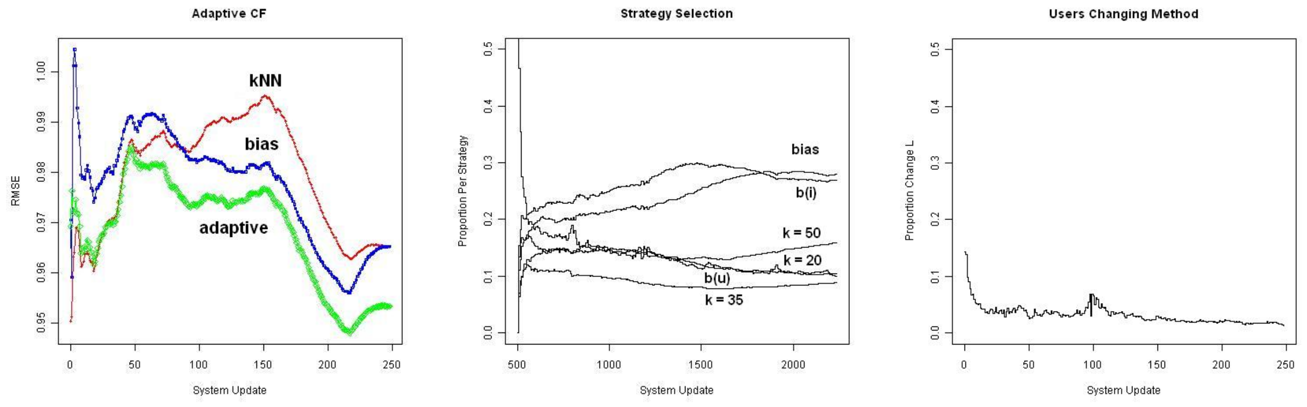
Comparing Figures 6(a) and 6(b): Although the bias model outperforms kNN on the raw data when predicting the Netflix probe, it does not consistently outperform kNN on the temporal scale. For example, in the 4th update, the bias model time-averaged result is 1.004, while the kNN result is 0.964. From these results, kNN with $k = 50$ emerges as the most temporally accurate method. However, we also explored how prediction error is distributed across a community of *individuals* by, once again, splitting users into groups according to profile size and plotting the group's 5-fold cross-validated time-averaged performance. As expected, group performance is proportional to the range of ratings that defines the group: the group of users who have fewer than 10 ratings also have the least accurate predictions, compared to the groups with more ratings. However, as shown as the rightmost plot of Figure 6(c), the k value performance in the group with fewer than 10 ratings is the opposite of what we observed when all groups were merged: larger neighbourhoods leads to *less* accurate results. In fact, there is often no consensus between the method that produces the best *global* performance and that which best suits *each user*, and predictive performance can be greatly enhanced if per-user adaptive methods are provided.

5. ADAPTIVE NEIGHBOURHOODS

Given this context, when iteratively updating the CF algorithm, we also select a future prediction method based on the current performance. As evaluated above, prediction methods are iteratively applied as the data grows; the only change from one step to the next is rating *data* that is input to the algorithm. In particular, no information on the current performance is fed forward to the next iteration of the algorithm. We therefore propose temporally *adaptive* collaborative filtering, which will make use of this information to change the algorithm that is used at each iteration. There are two adaptive methods that we explore and evaluate. The first selects between different algorithms (Section 5.1), while the second is based on only adapting kNN parameters (Section 5.2).

5.1 Adaptive CF

To implement temporally adaptive CF, we begin with a pre-defined set P of CF algorithms. In this work, the set includes kNN, with $k = \{0, 20, 35, 50\}$, and the bias model; the two algorithms that we previously evaluated independently from one another. A $k = 0$ value disregards neighbourhoods completely; in this case we can either return a baseline item ($b(i)$) or user ($b(u)$) mean rating (there are six candidate methods altogether). Each user u is assigned a label $L_{u,t}$ denoting which algorithm L best predicts their preferences at time t. At each time step t, all users u also have a corresponding error value $e_{u,t}$ denoting the time-averaged RMSE achieved on the predictions made to date on *their profile*. We therefore aim to optimise the per-user e value by selecting the $L \in P$ that would have maximised the improvement



(a) Adaptive CF (b) User Proportions Per Algorithm (c) User Change Proportions

Figure 7: Time-Averaged RMSE Comparing $k = 50$, the Bias Model, and Adaptive CF, Proportions of Users Who Selected Each Algorithm Over Time, and Proportions of Users Who Changed Method At Each Interval

on the current error:

$$\forall u : L_{u,t+1} = \max_{L_i \in P} (e_i - RMSE_{t,P_i}) \quad (3)$$

Although the previous analysis binned users according to profile size, and demonstrated that relative performance varies depending on the group being considered, we did not opt to adapt based on which "group" users belonged to. We did this for two reasons: first, the grouping was done with pre-defined values that could themselves benefit from fine-tuning; secondly, this form of grouping continues to mask the predictive performance on *individual* profiles, and the aim we envisage for adaptive filtering is based on addressing users' profiles individually. In doing so, the CF algorithm that provides personalised recommendations becomes itself personalised.

