Chapter 4

Temporal Accuracy of Collaborative Filtering

The primary task of a recommender system is to take user ratings and predict the values that users would attribute to content they have not rated, in order to generate personalised ranked lists of recommendations. Intuitively, the changes in the rating datasets that we have observed in the previous chapter will affect the performance of any learning algorithm that is iteratively retrained with the user ratings. In this chapter, we explore the extent to which this intuition is true: we first define a methodology for performing collaborative filtering temporal experiments and discuss a variety of design decisions that we made by reporting the results of two case study experiments (Section 4.1). We then perform and analyse a set of cross-validated temporal experiments with the Netflix data (Section 4.2). The key observation that we make is that state of the art filtering algorithms that are regularly batch-updated are not aware of their own temporal performance; we thus hypothesise that introducing this feature will improve an algorithm's temporal accuracy. We test this hypothesis in Section 4.3 by designing and evaluating a hybrid-switching CF algorithm that modifies how it predicts user ratings according to its performance to date.

4.1 Measuring Temporal Performance

At the broadest level we consider a scenario where, given a time t, we will train the CF algorithm with all ratings that have been input prior to t and want to predict the ratings (or a subset theoreof) that will be input *after* t. We then require a means of tracking performance over time. In this section, we examine the range of choices available when designing an experiment that mimics recommender systems that are updated.

4.1.1 Simulating Temporal Updates

Our generic description above prescribed a method for iteratively retraining CF algorithms. The simulated recommender system begins at time ϵ and will be updated at different times t. When an update occurs, the CF algorithm is retrained with the currently available ratings and then it derives predicted ratings of unrated items in order to present each user with personalised recommendations. There are a number of challenges that we face:

1. **Starting Point**: When should we begin the train-test cycle? If we begin at the first available date in the dateset, we will be making predictions with no ratings to learn from. In other words, how many ratings are enough to bootstrap a recommender system?

2. **Updates**: how often should the system be updated? Should we retrain the algorithm with all ratings input prior to the one we would like to predict? Or should the system be updated at some predefined regular interval (daily, weekly, monthly)?
3. **Test Sets**: how are they to be defined? By retraining CF algorithms with a growing dataset, we are simply performing a sequence of updates where the training set has been augmented. However, what should we be predicting? The test set could be a *static* set of ratings that will be input in some arbitrary time in the future, or a *changing* set of ratings based on what will be rated after the current update. Unlike the traditional methodology, we may also encounter a situation in which multiple predictions can be made for a user-item pair before the user rates the item. For example, if we are updating the system weekly, predicting all unrated items, and a user will rate an item one month after the current update, then we will make four predictions of the same rating prior to the user rating the item. Should they all be included in error measurements? If not, which one is the most *relevant*?

We explore these questions in Section 4.1.3 by comparing the results of different experiments; however, we first define the options available to measure temporal accuracy.

4.1.2 Metrics: Sequential, Continuous, Windowed

In terms of prediction error, there are three ways that a set of recommender systems' temporal updates can be evaluated. The first is a *sequential* view, where we compute the RMSE on each experiment separately. The alternative is the *continuous* time-averaged RMSE metric. To observe the dependence of prediction error on time, we modified the RMSE calculation, in a manner akin to that of the Time Averaged Rank Loss that is described in [CS01]. 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:

$$RMSE_t = \sqrt{\frac{\sum_{\hat{r}_{u,i} \in R_t} (\hat{r}_{u,i} - r_{u,i})^2}{|R_t|}} \tag{4.1}$$

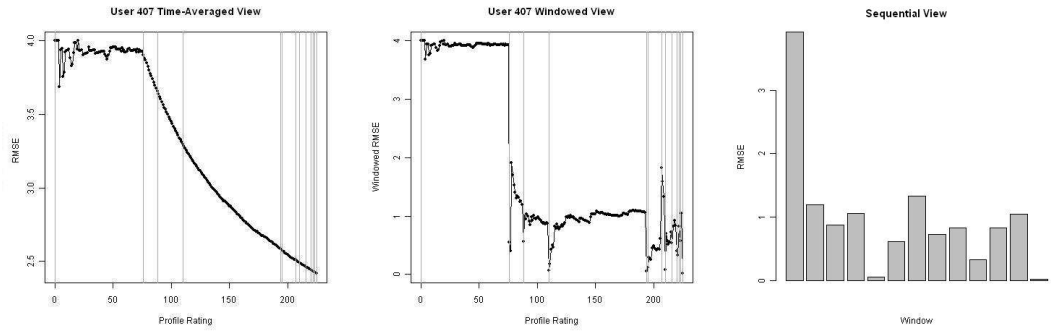
Similarly, we can define the time-averaged mean absolute error (MAE):

$$MAE_t = \frac{\Sigma_{i=0} |\hat{r}_{u,i} - r_{u,i}|}{|R_t|} \tag{4.2}$$

The last possibility is the *windowed* view. Error is accumulated and tracked using the continuous equations above, but once an update has been performed, we reset the error count to zero. This allows us to see how prediction error is distributed within a single update: are predictions more accurate immediately after the update? Do they become less accurate as time passes (since new ratings have not been accounted for in the CF algorithm)?

