SWAMI: A Framework for Collaborative Filtering Algorithm Development and Evaluation

Danyel Fisher, Kris Hildrum, Jason Hong, Mark Newman, Megan Thomas, and Rich Vuduc
Dept. of EECS, CS Division, University of California, Berkeley
{danyelf,hildrum,jasonh,newman,mct,richie}@cs.berkeley.edu

ABSTRACT

We present a Java-based framework, SWAMI (Shared Wisdom through the Amalgamation of Many Interpretations) for building and studying collaborative filtering systems. SWAMI consists of three components: a prediction engine, an evaluation system, and a visualization component. The prediction engine provides a common interface for implementing different prediction algorithms. The evaluation system provides a standardized testing methodology and metrics for analyzing the accuracy and run-time performance of prediction algorithms. The visualization component suggests how graphical representations can inform the development and analysis of prediction algorithms. We demonstrate SWAMI on the EachMovie data set by comparing three prediction algorithms: a traditional Pearson correlation-based method, support vector machines, and a new accurate and scalable correlation-based method based on clustering techniques.

1. INTRODUCTION

The goal of the SWAMI project is to create an open, extensible framework for designing and evaluating algorithms for collaborative filtering. Collaborative filtering systems have been built to help users get recommendations on content like newsgroup postings [9], music albums [10], movies [2], and jokes [7], to name a few. However, comparing approaches can be difficult since different developers often use different data sets and evaluation methodologies; the most notable systematic comparison appears in [3]. To overcome this difficulty, SWAMI (implemented in Java) provides common interfaces for supplying data sets and prediction algorithms to an evaluator. The evaluator benchmarks the prediction algorithms against one another, reporting metrics which help developers gauge prediction accuracy and performance.

We show how we used the framework to study and evaluate three very different prediction algorithms, including one new highly accurate, scalable method, on the full EachMovie [1] data set (see **Section 4**). We also describe our use of both old and new evaluation metrics, as well as the use of visualization in our study. We believe our system will enable investigators to develop new algorithms, as well as tune existing implementations. SWAMI's design encourages developers to add extensions, in the hopes of fostering new and insightful analyses of both algorithms and data sets. A full report and detailed discussion of the software architecture[1] can be found elsewhere [5].

2. THE PREDICTION TASK

The basic algorithmic task is to predict the rating that a user will give to an item, given that user's other ratings and the set of ratings given by all other users of the system. Below, we describe the Pearson correlation method, the support vector method, and a Pearson correlation method that uses clustering to improve scalability and accuracy.[2]

2.1 Simple Pearson Predictor

Herlocker, et al. [8] propose the following general form for computing a prediction $p_{u,m}$ for user u and item m:

$$p_{u,m} = \bar{v}_u + \kappa \sum_{i \neq u} w(u,i)(\bar{v}_u - v_{i,m}) \qquad (1)$$

where $v_{i,m}$ is the explicit vote (i.e., rating) given by user i on item m, $\bar{v}_u$ is user u's mean vote, $w(u,i) \in [-1,1]$ is a weighting that reflects the similarity between users u and i, and κ is a normalizing constant. The sum can be taken over all other users or restricted to some sensible neighborhood to improve run-time performance and/or accuracy.[3] The most widely used weight is the *Pearson-correlation*: $w(u,i) = \sum_k \frac{(v_{u,k}-\bar{v}_u)}{\sigma_u} \frac{(v_{i,k}-\bar{v}_i)}{\sigma_i}$ where the sum is over the items on which both u and i voted, and σ_u is the standard deviation of votes by user u on

[1]The SWAMI software is available for download at http://guir.cs.berkeley.edu/projects/swami.

[2]The latter two expand the set of techniques applied to EachMovie, which include vector similarity-based methods, graphical models [3], and boosting [6].

[3]Based on both our own experiments on EachMovie and suggestions in [8], we chose neighborhoods of size fifty.

the items $\{k\}$. We applied an additional linear penalty to the weight if the number of items rated in common was below some pre-determined threshold[4] [8]. The matrix of weights among all users (or movies) is the user (movie) correlation matrix. This *Simple Pearson Predictor* (SPP) is the most commonly used technique due to its simplicity. The primary disadvantage is poor prediction time ($O(n)$ for n users) since the sum in equation (1) accesses all users in the database even when restricted to a neighborhood.

