Neighborhood based methods for Collaborative Filtering

Ramesh Dommeti

1. Introduction

Predicting user ratings has tremendous applications in e-commerce systems. A common use of such capability is to introduce the user to items that the user is more likely to find interesting. Systems that manifest this capability are commonly known as recommendation systems or collaborative filtering systems and can have significant impact on the sales of an e-commerce business. They can also be used as effective navigational aids in a web site, by way of suggesting web pages and links to users, thus improving the overall user experience of a web site. This has led companies such as Amazon and Netflix to make significant investments in creating and deploying collaborative filtering systems.

Neighborhood-based methods have shown promise in predicting user-ratings [1,2] In the ensuing sections we explore some challenges and approaches for neighborhood computation. In section 2, we describe some of the common manifestations of a neighborhood-based approach. In section 3 we describe our implementation of a neighborhood-based algorithm on the Netflix data set. In section 4, we examine approaches for combining multiple algorithms for predicting user ratings.

2. Neighborhood-based methods

The general problem in recommendations systems can be cast as, given an $\{item_m, user_i\}$, determine the mostly likely $rating_{mi}$. A common framework for generating such predictions is a neighborhood-based approach. Here similar items that a $user_i$ has rated are computed and a $rating_{mi}$ is estimated as some weighted average of the ratings on those items. Some of the methods commonly used for neighborhood-based computation are:

- K-Nearest Neighbors (KNN)
- k-Means
- k-d Trees
- Locality Sensitive Hashing

A KNN approach involves finding the top K nearest neighbors for an item. Ratings from the list of nearest neighbors are combined to predict the unknown rating. This normally involves finding all user-user correlations or item-item correlations and poses the biggest challenge to this approach. Given that for most real-world systems, the number of users is much larger than the number of items, it becomes even more of a challenge to use a straightforward user-user similarity comparison for most practical scenarios. For large datasets such as the Netflix dataset, this involves about 230,000,000,000 user-user similarity computations. The k-Means approach also imposes significant computational burden with increasing *k,* and with the need for multiple iterations before convergence.

Another problem facing KNN and k-means approaches is that the notion of similarity is often arbitrary. Some of the common approaches to similarity are Pearson's correlation, mean squared error based similarity and vector product based similarity. The large number of dimensions present in most real word data reduces the effectiveness of common similarity functions. The curse of dimensionality makes it hard to use approaches such as kdTree for similarity computation [4].

Locality Sensitive Hashing (LSH) [4] offers an alternative approach and has been shown to be effective for large datasets [5]. Given a proper choice of the hash function, it is not susceptible to the curse of dimensionality. In the subsequent sections, we explore the use of Locality Sensitive Hashing for the Netflix dataset.

2.2 Netflix Data

The Netflix dataset [3] is well known, and comprises about 100 million ratings for 17770 movies and 48000 users. A test data set is provided and the evaluation metric is the RMSE on this data set. Netflix has provided an online submission mechanism to evaluate the RMSE of the predictions. In addition a part of the dataset is marked as a probe set and this data set has a composition similar to the final test set.

3. Implementation

3.1 Locality Sensitive Hashing

Locality Sensitive Hashing has been described in [4] and is a fast way to approximate the similarity function. It essentially relies on a hash function H, which exhibits the following properties for any items **p** and **q**.

$$\text{If } \|p - q\| < r, \text{ then } \Pr(Hp == Hq) \text{ is not small}$$

$$\text{If } \|p - q\| > cr, \text{ then } \Pr(Hp == Hq) \text{ is small}$$

Google news personalization is an application of LSH [5] where the hash value is simply the first news item that a user clicked on, in a random permutation of the news article list. The Netflix data has more information, specifically a rating on the scale of 1 to 5. The ratings bring in the inherent biases of the users and movies, which need to be handled in the design of the hash function. For example a rating of 3 by a user whose average rating is 4 is not the same as a rating of 3 by a user who on an average, rates a 2.

To handle the above biases, a 3-dimensional hash function has been designed with each of the dimensions defined as:

Given a random permutation of the list of movies,

$$H[1] = \text{first movie where: } |\text{rating} – \text{user\_average}| > x$$

$$H[2] = \text{first movie where: } |\text{ rating} – \text{user\_average }| < -x$$

$$H[3] = \text{first movie where: } x <= |\text{ rating} – \text{user\_average}| <= -x$$

Here, x is partition value that ensures that each dimension has equal representation and could be a function of the variance for that user. However, for our evaluation this was fixed to a constant for all users. A greedy search suggests a value of 0.8.

