Executive Summary

Internet users provide feedback about their likes and dislikes and a lot of data is generated in this process. Recommender Systems utilize this data to infer customers' interests.

Going by standard notation, the entity to which the recommendation is provided is called the user and the product being recommended is the item.

Recommender engines analyze the past interactions between the users and the items & predict the future choice of the user.

For example, Netflix allows its users to rate movies. This information is used by Netflix to make recommendations based on the movies seen and rated by users with similar interests.

This analysis encompasses a lot of different techniques, some of which will be focused upon in this tutorial.

The following topics will be discussed in this coursework -

- Recommendation techniques
- Model based techniques and Latent Factor approach
- Geometrical intuition for Latent Factor Models
- Un-constrained Matrix Factorization
    - ➢ Stochastic Gradient Descent
    - ➢ Incremental Latent Component Training
- Alternating Least Squares
- Regularization
- Coordinate Descent
- Incorporating User and Item Biases

The warmup questions test the fundamental understanding of these concepts and in the Lab questions we focus on implementing one of the Algorithms namely, Alternating Least Squares method and tests the understanding of other concepts in some more depth.

We have made use of a real life Movie Lens dataset to analyze various concepts involved in Movie Recommender systems.

Background

1. Recommendation techniques:

The type of Recommendation system that we can use depends on the two kinds of data available about users and items.

1. User-item interactions
2. Attribute information about the users and items: textual profiles/relevant keywords

The method used for the first type of data i.e User-item interactions is **Collaborative filtering (CF)**.

User-item interactions are typically captured in the form of a **Ratings Matrix** where every cell corresponds to the rating provided by **User u** to **Item i**. It is a common phenomenon that the Ratings Matrix is observed to be sparse in nature owing to huge catalog of items and large number of users not all of whom are associated/interested in all the products.

Hence, Collaborative filtering method uses this collaborated knowledge of multiple users and their ratings captured in the such a **Ratings Matrix** to make predictions to the users who did not see the item or purchase before.

Collaborative filtering method is based on the concept of identifying the ***underlying correlation*** of the ratings of every user to the type of items.

There are two types of methods that are commonly used in Collaborative filtering:

- **Memory Based Methods** (Neighborhood based collaborative filtering algorithm) -

  Based on their respective neighborhoods (user-based or item-based), it predicts the ratings of user-item combinations.

  User-based- E.g. Users that like watching romantic movies.
  Item-based - E.g. Movies which are romantic.

- **Model Based Methods** -

  Machine learning and data mining methods are used to find patterns on training data which is then used to make predictions for real data.

Content Based techniques are used when attribute information about the users and items are available. This method also uses the ratings matrix but the only difference here is that the model is user-centric and user-specific.

Hence, Collaborative filtering differs itself from other content-based methods in the sense that user or the item itself does not play a role in recommendation but rather the how(**ratings**) and which users(**user**) rated a particular **item**.

In this tutorial, we are going to focus on the *Model Based techniques* for Collaborative Filtering especially the **Latent Factor Models**, which is a widely-used technique in many companies.

2. Model based techniques and Latent Factor approach:

In the Model based technique, a model is learnt for the entire matrix and then the missing values of the **Ratings Matrix** are directly filled out using these models **in one shot**.

In other words, there is a separate training and testing phase. In contrast, other techniques are more instance based and there is no such distinction. One of the model based techniques is the Latent Factors approach.

Latent Factor Approach -

Key Idea: Find some form of correlation between all the rows and columns of the matrix simultaneously. It's like saying that all rows and columns can be dependent on some **common hidden features,** for example, genre in case of User Movie ratings matrix. In other words, the given matrix can have a lot of redundancies and it can be well approximated by a low rank matrix.

This can be achieved by visualizing Collaborative Filter as a **Matrix completion** task where the goal is to predict the unknown ratings for a user given user the ratings that is already provided.

Matrix Factorization is a key technique employed for Matrix Completion and identifying the Latent factors.

Suppose we have a Ratings Matrix **R** that has **rank k**, then we can easily split R as a factor of two Matrices U and V as:

$$R=UV^T$$

where, U is a n x k matrix and V is a m x k matrix

Even if the rank of matrix R is larger than k, we can approximate R as UV^T. Hence, the error of this approximation is $|| R - UV^T ||_F$. This is the residual matrix which needs to be minimized for minimum noise. ||.|| is the squared Frobenius Norm of the residual matrix. To visualize this, consider the following figure:

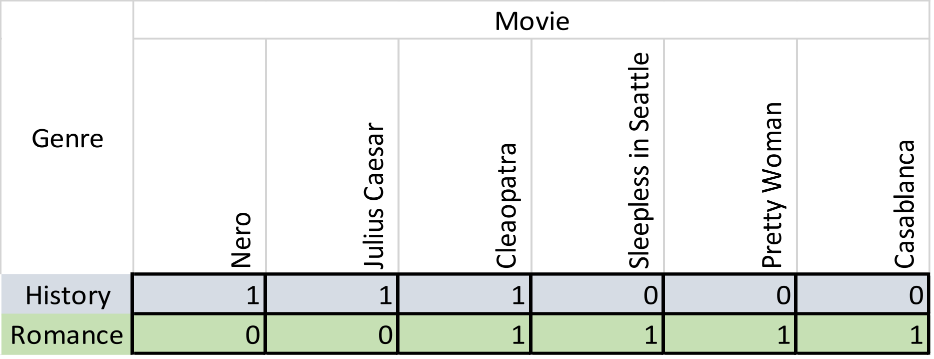
R

User	Movie					
	Nero	Julius Caesar	Cleaopatra	Sleepless in Seattle	Pretty Woman	Casablanca
1	1	1	1	0	0	0
2	1	1	1	0	0	0
3	1	1	1	0	0	0
4	1	1	1	1	1	1
5	-1	-1	-1	1	1	1
6	-1	-1	1	1	1	1
7	-1	-1	-1	1	1	1

