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*—The memory-based Collaborative Filtering (CF) methods are widely used in recommendation systems because they are easy-to-implement and highly effective. One of the most important challenges of these methods is the ability to scale with the increasing numbers of users and items. When number of existing users and items grows tremendously, memory-based CF algorithms suffer serious scalability problems. Researchers have been working on number of techniques and algorithms to deal with this issue such as dimensionality reduction techniques. Scalability and prediction performance have reciprocal relationship, hence it is a real challenge to preserve prediction performance while increasing scalability.**

Researchers have used number of methods including Locality Sensitive Hashing (LSH) to overcome the scalability issue with CF algorithms. We have been working on the evaluation of LSH with CF algorithms. In addition to evaluation of LSH in recommendation domain, we would like to find out if we can simply use LSH for prediction problem of recommendation. In this project we will implement LSH based prediction engine and compare the results with user-based CF algorithm.

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. Neighborhood-based collaborative filtering algorithms are widely used techniques in recommendation systems. These algorithms recommend items to the users such that the items are liked by the other users which have similar tests with those users [3]. In these 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 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." We can use the candidate pairs for making the predictions. In LSH, hashing items or users in the buckets operation is performed very efficiently in linear time complexity. We consider all items in the same bucket as near neighbor items for a target item. We can use these items to make a prediction for the target item.

In this project we will implement a prediction engine that will use LSH to query similar items of a target item then predict the rating for the target item. In the following sections we will briefly describe collaborative filtering and locality sensitive hashing methods. Then we will give the details of proposed method, using LSH for predictions and experiments. Finally we will discuss our findings from the experiments 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 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:

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

	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.

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].

We need to eliminate the need for comparing every item with every other item to find the most similar items in linear time. Locality Sensitive Hashing (LSH) provides an approach to find the candidate items or users that are likely to be similar 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

Locality-Sensitive functions take two items and decide about whether or not they should be a candidate pair. The function h will "hash" data points x and y and this is denoted as $h(x)$ and $h(y)$. Decision will be rendered based on whether the results 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 makes 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.

Previous researchers have proposed locality-sensitive hashing schemes for a variety of similarity measures thus far, including jaccard similarity, hamming distance, euclidean distance, cosine similarity, and kernelized similarity functions. We represent user and item data points as rating vectors, hence we use cosine distance/similarity to measure the similarity of two users or items. 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:

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

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

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

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

Eqn. 2 shows the probability of u and v being declared as a candidate pair. 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 concatenate these bits to create a hash key for the input vectors. After generation of hash keys, the input vectors that have the same hash keys 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) as described in banding technique. We slightly modified the bending technique described in [4] and used different hash functions for each bands (hash tables). 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 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, and therefore they will not be checked during the similarity calculation. Those dissimilar pairs that are hashed to the same bucket in at least one of the hash functions 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 functions. The actually similar pairs that do not map to the same bucket in any of hash functions are *false negatives* which will be only a small amount of the truly similar pairs [4].

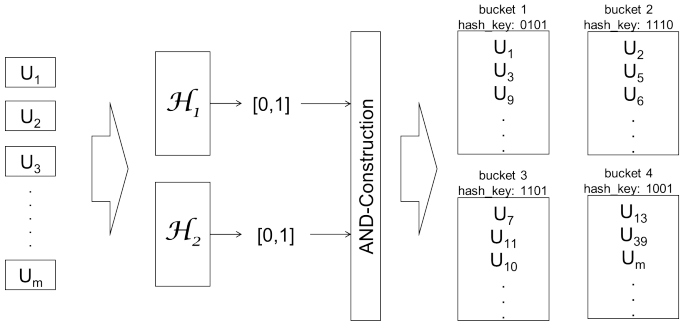


Fig. 1: Mapping user vectors $U_1, U_2, ..., U_m$ to the buckets of a hash table via LSH using AND−Construction.