The above scheme ensures that collisions in the hash values model similarities among users. The first dimension in the hash function models movies that the user rated higher than his average rating. The second represents movies that were relatively disliked. A multi-dimensional vector increases the probability of finding relevant neighbors. At the same time, it does not compromise cluster quality, since the hash vector can be used to compute similarities, and weights, within a given cluster.

3.2 Prediction

Prediction of a rating pr_{mi} for movie *m* and user *i* is made using a weighted average of all similar users who rated movie *m*, using the following computation:

Compute weights between user *i* and user *j* using the similarity of the hash vectors:

$$w_{i,j} = 1\{H_i[1]==H_j[1]\} + 1\{H_i[2]==H_j[2]\} + 1\{H_i[3]==H_j[3]\}$$

Make a rating prediction using

$$Pr_{mi} = user\_avg_i + (\sum w_{i,j} \,.\, residual_{mj}) / \sum w_{i,j}$$

With $residual_{mj}$ defined as:

$$\text{rating}_{mj} - \text{user\_avg}_j$$

It is possible that some predictions are not feasible using the above approach due to the absence of qualifying neighbors. While using a 3-dimensional hash function alleviates this problem a bit, compared to a single hash value, omissions were still noticed. However, this is a function of the permutation used and using a different permutation solves the problem to a certain degree. This is partly the motivation for the ensemble approach described later.

3.3. Data preparation and cross-validation

Netflix has marked about 1.4 million ratings from the data as a probe set, stating that the final test ratings are similar in nature to the probe set. For our purposes, the probe ratings were filtered out to create a separate training data set. The algorithms were run against the entire training set and the resulting hypothesis was evaluated against the probe ratings. Some of the best hypothesis were run against the test data and submitted to the online Netflix evaluation system for verification. The results on the probe set and test set were found to be similar.

3.4 Ensemble of Predictors

A single run of LSH misses out many values and gives a rating around **1.05**. While this may seem disappointing, it should be noted that LSH is an approximation and is cheap. A different permutation of the original movie vector will give different ratings. Combined with the low cost of a single iteration, this opens up the possibility for ensemble based LSH.

Ensemble learning [6] suggests that when combining results from multiple classifiers, the resultant error is less than the single worst classifier. When classifiers with similar errors are combined together, this often has the effect of improving the resultant error quite significantly. An ensemble approach also comes in handy when no single predictor can generate all of the predictions. We have seen this to be the case for LSH, since a random permutation could create a configuration where some customers do not have neighbors that have rated movies whose ratings need to be predicted.

It is computationally easy to generate multiple LSH permutations and cluster sets. 45 ensembles were computed and the resultant predictions are combined linearly. Significant improvement was noticed with the final RMSE converging to about **0.967** after 45 ensembles.

3.6 Computation based on neighborhood health

The ensemble approach described above used a straightforward linear combination. An alternative approach would be to weigh the contribution of each ensemble as a function of the "health" of the LSH cluster. The 'health' is in-turn computed as the inverse of the RMSE value of the error on a training set comprising all the ratings for that LSH cluster. This resulted in an improvement in the overall prediction from **0.967 to 0.956** after 45 ensembles. However this increases computation time significantly.

Other measures of cluster health could be explored. For example, instead of using the same weight for the entire LSH cluster, a RMSE value could be computed for every movie for every LSH cluster.

3.5 Parallelizing the algorithm

The LSH algorithm lends itself to easy parallelization and the algorithm is well suited for frameworks such as Map-Reduce. Partitioning is straightforward, since after the initial preprocessing stage, individual nodes of the framework need to access only the parameters for a particular movie or customer. Parallelization is easier to achieve here, than for an iterative gradient descent approach such as incremental SVD [7], where the latent parameters are modified by all nodes leading to more collisions and synchronization issues. In the gradient descent approach for SVD, updates need to be propagated immediately as they are needed for subsequent computations.