$\approx$

U

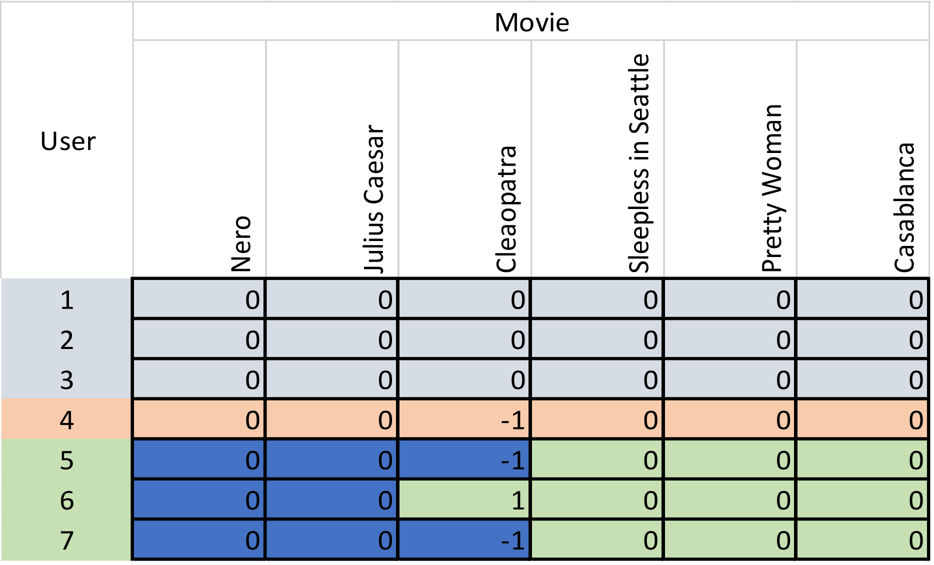
User	Genre	
	History	Romance
1	1	0
2	1	0
3	1	0
4	1	1
5	-1	1
6	-1	1
7	-1	1



Genre	Movie					
	Nero	Julius Caesar	Cleaopatra	Sleepless in Seattle	Pretty Woman	Casablanca
History	1	1	1	0	0	0
Romance	0	0	1	1	1	1

$\mathbf{V}^T$

(a) Example of rank-2 matrix factorization



User	Movie					
	Nero	Julius Caesar	Cleaopatra	Sleepless in Seattle	Pretty Woman	Casablanca
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	-1	0	0	0
5	0	0	-1	0	0	0
6	0	0	1	0	0	0
7	0	0	-1	0	0	0

(b) Residual Matrix

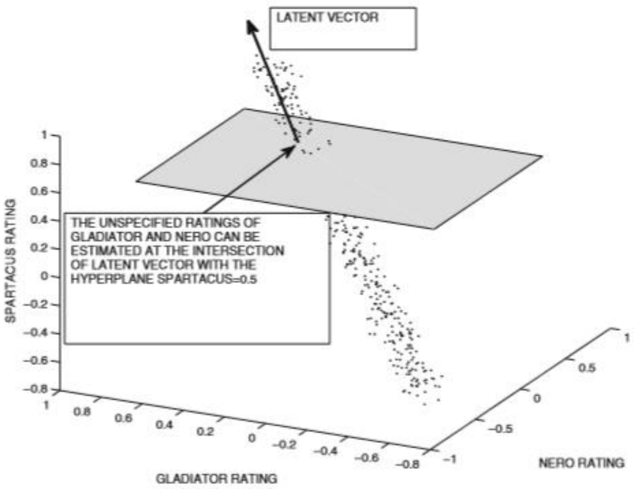
Fig. 2.1 (*Charu C. Aggarwal, Recommender Systems pg 95*)

In the above matrix, the rows correspond to the users and the columns correspond to the movies. The top left matrix in the figure is the Rating Matrix R. A technique is used (*will be discussed in the following sections*) to get/factorize the matrix R into two matrices: U and V which are of n x k and m x k dimensions. **U** gives a correlation between the users and the hidden features and **V** gives the correlation between the movies and the same hidden features. We see that UV^T is not exactly equal to R. This is because it is an approximated factorization. That is why we calculate the residual matrix in Fig 2.1(b). Our job is to minimize the Frobenius norm of this Residual Matrix.

This method is not that helpful when we have a fully specified ratings matrix R, but it becomes really helpful when we have a sparse matrix. In that case, we can still get completely specified U and V matrices and the respective entries of U and V can now be used to fill in the missing entries of the original matrix R.

Note: This method and Collaborative Filtering methods in general cannot be applied to provide recommendations when all the entries are missing in the original matrix i.e when there is no data available between user and item interactions. This is also referred to as the **cold start problem**.

3. Geometrical intuition for Latent Factor Models:



THE UNSPECIFIED RATINGS OF GLADIATOR AND NERO CAN BE ESTIMATED AT THE INTERSECTION OF LATENT VECTOR WITH THE HYPERPLANE SPARTACUS=0.5

Fig. 3.1 (*Charu C. Aggarwal, Recommender Systems pg 92*)

Let's say we have a movie ranking matrix of three movies - Gladiator, Nero and Spartacus which have been rated by n users. Every element of this nx3 matrix can be plotted in a 3d scatter plot as shown in Fig 3.1

From Fig.3.1, we can conclude that all the data is correlated in one direction. Any vector in this direction is called as the **Latent Vector**. For the given example, since this is the only dimension which can be visualized, we have one latent vector. In simple terms, this is the hidden feature based on which the movies and users can be compared. Hence, the nx3 matrix has rank 1 if we remove all the noise.

