6160 Advanced Topics in Artificial Intelligence Term Project: Using Locality Sensitive Hashing for Prediction Problem in Recommendation

Ahmet Maruf Aytekin
Department of Computer Engineering
Bahcesehir University
Istanbul, Turkey
aaytekin@gmail.com

***Abstract*—We have been working on the neighborhood-based collaborative filtering (CF) algorithms and using Locality Sensitive Hashing (LSH) to solve the scalability issue of these algorithms. In addition to using LSH for scalability problem of CF methods, LSH can be used for prediction problem of recommendation. In this study we introduce LSH based prediction method for recommendation and compare it with pairwise clustering. In this method we eliminate similarity computation and use candidate sets coming from LSH hash tables as near neighbors and use these sets directly in prediction computation. For clustering based prediction, we used two approaches; using clustering with CF and using cluster objects as near neighbor objects and compute the predictions.**

In this project we implement a prediction engine that will use LSH to query similar items of a target item then predict the rating for the target item. Then we implement pairwise clustering to build up a hierarchical cluster tree and use for predictions.

I. Introduction

A fundamental data-mining problem is to search data for "similar" items. Similarity of sets is very important in recommendation and needs to be calculated efficiently. In neighborhood-based collaborative filtering methods, general approach is to investigate every user and item to find the most similar ones. Because of the need to investigate every pair in search space, similarity search space grows quadratically depending on the number of users or items. Hence, neighborhood-based collaborative filtering algorithms do not scale well with the size of data [2].

As oppose to investigating every item to find the similar pairs in the search space, R, and computing the predictions, we need to find a better way to find the similar users and items without investigating every pair in R. There is a well-known and effective technique for approximate nearest neighbor search in high dimensional spaces, called locality sensitive hashing (LSH). One general approach to LSH is defined by [4] as follows: "hash items several times, in such a way that similar items are more likely to be hashed to the same bucket than dissimilar items are. We then consider any pair that hashed to the same bucket for any of the hashings to be a candidate pair."

Instead of investigating all users/items in a data set to find the most similar items, we can use the candidate pairs in the same bucket as the similar users/items then use them to make predictions. This way we eliminate similarity computation hassle of CF methods. In LSH, hashing items or users in the buckets operation is performed very efficiently in linear time complexity.

Similar to LSH, we can use clustering to group similar user/items in clusters then search the clusters for a target item. In this case we search only the cluster that target belongs to instead of searching all data set, R. As a second method to compare, we use hierarchical agglomerative clustering method. We experiment two versions of using clustering to make the predictions as follows. Once we build up the cluster tree, we run a query for a target user/item and find the cluster neighbors of the target then pass these neighbors to user-based CF algorithm to make the predictions. In the second method of using clustering we directly use cluster neighbors of the target to compute the predictions by treating all cluster members equally weighted.

In the following sections we will briefly describe collaborative filtering, locality sensitive hashing, and clustering methods. Then we will give the details of using LSH and clustering for predictions. Finally we will give the experimental results and discuss our findings and conclude.

II. Collaborative Filtering

Collaborative (social) filtering methods rely on the ratings of the users in the system. The key idea is that the rating of a target user for a new item is likely to be similar to other users' ratings if they behave in similar way to the target user. Likewise, if two items i and j are rated in similar way by the other users then the target user will rate these two items in a similar fashion. Collaborative filtering methods are grouped as neighborhood-based and model-based methods.

In neighborhood-based models the user-item ratings stored in the system are directly used to predict ratings for new items. In contrast to neighborhood-based systems, model-based approaches use these ratings to learn a predictive model [3].