Multi-core systems are becoming increasing common and an algorithm that can be parallelized across CPU cores can more effectively use commodity multi-core hardware resources. All the above computations were run on a Dual Socket Quad Core system with each core at 1.6GHz. The processing was split across all 8 cores using standard Posix threads. Each movie was processed exclusively by a single core. The work allocation for each core was done using the movie number modulo the total number of cores. Using this setup, a single iteration of LSH takes about 1 min.

4. Integration with other algorithms

4.1 KNN based computation

Item KNN based computation has been explored extensively elsewhere. However, motivated by a desire to test out hybrid computations mixing LSH with KNN and SVD, item KNN neighborhoods were computed. The results matching those reported elsewhere [2]. The following results for KNN were obtained using similarity functions and double centering.

K	**RMSE on probe**
15	0.9453
30	0.9493

Having the results from KNN for items and LSH for users allows for the exploration of confidence-based combination of ratings from the two prediction sets. Similar to the approach based on LSH cluster health described above, for every item-KNN prediction, we have a measure of confidence, indicated by the distance of the nearest neighbors relevant for the prediction. For a low confidence prediction from KNN, the LSH prediction could be preferentially weighed and vice versa. While preliminary trials showed some improvement, enough experiments were not conducted to conclusively determine trends.

4.2 Improving SVD using LSH and KNN based clustering.

The SVD method has been one of the most effective on the Netflix dataset [7]. However, for users or movies with just a few ratings, the resultant user latent parameters could be quite noisy. In such scenarios, neighborhood averages could be used for both the user and movie latent parameters. LSH comes in handy for user neighborhood computation. The following are the results of such interpolation on the probe set.

Method	***RMSE***
SVD	0.9198
SVD (with weighted user parameters, LSH ensemble for users)	0.9169
SVD (with weighted item and user parameters; kNN for items and LSH ensemble for users)	0.9155

5. Conclusion

While the LSH method as described above does not surpass SVD based factorization or KNN based approach, it is still relevant as it offers an effective way of performing nearest neighborhood search across users. Typically the number of users is much larger than the number of items, so this is a relevant problem. The results of LSH based nearest neighbor search results can be fed into other algorithms to achieve improvements as described above.

The ensemble approach has shown that results obtained from different sources can be combined effectively to reduce prediction errors. Ensemble learning is more effective when the underlying algorithms for prediction are different. While ratings from an LSH algorithm with an RMSE of 0.96 would probably not add much value directly to SVD or KNN based predictions with a much lower RMSE, if the LSH can be tuned to obtain a comparable RMSE, then the resulting ensemble RMSE would very likely be lower. Alternatives approaches to representing the LSH hash vector could be explored to tune the RMSE of LSH.

User based neighborhood computation is also relevant for predicting recommendations for new items. A new item has very little history, and hence computed values of latent variables or nearest item neighbors are unlikely to have converged. While Netflix has given equal weight to all ratings for the purposes of the RMSE calculation, this is unlikely to be the optimal weight distribution across various domains. Domains such as online news and collaboration (emails, shared documents) definitely place greater emphasis on recent items. When items are from a completely different domain, user neighborhood may still be useful even when item based neighbors or latent variables become irrelevant. It is possible that users who were similar in their movie preferences may transfer some of this similarity to their preferences of news articles or books.

References

[1] B.M. Sarwar,G. Karypis, J.A. Konstan and John Reidl, 2001, "Item-based collaborative filtering recommendation algorithms", Proc. of the 10th International World Wide Web Conference, Hong Kong
[2] R. Bell, Y. Koren, C. Volinsky, 2007, "The BellKor solution to the Netflix Prize", http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf
[3] The Netlix Prize: http://www.netflixprize.com
[4] Piotr Indyk and Rajeev Motwani, 'Approximate nearest neighbors: towards removing the curse of dimensionality', Proc. of 30th STOC, 1998, pp 604—613
[5] Abhinandan Das, Mayur Datar, Ashutosh Garg, Shyam Rajaram, "Google News Personalization: Scalable Online Collaborative Filtering", Proceedings of WWW 2007, pp. 271-280
[6] Dietterich, T.G., "Ensemble methods in machine learning.", Lecture Notes in Computer Science, 2000.
[7] S Funk. "Netflix Update: Try this at home", sifter.org/~simon/journal/20061211.html, 2006