Eigentaste 5.0: Constant-Time Adaptability in a Recommender System Using Item Clustering

Tavi Nathanson
Automation Sciences Lab
EECS Department
University of California, Berkeley
dnathanson@berkeley.edu

Ephrat Bitton
Automation Sciences Lab
IEOR Department
University of California, Berkeley
ebitton@berkeley.edu

Ken Goldberg
Automation Sciences Lab
IEOR and EECS Departments
University of California, Berkeley
goldberg@berkeley.edu

ABSTRACT

Recommender systems strive to recommend items that users will appreciate and rate highly, often presenting items in order of highest predicted ratings first. In this working paper we present Eigentaste 5.0, a constant-time recommender system that dynamically adapts the order that items are recommended by integrating user clustering with item clustering and monitoring item portfolio effects. This extends our Eigentaste 2.0 algorithm, which uses principal component analysis to cluster users offline. In preliminary experiments we backtested Eigentaste 5.0 on data collected from Jester, our online joke recommender system. Results suggest that it will perform better than Eigentaste 2.0. The new algorithm also uses item clusters to address the cold-start problem for introducing new items.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering*

General Terms

Algorithms, Experimentation, Human Factors, Performance

Keywords

Collaborative filtering, recommender systems, adaptability

1. INTRODUCTION

The purpose of a recommender system is to eliminate the need for browsing the entire item space by presenting the user with items of interest early on. As such, our objective in developing an effective recommender system is to sustain higher ratings over earlier recommendations.

Eigentaste 2.0 [4] is a constant-time collaborative filtering algorithm that collects real-valued ratings of a gauge set of items for all new users. Offline, it applies principal component analysis (PCA) to the ratings matrix to compute user clusters. Online, determining the appropriate cluster for a new user requires a single dot product with the stored eigenvector which is computed in constant time for fast response. Items are then recommended based on average ratings of other users in that cluster.

Although Eigentaste 2.0 provides a fast cold-start mechanism for new users, each user is permanently assigned to a user cluster and thus a fixed item presentation order that is determined by predicted ratings for that cluster. A disadvantage is that the system does not respond to the user's subsequent ratings. Another disadvantage is the potential for presenting similar items sequentially, referred to as the *portfolio effect* [21, 7]. For example, in Jester, our joke recommender system, a user who rates a Chuck Norris joke highly might then be recommended several more Chuck Norris jokes. In humor, as in many other contexts like movies or books, the marginal utility of very similar items decreases rapidly.

To address these problems we propose Eigentaste 5.0, an adaptive algorithm that, in constant online time, dynamically reorders its recommendations for a user based on the user's most recent ratings. We observe that the problem of finding item clusters based on users' ratings is the dual to the problem of finding user clusters based on item ratings. We develop a hybrid approach that allows us to take advantage of the computational efficiencies provided by solutions to both problems: PCA facilitates fast placement of new users in appropriate user clusters, and maintaining clusters or *portfolios* of similar items allows us to monitor portfolio effects.

Item clustering can also be used to address the cold-start problem for integrating new items into the system. A new item can be quickly matched with similar items, and the ratings of those items can be used to predict ratings for the new item.

2. RELATED WORK

Thornton maintains an excellent online survey of the literature on collaborative filtering: [16].

Quan et al. [12] propose a collaborative filtering algorithm that adds item clusters to a *user-based* approach: when predicting an item's rating for a user, it looks at similar users with respect to ratings within that item's cluster. George and Merugu [3] use Bregman co-clustering of users and items in order to produce a scalable algorithm. Vozalis

RecSys'07, October 19–20, 2007, Minneapolis, Minnesota, USA.

and Margaritis [17] compare other algorithms that combine *user-based* and *item-based* collaborative filtering.