In neighborhood-based methods general approach is to investigate every user and item to find the most similar ones. Then the similar users or items are used to make a prediction. Neighborhood-based methods are used widely because they are simple and easy to understand and implement. They are

	Space	Time	
		Model Build	Query
User-based	$O(\|U\|^2)$	$O(\|U\|^2 p)$	$O(\|I\|k)$
Item-based	$O(\|I\|^2)$	$O(\|I\|^2 q)$	$O(\|I\|k)$

TABLE I: The space and time complexity of user-based and item-based neighborhood methods, as a function of the maximum number of ratings per user $p = max_u|I_u|$, the maximum number of ratings per item $q = max_i|U_i|$, and the maximum number of neighbors used in the rating predictions k.

grouped in two as user-based and item-based predictions. User-based methods rely on the opinion of like-minded users to predict a rating. With user-based method rating of a user u on item i can be estimated as follows:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in N_i(u)} W_{uv} r_{vi}}{\sum\limits_{v \in N_i(u)} \mid W_{uv} \mid},$$

where; W_{uv} is the similarity of user u and v, r_{vi} is the rating value of user v for item i, and $N_i(u)$ is the set of neighbors who have rated for item i.

While user-based methods considers the opinion of similar users to predict a rating, item-based approaches look at ratings given to similar items. Item based algorithm works as follows. To predict a rating for a target item for a user, we use target users ratings on similar items to the target item. We weight these ratings with similarity degree and estimate the rating as follows:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in N_u(i)} W_{ij} r_{ui}}{\sum\limits_{j \in N_u(i)} \mid W_{ij} \mid},$$

where; W_{ij} is the similarity of items i and j, r_{uj} is the rating value of user u for item j, and $N_u(i)$ is the set of items rated by user u.

In order to predict a rating of an item for a user k nearest neighbors are used for computation in CF methods. k varies according to data set and is determined with cross validation. Another important parameter of CF methods is taking into account the significance of a similarity weight, will be denoted by Y. With significance of a similarity weight we mean to reduce the magnitude of a similarity weight when this similarity is computed using only a few ratings. A user similarity weight w_{uv} is penalized by a factor proportional to the number of commonly rated items, if this number is less than a given parameter $Y > 0$: $w'_{uv} = \frac{min\{|I_{uv}|, Y\}}{Y} \times w_{uv}$. Similar approach used for an item similarity weight as well. Optimum value for this parameter is data dependent and needs to be determined using a cross-validation approach [2].

In order to find the the k nearest neighbors of a user or item In CF methods, we need to compare every user or item with every other user or item. Therefore, CF methods exhibit quadratic growth, in terms of computational complexity, as a function of number of users or items. Table I shows user-based and item-based complexity where U and I are denoted for the user and item sets respectively [2].

LSH provides an approach to find the candidate items or users that are likely to be similar, without comparing every item with every other item, and referred as approximate nearest neighbor search [4]. Instead of using k nearest neighbors to make a prediction, we can use the candidate set lists provided by LSH to compute the prediction of a rating.

III. Locality Sensitive Hashing

The idea of LSH is as follows. Locality-Sensitive functions, h, take two data points (users/items) and decide about whether or not they should be a candidate pair. Let the function h that will "hash" data points x and y be denoted as $h(x)$ and $h(y)$. Decision will be rendered based on whether the output of $h(x)$ and $h(y)$ are equal or not. It is convenient to use the notation $h(x) = h(y)$ means that "x and y are a candidate pair". $h(x) \neq h(y)$ is used to mean that "x and y are not a candidate pair." A group of functions of this type called locality-sensitive functions or *family of functions* which is denoted as $\mathcal{H}$ [4, p. 86].

In general, LSH family of functions defined as follows: Let $d1 < d2$ be two distances, for data points x and y. According to some distance measure D, a family of functions, $\mathcal{H}$, is said to be $(d1; d2; p1; p2) - sensitive$ for D, if for every h in $\mathcal{H}$:

- If $d(x, y) \leq d_1$, then the probability that $h(x) = h(y)$ is at least p_1
- If $d(x, y) \geq d_2$, then the probability that $h(x) = h(y)$ is at most p_2

where p1 and p2 are two probability values and $p_1 > p_2$ for the LSH method to be useful.

