CS532 Lab
Recommender Systems

Lokesh Jindal Mehreen Ali Urmish Thakker

Fall 2015

Contents

1 Motivation

User modeling, adaptation, and personalization techniques have hit the mainstream. The explosion of social network websites, on-line user-generated content platforms, and the tremendous growth in computational power of mobile devices are generating incredibly large amounts of user data, and an increasing desire of users to "personalize" (their desktop, e-mail, news site, phone).

The potential value of personalization has become clear both as a commodity for the benefit or enjoyment of end-users, and as an enabler of new or better services a strategic opportunity to enhance and expand businesses.

An exciting characteristic of recommender systems is that they draw the interest of industry and businesses while posing very interesting research and scientific challenges.

In spite of significant progress in the research community, and industry efforts to bring the benefits of new techniques to end-users, there are still important gaps that make personalization and adaptation difficult for users. Research activities still often focus on narrow problems, such as incremental accuracy improvements of current techniques, sometimes with ideal hypotheses, or tend to overspecialize on a few applicative problems (typically TV or movie recommenders - sometimes simply because of the availability of data). This restrains the range of other applications where personalization technologies might be useful as well.

Thus, we may have reached a good point to take a step back to seek perspective in the research done in recommender systems.

	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

2 User Based Recommendation

2.1 Idea of *similarity*

Let's say the users of a website like Imdb [TODO FIXME] rate the 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 from 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 [TODO FIXME] that can be represented in the form of utility matrix *utility matrix* U as shown in equation [TODO FIXME]. 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

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

2.1.1 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 possible. In real data sets, we need 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 *neighborhood-based* recommendation systems, since it can have a significant impact on the performance and accuracy of a 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 the equation [TODO FIXME].

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

where $\mathrm{I_u}$ represents 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

EXERCISE2: In the example data set of Table [TODO FIXME], find the users that are most similar to user Eric. You can use the function *CosSimVecMatrix* provided to generate the similarity of user *Eric* with other users.

Hopefully that was easy and your 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 respective 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.

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 the equation [TODO FIXME].

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

where $\bar{r_u}$ [TODO FIXME] 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.

EXERCISE3: 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 *similar* to *Eric*. You can do a similar analysis for comparing the Cosine similarities and Pearson Correlation similarities of other users.

2.2 Calculating Predicted Ratings

The Recommender Systems exist so that they can make recommendations to the user. As highlighted in section [TODO FIXME], 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 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 the only neighbors v of user u who have rated the movie i, $N_i(u)$ representing the set of such neighbors.

EXERCISE4: Carefully look at the formula in the equation [TODO FIXME]. 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 EXERCISE4 [TODO FIXME], equation [TODO FIXME] that is more sophisticated in its calculation of the predicted rating should make sense.

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

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.

2.3 Rating normalization

Various users rate the movies differently, each having his/her own personal scale even when they have to give a rating on the same scale of 1 to 5. 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 use. To tackle this two normalization techniques are: Mean-Centering and Z-score normalization.

2.3.1 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 $\bar{r_u}$ [TODO FIXME] is the mean of ratings of user u.

2.3.2 Z-score normalization

While mean-centering takes removes the user bias by removing the mean 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*.

EXERCISE5: Consider the Table ?? 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:

	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

- **Calculate the PC similarity of Kim with other users using the provided MATLAB function *PCSimVecMatrix***
- **Find the nearest neighbors of Kim using a threshold**
- **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 the EXERCISE above.

2.4 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 predictions 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.

2.4.1 Root Mean Square Error

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

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.

2.4.2 Top Five Rated Movies

We sort the movies in our test dataset in a decreasing order and look at the top 5 rated movies, we then look at the predictions for all the movies rated in the test vector and sort them in a decreasing fashion. 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 predictions for the movies rated in test dataset for the user be represented by:

$$S_{pred}$$

Thus,

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

2.4.3 Number of Good Movies Not Predicted

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. Any movie with a rating more than the threshold value above average is considered good. Thus, if S_{gtb} represent the number of good movies that the recommender system predicted as bad and S_g are the set of all good movies in the test data set, then the error is simply,

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

2.4.4 Number of Flipped Movies

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

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

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

Table 4: Truncate to Die Hard

3 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 coverage and accuracy [TODO FIXME ref1]. 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 [TODO FIXME ref2]. 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 from missing certain obvious neighbors.

  For example - John and Susan are highly correlated, while Susan and Jack are 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 seperate 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 1

Take a look at Table 4. The table illustrates the example we described while discussing "Sparsity". So how do you ensure that John and Eric are considered similar to each other? Look at the code written in "Examples/ToyExampleModelSVDMotivation.m" and see how low rank approximation helps us magically establish this relationship between John and Eric. We will discuss what low rank approximation in the next section.

Exercise 2

Look at "Examples/ToyExampleModelSVDMotivation2.m". It explains a scenario similar to the talked about in Synonym Problem and looks at a potential solution.

