CS532 Lab
Recommender Systems

Lokesh Jindal Mehreen Ali Urmish Thakker

Fall 2015

Contents

1 Motivation

User modeling, adaptation and personalization methods have reached the mainstream. The growth of social networking websites and the computational power of mobile devices are generating huge amounts of user data and increasing the need of users to "personalize" their e-mail, phone etc.

The potential value of personalization is now clear both as a commodity for the benefit of end-users and as an enabler of better (or new) services. It serves as a tactical opportunity to expand and improve businesses.

The main characteristic of recommender systems is that they attract the interest of industry and businesses while posing extremely interesting challenges.

In spite of noteworthy progress in the research community and the efforts of the industry to provide the end users with the benefits of new techniques, there are still many important gaps that make personalization and adaptation challenging for users. Research activities still focus on narrow problems, such as incremental accuracy improvements of current techniques or on a few applicative problems. This confines the range of other applications where personalization technologies might also be useful.

Thus, we have come to a good point where we can take a step back to obtain a perspective in the research done in recommender systems. In this study we will be using MovieLens data set [1].

2 Introduction

There are two popular techniques used for developing a recommender system [3]. These are discussed in the following section.

2.1 Content-based

Content-based System is used to recommend an item to a user based upon a description of the item and a profile of users interests and other metadata. This metadata could be information like age, sex, demography etc. The content-based approach to recommendation has its roots in the information retrieval (IR) community, and employs many similar techniques. The system *learns* to recommend items that are similar to the ones that the user liked in the past.

2.2 Collaborative Filtering

Collaborative filtering is the process of finding information using collaboration among multiple agents. Typically, a set of "nearest neighbors" are found whose past ratings have a strong correlation with candidate user. Scores for unseen items are predicted based on a combination of the scores estimated from the nearest neighbors.

Main approaches for collaborative filtering:

- User based collaborative filtering is a straightforward algorithmic interpretation of the core premise of collaborative filtering: find other users whose past rating behavior is similar to that of the current user and use their ratings on other items to predict what the current user will like.
- Item based collaborative filtering generates predictions by using the users own ratings for other items combined with those items similar to the target item.

	Matrix	Titanic	DieHard	ForrestGump	Wall-E
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2		3	5	4
Diane	4	3	5	3	

Table 1: Example ratings

3 User Based Recommendation

3.1 Idea of Similarity

Let's say the users of a website rate movies on a scale of 1 to 5, where a rating of 5 implies the user really liked the movie and a rating of 1 implies the opposite. This data can be stored in the form of a *utility matrix* where each row of the matrix corresponds to a user and each column corresponds to a movie. Consider an example rating set shown in Table 1 that can be represented in the form of *utility matrix* U as shown in equation 1. This 4x5 matrix stores the ratings given by 4 users to a set of 5 movies. As expected, not every user would have rated each of the 5 movies. The user-movie combination for which a rating does not exist is indicated by a 0.

$$U = \begin{bmatrix} 5 & 1 & 0 & 2 & 0 \\ 1 & 5 & 2 & 5 & 5 \\ 2 & 0 & 3 & 5 & 4 \\ 4 & 3 & 5 & 3 & 0 \end{bmatrix} \tag{1}$$

Exercise 1

Look at the ratings of Eric in Table 1. Also look at the ratings given by other users in the table. Can you guess which user(s) is(are) similar to Eric?

3.1.1 Similarity Weight Computation

The example data set in Table 1 is very small in size for which finding users that are similar to the user of interest might be trivial. For real data sets, we need to define a measure of similarity that can be used to find the items/users that are similar to each other. The computation of of the similarity weights is one of the most critical aspects of building a *neighborhood-based* recommendation systems, since it can have a significant impact on the performance and accuracy of the system.

Cosine Similarity: We apply the traditional notion of Cosine Vector (CV) similarity to find the similarity between two users u and v. If $\mathbf{x_u}$ is vector of ratings $\mathrm{r_{ui}}$ of a user u, then the cosine similarity of users u and v can be calculated using equation 2.

$$CV(u,v) = cos(\mathbf{x}_u, \mathbf{x}_v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2 \sum_{j \in I_v} r_{vj}^2}} \tag{2}$$

where $\mathrm{I_u}$ and $\mathrm{I_v}$ represent the set of ratings of users u and v respectively, and $\mathrm{I_{uv}}$ is the set of ratings for the movies that have been rated by both users.

	Matrix	**Titanic**	**DieHard**	**ForrestGump**	**Wall-E**
Eric	-1.5	0	-0.5	1.5	0.5
Diane	0.25	-0.75	1.25	-0.75	0

Table 2: Mean Centered Ratings

Exercise 2

In the example data set of Table 1, find the users that are most similar to user. You can use the function *CosSimVecMatrix* (provided with the lab) to generate the similarity of Eric with other users.

Hopefully that was easy and you have the answer - users that seem most similar to Eric are Lucy and Diane. That was easy. Let's look at the ratings of Diane who seems to be similar to Eric. If we calculate the means of ratings of Eric and Diane, 3.5 and 3.75, and subtract the means from their respective ratings, the new ratings look like as shown in Table 2.