Key Idea: Latent vectors are created such that the squared distance of the data points from the hyperplanes formed by these latent vectors is as small as possible.

Therefore, we must use the partially specified dataset to recover the low-dimensional hyperplane on which the data approximately lies because the dimensional space over which the given incomplete matrix is visualized is reduced and rotated.

Now based on the new axis system, we get a complete matrix. This complete matrix is used to fill the missing entries in the original incomplete matrix.

From the geometric intuition explained above, we realize that in the Matrix Factorization of Ratings Matrix R = UV^T , each column of the U vector is a **latent vector** and each row is a **latent factor**.

Also, each row of U is that particular user’s affinity towards the hidden features and is called as **user factor**.

Each row of V is that particular movie’s affinity towards the same hidden features, and is called as **item factor**. Rating of a particular movie j by a particular user i (r_{ij}) is approximately equal to $u_i v_j^T$.

$$R_{ij} \approx \sum_{s=1}^{k} u_{is} u_{js}$$

$$= \sum_{s=1}^{k} (Affinity\ of\ user\ i\ to\ concept\ s) \times (Affinity\ of\ item\ j\ to\ concept\ s)$$

Based on the Fig 2.1, we can write the above equation as:

R_{ij} ≈ (Affinity of user i to history) x (Affinity of item j to history)

+ (Affinity of user i to romance) x (Affinity of item j to romance)

Note: Note that the latent vectors are not always orthogonal. Thus, it may be difficult to interpret them semantically always.

Note: It is possible to impose some constraints on U and V (i.e. making the latent vectors orthogonal or the latent factors non-negative, etc.)

The difference between different methods of recommendation analysis/ Matrix factorization we consider here are because of the following two reasons:

- Different constraints used for U and V
- The type of cost function that we choose to minimize

We will now look into Matrix factorization techniques to obtain the latent vectors in detail.

4. Unconstrained Matrix Factorization

In this approach, we do not set any constraints on the type of latent vectors that we are expecting.

Note: There are other approaches such as SVD which lays constraints such as latent vectors to be orthogonal.

The cost function that we will consider for this approach is :

$$Minimize\ J = \frac{1}{2} ||R - UV^T||^2$$

Subject to: No constraints on U and V

Fig. 4.1

The || . || norm used here is the Frobenius norm. The objective function J needs to be minimized. Fig. 3 makes sense when R is completely specified, but when dealing with sparse matrices with unspecified entries, we will simply modify the way we formulate J.

Let us consider (i,j) as the individual entries in the rating matrix R

Each prediction can be written as:

$$\hat{R}_{ij} = \sum_{s=1}^{k} u_{is} u_{js}$$

The net error in prediction can be broken down into the following individual entry:

$$e_{ij} = (r_{ij} - \hat{r}_{ij})$$

The new cost function that needs to be minimized is:

$$Minimize\ J = \frac{1}{2}\sum_{(i,j)\,\in\, S}^{k} e_{ij}^{2} = \frac{1}{2}\sum_{(i,j)\,\in\, S}(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js})^{2}$$

Subject to: No constraints on U and V

4.1) Gradient Descent

We can consider this as a problem in a feature space of (n*k + m*k) variables, all of which will get updated using the gradient descent approach. Thus, the gradient of the loss function with respect to every feature will be:

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-v_{jq}) + \lambda u_{iq}\ \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$= \sum_{j:(i,j)\in S}(e_{ij})(-v_{jq}) + \lambda u_{iq}\quad \forall i \in \{1 \ldots m\}, q \in \{1 \ldots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)(-u_{iq}) + \lambda v_{jq}\ \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

$$= \sum_{i:(i,j)\in S}(e_{ij})(-u_{iq}) + \lambda v_{jq}\quad \forall j \in \{1 \ldots n\}, q \in \{1 \ldots k\}$$

We first get the gradients of every parameter and then update all the (m*k + n*k) parameters together. Once updated, we check for convergence of the model. If not converged, we go on to get the new cost and then repeat the same procedure of updating the parameters using iterative method.

Note: In this approach, we are visualizing over a feature space of (m*k + n*k) features which in the cost function, appear as f_1f_2 combinations, hence we are optimizing a **non-convex function**. Thus, we do not use Normal Equations here as they will give incorrect solutions.

Technically, using the basic gradient descent algorithm is also not a good choice here as it can converge at a local minima and give a sub-optimal solution. So, we should choose some of the advanced gradient descent methods like the Bold Driver algorithm, Adam optimizer, Nesterov optimizer, Momentum, etc.

```
Algorithm GD(Ratings Matrix: R, Learning Rate: $\alpha$)
begin
  Randomly initialize matrices U and V ;
  S = {(i, j) : $r_{ij}$ is observed};
  while not(convergence) do
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$
    for each user-component pair (i,q) do $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq}$
    for each item-component pair (j,q) do $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq}$
    for each user-component pair (i,q) do $u_{iq} \Leftarrow u_{iq}^{+}$
    for each item-component pair (j,q) do $v_{jq} \Leftarrow v_{jq}^{+}$
  end
   Check convergence condition;
  end
end
```

(Taken from Charu C. Aggarwal, Recommender Systems pg 98)

We will be using the above algorithm with α as the learning rate.