3.1 Singular Vector Decomposition

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

3.2 Generating Predictions

We can use SVD in recommender system to capture certain latent relationship between customers and products/movies. We can use this relationship to capture the predicted likeliness of a certain product/movie by 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.

3.2.1 Latent Matrix Factorization

Recommendations can be generated by a wide range of algorithms. While user-based or item-based collaborative filtering methods are simple and intuitive, matrix factorization techniques are usually more effective because they allow us to discover the latent features underlying the interactions between users and items. Of course, matrix factorization is simply a mathematical tool for playing around with matrices, and is therefore applicable in many scenarios where one would like to find out something hidden under the data.

In this section, we will go through the basic ideas and the mathematics of matrix factorization, and then we will present a simple implementation in Python. We will proceed with the assumption that we are dealing with user ratings (e.g. an integer score from the range of 1 to 5) of items in a recommendation system.

Basic Idea

Just as its name suggests, matrix factorization is to factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix.

As mentioned above, from an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. And one obvious application is to predict ratings in collaborative filtering.

In a recommendation system such as Netflix or MovieLens, there is a group of users and a set of items (movies for the above two systems). Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users. In this case, all the information we have about the existing ratings can be represented in a matrix. Assume now we have 5 users and 10 items, and ratings are integers ranging from 1 to 5, the matrix may look something like this:

Table 5: User Ratings

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

Hence, 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 intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine 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 action movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to predict 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 $\mathbf{R}$ 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 $\mathbf{P}$ (a $|U| \times K$ matrix) and $\mathbf{Q}$ (a $|D| \times K$ matrix) such that their product approximates $\mathbf{R}$:

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

In this way, each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ 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 $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is the first initialize the two matrices with some values, calculate how 'different their product is to $\mathbf{M}$, 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.

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

Implementation in Python

```
import numpy

def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in xrange(steps):
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in xrange(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in xrange(len(R)):
            for j in xrange(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in xrange(K):
                        e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        if e < 0.001:
            break
    return P, Q.T
```

We can try to apply it to our example mentioned above and see what we would get. Below is a code snippet in Python for running the example.

```
R = [
     [5,3,0,1],
     [4,0,0,1],
     [1,1,0,5],
     [1,0,0,4],
     [0,1,5,4],
    ]

R = numpy.array(R)

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

nP, nQ = matrix_factorization(R, P, Q, K)
nR = numpy.dot(nP, nQ.T)
```

And the matrix obtained from the above process would look something like this:

We can see that for existing ratings we have the approximations very close to the true values, and we also get some 'predictions' of the unknown values. In this simple example, we can easily see that U1 and U2 have similar taste and they both rated D1 and D2 high, while the rest of the users preferred D3, D4 and D5. When the number of features (K in the Python code) is 2, the algorithm is able to associate the users and items to two different features, and the predictions also follow these associations. For example, we can see that the predicted rating of U4 on D3 is 4.59, because U4 and U5 both rated D4 high.

Table 6: Output of Matrix Factorization

	D1	D2	D3	D4
U1	4.97	2.98	2.18	0.98
U2	3.97	2.40	1.97	0.99
U3	1.02	0.93	5.32	4.93
U4	1.00	0.85	4.59	3.93
U5	1.36	1.07	4.89	4.12

3.2.2 Average Value Based SVD Completion

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

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 product/movies in that column.
3. Calculate the average rating for each customer $\overline{C}$ using the non zero values
4. Let $R_{norm} = R - \overline{C}$
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 $Pred = Pred_{norm} + \overline{C}$

Exercise 3

Implement a function for average value SVD completion. Use the matrix generated from table 1. Compare the result with our implementation located in Examples/ToyExampleAverageValueSVD.m

3.2.3 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 the process from Step 3 t number of times.

Exercise 4

Implement a function for iterative SVD completion. Use the matrix generated from table 1. Compare the result with our implementation located in Examples/ToyExampleIterativeSVD.m

3.3 Nearest Neighbor

3.3.1 Using Predicted Matrix

Once we have a prediction for all the movies/products for all the users, we can use nearest neighbors to generate item recommendation for a user. One may ask what use is this method if 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.

3.3.2 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.

3.3.3 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.

3.4 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.

Exercise 5

Look at the code in "ModelBasedPrediction / EvaluateModelBasedSVD.m" Play around with parameters, look at the impact it has on different error rates. It is upto you to decide what factors do you care about and what according to you is the best system. We have provided you with outputs for many different parameters in the table-reference-here[TODO FIXME].

4 Miscellaneous Topics

4.1 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 users 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 in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment 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. 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.

4.1.1 Advantages and Drawbacks 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 solely ratings provided by the active user to build her own 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.

4.2 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 might end up

liking that movie. Now consider a scenario where I end up watching a movie of a different genre altogether, say "Lincoln" and end up liking 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 measures how surprising the unexpected recommendations that is, serendipity is concerned with the novelty of recommendations and how far such recommendations may positively affect the user. It is generally not easy to measure and most systems consider a trade-off between various forms of accuracy and serendipity.