Eigentaste 2.0 deals with rating sparseness by ensuring that all users rate the common set of items, but there are many alternative solutions to this problem. Wilson et al. [19] approach sparseness by using data mining techniques to reveal implicit knowledge about item similarities. Xue et al. [20], on the other hand, fill in missing values by using user clusters as smoothing mechanisms. Wang et al. [18] fuse the ratings of a specific item by many users, the ratings of many items by a certain user, and data from similar users (to that user) rating similar items (to that item) in order to predict the rating of that item by the given user. This approach both deals with sparsity and combines *user-based* and *item-based* collaborative filtering. Herlocker et al. [5] evaluate several different approaches to dealing with sparseness.

Eigentaste 2.0 scales well because in constant online time, it matches new users with user clusters that are generated offline. Linden et al. [9] at Amazon.com use an *item-based* collaborative filtering algorithm that scales independent of the number of users and the number of items. Rashid et al. [14] propose an algorithm, ClustKnn, that combines clustering (*model-based*) with a nearest neighbor approach (*memory-based*) to provide scalability as well as accuracy. Earlier, Pennock et al. [11] evaluated the method of *personality diagnosis*, another technique that combines *model-based* and *memory-based* approaches by aiming to determine a user's personality type. Deshpande and Karypis [2] evaluate *item-based* collaborative filtering algorithms and show that they are up to two orders of magnitude faster than *user-based* algorithms.

As described above, Eigentaste 2.0 is well-suited to the cold-start situation for new users. Park et al. [10] use *filterbots*, bots that algorithmically rate items based on item or user attributes, to handle cold-start situations. Schein et al. [15] discuss methods for dealing with the cold-start problem for new items by using existing item attributes, which is symmetric to the cold-start problem for new users when user attributes are available. Rashid et al. [13] survey six techniques for dealing with this situation. Cosley et al. [1] investigate the rating scales used with recommender interfaces.

Other improvements to Eigentaste 2.0 include Kim and Yum's [6] iterative PCA approach that eliminates the need for a common set of items. Lemire's [8] scale and translation invariant version of the Eigentaste 2.0 algorithm improves NMAE performance by 17%.

Ziegler et al. [21] propose a new metric for quantifying the diversity of items in a recommendation list, which is used to address the portfolio effect. Konstan et al. [7] extend this work to apply recommender systems to information-seeking tasks. While these works improve user satisfaction by increasing topic diversity in recommendation lists, we do so by dynamically reordering recommendations in response to new ratings.

3. EIGENTASTE 5.0

In this section we describe Eigentaste 5.0, an extension of Eigentaste 2.0. Although there are no versions 3.0 or 4.0, we adopted this version numbering to maintain consistency with our recommender system; that is, the next version of Jester (5.0) will incorporate Eigentaste 5.0.

3.1 Notation

U	the set of all users
I	the set of all items
$r_{u,i}$	user u's rating of item i
$\overline{r_i}$	the mean rating of item i
$\overline{r_u}$	user u's mean item rating
$\vec{a}_u$	moving average of u's ratings per item cluster

3.2 Dynamic Recommendations

We seek to enhance the Eigentaste 2.0 algorithm so that it considers changes in a user's preferences when selecting the next item to recommend. In order to maintain the constant online running time of the algorithm, we exploit the dual nature between users and items in a recommendation system. By partitioning the item set into groups of similar items, we can make recommendations based on user preferences for certain classes of items, instead of moving users to different user clusters as their preferences change. While clustering users into groups with similar taste and aggregating their ratings is helpful in predicting how a new user would rate the items, we can tailor recommendations in real-time as we learn more about the user's preferences.

We use the k-means algorithm to cluster the item space offline and across all user ratings. The Pearson correlation function is used as a distance metric between items, where the correlation between item i and item j is defined as follows:

$$P(i,j) = \frac{\sum_{u \in U}(r_{u,i} - \overline{r_i})(r_{u,j} - \overline{r_j})}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r_i})^2}\sqrt{\sum_{u \in U}(r_{u,j} - \overline{r_j})^2}}$$

Pearson correlation provides a real-valued measure on the scale of $[-1, +1]$, where greater values correspond to higher correlation. To use this as a distance metric, we compute 1 minus the Pearson correlation, where larger values reflect greater distances. Note that it is standard procedure to only consider users who have rated both items when computing Pearson correlation, but this is problematic for k-means clustering when dealing with sparse data, as we discuss in section 5.