IV. LSH For Prediction

As described in Section II, 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. With LSH we can find the similar users or items without comparing every user or item with each other. We can retrieve the similar items or users, called candidate pairs, to a target item or user then compute the prediction with the retrieved candidate set. We can make a prediction of an items rating for a user with LSH 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. if a candidate user comes from multiple hash tables, we weight it with the number of hash tables it occurs call it frequency. Other wise all users treated equally weighted. Once we retrieved candidate sets from all hash tables, we take the union of all candidate sets and select the users who rated for the target item. Then we use their ratings, weighted with frequency, to compute the prediction for the target user. Computational complexity of this method shown in Table II.

	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

Let $\hat{r}_{ui}$ is the rating of user u on item i that will be predicted. We use following formula (Eq. 3) 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.

V. 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 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.

A. Data Sets

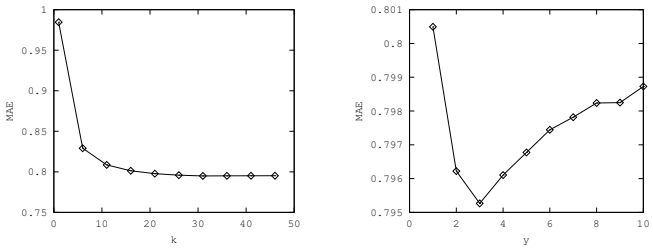
MovieLens data set is used to train, test, and evaluate the system. Table III shows the number of users, items and sparsity level of the data data.

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

TABLE III: MovieLens data set used in the experiments.

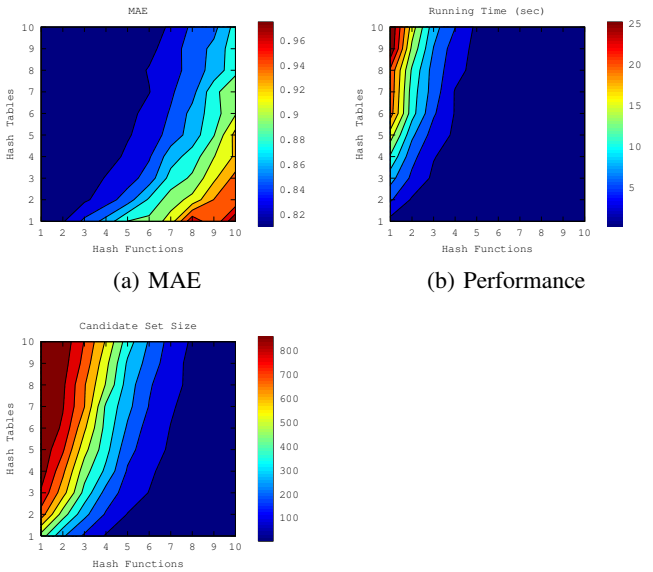
B. Results

We run parameter detection tests on validation data set first. We run CF 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. 2a. We found out that for our data set user-based



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

Fig. 2: User-based CF accuracy against k, nearest neighbor, and Y, significance, parameters.



(a) MAE (b) Performance

(c) Candidate Set Size

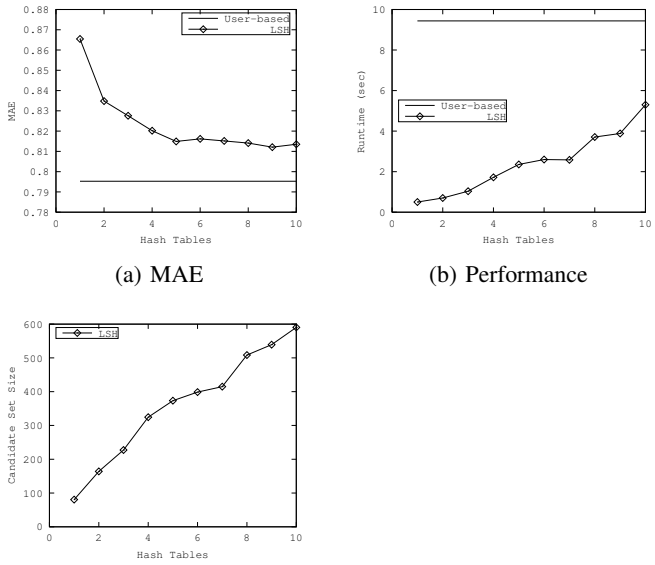
Fig. 3: How accuracy and performance effected by Number of Hash Tables and Functions change in LSH.