Based on the method used to parametrize and update, there are many possible variations of this method. Some of them are listed below:

- Stochastic Gradient Descent
- Incremental Latent Component

4.2) Stochastic Gradient Descent

This is simply a batch update method. It is very similar to the previous method with the only difference being as to how the parameters are updated.

In one of the variations of SGD algorithm, we pick a random movie and a random user. Update all the parameters for that movie and the user. Then check for convergence. if not converged, we proceed to get the new cost and repeat the procedure.

$$u_{iq} \Leftarrow u_{iq} - \alpha \cdot \left[\frac{\partial J}{\partial u_{iq}}\right]_{Portion\ contributed\ by\ (i,j)} \qquad \forall q \in \{1 \dots k\}$$

$$v_{jq} \Leftarrow v_{jq} - \alpha \cdot \left[\frac{\partial J}{\partial v_{jq}}\right]_{Portion\ contributed\ by\ (i,j)} \qquad \forall q \in \{1 \dots k\}$$

Hence, instead of updating (m*k + n*k) parameters, we update 2*k parameters only. This is a tradeoff between smoothness of the descent and speed. Each iteration is computationally less expensive as compared to the previous method but at the same time it is not as accurate as the previous method. However, over a large number of iterations, they both end up roughly at the same point.

The update rule for every parameter is given by:

$$u_{iq} \leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq} \;\; \forall q \in \{1 \dots k\}$$
$$v_{jq} \leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq} \;\; \forall q \in \{1 \dots k\}$$

For better efficiency, each of these k entries can be updated simultaneously in vectorized form. Let u_i-bar be the ith row of U and v_j-bar be the jth row of V. Then, the updates described above can be rewritten in k-dimensional vectorized form as follows:

$$\overline{u_\imath} \leftarrow \overline{u_\imath} + \alpha e_{ij} \overline{v_\jmath}$$
$$\overline{v_\jmath} \leftarrow \overline{v_\jmath} + \alpha e_{ij} \overline{u_\imath}$$

The above method affects the smoothness of the cost curve. The smoothness of the curve can be improvised by using a batch of movies and users instead of considering a single movie and a user in an iteration. We can use Bold Driver algorithm over this and at the same time selectively keep modifying the learning rate α while getting the convergence to improvise on this approach. One other disadvantage of this approach apart from the smoothness of the descent is with parameter initialization. Based on where we initialize the parameters, we may have a number of different solutions that can minimize the cost function by reaching various possible local minima (one of them being the most optimized solution when convergence occurs at global minimum).

4.3) <u>Incremental Latent Component Training</u>

This is another variant of the same approach. Here, we train the latent components separately, reach convergence, then remove them from the original matrix and go on to train for the next latent component. It uses the fact that every element of the rating matrix is the sum of the product of individual latent factors. Since these individual latent factors are all independent of each other, they can be learnt separately. Once they have been learnt, they aren't required for future training, so we remove them from the original data/rating matrix and start the training procedure for the remaining data matrix.

The above algorithm can be summed up in the following pseudo code:

```
Algorithm ComponentWise-SGD(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V ;
  S = {(i, j) : $r_{ij}$ is observed};
  for q = 1 to k do
  begin
    while not(convergence) do
    begin
      Randomly shuffle observed entries in S;
      for each (i, j) ∈ S in shuffled order do
      begin
        $e_{ij} \leftarrow r_{ij} - u_{iq}v_{jq}$
        $u_{iq}^{+} \leftarrow u_{iq} + \alpha \cdot (e_{ij} \cdot v_{ij} - \lambda \cdot u_{iq})$
        $v_{jq}^{+} \leftarrow v_{jq} + \alpha \cdot (e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq})$
        $u_{iq} = u_{iq}^{+};\ v_{jq} = v_{jq}^{+}$
      end
      Check convergence condition;
    end
    { Element-wise implementation of $R \leftarrow R - \overline{U_q}\overline{V_q}^{T}$ }
    for each (i, j) ∈ S do $r_{ij} \leftarrow r_{ij} - u_{iq}v_{jq}$;
  end
end
```

(Taken from Charu C. Aggarwal, Recommender Systems pg 104)

This approach is different from the previous version of the problem. It can be seen in the structure of the loop formation in the pseudo code. Here, the outer loop cycles through each latent factor / feature separately thereby learning it individually. Whereas, in the previous problem, the outer loop cycled through the observed entries instead. Since, there are only 2 updates, this method is faster (Since values of first latent component can be independently trained for all movies' and users' 2 latent components, we can also have 2*(m+n) updates). Note that, in this approach the matrix R is changed effectively if that latent feature is learnt.

Note: This approach is similar to solving it using SVD. The major difference with SVD is that the latent vectors that are learnt will not be mutually orthogonal in this case. We can get that by using projected gradient descent approach, or getting orthogonal vectors in the very end by using simple techniques like Gram-Schmidt, etc.

Disadvantages of the Gradient Descent Approaches-

- Gradient Descent methods are rather sensitive both to the initialization and the way in which the step sizes are chosen.

5. Alternating Least Squares(ALS)

Alternating Least Squares is another Iterative algorithm with a different approach for matrix completion and hence predicting missing ratings with a different approach. Unlike previous methods, there is no gradient involved in the optimization step.

Key Idea: The key insight is that you can turn the non-convex optimization problem considered above into two separate quadratic problems which are obtained by fixing the U and V matrix alternatively. When one is fixed, the other one is computed, and vice versa. This process is repeated until we reach a convergence point where neither of the matrices U and V no longer change or the change is quite small.