For each user $u \in U$ we maintain a vector $\vec{a}_u$ of a moving average of u's ratings of the items in each item cluster; that is, $\vec{a}_u[c]$ corresponds to user u's average rating of the last n items in item cluster c, where n is some constant. We initialize a new user's moving average slots with his ratings of items from the common set. For new users who do not have n ratings for each item cluster, we seed the remaining slots of their moving average with the average ratings for the corresponding item cluster across users from the same user cluster.

In Eigentaste 2.0, a user is always recommended items in decreasing order of their predicted ratings, where predictions are determined by the average rating for that item by users in the same user cluster. In essence, a user's ratings of additional items have no influence on which item is recommended next.

We give a constant online time solution as follows: user u is recommended the top-predicted item (not yet rated by u) from the item cluster corresponding to the highest value in $\vec{a}_u$. As u tires of the items from that cluster, the moving average of his ratings for the cluster will begin to decrease and eventually fall below that of another item cluster.

For clarity, we provide a numerical example that walks

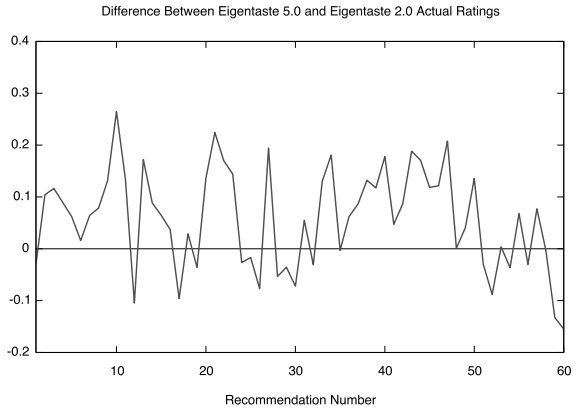

Figure 1: Average difference (across 7,000 users) between actual ratings for Eigentaste 5.0 and 2.0 for the i^{th} recommended item.

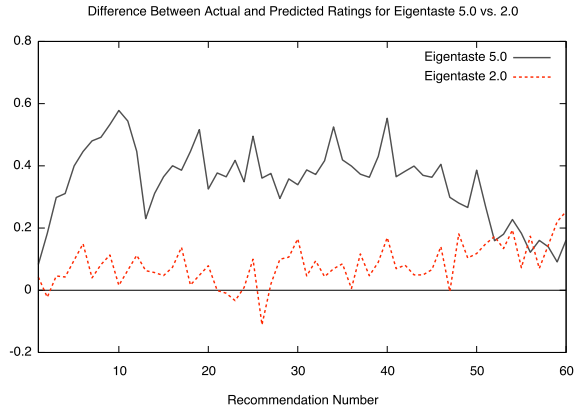

Figure 2: Average differences (across 7,000 users) between actual and predicted ratings for Eigentaste 5.0 and 2.0 for the i^{th} recommended item.

through the process of recommending the next item to some user u. Suppose u maintains the following table, $\vec{a}_u$, of her last 5 ratings of items in each item cluster.

Item Cluster	User u's last 5 ratings	Average
1	4.2, 5.3, 3.8, 2.1, 2.7	3.62
2	7.2, 6.5, 5.9, 0.8, -1.2	**3.84**
3	1.9, -2.4, 3.8, 2.1, 0.5	1.18

At present, user u's moving average of items in cluster 2 is the highest, and so Eigentaste 5.0 presents u with the item from cluster 2 that has the highest predicted rating and has not yet been evaluated by u. Suppose u rates this item with -2.3. Her new moving average of item ratings for cluster 2 becomes 1.94, which is lower than the moving average for item cluster 1. Subsequently, in the next iteration, user u will be recommended the item from cluster 1 that has the highest predicted rating, and the process is repeated.