CF performs best when k is around 30. We set k to 30 and run another experiment to detect the optimum Y, significance, parameter. Fig. 2b shows that user-based CF performs best, in terms of accuracy, when Y is around seven.

With the optimum k and Y, we run the prediction tests on the test set for user-based CF. We found average MAE as 0.79527 and average running time as 9.437 seconds. We compare this results with the results of proposed LSH method.

We run 10x10 experiments to detect the effect of LSH parameters, number of hash tables and functions. Fig. 3 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. 3). Note that increasing number of hash tables effects MAE in a better way (Fig. 4a). 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. 5a), 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. 4b), because the candidate set size decreases (Fig. 4c). In addition to decrease in candidate set size we eliminate the similarity computation with LSH prediction (3).



(a) MAE (b) Performance

(c) Candidate Set Size

Fig. 4: User-based and LSH in terms of accuracy, performance, and hash tables.

We compare LSH method with user-based collaborative filtering in terms of hash functions and hash tables.

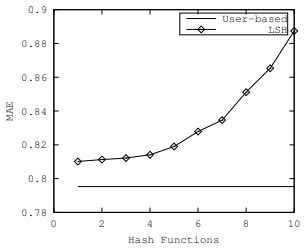
Fig. 4 shows how the change of hash tables effect MAE and performance. As we said before, increasing number of hash tables increases the candidate set size (Fig. 4c) and this reduces false negatives hence MAE is improved and got closer to the MAE in user-based method (Fig. 4a). Even though MAE of LSH is still worse then user-based method, we gain a lot in terms of the performance (Fig. 4b).

Fig. 5 shows 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. 6c) and this increases false negatives, hence, MAE gets worse (Fig. 5a). Even though MAE of LSH gets worse then user-based method, we gain a lot in terms of the performance as we increase the number of hash functions (Fig. 5b).

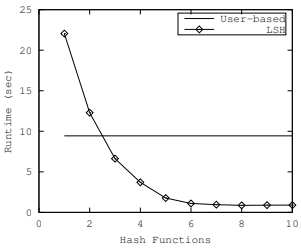
VI. Conclusion

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.

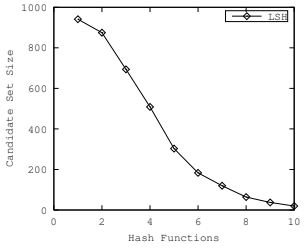
We found out that proposed LSH calculation of the predictions tremendously improved the scalability of recommendation as appose to traditional CF methods and maintained the accuracy in acceptable ranges. Proposed LSH method performs worse, which is expected, then user-based CF method but gains a lot in performance. This is because we eliminate the



(a) MAE

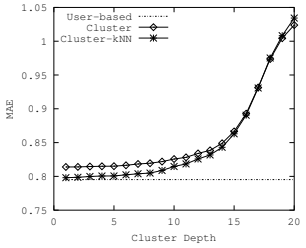


(b) Performance

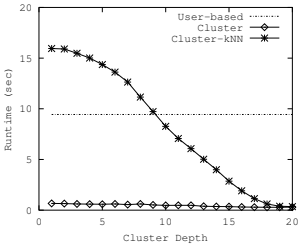


(c) Candidate Set Size

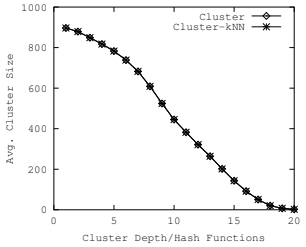
Fig. 5: User-based and LSH in terms of accuracy, performance, and hash functions.



(a) MAE



(b) Performance



(c) Average Cluster Size

Fig. 6: User-based, Clustering, and user-based with clustering results in terms of accuracy, performance, and cluster depth.

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.