4.1.3 Case Study

Prior to committing to a particular methodology, we explore the options available by running a number of experiments. In the first, we focus on a single user from the MovieLens dataset; we then expand our analysis to include all of the MovieLens users. These experiments allow us to reason about what choices to make regarding the experimental starting point, update frequency, and predictions. Lastly, we run a



(a) Time-Averaged RMSE (b) Windowed RMSE (c) Sequential RMSE

Figure 4.1: User 407: Three Views of Temporal Error

group of experiments that look at temporal updates with a *static* test set and conclude that, while this form of experiment does not reflect the reality of recommender systems, they provide important insight into the influence of rating data on prediction performance.

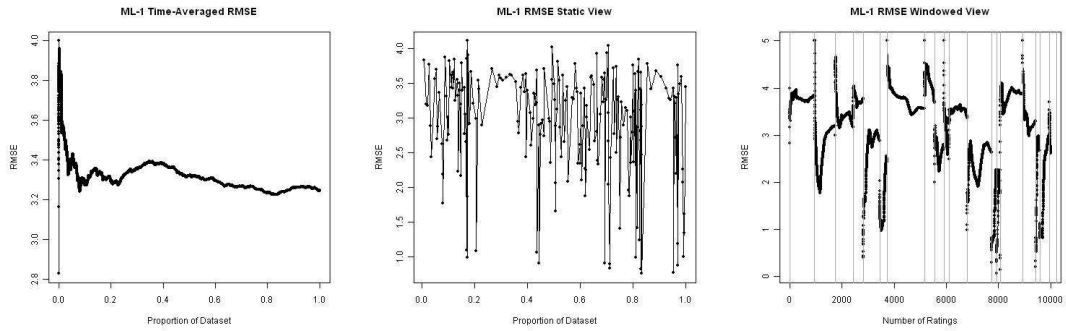
Single User

We begin with a single user; we picked the ratings of user 407 from the ML-1 dataset since they span a relatively lengthy time scale. We also make the following assumptions:

- The system will be updated *daily*; at each update user similarity is recomputed with all ratings input to date and predictions are made for any ratings that will be input before the next update. The first update occurs exactly one day after the first rating is input to the system. This allows us to (a) minimise how far into the future we have to predict (thus minimising any bias that may result from this) and (b) have a very fine-grained view of the system.
- Predictions will be made by a user-based kNN algorithm, where $k = 10$ and user-similarity is computed with the Pearson Correlation Coefficient. We therefore use a simple algorithm which has been extensively studied in the past [HKBR99].

We plot three temporal perspectives of the error in predicting user 407's ratings in Figure 4.1. Vertical gray lines in Figures 4.1(a) and 4.1(b) denote when the system was updated; each point represents the input of a successive rating. From these, we can make a number of observations:

- The user does not rate items consistently; for example, the number of items that were rated before the first update are far greater than those input before the second.
- All ratings input prior to the first update have a distinctly large error. From the algorithm's perspective, there is no data to use to predict this user's interests. The *cold-start* problem is often described as an issue that users with *few* ratings face; what we observe here, where the user has no historical profile, is an extreme version of it (i.e., had no ratings to generate a neighbourhood at the previous update and no mean rating value to provide an appropriate baseline prediction). Figures 4.1(c) and 4.1(b) show that the cold start region lasts until the next update; unfortunately, the time-averaged results in Figure 4.1(a) are skewed throughout the entire duration of the predictions by these initial predictions (the plotted error is on the range $[2.5, 4]$).



(a) Time-Averaged RMSE (b) Sequential RMSE (c) Windowed RMSE

Figure 4.2: ML-1 Dataset: Three Views of Temporal Error

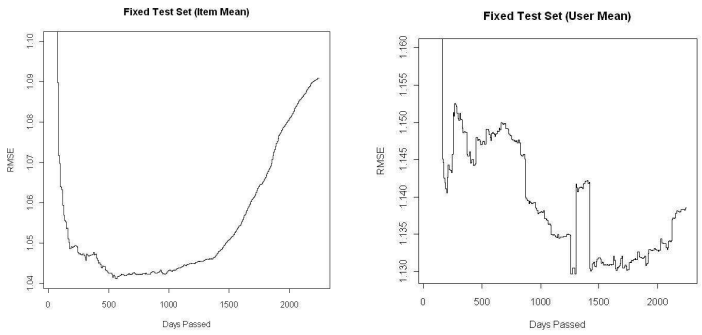
- Prediction error does not improve over time for this user. The time-averaged results seem to imply that predictions are improving; however, as stated above, this is due to the skew from the initial predictions. After the cold-start day, the error of predictions made in any given window range lies between slightly above 0 to just under 2.