2.2 Support Vector Predictor

We can also view prediction of discrete votes as a classification task. For any item m, let users who gave m the same vote be members of the same class. We want to build a classifier that maps voting patterns to classes; the support vector (SV) method is one statistical approach [11]. A user is a point whose coordinates are his/her votes,[5] labeled by class. The SV training algorithm finds optimal geometric class separations given a set of example points. The resulting classifier outputs a class label given an new point. One advantage of the *Support Vector Predictor* (SVP) is that the prediction time depends only on the number of training samples used, which is constant with respect to the number of users in the system, assuming infrequent updates of the classifier. However, training time grows as $O(n^2)$ for n training samples. Also, we must supply data for missing votes even if it is inappropriate for a given data set.

2.3 Clustered Pearson Predictor

SPP can be made more scalable by reducing the number of users that are examined for similarity in equation (1), for example, by randomly sampling the full data set. However, we propose an alternative approach: clustering users. Our *Clustered Pearson Predictor* (CPP) is based on the well-known k-means clustering algorithm [4]. Informally, our algorithm creates a fixed number, k, of clusters of users, and then creates composite user profiles for each cluster from its members. Each cluster has a "center," which is a user from the cluster that best correlates with other members. After clustering, a profile is created for each cluster. The profile is a composite user where the vote for an item is the average rating given to that item by members of the cluster. Predictions are computed using SPP where we only consider the k profiles generated above as potential neighbors.[6] Thus, scalability is achieved since the entire user data set has been reduced to a much smaller, constant number of composite users. Furthermore, these composites possess a higher vote density since they are aggregations of their cluster's votes, which helps maintain accuracy. However, the Clustered Pearson method suffers from a large off-line training time requirement.

3. EVALUATION FRAMEWORK

The evaluation framework provides a standard testing methodology for comparing the accuracy and performance of predictors. Below, we describe its three components: baseline predictors, test set selection, and metrics for analysis.

3.1 Baseline predictors

Baseline predictors [10] are fast and common "naive" algorithms against which the accuracy of any predictor can be compared. SWAMI provides two baseline predictors: one which returns a user's average vote and one which returns an item's average vote.

3.2 Test Set Selection

Our method of selecting testing and training sets helps developers determine (1) what minimum amount of data is needed to achieve some desired level of accuracy, and (2) if and when overfitting occurs. The SWAMI interfaces allow each algorithm to have a training phase, separate from prediction, in which to do any preprocessing work. The evaluator builds the training set from the full data set by randomly selecting a given percentage of the users[7] and passing all of their votes to the prediction algorithm. A testing set is constructed from the remaining users. As in Breese, et al. [3], the system performs *Given X* tests: the number of votes known per test user is set to X and we ask for predictions from among the withheld votes. These tests are repeated several times to improve reliability.

3.3 Metrics

We report the *mean absolute error* (MAE), which is usually reported, the variance of the MAE as a measure of predictive reliability, and the mean error which is a rough indicator of bias. However, since MAE does not account for predictive difficulty, we propose the following *weighted mean* $\mu = \sum_u |(v_{u,m} - \bar{v_u}) \times (v_{u,m} - p_{u,m})|$, where the sum is over test users, and the first factor is a weight that is higher when the actual vote is far from the user's mean vote. Finally, we report average training and testing execution times for each algorithm.

4. EVALUATION RESULTS

We compared the three predictors on the EachMovie data set. EachMovie contains about 2.8 million votes cast by over 60,000 users on 1648 movies. The voting scale is 0-5 stars (integer). Our tests were performed using 20%, 40%, 60%, and 80% of the data set for training, with the remainder for testing. In Figure 1, we report results for 40% in the Given 5, Given 20, and Given All-but-one scenarios.[8]

Among the algorithms tested, CPP proved to be both the most accurate and the most scalable algorithm. The results for the SPP resemble the findings of other researchers: the algorithm is accurate (well surpassing the baseline predictors) but slow (over a minute per

[4]For EachMovie, we chose 30 votes.

[5]For missing votes, we use a slightly negative vote.

[6]We report our results on EachMovie with $k = 5000$.

