Collaborative Filtering via Euclidean Embedding

Mohammad Khoshneshin
Management Sciences Department
University of Iowa
Iowa City, IA 52242 USA
mohammad-khoshneshin@uiowa.edu

W. Nick Street
Management Sciences Department
University of Iowa
Iowa City, IA 52242 USA
nick-street@uiowa.edu

ABSTRACT

Recommendation systems suggest items based on user preferences. Collaborative filtering is a popular approach in which recommending is based on the rating history of the system. One of the most accurate and scalable collaborative filtering algorithms is matrix factorization, which is based on a latent factor model. We propose a novel Euclidean embedding method as an alternative latent factor model to implement collaborative filtering. In this method, users and items are embedded in a unified Euclidean space where the distance between a user and an item is inversely proportional to the rating. This model is comparable to matrix factorization in terms of both scalability and accuracy while providing several advantages. First, the result of Euclidean embedding is more intuitively understandable for humans, allowing useful visualizations. Second, the neighborhood structure of the unified Euclidean space allows very efficient recommendation queries. Finally, the method facilitates online implementation requirements such as mapping new users or items in an existing model. Our experimental results confirm these advantages and show that collaborative filtering via Euclidean embedding is a promising approach for online recommender systems.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

algorithms, theory.

Keywords

Euclidean embedding, collaborative filtering, multidimensional scaling, fast recommendation generation.

1. INTRODUCTION

Living in the information age, people use a large variety of information. Unfortunately, the volume of information is so huge that no one can use all of it, even in a very specialized area. For example, there are tons of movies that you have not seen, but how can you decide which one is best to watch in your limited time? Therefore, customization can play an important role. The idea of a recommender system, an automatic system that can recommend an appropriate item, has emerged in response to this problem. There are two main approaches in recommendation systems: *content-based* and *collaborative filtering* [5]. In a content-based recommendation system, items are recommended based on a user profile and product information. Collaborative filtering uses similarity to recommend items that were liked by similar users.

Collaborative filtering is the subject of our work. Assume that we have a number of users and items, and some users have rated some items (e.g. based on a 1-to-5 scale). The main task is recommending appropriate items to users based on their previous ratings. One natural approach, which is the goal of many collaborative filtering methods, is the prediction of unknown ratings. The user can then be given suggestions based on items with a high expected rating.

As the result of the *Netflix Prize*[1] competition, much work has been done in the collaborative filtering area recently. In October 2006, Netflix released a large movie rating dataset and offered a $1,000,000 prize for developing a system that could beat the accuracy of Cinematch (its current collaborative filtering package) by 10% [2]. In September 2009, a union of three teams used an ensemble of several methods to win the prize.

Although the prediction power gained via approaches used in the Netflix completion is remarkable, the focus on prediction means that some practical aspects of recommender systems are ignored:

1. These approaches are more appropriate for static settings; more precisely, incorporating new data to these models may not be a trivial task.

2. Rating prediction is their main goal; predicting all unknown ratings and then recommending based on the best predicted ratings is very computationally expensive in large datasets.

One of the most accurate and scalable collaborative filtering algorithms is matrix factorization (MF), which is based on a latent factor model [16]. Singular value decomposition (SVD) and related methods using gradient descent are often used in collaborative filtering [4]. In this paper, we propose a Euclidean embedding method that is comparable to MF in

[1]http://www.netflixprize.com

terms of both scalability and accuracy while providing several advantages. First, the result of Euclidean embedding is more intuitively understandable for humans, especially in a low-dimensional space. Second, using nearest neighbor searches and range queries in metric space, recommendation queries can be implemented very efficiently. Finally, it facilitates online implementation requirements such as mapping new users or items in an existing model.

In Section 2, the related literature of collaborative filtering is discussed. In Section 3, collaborative filtering is introduced more formally, and the well-known MF model is described. In Section 4, collaborative filtering via Euclidean embedding and its advantages are discussed. In Section 5, experimental results are presented and finally in Section 6, we discuss conclusions and future work.

2. RELATED WORK

Collaborative algorithms include two primary methods: neighborhood-based and model-based. K-nearest neighbors (KNN) associates to each user or item its set of nearest neighbors, and then predicts a user's rating on an item using the ratings of its nearest neighbors [5]. These algorithms are also known as memory-based algorithms because they utilize the entire database of user preferences when computing recommendations [22]. On the other hand, a model-based algorithm computes a model of the preference data and uses it to produce recommendations. Often, the model building process is time consuming and is only done periodically [22]. A combination of memory-based and model-based methods can be used as well [21]. A very successful model-based method in collaborative filtering, especially in large-scale applications, is matrix factorization [16, 18, 20] and non-negative matrix factorization [24]

Our primary approach is based on Euclidean embedding. One popular example of Euclidean embedding is multidimensional scaling (MDS). MDS is a branch of multivariate statistical analysis and often used to give a comprehensible visual representation. MDS has been applied in a variety of disciplines including psychology, marketing, and machine learning [3, 7]. A narrow definition of multidimensional scaling is the search for a low dimensional space, usually Euclidean, in which points in the space represent the objects, one point representing one object, and such that the distances between the points in the space match, as well as possible, the original dissimilarities [7].