The quadratic functions are convex in nature and hence give the most optimized answer (global minima) using simply the normal equation. But, each problem that we are solving is not the actual optimization problem as nearly half the parameters belonging to the fixed matrix are fixed.

Alternating least squares (ALS) can be thought of as block coordinate descent where the parameter vectors are split into two blocks (U and V) and then each iteration alternately updates each block of parameters

Additionally, the number of parameters that are updated at a time are reduced, we only optimize either for the movie or the user at a time and as a result, calculating the inverse in the normal equation is now computationally less expensive as now we will be calculating it over a k x k matrix.

Key steps in the Algorithm:

1.) Keep U fixed and solve for each of the n rows of V by treating the problem as a least-squares regression problem. Only the observed ratings in Ratings Matrix R can be used for building the least-squares model in each case.

More precisely each Linear Regression for a row in V can be setup as follows.

- Let $\overline{vj}$ be the jth row of V. In order to determine the optimal vector $\overline{vj}$, we wish to minimize the squared error which is a least-squares regression problem in $v_{j1}...v_{jk}$

$$\text{Minimize J} = \frac{1}{2}\sum_{(i,j)\epsilon S}^{k} e_{ij}^2 = \frac{1}{2}\sum_{(i,j)\epsilon S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}v_{js}\right)^2$$

- The terms $u_{i1}...u_{ik}$ are treated as constant values, whereas $v_{j1}...v_{jk}$ are treated as optimization variables.
- Therefore, the k latent factor components in v_j for the j^{th} item are determined with least-squares regression.

$$R_{i.} = U_{(fixed)}\, v_{.j}$$

- And the v.j can be obtained using the normal equations $(U^TU)^{-1}U^TR$ where U is the fixed matrix.

A total of n such least-squares problems need to be executed, and each least-squares problem has k variables.

2. Similar to step 1 but keep the V fixed and solve for each of the n columns of U by treating the problem as a least-squares regression problem.

3. Steps 1 and 2 are repeated for certain fixed iterations or as long as there is very less change in the U or V matrix.

Note: The movies that do not have ratings in the update rule can be penalized. By doing so, we will depend on only the movies that have ratings from the users and do not make any assumption around the movies that are not rated in the recommendation. This is called **Weighted ALS**.

Advantages of ALS -

- Since, each of the least-squares problem is independent for each item, that step of ALS can be parallelized easily.
- ALS preserves the sparse structure of the known matrix elements and is therefore storage-efficient.

Drawbacks of ALS -

The drawback of ALS is that it is not quite as efficient as stochastic-gradient descent in large-scale settings with explicit ratings.
Note: In coordinate-descent, the approach of fixing a subset of variables (as in ALS) is pushed to the extreme.

Warm Up

1) Based on the explanation, write the pseudocode of SGD algorithm.

2) Is it possible to use the latent vectors obtained by the factorization of Ratings Matrix to get an understanding of the semantic meaning i.e. can we understand what is the average rating of per genre of a movie by looking into the item vector factor matrix?

3) When will you do the conventional matrix factorization and when will you use SGD?

4) It is mentioned that ALS uses a convex function and thus is a better technique compared to SGD or the Conventional Unconstrained approach. Do we have a convex loss function for those approaches then? Give Mathematical proof of your answer.

Lab

1) **Regularization**

Regularization is used to prevent overfitting and make the model more robust. To carry out regularization, a regularizer term is added to the objective/cost function: $\lambda/2(||U||^2 + ||V||^2)$. Here, the norm used is again the Frobenius norm which will also give uniqueness to the solution in terms of null space. Hence, now to minimize over this function, derive:

a) the new cost function

b) gradient

b) update rules

2) Movie Recommender system using Alternating Least Squares Method for Matrix Completion

a) Using the algorithm for Alternating Least Squares(ALS), develop a program for Matrix Factorization using ALS. Then run the program on the below sample Ratings Matrix (each cell of the table represents the rating on a 5-star scale given by the user to a movie. 0 indicates missing value) and compute the Factor Matrices U (latent vectors) and V(latent factors) for k = 1 and the predicted matrix assuming the low rank value of k=1,2, 3. Also compute the accuracy of the prediction.

The accuracy of the prediction can be measured by computing the residual matrix which is the $||R - UV^T||_F$. for the observed entries of R. More concisely, the error is generally characterized by the RMSE value which is given as -

$$\mathbf{RMSE}(\widehat{R}, S) = \sqrt{\frac{1}{|\Omega_S|}\sum_{(i,j)\in\Omega_S}\left(\widehat{R}_{ij} - S_{ij}\right)^2} = \left\|\frac{1}{|\Omega_S|}\mathcal{P}_{\Omega_S}(\widehat{R} - S)\right\|_F.$$

where S is the predicted matrix.

What do u observe? Do you think rounding of the final estimate of low rank matrix obtained UV^T will improve the prediction in terms of RMSE?

Movies				
Users	1	2	3	4
	0	2	3	4
	0	0	3	4
	0	0	0	4

b) Movie-Lens is a website for personalized movie recommendations. MovieLens dataset can be downloaded from http://files.grouplens.org/datasets/movielens/ml-1m.zip This file contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000. Each entry is in discrete number domain {1, 2, 3, 4, 5}. Modify the program to predict the missing ratings of the users for the movies they have not rated. You may assume that there are 18 genres.
Calculate the RMSE value for the predicted ratings and rounded predicted ratings for different rank k values and plot them. Explain what you observe.

c) In many real world datasets, the value of k is not known in advance. However, to build a robust recommendation engine, we would like to get an optimal value of k which is as minimum as possible and at the same time gives the least RMSE for the prediction of the missing values in the ratings matrix. Can you devise a method to find the optimal number of latent factors or the low rank(k) value for the above dataset?