We thus find that the time-averaged metric will only be appropriate if we do not include cold-start predictions. The windowed and sequential metrics do not suffer from this problem; in fact, they have already highlighted the large variability in prediction accuracy as time passes.

Groups of Users

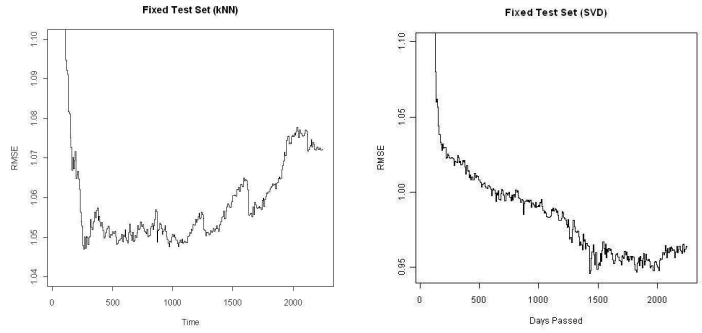
Based on the observations above, we broadened the scope of the experiment and included all the ML-1 users. This way we can view the same results as above for a large group of users: we plot these in Figure 4.2. The results highlight a number of points:

- Again, the time-Averaged RMSE is of little value if we include cold-start predictions. As above, users face the cold-start problem when they have no historical ratings; they thus have no mean rating value or neighbours. Our options here are to (a) modify our prediction algorithm in order to return an appropriate non-zero prediction for cold-start users, or to (b) disregard cold-start predictions. Since the cold-start problem has been approached from a variety of perspectives [NDB07, PPM+06] we opt for the latter in this work rather than limit ourselves to a single available solution.
- The windowed perspective (Figure 4.2(c)) shows that inter-window behaviour does not follow a single pattern. There are some windows that, as they progress, become ever more accurate; there are also windows that become less accurate as time passes. However each window is distinctly different from the others: how many ratings are input, along with what items are being rated, continuously changes.
- The sequential view (Figure 4.2(b)) is a summarised form of the windowed perspective; each point represents the *average* error of each window. In fact, since our prediction model is updated iteratively, the windowed results (Figure 4.2(c)) will be subject to the order that ratings are input. The static and time-averaged views, instead, are both not subject to this limitation and useful for



(a) Item Mean (b) User Mean

Figure 4.3: Temporal Experiments With a Static Test Set (User/Item Mean)



(a) kNN, k = 50 (b) SVD, f = 64

Figure 4.4: Temporal Experiments With a Static Test Set (kNN/SVD)

measuring performance across updates.

Predicting Static Test Sets

How should we define our test set? We can either keep the test set fixed and only change the training set, or update both as the simulated temporal updates are performed. In the context of a deployed recommender system, we assume that the prediction that has the greatest impact (in terms of the recommendations generated for each user) is the last one made before the rating is input. Why? Changes in the predictions as they are updated will affect each user's recommendations: when users rate items, only their current recommendations (i.e., ranked prediction values) will be influencing their decisions. Therefore, at time t (with update frequency μ), we decided to only make predictions for ratings that will be input 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 not be able to update predictions until the user inputs a rating. However, as described above, this allows us to focus on the predictions that will have the greatest impact on a user when they are rating an item. In the following chapters, we will remove this assumption when we evaluate the top-N lists created over time.

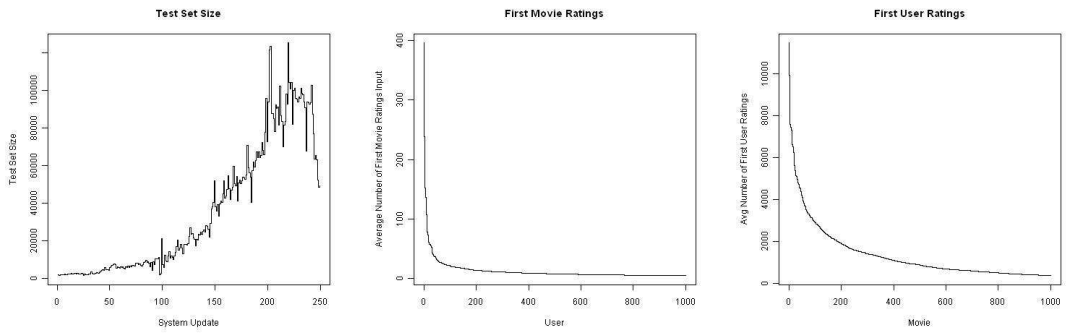
The alternative to the above would be to keep a *static* test set. In other words, we define an (unchanging) set of ratings that we would like to predict, and observe the prediction error as we add more