In the data mining and machine learning area, Euclidean embedding has been frequently used as a visualization approach [1], a dimensionality reduction technique [11], and in unsupervised learning [9]. Euclidean embedding is rarely used as a core of a supervised learning approach. For example, Trosset et al. [25] used MDS in the first step of a two-step data mining approach where the second step is training a classifier on the result of the MDS model. The MDS step can be considered as an unsupervised learning approach. In this paper, we propose a Euclidean embedding method that can be seen as a supervised version of MDS and uses the modeling results directly in prediction.

The traditional solution approaches for MDS often include eigenvector analysis methods for the matrix of dissimilarity between objects, resulting in a complexity of $O(N^3)$ where N is the number of objects [19]. Besides the cubic complexity, the computation must be repeated if data is slightly changed [19]. As a result, iterative optimization methods have been developed [6]. Numerical optimization techniques like gradient descent have been widely used in MDS [17].

To the best of our knowledge, Euclidean embedding has not been used as a direct optimization method to implement collaborative filtering. There is a literature about visualization for recommender systems. In [14], first a collaborative filtering algorithm such as classical SVD is run and then some recommendations will be proposed to a user in a visual manner. Note that items or users are not mapped on a space. Users are able to use some metadata such as genre or language to filter their result. In other work such as [10], recommended items are visualized via MDS. However, users are not mapped at the same time. Only items are scaled using classical MDS where dissimilarities between them will be computed via their correlation.

In contrast to the literature on collaborative filtering visualization, our method embeds both users and items in a unified space. This will facilitate visualization of a target user and items he likes where distance is strongly correlated with his personal preferences.

3. COLLABORATIVE FILTERING

In a collaborative filtering (CF) problem, there are N users and M items. Users have provided a number of explicit ratings for items; r_{ui} is the rating of user u for item i. The goal of collaborative filtering approaches is predicting unknown ratings given known ratings. In model-based approaches, a model is trained based on known ratings (training dataset) so that the prediction error is minimized. There are two popular error functions; mean absolute error (MAE) and root mean squared error (RMSE). Since the absolute value function is not differentiable at all points, RMSE is more desirable as an objective function. Therefore, the objective function of a model-based collaborative filtering approach can be defined as follows:

$$\min \sum_u \sum_i w_{ui}(r_{ui} - \hat{r}_{ui})^2 \tag{1}$$

where $\hat{r}_{ui}$ is the prediction of the model for the rating of user u for item i. w_{ui} is 1 if the rating r_{ui} is known and 0 otherwise.

Memory based methods, such as k-nearest neighbors, do not optimize a model based on a training dataset and instead estimate a rating r_{ui} based on the most similar users (items) to user u (item i). Therefore, the training phase is limited to computing similarity measures such as cosine distance or Pearson correlation. The rest of the computation will be done to predict a rating. In memory-based approaches, training is fast but prediction is computationally expensive whereas in the model-based approaches, the reverse is true.

Since our work is similar to collaborative filtering via matrix factorization, we briefly explain it in this section. Latent factor models such as SVD transfer users and items to a low-dimensional space to uncover the latent pattern of ratings. To predict a rating via MF, the following formula can be used [16]:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u q_i' \tag{2}$$

where μ is the total average of all ratings, b_u is the deviation of user u from average and b_i is the deviation of item i from average. b_u and b_i model the fact that some users tend to rate higher and some items are more likable. p_u and q_i are the user-factor vector and item-factor vector respectively in a D-dimensional space. $p_u q_i'$ is the dot product between p_u and q_i where a higher value means user u likes item i more than average.

Since in collaborative filtering problems the data matrix is highly sparse, classical SVD approaches have not been successful [16]. Therefore, a gradient descent approach is suggested to solve this problem by minimizing the following objective function [24, 20]:

$$\min_{p,q,b} \sum_{u,i} w_{ui}[(r_{ui} - \mu - b_u - b_i - p_u q_i')^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)] \tag{3}$$

where the $\lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$ term avoids overfitting by restricting the magnitude of parameters. λ is an algorithmic parameter. Using formula (3), the gradient descent updates for each known rating r_{ui} can be as follows [16]:

$$\begin{array}{lcl} b_u & \leftarrow & b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i & \leftarrow & b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u & \leftarrow & p_u + \gamma(e_{ui} q_i - \lambda p_u) \\ q_i & \leftarrow & q_i + \gamma(e_{ui} p_u - \lambda q_i) \end{array}$$

where e_{ui} is the current error for rating r_{ui} and γ is the step size of the algorithm. Therefore, in each iteration of the gradient descent algorithm, there are T (number of known ratings) steps to go through all ratings in the training dataset.