We represent data points as rating vectors. We will describe locality-sensitive hashing scheme on R^d, collection of vectors in ratings data set R. We use [1]'s proposal to define a locality-sensitive hashing scheme as a distribution on a family of hash functions $\mathcal{H}$ operating on a collection of vectors in R^d. Let $\vec{u}$ be user u's rating vector and $\vec{v}$ be user v's rating vector. The family of hash functions $\mathcal{H}$ defined as follows:

$$h_{\vec{r}}(\vec{u}) = \begin{cases} 1 & \text{if } \vec{r} \cdot \vec{u} \geq 0 \\ 0 & \text{if } \vec{r} \cdot \vec{u} < 0 \end{cases} \tag{1}$$

Then for vectors $\vec{u}$ and $\vec{v}$,

$$Pr[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi},$$

where $Pr[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v})]$ is the probability of declaring users u and v as a candidate pair.

Using this random hyper plane based hash function, we obtain a hash function family $\mathcal{H}$ for cosine similarity. Applying the above scheme gives a locality sensitive hashing scheme where the probability of u and v being declared as a candidate pair calculated as in Eqn. 2.

$$Pr[h(u) = h(v)] = 1 - \frac{\theta}{\pi}, \tag{2}$$

where $\theta = \cos^{-1} \frac{u \cdot v}{\|u\| \|v\|}$. The dot '$\cdot$' in the equation represents dot product of two vectors.

Each hash function from family of functions, $\mathcal{H}$, produces a hash value for input vectors, $\vec{u}$ or $\vec{v}$, which is a single bit. We use AND-Construction to concatenate these bits to create a hash key for the input vectors. After generation of the hash keys, the input vectors that have the same hash key are mapped to the same bucket, in other words they are hashed to the same bucket (Fig. 1). We repeat this procedure several times for number of bands (hash tables) with different randomly generated hash functions as described in banding technique then use OR-Construction to merge the results coming from different hash tables. [4]. In this method, similar (nearby) vectors are more likely to hash to the same bucket in at least one of the hash tables than dissimilar vectors. On the other hand dissimilar (distant) vectors are likely to be hashed to different buckets. Any pair that are hashed to the same bucket for at least one of the hash tables (OR-Construction) are called candidate pairs and they are likely to be similar pairs. It is assumed that most of the dissimilar pairs are not hashed to the same bucket in any of hash tables. Those dissimilar pairs that are hashed to the same bucket in at least one of the hash tables are *false positives* and these will only be a small amount of all pairs. In this method, most of the actually similar pairs are mapped to the same bucket under at least one of the hash tables. The actually similar pairs that do not map to the same bucket in any of hash tables are *false negatives* which will be only a small amount of the truly similar pairs [4].

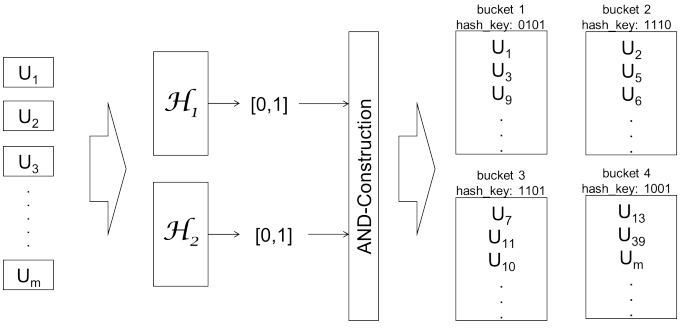


Fig. 1: Mapping user vectors $U_1, U_2, ..., U_m$ to the buckets of a hash table via LSH using AND−Construction.

In following section we will describe how we use the candidate sets for prediction calculation.

IV. LSH For Prediction

As described in Section II, in neighborhood-based methods, general approach is as follows. In order to find the similar objects investigate every object to find the most similar ones, then use these similar objects to make a prediction. With LSH we can find the similar objects without comparing every object with each other in linear time. We run a query to find the similar objects set, called candidate set, for a target object then compute the prediction with the retrieved candidate set. With the word "object", we mean a data point in our data set. It can be a user or item for MovieLens data set we use in this study. With LSH prediction, we can make a prediction of a target item's rating for a target user as follows. For each hash table, we compute a hash key for the target user and retrieve the candidate users with the same hash key from the hash tables.