data to the training set. This allows us to explore how prediction quality varies with time from the opposite point of view of the above: we can see the performance trajectory of the system as it approaches the state where the user will rate the predicted item. We tried this setup with the Netflix dataset: we first made a static test set, consisting of all ratings input in the first fifty days of the dataset, and reserved the rest as training data. We selected the ratings from the first (rather than last) fifty days since this guaranteed that the items we are predicting will already be in the system and will continue to be rated in the training data. We selected four CF algorithms: two baseline prediction methods (the user and item mean), an item-based kNN with $k = 50$ neighbours, and a SVD with 64 user and item features; this range of choices both reflects state-of-the-art CF and each manipulate the rating data in different ways. We then iteratively trained each algorithm with a growing dataset, incrementing it with one week's worth of ratings at each round. The choice of one week increments is arbitrary; in our case, it allows for a relatively high number of ratings to be added to the training set. Given that we used the Netflix dataset for these experiments (which spans a longer time frame), this choice also requires fewer iterations of the algorithms to be run. After each training phase, we queried the algorithms for predictions of the test set, and plot the time-averaged RMSE results in Figures 4.3 and 4.4. All of the results share common traits: on the left side of the plots, where very little training data is available, RMSE values are very high. These results hint at the fact that when more training data is available CF algorithms will be able to make better predictions. Each prediction method's results shows different amounts of variability; the most notable is the user mean, which has very clear changes in performance when the test set users add more ratings to their profile. However, all of the methods' best predictions were made prior to the full training data being made available to the algorithms. Even the SVD, with RMSE results that seem to decrease monotonically over time, hits a minimum value before all the data has been given to it. All of the minimum values occur at different times, highlighting how prediction algorithms will each be affected in different ways by the available data, and any noise within it [APO09].

The purpose of these experiments was to see how CF accuracy is affected by a growing training set, from the point of view of a fixed set of ratings that need to be predicted. In practice, these results show us how accuracy will vary as predictions for unrated items are updated. For example, assume that a user does not rate a movie for one month after it becomes available. The system does not know when the user will rate the item and will thus continue updating its prediction until the true rating is input. These results show us that the prediction (which determines whether or not the movie will be recommended) may suffer from high variability and will not necessarily improve as time passes. Deployed recommender systems, however, do not have the luxury of having a closed test set: as we saw in Chapter 3, the available items will continue to grow over time. Real systems will thus have to face a *dynamic* test set: they continuously have to predict the future. In the following sections, we perform experiments reflecting this context; we begin by explicitly defining how we will do so.

4.1.4 Methodology

Based on the exploratory work we reported above, we define here how we conducted temporal experiments. Given a dataset of timestamped user ratings, a start time ϵ and update frequency μ, we define:

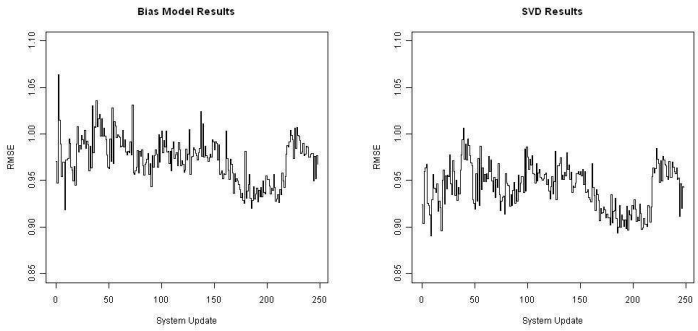


(a) Test Set Size Over Time (b) Users Ratings Removed (c) Items Ratings Removed

Figure 4.5: Temporal Experiment Test Sets' Characteristics: Size, and Distribution of Users Who Rate Items First and Items that Are Rated First