4. CF VIA EUCLIDEAN EMBEDDING

In this section we present collaborative filtering via Euclidean embedding. Assume that all items and users are embedded in a unified Euclidean space. Let the location show the characteristics of each person. This is a legitimate assumption since one of the early psychological applications of MDS is related to embedding people on a low-dimensional space based on their preferences [7]. A similar assumption can be set for items where the location of each movie reflects its characteristics, such as genre. Note that this is simply the meaning of latent factor model in the framework of Euclidean embedding. Therefore, if an item is close to the user in the unified space, its characteristics are attractive for the user. As a result, there will be a negative correlation between the distance and the likability. Figure 1 represents this idea.

In the Euclidean embedding framework, equation (2) can be re-written as

$$\hat{r}_{ui} = \mu + b_u + b_i - (x_u - y_i)(x_u - y_i)' \tag{4}$$

where x_u and y_i are point vectors of user u and item i in a D-dimensional Euclidean space and $(x_u - y_i)(x_u - y_i)'$ is the squared Euclidean distance. The reason we used squared Euclidean distance instead of Euclidean distance is that the former is computationally cheaper while the accuracy of the method is empirically the same.

The main difference between the matrix factorization model and the Euclidean embedding model lies in the latent factor space characteristics. In the space of Euclidean embedding, the interpretation of user points and item points are the same, since the maximum rating can be reached when they are at the same point. However in matrix factorization the user and item space are not unified. As an example let a user be at $p = (.5, .5)$ in the matrix factorization model and $x = (.5, .5)$ in the Euclidean embedding model. The most similar item to this user in Euclidean embedding is located at $y = (.5, .5)$. However for matrix factorization there is no ideal solution and $q = (.5, .5)$ is worse than, for example, $q = (1, 1)$.

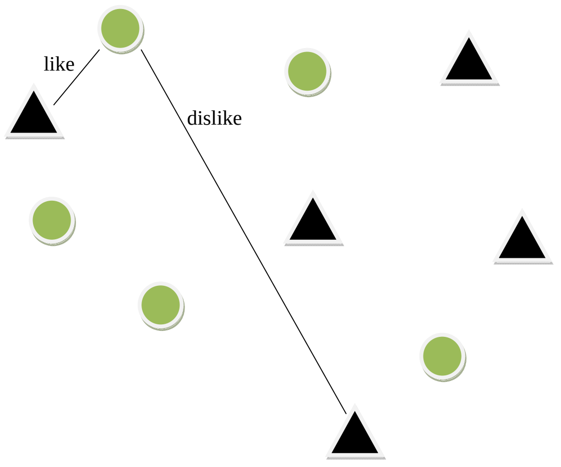

Figure 1: Users (circles) and items (triangles) are embedded in a unified Euclidean space. A user is expected to like an item which is close in the space.

Similar to matrix factorization, the goal of the Euclidean embedding model is embedding users and movies in a low-dimensional Euclidean space based on the known ratings and then predict the unknown ratings based on the trained model. Therefore, Euclidean embedding is a supervised learning approach in which the training phase includes finding the location of each item and user to minimize a loss function. Modifying equation (3) to Euclidean embedding, we have:

$$\min_{x,y,b} \sum_{u,i} w_{ui}[(r_{ui} - \mu - b_u - b_i + (x_u - y_i)(x_u - y_i)')^2 + \lambda(\|x_u - y_i\|^2 + b_u^2 + b_i^2)]. \tag{5}$$

We control the magnitude of the $(x_u - y_i)$ instead of individual points since the distance does not depend on the absolute value of the points but the relative position of x_u to y_i.

This optimization problem is different from typical MDS problems in several respects. First, here we are embedding two different kinds of objects (items and users) in a low-dimensional space, while in standard MDS there is only one object type. Second, the rating matrices are always extremely sparse, which makes the matrix operations less trustable. Finally, in addition to coordinates of points, the b parameters are also being optimized, while in MDS, only the object locations are of interest. Therefore, conventional MDS techniques cannot be applied directly in our Euclidean embedding problem.

Using gradient descent to minimize the Euclidean embedding objective function (equation 5), updates in each step can be defined as

$$\begin{array}{lcl} b_u & \leftarrow & b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i & \leftarrow & b_i + \gamma(e_{ui} - \lambda b_i) \\ x_u & \leftarrow & x_u - \gamma(x_u - y_i)(e_{ui} + \lambda) \\ y_u & \leftarrow & y_u + \gamma(x_u - y_i)(e_{ui} + \lambda) \end{array}$$

where γ is the step size.

4.1 Time complexity

For comparing the time complexity of Euclidean embedding and matrix factorization models, three tasks are important:

1. Training
2. Prediction
3. Recommendation

We address the first two tasks in this section and the third in Section 4.3.