The five-fold cross-validated time-averaged RMSE results are plotted in Figure 7(a), compared to the two best individual methods; kNN with $k = 50$ and the bias model. As the plot shows, the adaptive method begins by following the same pattern as the kNN curve, but then (as the bias model becomes more accurate) departs from this pattern and becomes more accurate than either model alone. In fact, adapting on a per-user basis offers better temporal accuracy than if we simply selected the minimum of the two methods. We also plotted in Figure 7(b) the proportion of users who select each method over time. The results show that, while the bias model dominates over the others (in terms of the proportion of users that the algorithm selects the bias model for), it is selected for less than 30% of the users: no single model 'best' predicts the majority of end users.

To gain insight into how often the algorithm needs to change a decision it had previously made, we plotted the proportion of users with different labels than at the previous update in Figure 7(c). Overall, very few of the growing population of users changes method from one update to the next; overall, the change is consistently between 1.3% and 14.3% of the growing user community, and on average is $3.1 \pm 1.6\%$.

5.2 Adaptive kNN

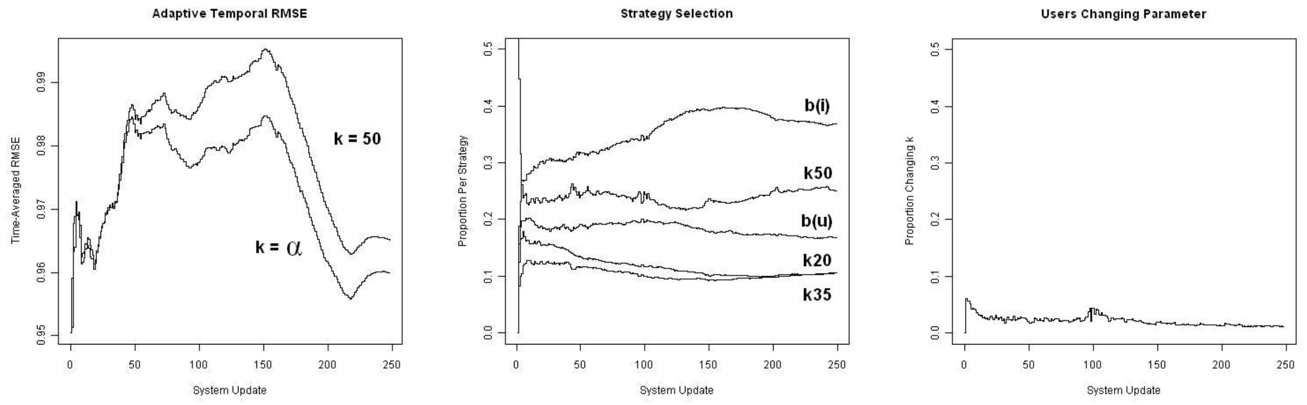
While the above method offers greater accuracy, it does so at a great expense: multiple CF algorithms must be implemented and independently trained at each update on the growing data. Given the volume of data that large scale recommender systems must handle and the time it takes to train CF algorithms [8], repeating this process with multiple algorithms may be prohitive and difficult to scale.

We therefore also explored a method that only tunes the kNN parameters. To do so, we first select a subset of potential k values $P \subset \mathbb{N}$. In this work, $P = \{0, 20, 35, 50\}$, following the experiments that we performed above. We then proceed to set a value $k_{u,t} \in P$ for each user u at time t. When new users enter the system, their $k_{u,t}$ value is bootstrapped to a pre-determined member of P. The idea is for each $k_{u,t}$ to be set to that which would have provided the steepest improvement on the users' $e_{u,t}$ value in the last time step, just as shown in Equation 3:

$$\forall u : k_{u,t+1} = \max_{k_i \in P} (e_i - RMSE_{t,P_i}) \quad (4)$$

It is important to note that this parameter update method is independent of the particular flavour of kNN that is implemented. In other words, it is equally applicable to both the user-based and item-based approaches; for example, if the item-based approach is implemented (as we have experimented with above), then a prediction $\hat{r}_{u,i}$ of item i for user u is done by aggregating ratings by k similar *items.* It could also be applied to the user-based approach, where predictions would aggregate ratings from k similar *users.* We still aim to optimise performance on a per-user basis.