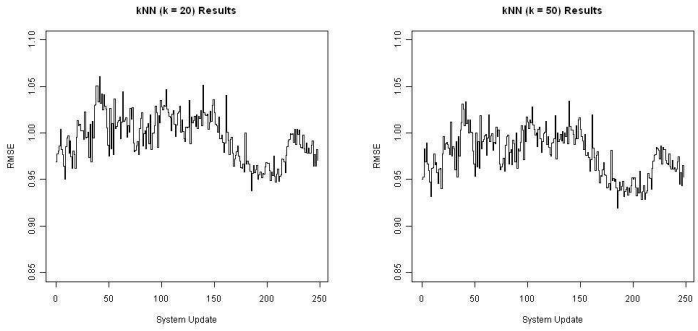
1. **Starting Point**: we define the starting time as ϵ and elected to wait for an arbitrary number of days before beginning the train-test cycle; this allowed us to observe a system that (in terms of number of ratings) is not suffering from system-wide cold start problems. We denoted this number of days as the "edge." In the Netflix experiments below these are any ratings input in the first 500 days of the dataset; our data thus allows for 250 temporal updates.
2. **Updates**: we elected to update the system based on accounts of deployed large-scale recommender systems [Mul06]; the system will be updated weekly. When it is updated, it will train with all ratings input up to the current time.
3. **Test Sets**: After each update, the system will be queried for predictions concerning ratings that will be entered before the next update, only if both the user and item have at least 1 historical rating. We thus still expect to see how our algorithms cope with the cold start problem; however, this assumption will remove the need to define a default prediction to return in the case of no history.

This setup has two implications, due to the temporal structure of the dataset: 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 4.5. It is interesting to note that pruning the test sets of items and users who have no history tends to exclude some users more than others. Figure 4.5 includes two plots that highlight this feature. Figure 4.5(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 4.5(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 [HKTR04]. There are also movies that consistently appear as users' first rating, which may give insight into what recommendations Netflix was offering to new users.



(a) Bias Model (b) SVD

Figure 4.6: Sequential RMSE Results for User Bias Model and SVD



(a) kNN, $k = 20$ (b) kNN, $k = 50$

Figure 4.7: Sequential RMSE Results for kNN Algorithm With $k \in \{20, 50\}$

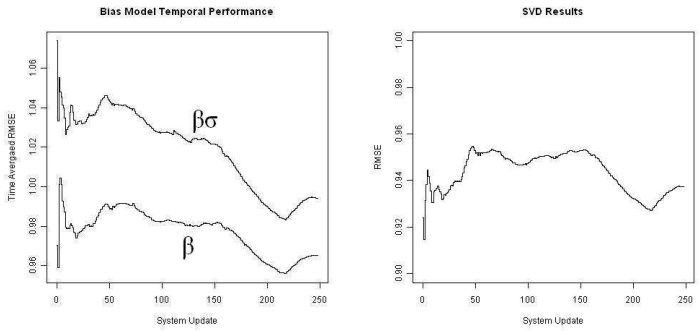
4.2 Results

We now evaluate CF 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 time ϵ. 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.

We focus on three algorithms: Potter's bias model [Pot08], an item-based kNN (with $k \in \{20, 35, 50\}$), and a SVD with 64 user and item features. In doing so, we cover *baseline* models, the ever-popular *nearest-neighbour* method and a *factorisation*-based approach, which represent three different and important state-of-the-art algorithms.

4.2.1 Sequential Results

The sequential results for the bias model, kNN with $k = 20, 50$, and the SVD are in Figures 4.6 and 4.7. 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 [LHC09b], its temporal performance is between 0.9186 and 1.0637: at best, it shows a minor



(a) Bias Model (b) kNN For Users With $|R_u| \in [0, 10)$

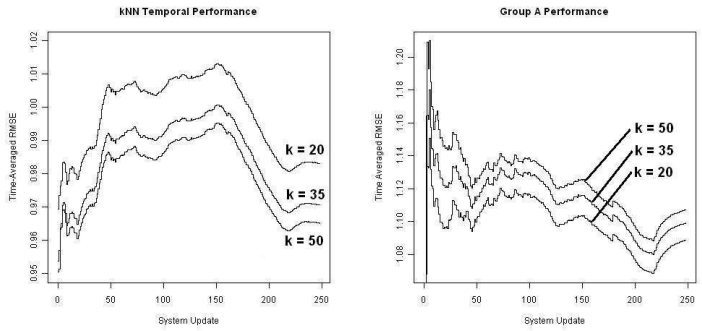
Figure 4.8: Time-Averaged RMSE for User Bias Model and SVD

improvement over $k = 50$, while in other cases is outperformed by $k = 20$. Similarly, the SVD values range between $[0.8907, 1.0061]$; while achieving the best minimum, that values are not consistently lower than the other algorithms. The performance across all methods falls by approximately 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 achieves both the best and worst performance within the first 10 updates. The SVD, instead, hits its minimum on the 8th update, and maximum at its 39th update.