	Space	Time	
		Model Build	Query
LSH	$O(\|U\|\|L\|)$	$O(\|U\|)$	$O(\|C\|\|L\|)$

TABLE II: The space and time complexity of LSH method, as a function of U, L, and C

We weight the candidate users with the number of occurrences in hash tables. If a candidate user comes from two hash tables, it will have a weight of two as oppose to a user comes from only one table which will have a weight of one. We call these weights as "frequency". Once we retrieved candidate sets from all hash tables, we find the users who rated for the target item, then we use their ratings, weighted with frequency, to compute the prediction. Let $\hat{r}_{ui}$ is the rating of user u on item i that will be predicted, we will use following formula to predict the ratings of user u on the item i;

$$\hat{r}_{ui} = \frac{\sum_{t=1}^{L} \sum_{v \in C_i(t)} r_{vi}}{\sum_{t=1}^{L} \mid C_i(t) \mid}, \tag{3}$$

where; L is the number of hash tables (bands) and $C_i(t)$ is the set of candidate pairs who have rated for item i retrieved from hash table t.

Computational complexity of this method is linear and shown in Table II.

V. Clustering For Prediction

With clustering, similar to LSH, we can find the similar objects without comparing every object with each other. We first create the distance matrix of users with cosine similarity measure, then create the cluster tree of data set based on the distance matrix. As the clustering algorithm we use agglomerative hierarchical clustering algorithm with "grouped average linkage" method. With this method all objects in the cluster contribute to inter-cluster similarity.

The point where to cut the hierarchical tree into clusters decisive on the inter-cluster similarity of the clusters. In other words, if we cut the hierarchical tree close to the leaves, we will have many clusters with high inter-similarity, but, if we cut it close to the root of the hierarchical tree we would have less clusters with low inter-similarity. Once we cut off the hierarchical tree at an arbitrary point, we assign all the objects below each cut point to a single cluster (Fig. 2).

Once we have sub clusters of data set, we use the clusters for predictions as follows. For a target user, u, first we find the cluster that u belongs to, then we use all the users in this cluster, c, (cluster-neighbors) as nearest neighbor list for u to make predictions. For the predictions we tried two different approaches. First we pass cluster-neighbors to user-based CF algorithm to make the predictions and called this method "clustering-kNN". As the second approach, we directly use cluster neighbors of the target to compute the predictions by treating all cluster members equally weighted and call this method "clustering".

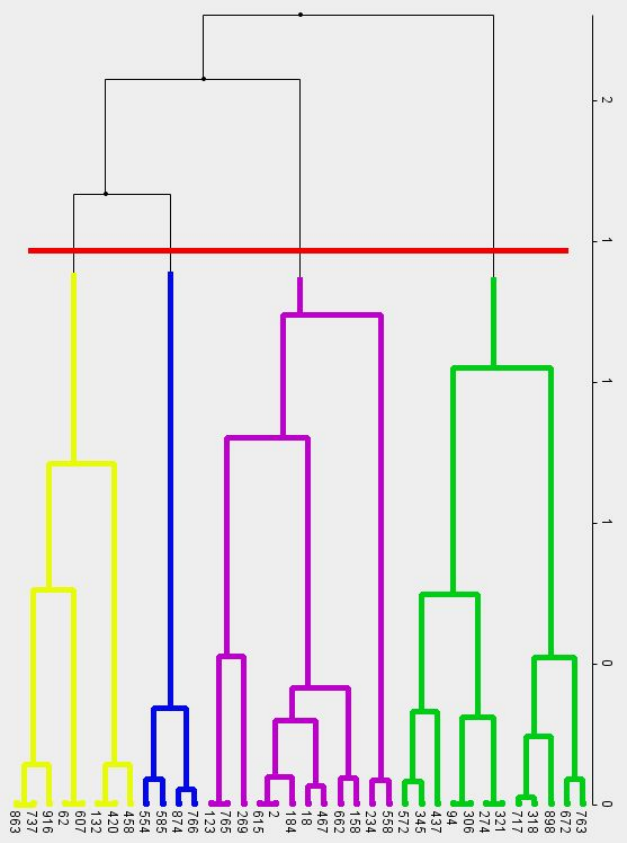


Fig. 2: Sample cut-off point and resulted sub clusters.