For training, if we pre-compute the $(x - y)$ terms in each step, the other operations will be almost the same as matrix factorization. Therefore, Euclidean embedding needs D (the dimension of space) more operations than matrix factorization. The time complexity of both methods for a step of an iteration is $O(D)$.

The total number of iterations to converge depends on the algorithmic parameters.

For the prediction of a rating, with a similar approach as the training task, we need only D more operations and the time complexity of prediction for both models is $O(D)$. However, we can decompose $(x-y)(x-y)'$ to $xx'+yy'-2xy'$. xx' and yy' can be pre-computed when the system is offline and then the number of operations for both models in the online phase will be exactly the same.

4.2 Visualization

Another advantage of Euclidean embedding is its representation. However, to represent items to users, Euclidean embedding must be implemented only using 2 or 3 dimensions. This can deteriorate the accuracy of recommendations. To avoid this problem, we use the following strategy:

1. Implement CF via Euclidean embedding in a high-dimensional space;
2. Select the top k items for an active user;
3. Embed user, selected items, and some favorite items in a 2-dimensional space via classic MDS, using distances from the high-dimensional space in step 1.

Note that in the third step, MDS can be run very fast because there are only a few objects to embed, all distances are known (from the high-dimensional space solution), and all distances satisfy the triangle inequality (they are real distances in the high-dimensional space).

While classic MDS has been used to visualize items in a CF setting, existing methods use correlation between items as a similarity measure. Using this approach it is not possible to embed a user in the same space because correlation is not defined between an item and a user. Therefore, the picture only shows what items are similar to each other, but it does not indicate how much a user might like each movie. Also sparseness may cause the distance based on correlation to be unknown, and correlation does not satisfy the triangle inequality which makes implementing MDS harder.

Using this low-dimensional unified user-item space, we can represent items to users via a graphical interface such as Figure 2. Highly-rated items can be presented to give an idea about movies. For example a user might be in a mood to watch a romance movie so he can check the movies near to his favorite romance movies.

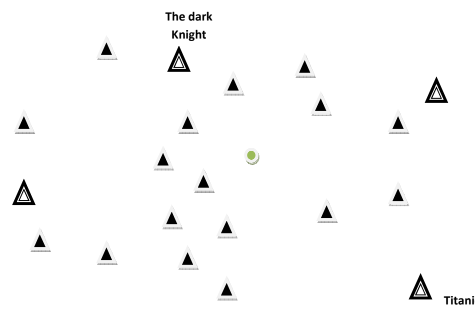

Figure 2: Representing close items (triangles) to a user (circle) besides the movies he has already liked (bold triangles) to assist him in selection.

4.3 Fast recommendation generation

Although both matrix factorization and Euclidean embedding algorithms are fast in predicting a rating, the main task of a recommendation system is finding desirable items for a query user. This problem is rarely addressed in the literature. Das et al. [8] proposed some strategies to select candidate items in the context of a news recommendation system in which news items are selected based on a content-based criteria. Also, they proposed to select items that are chosen by a cluster of users. However, these approaches are not applicable to our problem since we are not using any extra information about items, and we choose to use highly accurate latent factor models' features in the selection task.

A key advantage of Euclidean embedding over matrix factorization is that the nature of the mapped space allows candidate retrieval via neighborhood search. Consider an example with user v as the query user, and $D = 2$. Also let $p_v = [.5\ .5]'$ in the matrix factorization model and $x_v = [.5\ .5]'$ in the Euclidean embedding model. In the Euclidean embedding model, the smaller the distance, the more desirable an item will be. Therefore, we need only search for movies near the point $[.5\ .5]'$. On the other hand, for the matrix factorization model, the larger the value of the term $.5(q_{i1})+.5(q_{i2})$ (where q_{i1} and q_{i2} are movie coordinates), the more desirable the item will be. Therefore, a large area of space may be possible. Figure 3 illustrates the difference between matrix factorization and Euclidean embedding search space for a query user.

Therefore, using Euclidean embedding, the top k closest items to an active user can be found via K-nearest neighbors search. Then, ratings for these top k selected items can be estimated and the ranked list can be presented to the user. There is a highly developed literature on spatial indexing and searching for nearest neighbors, using structures such as R-trees [13] which can be used to find recommendation candidates efficiently.