Positive values in Table 2 represent a preference for the movie by the user since it's rating is more than the average, whereas a negative value implies a disliking for the movie. On analyzing these *mean-centered ratings*, Eric and Diane don't seem to be so similar anymore. In fact, they seem to have quite the opposite taste. What's going on?

Pearson Correlation Similarity: A major flaw in using *Cosine Similarity* to find the similarity between users is the fact that Cosine Similarity does not take into account the differences in the mean and variance of the ratings made by the users. Pearson Correlation (PC) similarity is a popular measure that can be used to compare the ratings of users since it removes the effects of mean and variance in ratings while calculating the similarity between users. The PC similarity between users u and v can be calculated using equation 3.

$$PC(u,v) = PC(\mathbf{x}_u, \mathbf{x}_v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r_u})(r_{vi} - \bar{r_v})}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r_u})^2 \sum_{i \in I_{uv}} (r_{vj} - \bar{r_v})^2}} \tag{3}$$

where $\bar{r_u}$ represents the mean of the ratings given by user u. Note that PC accounts for variance in ratings only for the ratings of the movies $\mathrm{I_{uv}}$ that have been rated by both users u and v. The sign of similarity weight calculated using PC indicates whether the correlation between the two users is direct or inverse, and the magnitude of similarity weight (ranging from 0 to 1) represents the strength of correlation. These will be used to predict unknown ratings for a user using *neighborhood-based* methods later.

Exercise 3

Calculate the Pearson Correlation (PC) similarity of Eric with other users in Table 1. You can use the function *PCSimVecMatrix* provided with the lab. Which user(s) would you say is(are) similar to Eric? You can do a similar analysis by comparing the Cosine similarities and Pearson Correlation similarities of other users.

3.2 Calculating Predicted Ratings

The Recommender Systems exist so that they can make recommendations to the user. As highlighted in Section 2, collaborative filtering relies on finding users similar to a user of interest based on the ratings given by them. We now have two important measures of finding similarity between users, namely Cosine similarity and Pearson Correlation similarity. It follows that these similarity measures can now be used to find the neighbors nearest to a user, and subsequently their ratings can be *combined* to give us predicted ratings for a number of movies that have actually not been rated by the user of interest. It is natural that the movies whose predicted ratings are higher are the potential candidates that the recommender system can recommend to the user.

Thus, the rating $\mathrm{r_{ui}}$ of a user u for a new item i, can be predicted using the ratings given to i by users most similar to u. Let $\mathrm{w_{uv}}$ denote the similarity weight between users u and v, and N(u) be the set of k neighbors of u, then the predicted rating $\mathrm{r_{ui}}$ can be calculated as:

$$r_{ui} = \frac{\sum_{v \in N_i(u)} r_{vi}}{|N_i(u)|} \tag{4}$$

Note that the formula considers only the neighbors v of user u who have rated the movie i, $\mathrm{N_i(u)}$ representing the set of such neighbors.

Exercise 4

Carefully look at the formula in equation 4. Can you find a flaw in the formula? What does it use the similarity weights of user u with different users v for? Explain qualitatively, how the formula is not making full use of the similarity weights.

If you have an intuitive answer to Exercise 4, equation 5, which is more sophisticated in its calculation of the predicted rating, should make sense. It accounts for the fact that there can be some neighbors who are more "similar" than others i.e. it distinguishes between neighbors and does not treat them alike while generating predicted ratings.

$$r_{ui} = \frac{\sum_{v \in N_i(u)} w_{uv} r_{vi}}{\sum_{v \in N_i(u)} |w_{uv}|} \tag{5}$$

We now take a small detour to look at the concept of *Rating Normalization* and will then come back to learn how to really generate a recommendation for a user.

3.3 Rating Normalization

Various users rate the movies differently, each having his/her own personal scale. For example, while a rating of 4 by one user might imply a strong preference for a movie, the same rating by another user whose average rating across movies is 4, might not give us any interesting information. To tackle this, there are two popular normalization techniques: *Mean-Centering* and *Z-score Normalization*.

3.3.1 Mean-Centering

The idea of mean-centering is quite simple. It determines whether a rating is positive or negative by comparing it to the mean of the ratings for that user. Once we have the ratings for nearest neighbors, they can be *normalized* by subtracting their corresponding means and then used to generate the predicted rating for a user using the following relation.

$$r_{ui} = \bar{r_u} + \frac{\sum_{v \in N_i(u)} w_{uv}(r_{vi} - \bar{r_v})}{\sum_{v \in N_i(u)} |w_{uv}|} \tag{6}$$

where $\bar{r_u}$ is the mean of ratings of user u.