The results are plotted in Figure 8. Figure 8(a) compares the five-fold cross validated time-averaged RMSE results of the best *global* parameter setting ($k = 50$) and the adaptive technique. The results highlight a number of benefits of adaptive CF. In particular, the adaptive strategy at first rivals the performance of $k = 50$, but then improves the overall time-averaged RMSE, without requiring any manual parameter tuning. In these runs we opted for the bootstrapping setting to be $k = 50$, since it performed the *worst* when predicting users with very small profiles, as plotted in Figure 6(c). It will thus tend to disadvantage new entrants to the



(a) Adaptive kNN (b) User k-Value Proportions (c) User Change Proportions

Figure 8: Time-Averaged RMSE Comparing $k = 50$ and Adaptive ($k = \alpha$) kNN, Proportions of Users Who Selected Each k Value Over Time, and Proportions of Users whose k Value Changed At Each Interval

system; however, we still can see an improvement in temporal accuracy. Changes to the bootstrapping value affected the first number of updates (producing curves similar to the left plot of Figure 6), but, after a number of updates, all the values we tested differed in performance by less than 0.001; they all outperformed $k = 50$.

To explore how the different parameter settings are distributed amongst members of the system over time, we plotted the proportions of current users who have adopted each setting, shown in Figure 8(b). From this, we see that users both do not converge to a single parameter and that the dominant strategy (selected by up to 40% of the current users) is the baseline item mean, followed by $k = 50$, the user mean, $k = 20$, and lastly 35. The most selected method, when operating alone, was consistently outperformed by all other k values. However, it plays an important role in providing greater temporal accuracy to the adaptive case.

The method we have outlined allows the k value for each user to be updated at *every* interval. The k value is changed if a different value would have yielded better predictions at the current time; it is possible, therefore, that this k values would continuously fluctuate without finding a stable value. To explore this possibility, we graphed the proportion of users who *change* neighbourhood size over time in Figure 8(c), and found that only a very small proportion of the user neighbourhood sizes are being changed at any given update. On average, only 2% of the current users change neighbourhood size; at most, 6% adopt a new size for the next interval. While this does not imply that users are converging and remaining on the optimal strategy, it highlights the proportion of users with parameters *not* set to the best member of P.

The improved accuracy of adaptive-kNN comes at little cost: the computational overhead is minimal. The cost of computing predictions remains the same, since, for example, the computations for both the $k = 20$ and 35 predictions for a user-item pair are contained within those required to compute $k = 50$. User profiles need to be augmented to include e_i, the error achieved to date, and a set of error values that each k has achieved in the current time step.

While the notion of adaptive-CF has been applied here to *temporal* collaborative filtering, it can also be applied to the static case. In the latter context, the problem is that of determining appropriate k values from in a single step. We leave a full analysis of adaptive CF in the static case as a topic of future work; however, here we explore the potential for improvement by reporting the results of the *optimal* case. Given $P = \{0, 20, 35, 50\}$, if we select the optimal parameter setting for each user (assuming full knowledge of the RMSE each method produces for each user), the probe RMSE would be 0.8158. This error lies below the threshold for the Netflix prize, and is achieved by adaptively selecting from 5 techniques that *alone come nowhere close to this mark*. Furthermore, there is no single method that dominates over the others: 22% select $k = 20$, 12% opt for $k = 35$, 14% select $k = 50$, 24% the item mean, and 26% the user mean rating. Interestingly, the two baseline (mean rating) based methods together compose half of the users in the dataset.

6. RELATED WORK

A large proportion of previous work in this area attempts to optimise performance of a *single* train-predict-recommend iteration; one approach has been to produce hybrid algorithms by merging candidates from the wide range of classifiers available [9]. Adaptive-CF differs from hybrid methods since, rather than focusing on merging different predictive models, individual methods are selected based on current performance. To that extent, adaptive-CF is independent of the particular set of selected classifiers that it alternates between, and falls under the broader category of available meta-learners [10], although we strictly consider the temporal scenario. It is therefore also possible to widen the set of choices available in order to further improve accuracy; for example, some users' ratings may be best predicted by performing a SVD with a varying number of features. We have not included this possibility here since doing so may well also introduce the potentially prohibitive cost of computing many models in a deployed system.