4.4 Incorporating new users and items

One drawback of model-based approaches is that if a new user or item arrives, it is hard to incorporate them into the model. However, in practice there are always new users and items which must be considered in the recommendation system. This problem has been addressed as *incremental collaborative filtering* in the literature. George and Merugu

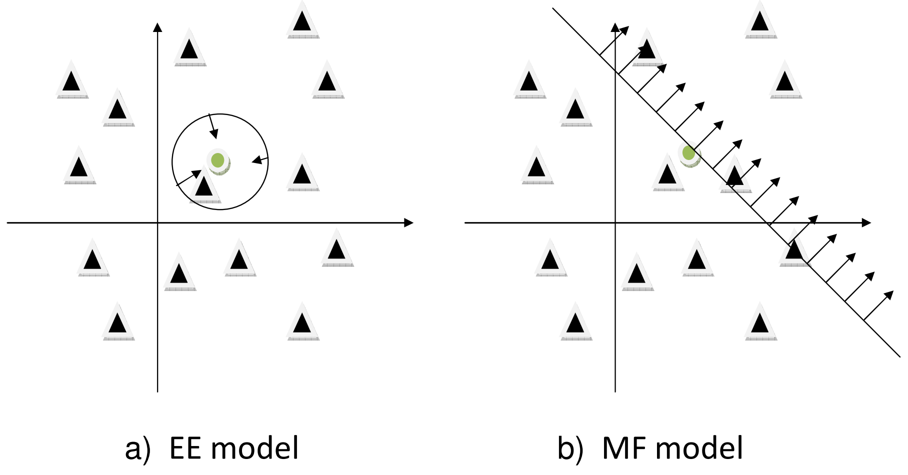

Figure 3: The search space for a query user (the circle). For Euclidean embedding (EE), we need to search for nearest neighbors while in matrix factorization (MF) a large space must be explored.

[12] used co-clustering as a very scalable incremental collaborative filtering approach. They showed the result of incremental co-clustering is comparable to incremental SVD while co-clustering is more scalable. However, they did not address the problem of incorporating new users and items in their model and only updated relevant parameters for old users and items. Sarwar et al. [23] and Brand [4] proposed using singular value decomposition as an online collaborative filtering strategy. In this work, the original version of SVD is optimized using eigenvalue-vector operations. Furthermore, since it is hard to apply classical SVD on sparse data, Sarwar et al. and Brand used imputation to fill in unknown ratings which results in very poor accuracy. Then they used algebraic manipulation to map new users/items in the existing space. However, their approach is not applicable to the MF algorithms which use gradient descent since the user and item factor vectors are not orthogonal.

Generally, for a new user or item, there are $D+1$ unknown values (D for the vector p or q and one for the scalar b). So if there are K ratings for the new object, then we have a $(D+1)\times K$ system of equations (for Euclidean embedding these equations are non-linear). To have a unique estimation, at least $D+1$ ratings are required. However, with fewer than $D+1$ ratings, it is still possible to have an estimation via a ridge regression approach where the accuracy might compensate. The values used for dimension in the literature are always very high (more than 50 factors). For new items, this number of ratings can be gained easily since the number of users usually is more than the number of items. However, gathering this number of ratings might be very tedious for a new user.

In practice many users tend to rate favorite movies first. Also, a recommender system may employ an active learning strategy by asking new users to provide their favorite items. This way, since we know the point vector of the items in the space, and it is very probable that the new user is very close to her favorite items in the Euclidean embedding space, we can estimate the user vector by

$$x_u = \frac{\sum_j y_j}{n} \tag{6}$$

where j indexes the items that the new user u has selected as her favorites and n is the number of selected items.

In the matrix factorization model, knowing the favorite items for a new user may not be as helpful as in Euclidean embedding, since closeness is not related to likability. However, we can argue that the items similar to the favorites are probably interesting as well. As a result, a similar function to equation (6) can be used in the MF model. We address the power of both methods in the next section.

5. EXPERIMENTAL RESULTS

In this section, we present the results of experiments performed to evaluate the effectiveness of the presented methods. All of the experiments are implemented via MATLAB on a 3.2 GHz PC with 4 GB RAM.

In these experiments, two datasets were used. The first dataset is the popular Netflix dataset which consists of 17770 movies, around 480,000 users, and around 100,000,000 ratings, which we only used for comparing accuracy (RMSE). We set step size $\gamma = .005$, dimension $D = 50$, and the regularization parameter $\lambda = .005$ for Euclidean embedding and $\lambda = .01$ for matrix factorization (for MF we used the values from literature). The RMSE on the probe dataset for MF is .9097 (after 23 iterations) and for EE is .9124 (after 27 iterations). The results are similar despite the fact that fine-tuning was not performed on the Euclidean embedding model.