	Matrix	**Titanic**	**DieHard**	**ForrestGump**	**Wall-E**
John	5	1		2	2
Lucy	1	5	2	5	5
Eric	2		3	5	4
Diane	4	3	5	3	
Kim	1	4		5	

Table 3: Example ratings

3.3.2 Z-score Normalization

While mean-centering removes the user bias by removing their respective means from their ratings, it still does not consider the spread of ratings given by a user. For example, consider two users both of whom have an average rating of 2.8. Further, user X shows a lot of variation in his ratings from 1 to 5, whereas user Y has most ratings very close to 2.8. In this case, a rating of 4.5 by Y would imply an exceptional liking for the movie for user Y. The same cannot be said for user X. Z-score normalization takes care of this by normalizing the ratings so the ratings of a user have a mean of zero and a variance of 1. You can explore the MATLAB function *zscore*.

Exercise 5

Consider Table 3 which has a new user Kim added to our earlier toy example. We are interested in generating a recommendation for Kim. Can you guess which movie we should recommend just by looking at the table? Now write a small code in MATLAB to generate those predictions following the steps mentioned below:

- Calculate the PC similarity of Kim with other users using the provided MATLAB function *PCSimVecMatrix*
- Find the nearest neighbors of Kim using a threshold for the similarity
- Use mean-centering to normalize the ratings of the neighbors
- Generate a recommendation for Kim using the normalized ratings of neighbors and similarity weights

Carefully read the specification of the provided MATLAB function *PCSimVecMatrix*. Can you use it to skip one of the steps mentioned in Exercise 5? Refer to our implementation for Exercise 5 in file *UserBasedPredictionToy.m*.

3.4 How to Evaluate a Recommender System?

We need some methodology to evaluate our recommendations. In this section we will discuss multiple ways to do so. A recommender system generally looks at a mix of these values to evaluate its results. We first seperate our database into a test set and a train set. We compute our results based on the train set and make our predictions. We extract the ratings from our test set and look at the corresponding predictions we genrated for them. In order to quantify the difference, we use the following techniques.

3.4.1 Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{\sum_{(x,i)\in T}(r_{xi} - r^*_{xi})^2}{N}}$$

where T : The set of test data

$N = |T|$: The number of elements in the test data

r_{xi} : The actual rating of item 'i' by user 'x'

r^*_{xi} : The rating that our system predicts user 'x' will give item 'i'

We can try to design our algorithm to reduce this RMSE value, by minimizing the sum of squared errors (SSE) thereby giving us more accurate predictions of user ratings and hence improving our recommendations.

3.4.2 Top Five Rated Movies (TFRM)

Sort the movies in the test dataset in a decreasing order of ratings and look at the top five rated movies. For the movies rated in the test dataset, look at their corresponding predicted ratings and sort the predictions in a decreasing order. We find the intersection of the top 5 movies for both these sets to see how close we came to understanding user preference. Let the top five movies in the test dataset for a user be represented by S_{test} and the top five movies using the predicted ratings be S_{pred} Then,

$$Error = \frac{S_{test} \cap S_{pred}}{|S_{test}|}$$

3.4.3 Number of Good Movies Not Predicted (NMNP)

Another measure of error could be the number of movies that were rated by the user in the test dataset as good which the recommender system predicted as bad. For the notion of good, we use a threshold value t. Given a mean rating of m, any movie with a rating greater than $m + t$ is considered good. Thus, if S_{gtb} represents the number of movies that were good as per test data but were bad as per our predictions, and S_g are the set of all the good movies in the test data set, then the error is,

$$Error = 1 - \frac{|S_{gtb}|}{|S_g|}$$

3.4.4 Number of Flipped Movies (NFP)

This is an extension of the previous error metric. Here we look at all the movies that user rated as good and bad. The definition of good remains the same as before. We define bad movies as movies which have ratings lower than $m - t$. S_{flip} is set of good movies that the recommender system predicted as bad and bad movies that the recommender system predicted as good, S_{gb} is the set of all good and bad movies in the test data set, then the error is,

$$Error = 1 - \frac{|S_{flip}|}{|S_{gb}|}$$

Now that we have all the pieces required to build a user based recommendation system, let's work out an example! We will be using the MovieLens dataset for this. The train dataset has 943 users and 1682 movies while the test dataset has 463 users and 1682 movies.

Similarity measure	***kn***	***ks***	**Normalization**
Pearson Correlation	30	0.95	Mean-Centering

Table 4: User-based recommendation parameters

Exercise 6

In this exercise, you will extend the MATLAB code from Exercise 5 to generate recommendations using user-based method on a large data set. The training data is available in file u1.base and test data in u1.test. You can read these files into *utility matrix* form using the function *ConvertUDataToMatrix* that takes filename as input. You should pick a user from the test data, generate recommendations for him and calculate the *error* in the recommendation. You can repeat this for all the users in the test set and calculate the mean *error* across the users. This will give you a measure of how well the recommendation system performs for one *configuration*. A *configuration* is defined by the chosen values for a number of parameters:

- Choice of similarity criteria - Cosine similarity, PC similarity (Use functions *CosSimVecMatrix* and *PCSimVecMatrix*)
- Choice of threshold value for similarity *ks* such that users with similarity weights greater than *ks* are selected as neighbors
- Choice of threshold number *kn* such that *kn* users with highest similarity weights are selected as neighbors *(You may choose to select neighbors based on either threshold ks or threshold kn. You can also use a combination of the two approaches.)*
- Choice of normalization used - Mean Centering (Check out function *PCSimVecMatrix*), no normalization
- Choice of metric to calculate the error - RMSE, TFRM (use *Top5Accuracy*), NMNP, NFP (use *StepErrorFunction*)

A set of values selected for these parameters will give you one configuration. Vary one or more of these parameters to run for different configurations and calculate the mean error for each. You DO NOT need to cover the entire space of configurations. Refer to Table 4 that gives the values of parameters that gave us best results for this data set. Do some choices perform better than others? Try explaining any interesting observations. Our implementation is also provided for reference in files "UserBasedPredictionReal_kn.m" and "UserBasedPrediction-Real_ks.m".

	Die Hard 1	Die Hard 2	Die Hard 3	Die Hard 4
John	4	4		
Susan	4	2	3	3
Eric			4	4

Table 5: Truncate to Die Hard

4 Model Based Systems

The systems based on methods discussed earlier in the lab have been widely used and produce good results. But such algorithms have been shown to have several limitations.

- Sparsity - Nearest Neighbor (NN) Algorithms need exact matches. As a result algorithms sacrifice recommender system's coverage and accuracy [4]. To be more precise, correlation coefficient is only defined between customers who have products (movies in our database) in common. In an ecommerce environment where there are large number of items, one may find many customers who do not have any correlation with other customers [2]. As a result Nearest Neighbor based algorithms are not able to recommend anything to these customers. This problem is known as the reduced coverage problem. The sparsity could also lead to a recommender system to miss certain obvious neighbors.

  For example - John and Susan are highly correlated. Susan and Jack are also highly correlated. Conventional wisdom might suggest John and Jack should also have similar choices, however if John and Jack have very few ratings in common, such patterns could be easily missed.

- Synonym Problem - In real life scenarios, different product names can refer to similar items. Correlation based recommender systems cannot find such latent association and thus end up treating these objects as two separate entities.

  For example, let us consider two customers one of whom rates 10 different writing pad products as "high" and another customer rates 10 different notepads as "high". NN based recommender system will not capture their association.

One of the methods used to handle both the problems is the low rank approximation method. Let us do an exercise to understand the issues discussed above and see how low rank approximation can help us.

Exercise 7

Take a look at Table 5. The table illustrates the example we described while discussing *Sparsity*. How do you ensure that John and Eric are considered similar to each other? Look at the code written in *Examples/ToyExampleModelSVDMotivation.m* and run it to see how low rank approximation helps us magically establish this relationship between John and Eric!! We will discuss what low rank approximation is in the next section. For now, do you agree that this method is helping us with the sparsity problem?

Exercise 8

Look at *Examples/ToyExampleModelSVDMotivation2.m.* It explains a scenario similar to the one talked about in Synonym Problem and looks at a potential solution. Run the code and observe the output? Do you think Low Rank approximation is finding some sort of latent association? What are the predictions for values not rated by the user?

4.1 Singular Vector Decomposition

The Singular Value Decomposition (SVD) is a well known matrix factorization method. Formally, the SVD of an $m \times n$ matrix A is a factorization of the form $A = U\Sigma V^T$ where U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix with non-negative terms on the diagonal, and V is an $n \times n$ orthogonal matrix. The diagonal entries of Σ are known as the singular values of A. The m columns of U and the n columns of V are known as the left and right singular vectors of A respectively.

The SVD gives the 'best' low-rank approximation of a matrix. To put this in a more formal notation, it minimizes the Frobenius form of the difference between the approximation and the original matrix.

Assume that matrix $A \in R^{m\times n}$ with rank $r > k$. The Frobenius norm approximation problem $min||A-Z||_F$ where $rank(Z) = k$ has the solution:

$$Z = A_k = U_k\Sigma_k V_k^T$$

where U_k, Σ_k and V_k are the matrices obtained by truncating the SVD to contain only the first k singular vectors/values.

The SVD is implemented by the function *svd* in MATLAB. You can try the following code in matlab to get an idea:

```
A = magic(3)
[U,D,V] = svd(A)
```

You should observe that the columns of the matrices U and V are orthonormal. Also note that Σ is a diagonal matrix with non-negative and monotonically decreasing entries along the diagonal. These are the singular values of A.

4.2 Generating Predictions

We can use SVD in a recommender system to capture a certain latent relationship between customers and products/movies. We can use this relationship to capture the predicted preference for a certain product/-movie of a consumer. *So the next big question is*, how do you calculate the SVD of an incomplete matrix? Or rather, how do you complete a matrix in general? In order to handle this, we will study various approximations. We will then run these methods and collect empirical evidence to validate their effectiveness.