What do we learn from these results? Viewing the sequence of RMSE results emphasises the difficulty of identifying which algorithm outperforms the others. Ranking the algorithms according to performance is dependent on what snapshot of the data is currently being trained with. However, the $k = 50$ parameter was more accurate than $k = 20$ in 248 (of the 250) iterations. Similarly, the SVD is more accurate than the $k = 50$ kNN for 245 updates. The balance between $k = 50$ and the bias model is not as one-sided: the bias model is more accurate in about two-thirds of the updates (158), while in the other 91 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.

4.2.2 Time-Averaged Results

The time-averaged results of 5-fold cross validated experiments are shown in Figure 4.8 and 4.9. This visualisation provides a different perspective on the experimental results, and there are a number of observations that can be made. Figure 4.9(a) shows that $k = 50$ tends to outperform $k = 20$ over time. We also tried experiments with $k = 0$; in this case the current item mean is returned. All values of $k \neq 0$ consistently outperformed this 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



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

Figure 4.9: Time-Averaged RMSE for kNN Algorithm and Users With Fewer Than 10 Ratings

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 data they train on.

Figure 4.8(a) 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 differences 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; 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 4.9(a) and 4.8(a): although the bias model outperforms kNN 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 in Figure 4.9(b), 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.

4.2.3 Discussion

The above results provide insight into a number of characteristics of recommender systems. The foremost observation to be made is that recommender systems are not built to be *aware* of their own temporal performance. Each update is treated independently of the rest: algorithms are retrained with all of the available data, and no changes are made based on the temporal performance to date. The experiments

also show the range of results that algorithms produce: a single snapshot of algorithm performance is not sufficient to conclude that one algorithm is indeed more accurate than another. In fact, there is often no consensus between the method that produces the best *global* performance and that which best suits *each user*.

These conclusions led us to formulate the following hypothesis: collaborative filtering algorithms that modify how they predict user ratings (by *switching* algorithm [Bur02] or *updating* parameters) based on their temporal performance will be more accurate than algorithms that do not. In the following sections, we test this hypothesis by designing and evaluating temporal hybrid switching algorithms.

4.3 Adaptive Temporal Collaborative Filtering

Currently, prediction methods are applied iteratively 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 4.3.1), while the second is based on only adapting kNN (Section 4.3.2) or SVD (Section 4.3.3) parameters.

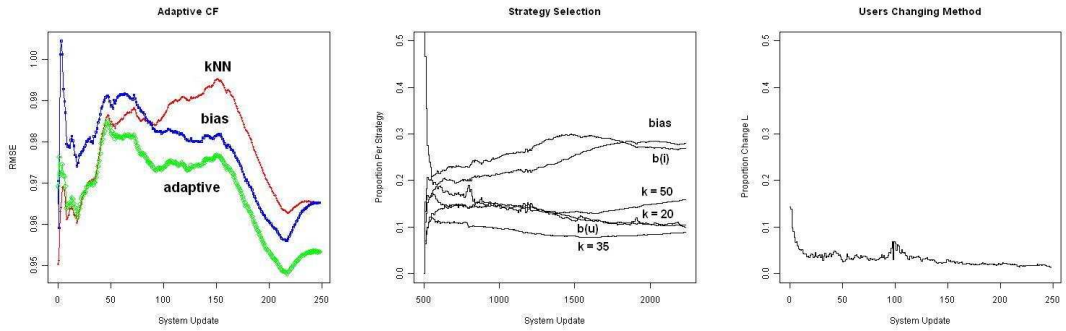
4.3.1 Adaptive CF

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. 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, each user also has a corresponding error value $e_{u,t}$ denoting the time-averaged RMSE achieved on the predictions made to date on *the individual profile*. We therefore aim to minimise the per-user $e_{u,t}$ value by selecting the $L \in P$ that would have maximised the improvement on the current error:

$$\forall u : L_{u,t+1} = \max_{L \in P} (e_{u,t} - RMSE_{u,t}) \tag{4.3}$$

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 for the adaptive method are plotted in Figure 4.10(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