For exploring the ideas presented in this paper, we used the Movielens dataset[2] consisting of 100,000 ratings (1-5 scale) by 943 users on 1682 movies. We employed a 5-fold cross-validation strategy, and all of the values presented in this section (including time, accuracy, etc.) are the average

[2]http://www.grouplens.org/data/

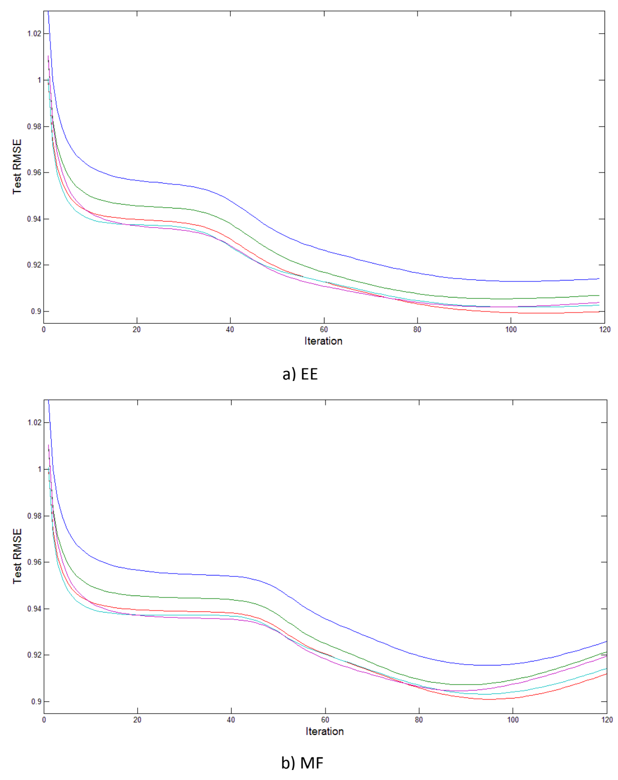

Figure 4: Test RMSE of EE (a) and MF (b) in each iteration of the gradient descent algorithm for five different folds.

values of the five folds. We set step size $\gamma = .005$ and dimension $D = 50$ (unless otherwise is noted), the regularization parameter $\lambda = .03$ for Euclidean embedding and $\lambda = .04$ for matrix factorization. As a stopping criteria for the gradient descent algorithm, we used a tuning set with a size of 5% of the whole training dataset.

Learning curve: Figure 4 shows the values of RMSE in the test dataset as the gradient descent algorithm proceeds. Note that the shapes of these curves are very similar. The only difference is that MF is more prone to overfitting; as it passes the optimal point, the error increases faster.

Dimension, accuracy, time: Table 1 shows the result of implementing EE and MF in 5, 25, and 50 dimensions. Again, both methods give similar results in the sense that increasing the dimension adds little to time requirements, and the accuracy gain from larger dimensionality is small beyond 25-50 dimensions.

Besides RMSE, another set of measurements that are popular in retrieval problems is precision and recall. Since the main goal of recommender systems is suggesting some items of interest, it is important to measure what percent of recommendations are desirable (precision) and what percent of interesting items are retrieved (recall). Here, ratings of 4 and 5 are considered as desirable. Figure 5 shows the precision and recall curve for different dimensions and different methods. The values are the precision and recall for the top-k recommendation scenario for different values of k. As k increases, recall will increase while precision decreases. Again, it seems the overall accuracy of both EE and MF are

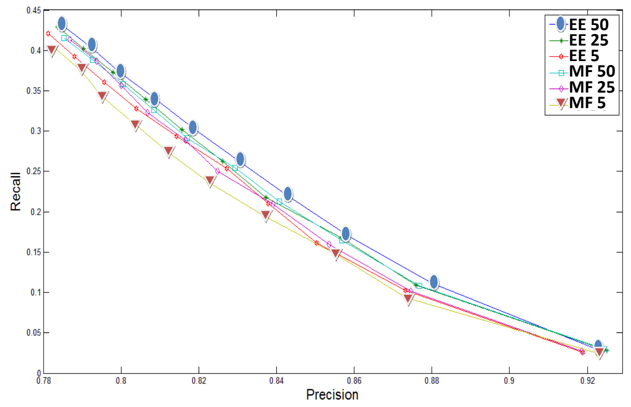

Figure 5: Precision/recall curve for different dimension size

not very sensitive to the size of dimension. In general, EE performs better than MF.

Visualization: Figure 6 shows user visualizations constructed in the following manner. For a typical user, the top 50 movies were selected based on Euclidean embedding with $D = 50$ dimensions. Then, the active user, the top selected items whose ratings were known in the test dataset, and items which were rated as 5 by the active user in the training dataset were embedded in a 2-dimensional space. As a distance we used the Euclidean distance from the primary high-dimensional space. For the sake of comparison, we scaled the same items via MDS using $1-$correlation as the distance. Note that in the comparison method, the active user cannot be embedded. In visualization based on Euclidean embedding, movies that are closer to the active user are more probable to be liked. However, using classic MDS, the picture is harder to interpret since movies can only be compared to each other. For example, disliked movies "Junior," "IQ," and "A Pyromaniac's Love Story" are close to each other and far away from the active user in the first picture, but scattered in the second. The main reason is that, in the first picture items are embedded based on the taste of the active user while in the second picture it is based on the taste of all users.