4.2.1 Latent Matrix Factorization

Since we cannot generate an SVD of an incomplete matrix, we can try to find a low rank approximation to our matrix using factorization similar to SVD. To do this, we first fill the incomplete values with 0 or any random permissible number within the valid range of our ratings. Thus the factorization of a matrix R into two matrices P and Q could be viewed as a minimization problem where we are trying to reduce the error of our known values from their predicted values. In other words, our ideal factorization would be the one where after multiplying P and Q, we get back the same value of ratings that the user had filled initially along with some predictions. In order to avoid over-fitting and generate good predictions, we use regularized least squares approach to solve the factorization problem. Because Latent Matrix Factorization is difficult to implement, we discuss this topics in a more formal context in the Appendix. We encourage the user to go through the tutorial in Appendix and run the code provided in the end to get an idea of Latent Matrix Factorization. We now discuss two other simple, yet effective approaches for Matrix Completion.

4.2.2 Average Value Based SVD Completion

Below is the description of steps used to generate predictions using Average Values -

1. Let R be the original sparse matrix where rows represent the user and columns represent the movies

2. Fill the empty cells in each column with average values of the ratings of the product/movies in that column
3. Calculate the average rating for each customer $\overline{C}$ using the non zero values
4. Let $R_{norm} = R - \overline{C}$, i.e. mean center each row
5. Factor R_{norm} using SVD and obtain U, S and V
6. Truncate U, S and V to U_k, S_k and V_k
7. Compute the resultant matrix $Pred_{norm} = U_k * S_k * V_k$
8. Add the average customer value calculated in Step 3 to this matrix, i.e, $Pred = Pred_{norm} + \overline{C}$

Exercise 9

Implement a function for average value SVD completion. Use the steps described above and run the code on matrix generated from table 1. Compare the result with our implementation located in *Examples/-ToyExampleAverageValueSVD.m*. What do you think about the corresponding predictions? Do they make sense?

4.2.3 Iterative SVD Completion

Another simple approach to complete a matrix and generate predictions is to use iterative SVD.

1. Let R be the original sparse matrix where rows represent the user and columns represent the movies
2. Fill the empty cells in each row with 0. Let this new Matrix be R_{next}
3. Factorize R_{next} using SVD and obtain U, S and V
4. Truncate U, S and V to U_k, S_k and V_k
5. Compute the resultant matrix $R_{next} = U_k * S_k * V_k$
6. Use the known values in R to replace values in R_{next}
7. Repeat steps from 3 to 6, T number of times

Exercise 10

Implement a function for iterative SVD completion. Use the steps described above and run your code on the matrix generated from table 1. Compare the result with our implementation located in *Examples/-ToyExampleIterativeSVD.m*. Look at the corresponding predictions. What do you think about them? How do they compare to the predictions using the previous methods?

4.3 Nearest Neighbor

By now you should have learnt to predict the ratings for unrated items. So what do we do with these predictions? One thing that we can do after applying a matrix completion technique is to recommend the top rated predictions for each user. Additionally we can look into the Nearest Neighbor approach that we discussed earlier in the lab.

4.3.1 Using Predicted Matrix

Once we have a prediction for all the movies/products of all the users, we can use nearest neighbors to generate item recommendation for a user. Why do we do this when we already have a prediction? Well, nearest neighbor method can lead to something called as "Serendipity". We will discuss the meaning of Serendipity in a later section. This method is based on the assumption - given a high accuracy of prediction, we might find better neighbors and thus better ratings.

4.3.2 Original Matrix in Lower Dimension

Another work around to find better neighbors is to look for neighbors in a lower dimension space of the original sparse matrix. The motivation here is that a lower dimension space will lead to a denser matrix and might help us find better neighbors.

4.3.3 Which Nearest Neighbor approximation to Use?

To answer this question we will resort to the golden answer to life, universe and anything non-concrete - *"It Depends!!!"*. For the purpose of this lab we used Nearest Neighbor approach in a lower dimension space of predicted matrix. Empirically, it worked better. In general, one tries different approaches and sees what works best for their environment and their preference of error metrics.

4.4 Bringing it All Together

Now that you have all the concepts required to understand a Model Based Recommender System, we will encourage you to experiment with it to get an idea of the different trade-offs involved and the corresponding complexity in settling down on a method. We will be using the MovieLens dataset for this. The train dataset has 943 users and 1682 movies while the test dataset has 463 users and 1682 movies.

Exercise 11

Write a code to run Model Based Prediction on the entire movie lens dataset. Follow these steps -

1. Read u1.base and u1.test.
2. Create a Utility Matrix where unitialized values are set to zero. Use *ConvertUDataToMatrix.m* to do both these steps.
3. Create two matrix completion for the u1.base dataset, one based on Average Value SVD and one based on Incremental SVD. Use *IncrementalLowRankCompletion.m* and *AverageValueBasedMatrixCompletion.m* for the same.
4. Look at the Top 5 predictions from both of them for any random customer.
5. Pick a customer from u1.test and look at the movies he has rated.
6. Look at the corresponding ratings in your predictions. Do they agree?
7. Calculate the average errors for all the customers in the test dataset using one of the error metrics - RMSE, TFRM (use *Top5Accuracy*), NMNP, NFP (use *StepErrorFunction*)

We have already implemented this code in *ModelBasedPrediction / EvaluateModelBasedSVD.m*. Play around with the parameters i_singularValueThreshold, i_iterCount, i_NumNearestNeighbor, look at the impact it has on different error rates. It is upto you to decide what factors are important and what constitutes a good system. We have provided you with outputs for many different parameters in the section below.