3.3 Cold Starting New Items

Difficulty in introducing new items to the system stems from the fact that they have so few ratings overall, and even fewer ratings within individual user clusters. Hence, the predictions generated by Eigentaste 2.0 are subject to more variability due to the small sample sizes.

Eigentaste 5.0 uses sparse ratings for a new item i to find the closest item cluster based on the Pearson distance metric described in section 3.2. User u's predicted rating of i is determined by the average rating of all items within i's nearest item cluster across users in u's user cluster. We use confidence intervals to determine the appropriate time to switch from this estimate to the estimate based only on actual ratings of item i.

4. PRELIMINARY RESULTS

It is impossible to truly compare how a user would react to different item recommendation orders, as the first test would greatly bias the second. In the near future, we will release Eigentaste 5.0 for data collection and evaluate the algorithms by randomly assigning new users to either Eigentaste 2.0 or 5.0; in the meantime, we compare the algorithm with its predecessor by backtesting on data collected from Jester.

In this section we discuss recent improvements to Jester and report our initial findings on the performance of Eigentaste 5.0.

4.1 Jester: An Online Joke Recommender

Jester (`http://eigentaste.berkeley.edu`) uses jokes as an item class to demonstrate and evaluate Eigentaste 2.0. Jokes are an example of an item class that can be rated quickly and without prior knowledge. Between March 1999 and May 2003, 73,421 users registered and rated a set of 100 jokes using the Jester system, producing a total of 4.1 million continuous ratings in the range [-10.00, +10.00].

In November 2006 we released Jester 4.0, which introduced a number of improvements, including 50 new jokes, a redesigned user interface, and a visible slider that hovers above the continuous rating meter. To deal with the cold-start problem for new jokes, Jester 4.0 collects ratings for the 5 most sparsely rated jokes at the time of a new user's registration. It continues to interleave sparsely rated jokes during the recommendation phase. Jester 4.0 has collected approximately 110,000 joke ratings of 150 jokes from 3,000 users (as of August 10, 2007). Both data sets, including anonymous ratings, are available upon request.

The features of Eigentaste 5.0 (dynamic recommendations and bootstrapping new items) will be implemented in Jester 5.0.

4.2 Backtesting Results

We simulated Eigentaste 2.0 and the dynamic recommendations of Eigentaste 5.0 with the Jester data collected between 1999 and 2003. The users are randomly partitioned into equal sized sets of "existing" and "new" users, and "existing" users are clustered using principal component analysis. The item space is clustered using a k-value of 15, and we use a 5 item moving average for each item cluster to track user preferences. We iteratively introduce the "new" users into the system and determine the order that the respective algorithms would recommend items. The two sequences of ratings (and corresponding predictions) for each user are recorded in this order. We average the ratings for the i^{th} recommended item across all "new" users for both actual and predicted rating sequences.

Figure 1 shows the difference between the average actual ratings for Eigentaste 5.0 and 2.0. In accordance with our objective, we find that Eigentaste 5.0 provides a distinct advantage over Eigentaste 2.0 for earlier recommendations, particularly within the range of the first 50 items.

The differences between the average actual ratings and the average predicted ratings for each algorithm are shown in Figure 2, which illustrates that the error in predictions is significantly greater for Eigentaste 5.0 than for Eigentaste 2.0. This is because the user clusters used to generate predictions only consider ratings for the common set of items, while with Eigentaste 5.0, we can recommend items better suited to a user's interests by using item clusters to take into account the latest user-specific information.

5. FUTURE WORK

In this working paper we presented Eigentaste 5.0, a new constant-time algorithm to dynamically tailor recommendation sequences to user preferences by integrating user clustering with item clustering and monitoring item portfolio effects. We presented results from preliminary backtesting experiments, which we expect will be a lower bound on actual performance that we will evaluate with Jester 5.0.