Generating fast recommendations: As mentioned earlier, finding new recommendations for a user in the EE problem can be treated as a k-nearest neighbors search problem in a Euclidean space. Table 2 summarizes the result of top-10 recommendation to all users. In MF and EE we simply used exhaustive search to find best recommendations. For EE-KNN, first 100 movies for each user were selected as candidates using a brute search K-nearest neighbor search algorithm. Also we found it useful to filter out movies with parameter b_i less than the average of parameter b_i for all items. Then ratings were estimated for all candidates and the top 10 were selected.

In these experiments we used dimension $D = 50$. Note that since we used exhaustive search, the time complexity is a linear function of the number of movies for each user.

As Table 2 shows, the search time can be decreased drastically using KNN-search, while the accuracy is competitive.

New users: As discussed in earlier, if we ask a new user to provide a list of favorite movies, we could quickly map the user in the existing space. To simulate this setting, we

	MF			EE		
Dimension	5	25	50	5	25	50
RMSE	0.9175	0.9107	0.9097	0.9157	0.9104	0.9086
Iteration	105.6	95.0	96.6	93.2	100.4	98.2
Time (sec)	0.4810	0.5221	0.6068	0.4811	0.5226	0.6074

Table 1: RMSE, average number of iterations, and average time per iteration for MF and EE in 5, 25, and 50 dimensions

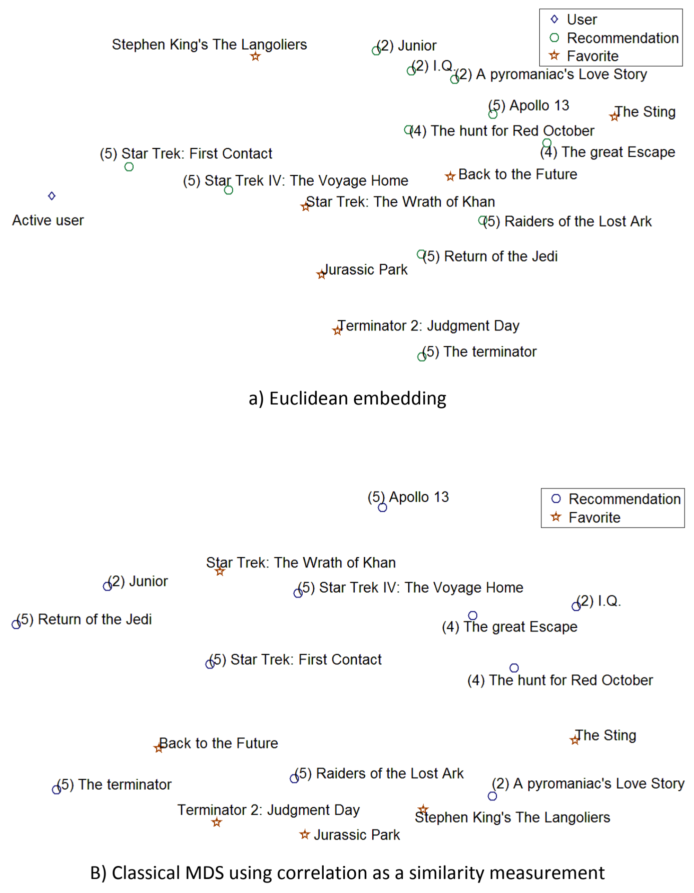

Figure 6: Visualization of movies using Euclidean embedding and classic MDS. The values in parenthesis in front of recommendations are ratings of the user in the test dataset.

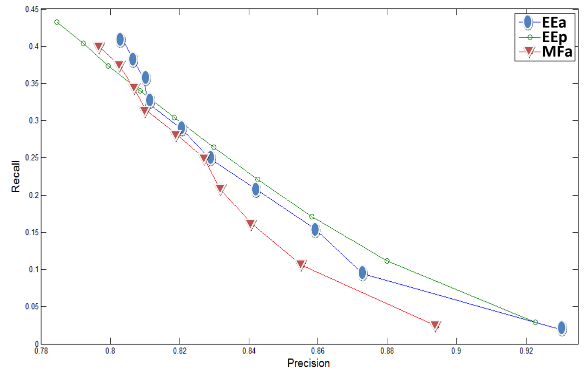

Figure 7: Precision and recall for EEa and MFa. The suffix "a" refers to mapping points via averaging. EEp represents the curve from simple EE.

selected 224 of the 943 users in the Movielens dataset, and 3 movies with ratings of 5 for each user were randomly selected to simulate the list of favorite movies. Then the space was learned via the ratings of the rest of the users. Afterwards, we estimated the point vector of a user by simple averaging for both EE and MF. The result is shown in Figure 7. MFa and EEa (a for mapping points via averaging) implement averaging for new users. EEp represents the precision/recall values for the regular settings when users are not new to the system and we included it for the sake of comparison.

As Figure 7 shows, EEa performs extremely well. Especially its top-5 recommendation precision (on the high-precision end of the plot) is remarkable. When retrieving a small number of recommendations, the precision of Euclidean embedding is much larger than MF, while the recall is about the same.