	Space	Time	
		Model Build	Query
Cluster	$O(\|U\|)$	$O(\|U\|^2)$	$O(1)$
Cluster-kNN	$O(\|U\|)$	$O(\|U\|^2)$	$O(\|S\|)$

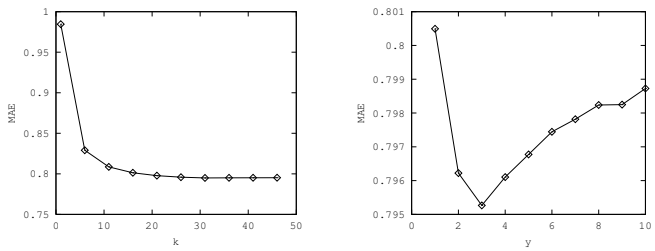
TABLE III: The space and time complexity of Clustering method, as a function of U and S, where S is the clustered data set.

$$\hat{r}_{ui} = \frac{\sum_{v \in S_i(c)} r_{vi}}{\mid S_i(c) \mid}, \tag{4}$$

where; $S_i(c)$ is the set of cluster neighbors who have rated for item i retrieved from cluster c.

VI. Experiments

We evaluate collaborative filtering and LSH method in terms of the accuracy and efficiency. Experiments are carried out as follows. We first divided dataset, R, into two subset of training, R_{train}, and test, R_{test} sets. We used the training data set to build LSH model. Then we evaluated LSH and CF algorithms on test data sets in terms of the prediction accuracy and performance. For these experiments we used holdout cross validation. We holdout %5 of data set as test, R_{test}, %5 of data set as validation, R_{val}, and used the rest as training data set, R_{train}, then run the experiments. We performed this experiment three times and averaged results



(a) MAE and k (b) MAE and Y

Fig. 3: User-based CF accuracy against k, nearest neighbor, and Y, significance, parameters.

over the rounds. We first run cross validation tests to detect optimum parameters k and Y on validation data sets. We then set these parameters and run the experiments on the test set to measure the performance of LSH and CF. We finally run tests to find out how the boundaries of LSH parameters effect accuracy and efficiency.

We also evaluated collaborative filtering and clustering. We used clustering for predictions as follows. We used hierarchical clustering to cluster the users based on the distance matrix. In order to make a prediction for a user, u, on an item, i, we use the cluster neighbors of u to estimate u's prediction on item, i. We tried this with different number of clusters starting from 1 up to 20 levels (depth). We use depth to describe the level of connections from bottom up to the root of the cluster.

A. Data Sets

MovieLens data set is used to train, test, and evaluate the system. Table IV shows the number of users, items and sparsity level of the data data.

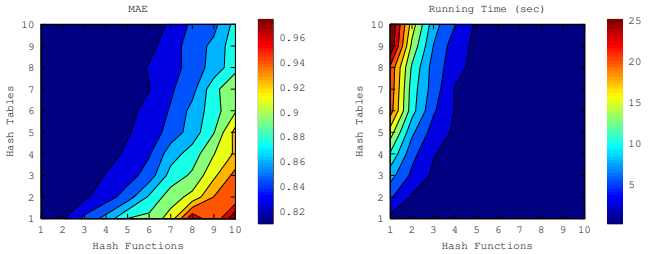
Data Set	Number of Users	Number of Items	Number of Transactions	Sparsity
MovieLens	943	1682	100000	0.0630

TABLE IV: MovieLens data set used in the experiments.