3) Incorporating User and Item Bias.

To make our solutions more general, let's say we incorporate a bias. So here we simply attach a bias term for every user and for every movie separately. The bias for the movie is high if that movie is liked more. High bias value for a user indicates if the user is generous enough to give points or not.

a) Write the formula for the prediction of movie j by user i.

b) What change can we do in the matrix U and V to make this type of problem similar to all previous approaches that we have seen.

c) What does the new error function look like?

d) Is this a type of constraint problem? Explain.

e) What is the new update rule?

Appendix

Warm Up solutions

1.

```
Algorithm SGD(Ratings Matrix: R, Learning Rate: $\alpha$)
begin
  Randomly initialize matrices U and V ;
  S = {(i, j) : $r_{ij}$ is observed};
  while not(convergence) do
  begin
    Randomly shuffle observed entries in S;
    for each (i, j) $\in$ S in shuffled order do
    begin
      $e_{ij} \leftarrow r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}$
      for each q $\in$ {1 . . .k} do $u_{iq}^{+} \leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$
      for each q $\in$ {1 . . .k} do $v_{jq}^{+} \leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$
      for each q $\in$ {1 . . .k} do $u_{iq} = u_{iq}^{+} \; and \; v_{jq} = v_{jq}^{+}$
    end
    Check convergence condition;
  end
end
```

(Taken from Charu C. Aggarwal, Recommender Systems pg 100)

2. The Matrix factorization of a matrix $R_{(mxn)}$ into UV^T is not necessarily unique. It is possible that the factorization results in U and V matrices having negative entries. Hence, the factorization of the Ratings matrix containing 5-star ratings although gives us two matrices of dimensions m x k and n x k but we cannot directly parse them as the User-Genre matrix and Movie-Genre matrix or either of them can have negative entries and hence we cannot just use the V matrix to compute the average rating per genre.

3. We will use the conventional matrix factorization techniques when the data/ratings matrix is small. This way, calculating gradient with respect to every user and item simultaneously over every epoch is computationally less expensive. In case of a large matrix, we will use sgd. It is a tradeoff between rate of convergence and the amount of training or the smoothness of the descent.

4. Computing ALS is better because here we are optimizing a convex solution. This is not true for SGD approach or even the conventional unconstrained approach. To prove this, we know that the loss function will be quadratic or a linear term in ALS. However, in other approaches, since the features being parametrized are part of both user as well as item vectors, thus, if the features being optimized are f_1 and f_2. For example, there will be a quadratic term of f_1f_2 feature. Hence, it is a square of a non-linear product of features.

Other way to prove that, approaches like SGD optimize over a non-convex function is:

Let's say, $U=U_0$ and $V=V_o$ are solutions of min|| $R-UV^T$||. Now, if we multiply an orthogonal square matrix B to U_0 and V_0 , we still get the same answer. That is:

$(U_oB)(V_OB)^T = U_0BB^TV_0^T = U_0V_o^T$ as $BB^T = I$, since B is orthogonal square matrix

Lab solutions

Solution 1)

a)

$$Minimize\ J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

$$= \frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

b)

The new gradients that we now have are:

$$\frac{\partial J}{\partial u_{iq}} = \sum_{j:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)\left(-v_{jq}\right) + \lambda u_{iq}\ \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$= \sum_{j:(i,j)\in S}\left(e_{ij}\right)\left(-v_{jq}\right) + \lambda u_{iq}\ \ \forall i \in \{1 \dots m\}, q \in \{1 \dots k\}$$

$$\frac{\partial J}{\partial v_{jq}} = \sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)\left(-u_{iq}\right) + \lambda v_{jq}\ \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

$$= \sum_{i:(i,j)\in S}\left(e_{ij}\right)\left(-u_{iq}\right) + \lambda v_{jq}\ \ \forall j \in \{1 \dots n\}, q \in \{1 \dots k\}$$

c)

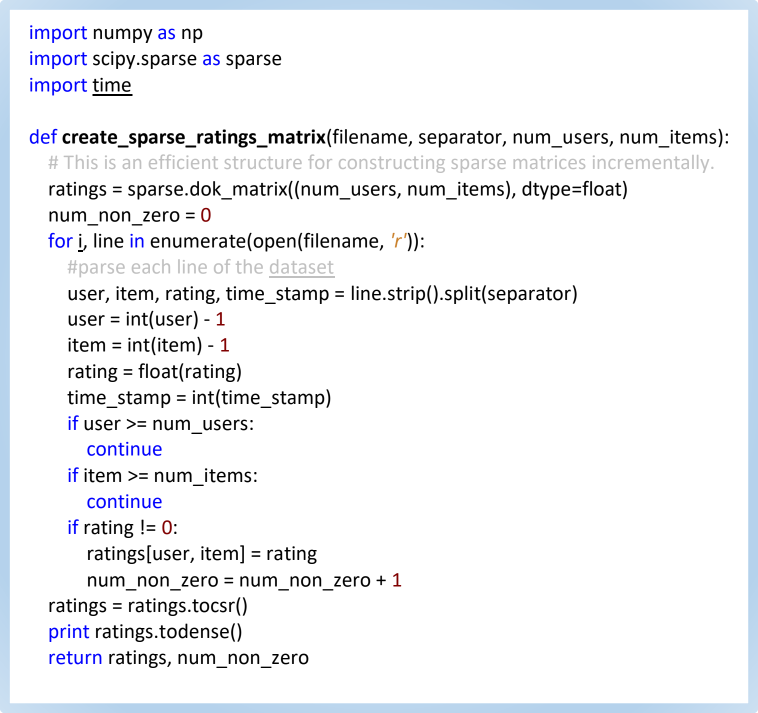
$$u_{iq} \leftarrow u_{iq} + \alpha\left(\sum_{j:(i,j)\epsilon S} e_{ij}\cdot v_{jq} - \lambda\cdot u_{iq}\right)\ \ \forall q \in \{1 \dots \mathrm{k}\}$$