	Prec.	Recall	Time (Sec)
MF	0.9065	0.0521	18.2455
EE	0.9063	0.0525	19.5189
EE-KNN	0.9012	0.0423	0.5396

Table 2: Precision and recall for simple MF, EE, and EE with candidate generation using KNN-search ($D = 50$ for all runs).

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel collaborative filtering algorithm based on multidimensional scaling, a latent factor model. We showed that CF via Euclidean embedding is comparable to one of the most accurate MF-based algorithm in both accuracy and scalability, while it provides some advantages.

First, using Euclidean embedding, finding recommendations is equivalent to a k-nearest neighbor search in a metric space which can be performed very fast. Second, using the fact that closeness in the transformed space corresponds to higher ratings, we proposed a very simple and fast approach to incorporate new users by asking them to provide their favorite items. Note that in our experiments we chose random movies with a rating of 5. It is likely that performance would improve in actual use, since users would be choosing actual favorites.

A similar approach can be applied to items. Although we cannot ask items to give the most favorite users, in practice when a new item arrives, it is likely that users who like the item most will tend to rate it faster.

A recent research topic in the context of recommendation systems is recommendation to groups [15]. In this context, the goal is finding a group of items that can satisfy a group of users simultaneously. Due to its representation, Euclidean embedding can be used as an effective approach for this problem. In this context we can replace the group of users with their centroid as a single user. Then, if we find a recommendation for the artificial user, the users will should be expected to, on average, like the movie.

7. REFERENCES

[1] W. Basalaj. Incremental multidimensional scaling method for database visualization. In *Proc. Visual Data Exploration and Analysis VI, SPIE*, volume 3643, pages 149–158, 1999.

[2] J. Bennett and S. Lanning. The Netflix Prize. In *KDD 2007, Netflix Competition Workshop*, 2007.

[3] I. Borg and P. Groenen. *Modern multidimensional scaling.* Springer New York, 1997.

[4] M. Brand. Fast online SVD revisions for lightweight recommender systems. In *SIAM International Conference on Data Mining*, pages 37–46, 2003.

[5] L. Candillier, F. Meyer, and M. Boulle. Comparing state-of-the-art collaborative filtering systems. *Lectures Notes in Computer Science*, 4571:548, 2007.

[6] M. Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 127–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.

[7] T. Cox and M. Cox. *Multidimensional Scaling.* CRC Press, 2001.

[8] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, pages 271–280. ACM New York, NY, USA, 2007.

[9] W. DeSarbo, D. Howard, and K. Jedidi. Multiclus: A new method for simultaneously performing multidimensional scaling and cluster analysis. *Psychometrika*, 56(1):121–136, 1991.

[10] D. Fisher, K. Hildrum, J. Hong, M. Newman, M. Thomas, and R. Vuduc. Swami: A framework for collaborative filtering algorithm development and evaluation. In *SIGIR 2000.* Citeseer, 2000.

[11] I. Fodor. A survey of dimension reduction techniques. https://computation.llnl.gov/casc/sapphire/pubs/148494.pdf, 2002.

[12] T. George and S. Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the IEEE Conference on Data Mining*, pages 625–628, 2005.

[13] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM.

[14] F. Igo Jr, M. Brand, K. Wittenburg, D. Wong, and S. Azuma. Multidimensional visualization for collaborative filtering recommender systems. *Technical Report TR20003-39, Mitsubishi Electric Research Laboratories*, 2002.

[15] A. Jameson and B. Smyth. Recommendation to groups. *Lecture Notes in Computer Science*, 4321:596–627, 2007.

[16] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *KDD '08: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge discovery and Data Mining*, pages 426–434, New York, NY, USA, 2008. ACM.

[17] J. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, 1964.

[18] M. Kurucz, A. Benczúr, and K. Csalogány. Methods for large scale SVD with missing values. In *KDD 2007: Netflix Competition Workshop.*

[19] A. Morrison, G. Ross, and M. Chalmers. Fast multidimensional scaling through sampling, springs and interpolation. *Information Visualization*, 2(1):68–77, 2003.

[20] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *KDD 2007: Netflix Competition Workshop.*

[21] D. Pennock, E. Horvitz, S. Lawrence, and C. Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 473–480. Stanford, California, 2000.

[22] A. Rashid, S. Lam, G. Karypis, and J. Riedl. ClustKNN: A highly scalable hybrid model & memory-based CF algorithm. In *Procedings of WebKDD 2006 - Knowledge Discovery on the Web.*

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.

[24] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. On the Gravity recommendation system. In *KDD 2007: Netflix Competition Workshop.*

[25] M. W. Trosset, C. E. Priebe, Y. Park, and M. I. Miller. Semisupervised learning from dissimilarity data. *Computational Statistics and Data Analysis*, 52(10):4643 – 4657, 2008.