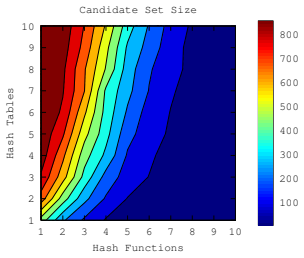
B. Results

1) Parameter Detection Tests: We run parameter detection tests on validation data set first. We run user-based prediction tests for k nearest neighbor parameter. In neighborhood based CF, k number of neighbors is used for prediction calculations. We start k from one and increased by five up to fifty and computed MAE as shown in Fig 3a. We found out that user-based CF performs best when k is around 30 for the data set we use. We set k to 30 and run another experiment to detect the optimum Y, significance, parameter. Fig. 3b shows that user-based CF performs best, in terms of accuracy, when Y is around seven.

2) LSH Tests: With the optimum k and Y from Sec. VI-B1, we run user-based CF prediction tests on the test set and found average MAE and running time 0.79527 and 9.437 seconds respectively. We compare this results with the results of proposed LSH method. For the proposed LSH method, we

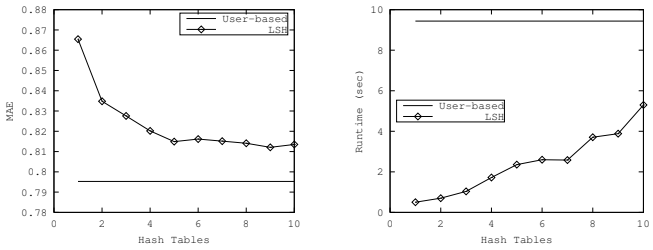


(a) MAE (b) Performance

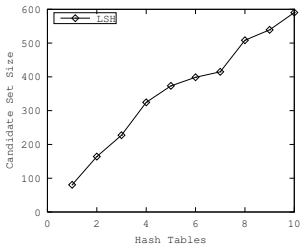


(c) Candidate Set Size

Fig. 4: How accuracy and performance effected by Number of Hash Tables and Functions change in LSH.



(a) MAE (b) Performance



(c) Candidate Set Size

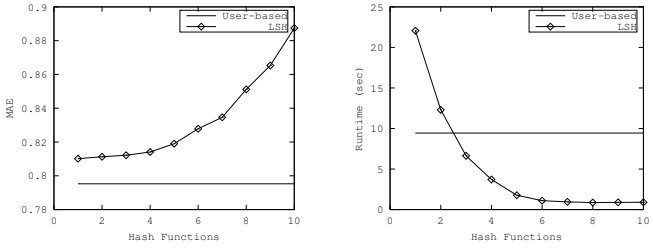
Fig. 5: User-based and LSH in terms of accuracy, performance, and hash tables.

run 10x10 experiments to detect the effect of LSH parameters, number of hash tables and functions. Fig. 4 shows the heat map figures of these experiments.

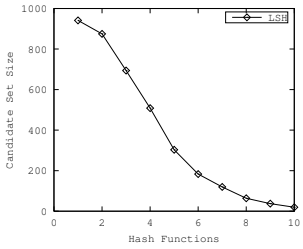
In order to show the effect of LSH parameters better, we draw heat map figures of hash functions and tables (Fig. 4). Note that increasing number of hash tables effects MAE in a better way (Fig. 5a). This is because, increasing the number of tables reduces false negatives. On the other hand, when we increase the number of hash functions, MAE gets worse (Fig. 6a), because, LSH selects only the most similar users and candidate set size, which is used for predictions, gets smaller, hence, false negatives are increased. As oppose to MAE, performance gets better when number of hash functions increase (Fig. 5b), because the candidate set size decreases (Fig. 5c). In addition to decrease in candidate set size we eliminate the similarity computation with LSH prediction (3).

We compare LSH method with user-based collaborative filtering in terms of hash functions and hash tables. Fig. 5 shows how the change of hash tables effect MAE and performance. Increasing number of hash tables increases the candidate set size (Fig. 5c) and this reduces number of false negatives and MAE is improved and got closer to the MAE in user-based method (Fig. 5a). Even though MAE of LSH is still worse then user-based CF method, we gain a lot in terms of the performance (Fig. 5b).