$$v_{jq} \leftarrow v_{jq} + \alpha\left(\sum_{j:(i,j)\epsilon S} e_{ij}\cdot u_{iq} - \lambda\cdot v_{jq}\right)\ \ \forall q \in \{1 \dots \mathrm{k}\}$$

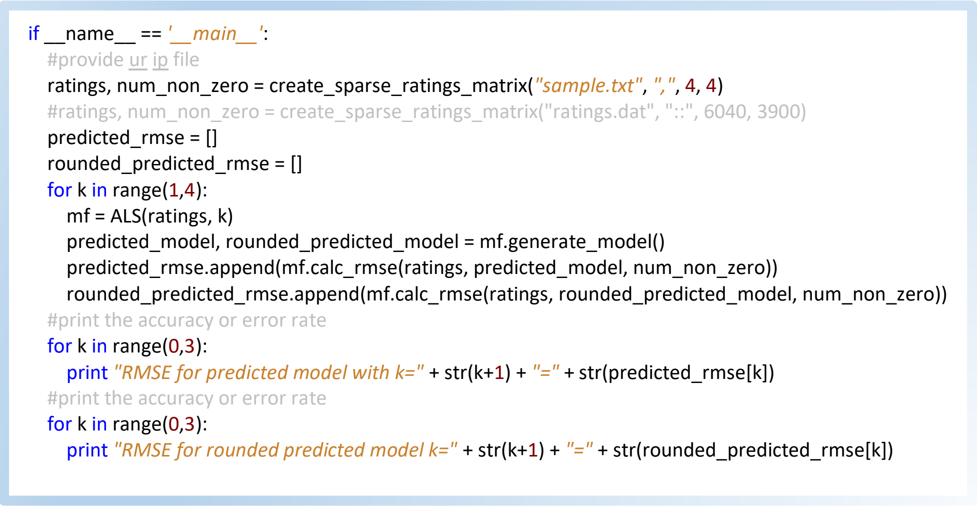
Solution 2)

a. Python code for Matrix completion that we developed is listed below

```
class ALS():
    def __init__(self, ratings, num_factors, num_iterations=5):
        self.ratings = ratings
        self.num_users = ratings.shape[0]
        self.num_items = ratings.shape[1]
        self.num_factors = num_factors
        self.num_iterations = num_iterations


    def generate_model(self):
        self.user_vectors = np.random.normal(size=(self.num_users,
                                               self.num_factors))
        self.item_vectors = np.random.normal(size=(self.num_items,
                                               self.num_factors))

        for i in xrange(self.num_iterations):
            print 'Solving for user vectors...'
            self.user_vectors = self.iteration(True, sparse.csr_matrix(self.item_vectors))
            print 'Solving for item vectors...'
            self.item_vectors = self.iteration(False, sparse.csr_matrix(self.user_vectors))

        predicted_model = self.user_vectors.dot(self.item_vectors.T)
        rounded_predicted_model = self.user_vectors.dot(self.item_vectors.T)

        for i in xrange(predicted_model.shape[0]):
            for j in xrange(predicted_model.shape[1]):
                rounded_predicted_model[i, j] = round(predicted_model[i,j])

        print "user vectors"
        print(self.user_vectors)
        print "item vectors"
        print(self.item_vectors)
        print "predicted model"
        print(predicted_model)
        print "rounded predicted model"
        print(rounded_predicted_model)
        return predicted_model, rounded_predicted_model

    def iteration(self, user, fixed_vecs):
        # m(users) - num_solve n(items)
        num_solve = self.num_users if user else self.num_items
        #create a holder of approx U or V  dim will be = m * k or n * k
        vec_holder = np.zeros((num_solve, self.num_factors))
        for i in xrange(num_solve):
            if user:
                #ratings[i] gives the ith row of the matrix i.e 1 users-ratings Ri.
                ratings_i = self.ratings[i].toarray()
            else:
                #ratings[:i] gives the ith col of the matrix i.e 1 users-ratings R.j
                ratings_i = self.ratings[:, i].T.toarray()
            #solve the linear regression problem to find
            ls_soln, residuals, rank, s = np.linalg.lstsq(fixed_vecs.todense(), ratings_i.T)
            vec_holder[i] = ls_soln.T
        return vec_holder

    def calc_rmse(self, orig_model, predicted_model, num_non_zero):
        residue = 0.0
        for i in xrange(predicted_model.shape[0]):
            for j in xrange(predicted_model.shape[1]):
                if ratings[i, j] != 0:
                    residue = residue + pow((ratings[i, j] - predicted_model[i,j]),2)
        return pow(residue/num_non_zero, 0.5)
```



```
import numpy as np
import scipy.sparse as sparse
import time

def create_sparse_ratings_matrix(filename, separator, num_users, num_items):
   # This is an efficient structure for constructing sparse matrices incrementally.
   ratings = sparse.dok_matrix((num_users, num_items), dtype=float)
   num_non_zero = 0
   for i, line in enumerate(open(filename, 'r')):
      #parse each line of the dataset
      user, item, rating, time_stamp = line.strip().split(separator)
      user = int(user) - 1
      item = int(item) - 1
      rating = float(rating)
      time_stamp = int(time_stamp)
      if user >= num_users:
         continue
      if item >= num_items:
         continue
      if rating != 0:
         ratings[user, item] = rating
         num_non_zero = num_non_zero + 1
   ratings = ratings.tocsr()
   print ratings.todense()
   return ratings, num_non_zero
```



```
if __name__ == '__main__':
   #provide ur ip file
   ratings, num_non_zero = create_sparse_ratings_matrix("sample.txt", ",", 4, 4)
   #ratings, num_non_zero = create_sparse_ratings_matrix("ratings.dat", "::", 6040, 3900)
   predicted_rmse = []
   rounded_predicted_rmse = []
   for k in range(1,4):
      mf = ALS(ratings, k)
      predicted_model, rounded_predicted_model = mf.generate_model()
      predicted_rmse.append(mf.calc_rmse(ratings, predicted_model, num_non_zero))
      rounded_predicted_rmse.append(mf.calc_rmse(ratings, rounded_predicted_model, num_non_zero))
   #print the accuracy or error rate
   for k in range(0,3):
      print "RMSE for predicted model with k=" + str(k+1) + "=" + str(predicted_rmse[k])
   #print the accuracy or error rate
   for k in range(0,3):
      print "RMSE for rounded predicted model k=" + str(k+1) + "=" + str(rounded_predicted_rmse[k])
```