[7]This selection method can be changed easily.

[8]We observed that more than 40% of the data did not significantly improve the accuracy of the algorithms.

		Metrics			
	Given	MAE	Abs. Var.	Wgt. Mean	Pred. Time (s)
By User Avg.	5	1.21	0.84	2.29	0.0
	20	1.17	0.75	2.13	0.0
	All-1	1.02	0.61	1.66	0.0
Simple Pearson	5	1.02	0.53	1.49	84.0
	20	0.88	0.46	1.35	82.9
	All-1	0.89	0.65	1.29	84.2
Support Vector	5	1.46	1.80	1.95	69.3*
	20	1.25	1.45	1.54	71.2*
	All-1	1.10	1.27	1.38	65.0*
Clustered Pearson	5	0.87	0.71	1.41	0.013
	20	0.91	0.67	1.47	0.017
	All-1	0.77	0.57	1.08	0.021

Figure 1: Prediction algorithm accuracy (training set size: 40% of users). *SVP times include training time (true prediction time ≈ 10 ms).

prediction). While SVP performance is disappointing, our expanded report shows a number of instances in which SVP does both significantly better than and significantly worse than the others [5].

5. VISUALIZATION

We tried a number of techniques for visualizing the EachMovie data set to create informative pictures for both developers and end-users. Such techniques included simple curves, views of the sparse user-by-movie vote matrix, multidimensional scaling, geometric views of clusters, and views of the structure of the user and movie correlation matrices. Due to space limitations, we show only one example of an interesting result of a visualization experiment. As suggested in [7], we performed a principal components analysis (PCA) of the movie vote data, which amounts to computing eigenvectors of the movie-movie correlation matrix. Figure 2 shows the largest and smallest (in magnitude) coordinates of the first eigenvector. The titles suggest a surprising but intuitively pleasing interpretation of the eigenvectors. The other largest eigenvectors exhibit a similar arrangement of movies by some sense of "topic."

	High-brow to low-brow
Top 4	Il Postino (The Postman) Mighty Aphrodite Richard III Eat Drink Man Woman
Bottom 4	Beverly Hillbillies Lawnmower Man 2 The Next Karate Kid Operation Dumbo Drop

Figure 2: Largest (top) and smallest (bottom) coordinates of the top eigenvector.

6. CONCLUSION AND FUTURE WORK

Our main contributions are (1) the creation of a portable, open framework for designing and evaluating collaborative filtering systems, (2) a new scalable Pearson correlation algorithm based on clustering, (3) a standardized method for conducting evaluations, and (4) a demonstration of the utility of visualization and data analysis techniques.

There are a number of future directions for this work. (1) We have not yet explored the full tuning parameter space to understand all of the data size, accuracy, and performance trade-offs. (2) A more thorough understanding of which algorithms perform best on what data is needed, possibly leading to data models and/or hybrid predictors. (3) Continued exploration of using PCA to understand the dimensionality of the data set is likely to be fruitful. (4) Validation on new data sets, such as the Jester data set [7] (in progress).

Acknowledgements

We thank Sridhar Rajagopalan, David Gibson, and Alex Berg for much thoughtful input.

7. REFERENCES

[1] Eachmovie collaborative filtering data set, 1997. `www.research.digital.com/SRC/eachmovie`.

[2] Movielens, December 1999. `www.movielens.umn.edu`.

[3] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. `msr-tr-98-12`, Microsoft Research, Seattle, WA, May 1998.

[4] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, Inc., 1973.

[5] D. Fisher, K. Hildrum, J. Hong, M. Newman, M. Thomas, and R. Vuduc. Swami project report. `www.cs.berkeley.edu/~mct/swami/paper.html`.

[6] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. In *Proceedings of the 15th Int'l Conference on Machine Learning*, 1998.

[7] K. Goldberg. Overview of the jester system (invited talk), November 1999. `eigentaste.berkeley.edu`.

[8] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 1999 Conference on Research and Development in Information Retrieval*, Berkeley, CA, August 1999.

[9] P. Resnick, N. Iacovou, M. Sushak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the Computer Supported Collaborative Work Conference*, 1994.

[10] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the Conference on Computer-Human Interaction*, 1995.

[11] V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, Inc., 1998.