Figures in Fig. 6 show how the change of hash functions effects MAE and performance. As we said before, increasing number of hash functions decreases the candidate set size (Fig. 7c) and this increases false negatives, then MAE gets worse (Fig. 6a). Even though MAE of LSH gets worse as we increase number of hash functions, we gain a lot in terms of the performance as oppose to user-based CF method. (Fig. 6b).



(a) MAE (b) Performance



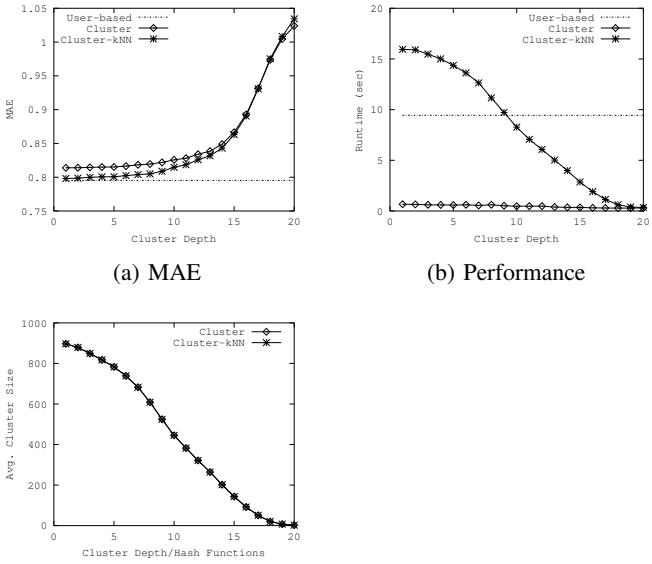
(c) Candidate Set Size

Fig. 6: User-based and LSH in terms of accuracy, performance, and hash functions.

3) Cluster Tests: With the optimum k and Y from Sec. VI-B1 we also compared the user-based method with clustering. We started experiments from level(depth) 0 which is the root of the cluster and increased the level up to 20. When $n = 0$ there is only one cluster ($2^0 = 1$). If we assume the equal distribution of the cluster, at level n, approximately 2^n clusters are generated where n is the depth of the cluster from bottom up. Since 943 users in the data set and we use pairwise clustering, $943/2 = 421$ clusters generated at the last level

of the cluster. Cluster of all users are shown in Fig. 8. A sample cluster of user with id:306 up to 5 level is shown in Fig. 9. We cut off the cluster at every level between 1-20 and measured the accuracy and performance for both "Clustering" and "Custering-kNN" methods then compare the with LSH in terms of the performance and accuracy.

[TODO]



(a) MAE

(b) Performance

(c) Average Cluster Size

Fig. 7: User-based, Clustering, and user-based with clustering results in terms of accuracy, performance, and cluster depth.

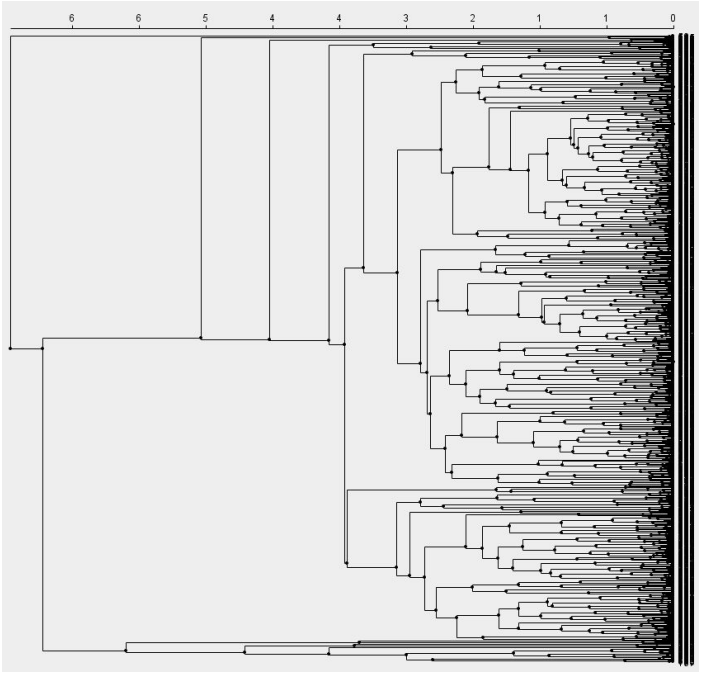


Fig. 8: Hierarchical cluster of all users in the data set. The top line shows the distance which scales from 1 to 10.