4.4.1 Results and Analysis

We will be using the same error metric as discussed in Section 3.4. The average number of movies rated as good or bad per user is 27.5 while the average number of movies rated as good is 14.85. We would look at the results of using the best ratings from our "Completed Matrix" (using Average value and Iterative Based) as our estimates, and the results obtained from using Nearest Neighbor approach on this "Completed Matrix" as our prediction estimates. Use the following analogy to interpret the results -

1. An RMSE value of 1 implies that on an average our predicted rating is off by a value of 1.
2. An NFP value of 20 implies that out of the 28 movies user considered good or bad, we were not able to classify 20 of those.
3. An NMNP value of 14 implies that out of the 15 movies a user considered good, we were not able to classify 14 of those.
4. A TFRM value of 1 implies that the top five movies based on predicted ratings and the top five movies based on test ratings were off by one movie. (Note: We only consider predictions for movies rated in test dataset)

Changing i_NumNearestNeighbor, i_singularValueThreshold = 0.03, i_iterCount = 40

We will vary the number of nearest neighbors here and see the results for Nearest Neighbor Approach. Table 6 gives you the results for using the "Completed Matrix". Table 7 and Table 8 show the results after varying the the number of nearest neighbors for average value based SVD method and iterative SVD method respectively.

Table 6: **Prediction 1**

	RMSE	NFP	NMNP	TFRM
Average Value SVD	0.4001	15.8126	4.7124	0.5599
Iterative SVD	0.9076	14.3246	3.8562	0.6614

Table 7: Nearest Neighbor Sweep - Average Value Based SVD

# Nearest Neighbors	RMSE	NFP	NMNP	TFRM
1	0.4972	27.5163	14.8497	0.5373
20	**0.4013**	**15.7952**	**4.7102**	**0.5595**
40	**0.4013**	15.7974	4.7124	**0.5595**

Changing i_singularValueThreshold, i_NumNearestNeighbor = 80, i_iterCount = 100

Table 9 shows the result for sweeping singular value threshold. Singular Value Threshold will dictate the number of top singular values picked up for low rank approximation. All values greater than $singularValueThreshold * maxSingularValue$ are part of the low rank approximation matrix.

Sensitivity to Iteration Count for Iterative SVD

Table 10 shows the results for different iteration counts for Iterative SVD provided, i_singularValueThreshold is fixed to 80 and Singular Value Threshold is fixed to 0.1.

Analysis

As we can see, different parameters can lead to the optimization of different error metrics. The parameter set which gives us the best result depends on the error metric a system developer considers important.

5 Miscellaneous Topics

5.1 Content Based Recommender Systems

Content-based recommender systems recommend an item to a user based upon a description of the item and the profile of the user's interests. Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists of matching up the attributes of the user profile against the attributes of an item. The result is a relevance judgement that represents the users level of interest in that object. If a profile accurately reflects user preferences, it is of tremendous advantage for the effectiveness of an information access process.

Table 8: Nearest Neighbor Sweep - Iterative SVD

# Nearest Neighbors	RMSE	NFP	NMNP	TFRM
1	1.4154	27.5163	14.8497	0.5373
20	**1.1235**	17.9935	5.6885	**0.6148**
40	1.242	**17.9586**	**5.4423**	0.6157

Table 9: Results for Singular Value Threshold Sweep

		Predictions				Nearest Neighbors			
	Threshold	RMSE	NFP	NMNP	TFRM	RMSE	NFP	NMNP	TFRM
Average Value	0.01	0.4001	15.8126	4.7124	0.5599	0.4015	15.841	4.7495	0.5625
	0.1	0.3987	**15.793**	**4.6841**	0.5621	**0.4013**	**15.7996**	**4.7146**	0.5595
	0.8	**0.3963**	15.9913	4.9216	**0.556**	0.4023	15.8715	4.8126	**0.559**
Iterative	0.01	1.5571	26.0392	14.841	0.7407	1.3346	21.281	8.6144	0.5956
	0.1	0.9026	14.0283	3.7059	0.6575	1.1042	17.5142	5	0.6122
	0.8	**0.4495**	**13.024**	**1.1808**	**0.5647**	**0.436**	**13.0719**	**1.159**	**0.5643**

Table 10: Iteration Count Sweep

	Predictions				Nearest Neighbors			
Iterations	RMSE	NFP	NMNP	TFRM	RMSE	NFP	NMNP	TFRM
40	0.9076	14.3246	3.8562	0.6614	1.1209	17.7778	5.1852	0.6096
100	**0.9026**	14.0283	**3.7059**	**0.6575**	1.1042	17.5142	5	0.6122
1000	0.92	**13.8344**	3.9346	0.6619	**1.0805**	**17.1155**	**4.719**	**0.6092**

For instance, it could be used to filter search results by deciding whether a user is interested in a specific Web page or not and, in the negative case, preventing it from being displayed.