Previous work that highlights the importance of time in CF (and in related fields, such as information retrieval [11]) tends to focus on the data, rather than the sequential application of an algorithm. For example, Potter [5] (whose bias model we explored above) and Bell & Koren [6] also consider the temporal nature of ratings, by looking at the variability of individual user ratings across different days of the week in order to improve predictive performance. However, these methods rely on knowing *when* users will rate items (in order to scale predictions accordingly), and are therefore not readily applicable in deployed systems. Temporality has also been explored from the point of view of changing user tastes [12]; in this case, ratings are scaled according to when they were input. The aim is to capture the most *relevant* ratings that represent current user tastes. In fact, both our adaptive-CF and this method could be merged: in this work we focus on the algorithm rather than modifying the ratings.

7. CONCLUSION

This work departs from traditional CF research by extending the analysis of prediction performance to incorporate a sequence of classification iterations that learn from a growing (and changing) set of ratings. We then implemented and evaluated a cheap method that automatically tunes parameters to provide greater temporal accuracy.

The focus of this work has revolved around optimising recommender system prediction performance from the point of view of RMSE. The results show that these algorithms do not output consistent error, and it becomes difficult to claim that one algorithm outpredicts another when only a static case is investigated (and especially when the static difference in performance is relatively small). For example, the bias model was more accurate than raw-data kNN on the Netflix probe, but did not maintain this advantage when a range of datasets (of varying size) were tested in an iterative set of cross-validated experiments. We plan on investigating the effect that the update interval μ has on these results. However, this observation also motivates research that departs from traditional mean-error based evaluations of CF algorithms toward evaluating the top-N recommendations generated by a prediction algorithm. These kinds of evaluations aim to explore the *ranking* that emerges from rating prediction, and the utility that users draw from the lists of recommendations they are offered. A further evaluation of adaptive-CF would therefore also encompass the variation in recommendations that results from user parameters being constantly updated, since a consequence of this method may be a strong temporal diversification of the recommendations provided [13].

This work has focused on the temporal performance of CF algorithms, without considering (a) the extent to which user preferences and interests will vary over large time intervals, and (b) the temporal effect of malicious ratings [14]. In particular, as the experiments in Section 4 highlight, performance does not necessarily improve as the available training data grows. We plan to explore how adapting the training set size affects performance, both to address the robustness and improve the temporal accuracy of online recommender systems.

8. REFERENCES

[1] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of ACM SIGIR Conference*, pages 230–237, 1999.

[2] G. Adomavicius and A. Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 17(6), June 2005.

[3] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM TOIS*, 22(1):5–53, 2004.

[4] N. Lathia, S. Hailes, and L. Capra. kNN CF: A Temporal Social Network. In *Proceedings of Recommender Systems (ACM RecSys '08)*, Lausanne, Switzerland, 2008.

[5] G. Potter. Putting the Collaborator Back Into Collaborative Filtering. In *Proceedings of the* 2^{nd} *Netflix-KDD Workshop*, 2008.

[6] R. Bell and Y. Koren. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *IEEE ICDM*. IEEE, 2007.

[7] A. Nguyen, N. Denos, and C. Berrut. Improving New User Recommendations with Rule-Based Induction on Cold User Data. In *ACM RecSys*, 2007.

[8] M. Mull. Characteristics of High-Volume Recommender Systems. In *Proceedings of Recommenders '06*, Bilbao, Spain, September 2006.

[9] A. M. Rashid, S.K. Lam, G. Karypis, and J. Riedl. ClustkNN: A Highly Scalable Hybrid Model-Memory-Based CF Algorithm. In *Proceedings of ACM KDD*, August 2006.

[10] R. Vilalta and Y. Drissi. A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review*, 18(2), October 2002.

[11] O. Alonso and M. Gertz. Clustering of Search Results Using Temporal Attributes. In *Proceedings of ACM SIGIR 06*, Seattle, USA, August 2006.

[12] Y. Ding and X. Li. Time Weight Collaborative Filtering. In *Proceedings of ACM CIKM 05*, Bremen, Germany, November 2005.

[13] J. A. Konstan G. Lausen C.N. Ziegler, S. M. McNee. Improving Recommendation Lists Through Topic Diversification. In *Proceedings of WWW 05*, Chiba, Japan, May 2005.

[14] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. In *ACM TOIT*, 2007.