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

Figure 4.10: 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

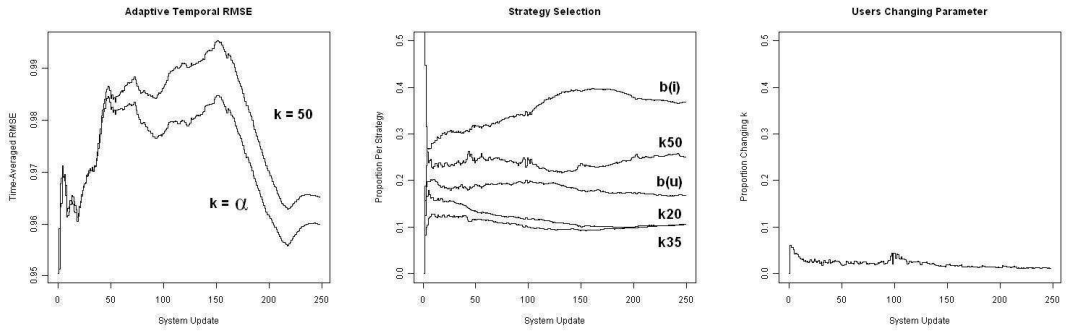
(as the bias model becomes more accurate) departs from this pattern and becomes more accurate than either model alone. In fact, adapting on a per-user basis offers better temporal accuracy than if we simply selected the minimum of the two methods. We also plotted in Figure 4.10(b) the proportion of users who select each method over time. The results show that, while the bias model dominates 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 who, during the update, changed algorithm from the one used during the previous window (Figure 4.10(c)). Overall, very few of the growing population of users changes method from one update to the next; the change is consistently between 1.3% and 14.3% of the growing user community, and on average is $3.1 \pm 1.6\%$.

4.3.2 Adaptive kNN

While the above method offers greater accuracy, it has two shortcomings. First, it is *expensive*: 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 [Mul06], repeating this process with multiple algorithms may be prohitive and difficult to scale. However, if the cost can be incurred, and the goal of doing so is to heighten accuracy, then blending the predictors (rather than switching between them) will offer better results. In fact, one of the first lessons to be learned from the Netflix prize is that greater accuracy can be achieved by blending a wide variety of predictors; the grand prize solutions combined hundreds of predictors in order to surpass the 10% improvement goal [Kor09b, TJB09, PC09].

In the interest of scalability, 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, $P = \{0, 20, 35, 50\}$. 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



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

Figure 4.11: 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

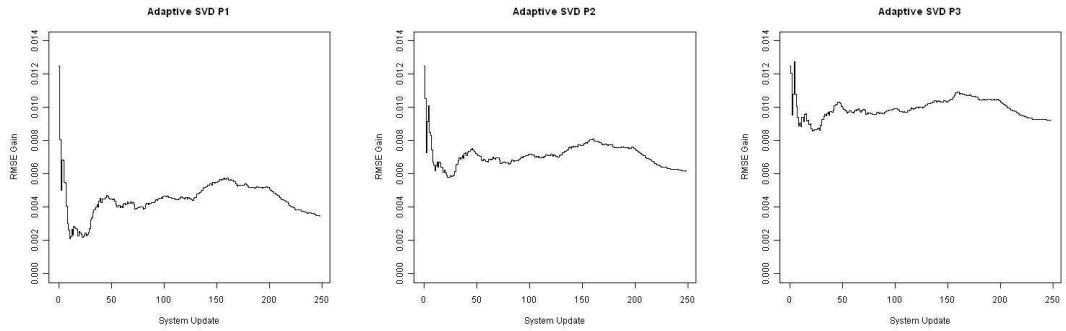
that which would have provided the steepest improvement on the users' $e_{u,t}$ value in the last time step, just as shown in Equation 4.3:

$$\forall u : k_{u,t+1} = \max_{k \in P} (e_{u,t} - RMSE_{u,t}) \tag{4.4}$$

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 4.11. Figure 4.11(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 *worst* when predicting users with very small profiles, as plotted in Figure 4.9(b) (we thus are considering a worst case scenario). It will thus tend to disadvantage new entrants to the system; however, we still can see an improvement in temporal accuracy. Changes to the bootstrapping value affected the first number of updates, 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 4.11(b). From this, we see that users do not converge to a single parameter and moreover, 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



(a) $P_1 \in (56, 64, 72, 80, 88, 96)$ (b) $P_2 \in (72, 80, 88, 96)$ (c) $P_3 \in (88, 96)$

Figure 4.12: Time-Averaged RMSE Gain of Adaptive-SVD With Different Subsets of Parameters

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 value 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 4.11(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 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.

4.3.3 Adaptive SVD

In the previous section, we showed that the neighbourhood size parameter k can be selected from a predefined subset of candidates and updated over time in order to improve the system's time-averaged performance. A natural question to ask is whether this technique is bound to how the kNN algorithm works, and whether the general principle of parameter update based on temporal performance can be applied to other CF algorithms as well.