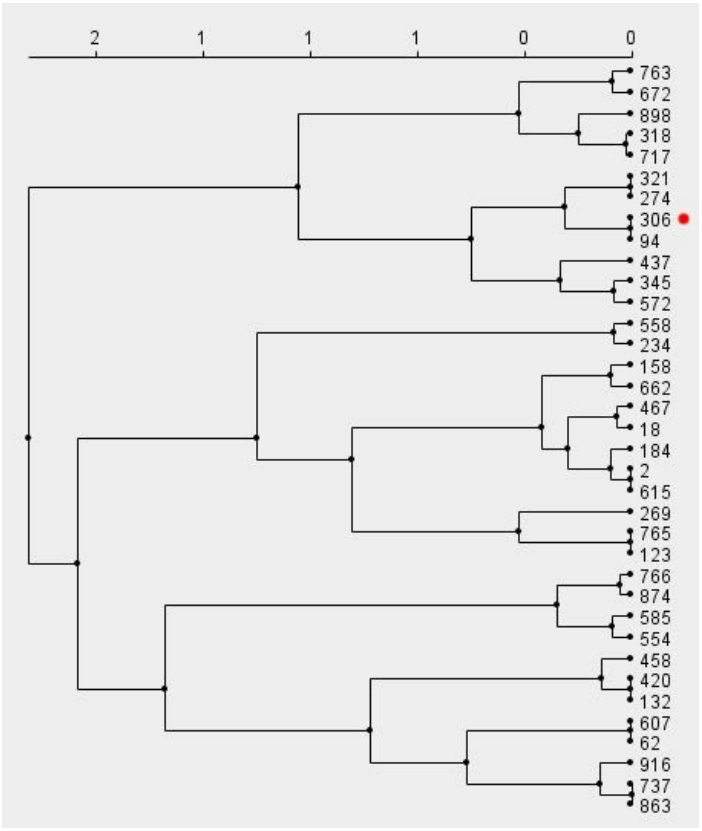


Fig. 9: Sub cluster of user id: 306 at level 5. The top line shows the distance which scales from 1 to 10.

VII. Conclusion

In this term paper we have evaluated application of LSH to the prediction problem of recommendation and compared it against user-based neighborhood method of CF.

We found out that proposed LSH calculation of the predictions tremendously improved the scalability of recommendation as appose to traditional CF methods and maintained the accuracy in acceptable ranges. Proposed LSH method performs worse, which is expected, then user-based CF method but gains a lot in performance. This is because we eliminate the similarity calculation in order to make a prediction. We let LSH algorithm to find the similar users or items (candidate set) and compute the predictions by using this candidate set.

With different configuration parameters of LSH, we can get scalable and better results than user-based method. LSH model needs to be configured to provide optimum outcome that balances MAE and running time according to the priorities of expectations from the system.

In order to complete the evaluation of this method in recommendation domain, we need to run more experiments to find out how it performs against the $top-N$ recommendation problem, which is one of the main problems in recommendation domains. We also need to study this algorithms in terms of other recommendation metrics such as diversity, novelty, serendipity, and aggregate diversity. Another important issue in recommendation domain is the stability of recommendation

system. Stability of proposed method needs to be discussed and evaluated too.

REFERENCES

[1] Moses Charikar. Similarity estimation techniques from rounding algorithms. pages 380–388. ACM, 2002.

[2] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. 2011.

[3] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5(4):287–310, 2002.

[4] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.