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

Table 1: Truncate to Die Hard

1 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 few 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 a recommender system from detecting 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: Take a look at Table 1. This 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 writted in "Examples/ToyExampleModelSVDMotivation.m" and see

how low rank approximation helps us magically establish this relationship between John and Eric. We will discuss what this low rank approximation is and how do we calculate it in the next section.

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

1.1 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 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 (i.e. the sum of squared errors).

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, i.e. orthogonal unit vectors. Also note that Σ is a diagonal matrix with non-negative and monotically decreasing entries along the diagonal. These are the singular values of A. The i^{th} singular value (Σ_{ii}) represents the energy content, or variance of the columns of A along the i^{th} left singular vector or the rows of A along the i^{th} right singular vector.

1.2 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. However, how do you calculate the SVD of an incomplete matrix? In order to handle this, we will study various approximations. We will then run these methods and collect empirical evidence to validate their effectiveness.

1.2.1 Latent Factor Model

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

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

1.3 Nearest Neighbor

1.3.1 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 negihbor 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.

1.3.2 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.

1.3.3 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.

1.4 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 tradeoffs involved and the corresponding complexity in settling down on a method.
EXERCISE: 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].

2 Miscellaneous Topics

2.1 Item Based Recommendation System

2.2 Seredipity

Recommender Systems exist to help users dicsover 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 disoveries are called seredipitious 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 tradeoff between various forms of accuracy and seredipity.

2.3 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.

2.3.1 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.

2.3.2 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,

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

2.3.3 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,

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

2.3.4 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,

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