In order to investigate this question, we turned to SVD-based CF. As introduced in Chapter 2 (Section 2.2.4), SVDs are given a parameter f that denotes how many features will be used to describe the users and movies once they are projected to a lower dimensional space. While nonparametric versions of this algorithm have been explored recently [YZLG09], we focus on the family of SVDs that are initialised with a predefined value of f. In our case, we ran an experiment where $f = 96$. We also output all predictions for any f in $P \in \{56, 64, 72, 80, 88, 96\}$); a number of arbitrary parameters, selected so as to be evenly spaced from each other (they are, in fact, all multiples of eight). Note that we do not recompute the user and movie feature values with a new parameter, but simply output a number of predictions, where each uses a varying subset of the features computed with $f = 96$. We then repeat the same update process that we implemented above; this time, instead, we select (for each user) a future f value based on the one that is currently performing best:

$$\forall u : f_{u,t+1} = \max_{f \in P} (e_{u,t} - RMSE_{u,t}) \tag{4.5}$$

In this case, we take our baseline to be the predictions computed using the full (96) user and movie features, since any hybrid switching approach will select to move away from the full feature matrix toward lower valued parameters. We also varied the range of f values we allowed in the full set P, in order to test the effect of excluding the smaller members of P. We tried $P_1 \in \{56, 64, 72, 80, 88, 96\})$, $P_2 \in \{72, 80, 88, 96\})$, $P_3 \in \{88, 96\})$: the results from these three experiments are plotted in Figure 4.12. In order to highlight how much we gain from the baseline, we plotted the difference between the baseline and each method's time-averaged RMSE. As with the kNN in the previous section, all P_i consistently improve the time-averaged RMSE of the baseline. However, we observe in this case that broadening the range of available f values does not always help. In fact, the group that achieves the highest gain from the baseline is the one with only two f candidates.

4.4 Related Work

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 [VD02], 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.

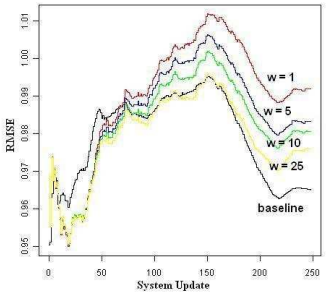


Figure 4.13: Time-Averaged RMSE of kNN With Limited History

Previous work that highlights the importance of time in CF (and in related fields, such as information retrieval [AG06]) tends to focus on the data, rather than the sequential application of an algorithm. For example, Potter [Pot08] (whose bias model we explored above) and Bell & Koren [BK07] 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. Temporality has also been explored from the point of view of changing user tastes [Kor09a, DL05]; 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.

Both our adaptive-CF and this method could be merged; in this work we focus on the algorithm rather than modifying the set of ratings we train with. However, there are a number of questions to be addressed in future research. One of them is the influence of the update interval μ. In this section, we highlight a different example: the balance between time-averaged accuracy and how long the ratings that are being trained with have been in the system.

We repeated our temporal kNN experiments, but limited the algorithm to computing item neighbours using only ratings that were input within the last $w \in \{1, 5, 10, 25, 50\}$ updates. In other words, if $w = 1$, then item similarity is computed using only the ratings input in the previous week; a potential majority of the ratings are excluded. Any ratings input before the allowed 'window' were only used to compute item means. The results, along with the baseline (where all historical data is used), are plotted in Figure 4.13. The figure can be roughly divided into three sections: in the beginning, the baseline performance degrades over time. Then, after a period where the baseline is relatively flat, performance improves for majority of the final updates. During the period where performance degrades, all of the limited/windowed kNNs are more accurate. However, when the baseline performance begins improving, the baseline overtakes the windowed kNNs, although the $w = 50$ is remarkably close (considering the difference in data that each method has available). Just as we found a relation between the *accuracy* and the CF *algorithm*, in order to design our hybrid method, there is also a relationship between *accuracy* and the *data*.

4.5 Summary

This chapter 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.

The contributions we made can be summarised as follows:

- **Methodology and Metrics**. We defined a novel approach with which to examine CF's temporal predictions, using three variations (continuous, sequential, windowed) of accuracy metrics.
- **Evaluation of State-of-the-Art Algorithms**. We ran a variety of experiments that evaluated CF algorithms' temporal accuracy from two perspectives, and highlighted the *variability* of both static and dynamic sets of predictions as training sets are augmented with new ratings.
- **Adaptive Algorithms for Improved Temporal Accuracy**. We implemented and evaluated two adaptive algorithms that improve temporal accuracy over time: a method to switch between CF algorithms and a computationally cheap technique to automatically tune parameters to provide greater temporal accuracy.

The focus of this chapter 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 have 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 [MBW07]. In particular, as the experiments in Section 4.2 highlight, performance does not necessarily improve as the available training data grows.

However, this observation also motivates research that departs from traditional mean-error based evaluations of CF algorithms. 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 temporal CF would therefore also encompass the variation in recommendations that results from the changing data; we begin in the following chapter by shifting our focus to CF's temporal diversity.