We are seeking a more robust method for determining similarity between items when data is very sparse. Substituting average ratings for null data dilutes the actual ratings obtained for the item, and the clustering algorithm is more likely to place the sparse items in the same item cluster. For our experiments, we only considered users that had rated all of the items in the set in order to avoid the sparsity issue.

We will also experiment with generalizations of the adaptive aspect of Eigentaste 5.0, where we recommend items by cycling through the top n item clusters for a user as opposed to just one. Doing so would introduce more diversity among recommendations and may further reduce portfolio effects. We will also experiment with giving users the ability to modify how much item similarity they desire.

6. REFERENCES

[1] D. Cosley, S. Lam, I. Albert, J. Konstan, and J. Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. *Proc. of the Annual SIGCHI Conf.*, pages 585–592, 2003.

[2] M. Deshpande and G. Karypis. Item-based top-N recommendation algorithms. *ACM TOIS*, 22(1):143–177, 2004.

[3] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. *Proc. of the 5th IEEE Int'l Conf. on Data Mining*, pages 625–628, 2005.

[4] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: a constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[5] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM TOIS*, 22(1):5–53, 2004.

[6] D. Kim and B. Yum. Collaborative filtering based on iterative principal component analysis. *Expert Systems with Applications*, 28(4):823–830, 2005.

[7] J. Konstan, S. McNee, C. Ziegler, R. Torres, N. Kapoor, and J. Riedl. Lessons on Applying Automated Recommender Systems to Information-Seeking Tasks. *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 2006.

[8] D. Lemire. Scale and translation invariant collaborative filtering systems. *Information Retrieval*, 8(1):129–150, 2005.

[9] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[10] S. Park, D. Pennock, O. Madani, N. Good, and D. DeCoste. Naïve filterbots for robust cold-start recommendations. *Proc. of the 12th ACM SIGKDD Int'l Conf.*, pages 699–705, 2006.

[11] D. Pennock, E. Horvitz, S. Lawrence, and C. Giles. Collaborative filtering by personality diagnosis: a hybrid memory and model-based approach. *Proc. of the 16th Conf. on Uncertainty in Artificial Intelligence*, pages 473–480, 2000.

[12] T. Quan, I. Fuyuki, and H. Shinichi. Improving accuracy of recommender system by clustering items based on stability of user similarity. *IAWTIC'2006 Proc.*, pages 61–61, 2006.

[13] A. Rashid, I. Albert, D. Cosley, S. Lam, S. McNee, J. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. *Proc. of the 7th Int'l Conf. on Intelligent User Interfaces*, pages 127–134, 2002.

[14] A. Rashid, S. Lam, G. Karypis, and J. Riedl. ClustKNN: a highly scalable hybrid model-& memory-based CF algorithm. *Proc. of WebKDD 2006*, 2006.

[15] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. *Proc. of the 25th Annual Int'l ACM SIGIR Conf.*, pages 253–260, 2002.

[16] J. Thornton. Collaborative filtering research papers.

[17] M. Vozalis and K. Margaritis. On the combination of user-based and item-based collaborative filtering. *International Journal of Computer Mathematics*, 81(9):1077–1096, 2004.

[18] J. Wang, A. de Vries, and M. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. *Proc. of the 29th Annual Int'l ACM SIGIR Conf.*, pages 501–508, 2006.

[19] D. Wilson, B. Smyth, and D. Sullivan. Sparsity reduction in collaborative recommendation: a case-based approach. *Int'l Journal of Pattern Recognition and Artificial Intelligence*, 17(5):863–884, 2003.

[20] G. Xue, C. Lin, Q. Yang, W. Xi, H. Zeng, Y. Yu, and Z. Chen. Scalable collaborative filtering using cluster-based smoothing. *Proc. of the 28th Annual Int'l ACM SIGIR Conf.*, pages 114–121, 2005.

[21] C. Ziegler, S. McNee, J. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. *Proceedings of the 14th international conference on World Wide Web*, pages 22–32, 2005.