5.1.1 Advantages and Disadvantages of Content-based Filtering

The adoption of the content-based recommendation paradigm has several advantages when compared to the collaborative one:

- **User Independence** - Content-based recommenders exploit the ratings provided by the active user to build her profile.
- **Transparency** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations.
- **New Item** - Content-based recommenders are capable of recommending items not yet rated by any user i.e., they do not suffer from the first-rater problem.

Nonetheless, content-based systems have several shortcomings:

- **Limited Content Analysis** - Content-based techniques have a natural limit in the number and type of features that are associated with the objects they recommend. Domain knowledge is often needed, e.g., for movie recommendations the system needs to know the actors and directors, and sometimes, domain ontologies are also needed. No content-based recommendation system can provide suitable suggestions if the analyzed content does not contain enough information to discriminate items the user likes from items the user does not like.
- **Over-Specialization** - Content-based recommenders have no inherent method for finding something unexpected. The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated. This drawback is also called serendipity problem to highlight the tendency of the content-based systems to produce recommendations with a limited degree of novelty.
- **New User** - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

5.2 Serendipity

Recommender Systems exist to help users discover an item that he or she would like among the large pool of items available. Most of the errors that we saw did not talk about the novelty of recommendation. For example, if I have seen Star Wars: A New Hope, Star Wars: The Empire Strikes Back and Star Wars: Revenge of the Sith, I will most likely find a lot of nearest neighbors whose collective votes leads to the recommendation for The Phantom Menace, Attack of the Clones and Revenge of the Sith. I will definitely end up liking that movie. Now consider a scenario where I watch a movie of a completely different genre, say *Lincoln* and enjoy it a lot. An item based recommendation system would have never led me to discover this movie as all the movies I have rated fall into the Action/Saga/Fantasy category. Such discoveries are called serendipitous discoveries.

Serendipity is related to unexpectedness and involves a positive emotional response of the user about a previously unknown item. It is concerned with the novelty of recommendations and how far such recommendations may positively affect the user. Serendipity is generally not easy to measure and most systems consider a trade-off between various forms of accuracy and serendipity.

Appendix

A Latent Matrix Factorization

We talked about Latent Matrix Factorization in the document earlier. In this section, we will go through the basics of matrix factorization [5]. Assume that we are dealing with user ratings (score in the range of 1 to 5) of items in a recommendation system.

Basic Idea

In recommendation system like Netflix or MovieLens, there are a group of users and a set of items. Given that each user has rated only some items in the system, we would like to predict how the users would rate the other not yet rated items, so that we can make appropriate recommendations to the users. We can represent all the information we have about the existing ratings in a matrix. Assume that we have 5 users and 4 items, and ratings range from 1 to 5 (integer values), the matrix may look something like this:

Table 11: User Ratings

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	-	4

The task of predicting the missing ratings can be considered as filling in the blanks such that the values would be consistent with the existing ratings in the matrix.

The basic idea behind using matrix factorization is that there should be some latent features that can predict how a user rates an item. For example, two users would give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is an comedy movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to determine a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item.

The Mathematics of Matrix Factorization

Having discussed the intuition behind matrix factorization, we can now go on to work on the mathematics. Firstly, we have a set U of users, and a set D of items. Let **R** of size $|U| \times |D|$ be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrices **P** (a $|U| \times K$ matrix) and **Q** (a $|D| \times K$ matrix) such that their product approximates **R**:

$$R \approx P \times Q^T = \widehat{R}$$

In this way, each row of **P** would represent the strength of the associations between a user and the features. Similarly, each row of **Q** would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$r_{ij} = p_i^T q_j = \sum_{k=1}^{K} p_{ik} q_{kj}$$

Now, we have to find a way to obtain **P** and **Q**. One way to approach this problem is to first initialize the two matrices with some values, calculate how 'different' their product is from **M**, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference.

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating.

To minimize the error, we have to know in which direction we have to modify the values of p_{ik} and q_{kj}. In other words, we need to know the gradient at the current values, and therefore we differentiate the above equation with respect to these two variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \widehat{r_{ij}})(q_{kj}) = -2e_{ij}q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \widehat{r_{ij}})(p_{ik}) = -2e_{ij}p_{ik}$$

Having obtained the gradient, we can now formulate the update rules for both p_{ik} and q_{kj}:

$$p^{`}_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij}q_{kj}$$
$$q^{`}_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}p_{ik}$$

Here, α is a constant whose value determines the rate of approaching the minimum. Usually we will choose a small value for α, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

A question might have come to your mind by now: if we find two matrices **P** and **Q** such that **P** $\times$ **Q** approximates **R**, isnt that our predictions of all the unseen ratings will all be zeros? In fact, we are not really trying to come up with **P** and **Q** such that we can reproduce **R** exactly. Instead, we will only try to minimize the errors of the observed user-item pairs. In other words, if we let T be a set of tuples, each of which is in the form of (u_i, d_j, r_{ij}), such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimize every e_{ij} for $(u_i, d_j, r_{ij}) \in T$. (In other words, T is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learnt.

Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process.

$$E = p_i^T q_j = \sum_{(u_i,d_j,r_{ij}) \in T} e_{ij} = \sum_{(u_i,d_j,r_{ij}) \in T} (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2$$

Regularization

The above algorithm is a very basic algorithm for factorizing a matrix. There are a lot of methods to make things look more complicated. A common extension to this basic algorithm is to introduce regularization to avoid over-fitting. This is done by adding a parameter β and modify the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)^2$$

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers.

In practice, β is set to some values in the range of 0.02. The new update rules for this squared error can be obtained by a procedure similar to the one described above. The new update rules are as follows.

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj})$$

Exercise 12

Study the code in *LatentMatrixFactorization.m* which implements the Latent Matrix Factorization technique. Use *LMFCall.m* to call this method. Do you get interesting and/or useful recommendations for the new user based on her ratings for a small set of movies? Try and add/change the ratings for the new user and see how the recommendations change. Try different values of regularization parameters in *LatentMatrixFactorization.m* and observe the differences in new user ratings.

B Code and Function Walkthrough

Once you unzip the files in a directory *RecommenderSystemLab*, it will contain the following five folders. Add the directory *RecommenderSystemLab* and its subfolders to your current matlab path.

1. **UsefulScripts -** This folder contains basic read write function implementations.
   - **ConvertUDataToMatrix.m -** Contains the function to read a database and convert it into a user*movies matrix with uninitialized value set to zero.
   - **GetMovieNameDatabase.m -** Contains the function call to read the movie names for each movie Id.
2. **LabFunctions -** This folder contains basic functions used throughout the lab.
   - **CosSimVecMatrix.m -** This function generates the Cosine similarity between a user and other users based on their ratings.
   - **PCSimVecMatrix.m -** This function generates the Pearson Correlation similarity between a user and other users based on their ratings. It also returns the normalized utility matrix, where the mean rating of each row has been removed from its ratings.
   - **AverageValueBasedMatrixCompletion.m -** This function is the implementation of Average Value Based Matrix Completion method.
   - **Top5Accuracy.m -** This function takes in the test vector and predicted vector to calculate how many top five movies in the test vector were predicted by the predicted vector.
   - **StepErrorFunction.m -** This function calculates the number of good and bad movies in a test vector that were not predicted as good and bad in the predicted vector.
   - **IncrementalLowRankCompletion.m -** This function implements the incremental low rank completion method.
   - **PrintOutMoivesOfAUser.m -** Given a 1682 length vector and a movie database, this function prints out the names of the movie rated by the user. The input representation can be changed using a mode flag.

- **TopKEigenValues.m** - Given a threshold with respect to the maximum sigma values and a diagonal matrix of sigma values, this matrix returns a matrix with sigma values below the threshold set to zero.
- **FillZeroEntryWithAverageInARow.m** - Function used by Average Value Based Matrix Completion for Matrix Initialization.
- **LatentMatrixFactorization.m** - This function implements the technique for Matrix Factorization.
- **LMFCall.m** - This calls the LatentMatrixFactorization function to factorize the given matrix.

3. **ModelBasedPrediction** - This folder has the final exercise of the Model Based System
   - **EvaluateModelBasedSVD.m** - This function runs the Model Based SVD for the entire movie lens dataset, parameters are configurable.
   - **ModelBasedPredictionTest.m** - EvaluateModelBasedSVD calls this function for each customer to generate the prediction.
4. **Examples** - This folder has all the toy examples used in the Lab.
   - **UserBasedPredictionToy.m** - contains code for generating prediction in Exercise 5.
   - **UserBasedPredictionReal_ks.m and UserBasedPredictionReal_kn.m** - are the implementations of user-based recommendation for reference.
   - **ToyExampleAverageValueSVD.m** - Contains code to implement Average Value SVD.
   - **ToyExampleIterativeSVD.m** - Contains code to implement Iterative Value SVD.
   - **ToyExampleModelSVDMotivation.m** - Contains example code to describe issues with Sparsity and a potential solution.
   - **ToyExampleModelSVDMotivation2.m** - Contains example code to describe issues related to Synonym and a potential solution.
5. **Data** *(You can read these files into utility matrix form using the function ConvertUDataToMatrix that takes filename as input.)*
   - **u1.base** - Train Data for our examples.
   - **u1.test** - Test Data for our examples.
   - **u.data** - Entire Data set for our examples.

References

[1] MovieLens 100K Dataset. `http://grouplens.org/datasets/movielens/100k/`.

[2] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[3] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook*, pages 107–144. Springer, 2011.

[4] Badrul M Sarwar, Joseph A Konstan, Al Borchers, John T Riedl, Badrul M Sarwar, Joseph A Konstan, and John T Riedl. Applying knowledge from kdd to recommender systems. *University of Minnesota, Minneapolis*, 1(612):625–4002, 1999.

[5] Albert Au Yeung. Matrix factorization: A simple tutorial and implementation in python.