K = 1, the factor matrices are

user vectors U =

```
[[ 0.93956699]
 [ 0.92924724]
 [ 0.8471448 ]
 [ 0.57871415]]
```

item vectors V =

```
[[ 0.3356974 ]
 [ 1.33541533]
 [ 2.91115078]
 [ 4.70860829]]
```

predicted model =

```
[[ 0.3154102   1.25471216  2.73522117  4.42405291]
 [ 0.31194588  1.240931    2.70517881  4.37546123]
 [ 0.28438431  1.13129016  2.46616625  3.98887304]
 [ 0.19427284  0.77282375  1.68472415  2.72493825]]
```

K = 2

Rounded predicted model=

```
[[ 0.  2.  3.  4.]
 [ 0.  2.  3.  4.]
 [ 0.  1.  2.  4.]
 [-0. -0.  0.  4.]]
```

K = 3

Rounded predicted model=

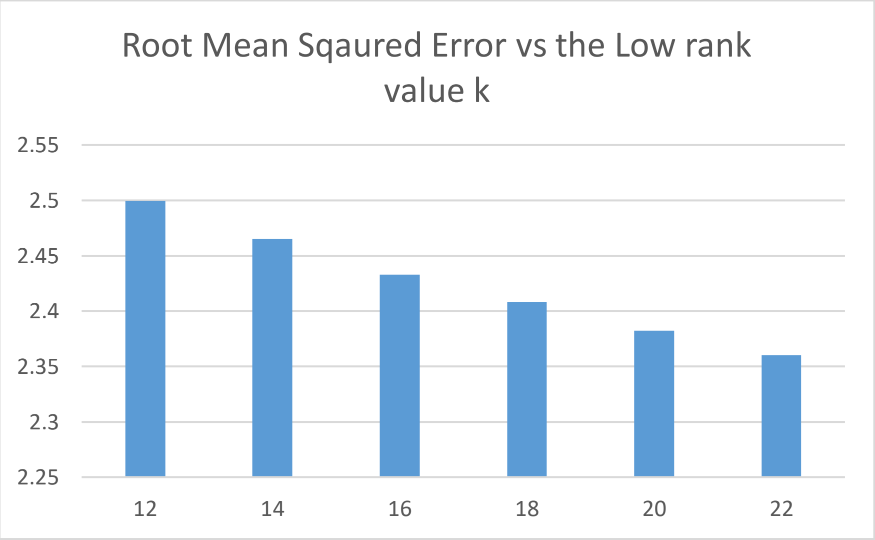
```
[[ 1.  2.  3.  4.]
 [ 0.  2.  3.  4.]
 [-0.  0.  3.  4.]
 [ 0. -0.  0.  4.]]
```

The RMSE values computed are as as follows

```
RMSE for predicted model with k=1=0.631585862695
RMSE for predicted model with k=2=0.349070211322
RMSE for predicted model with k=3=0.148240302149
RMSE for rounded predicted model k=1=0.707106781187
RMSE for rounded predicted model k=2=0.4472135955
RMSE for rounded predicted model k=3=0.0
```

From the RMSE values and the predicted (rounded) model we observe that rounding of the predicted model need not always improve the accuracy. We note for k=1 and 2, the rounded matrix has a higher RMSE and hence less accurate. Whereas for k=3, the RMSE value of the rounded predicted matrix is zero hence very accurate. But this might also lead to overfitting. One other implementation is we round off the prediction if its distance to the nearest integer is less than or equal to 0.1. It has been observed that this approach can improve the performance of the recommendation system.

b.



It is observed that the value of RMSE continuously decreases with the value of K which is varied from 12 to 18. This is expected because even though we know the dataset has 18 genres. It will continue to provide a better model at the cost of overfitting.

c)

Divide the data set of around 1M ratings using a 80%-20% split and use the former as training set and the latter as the test set. Further divide the former using a 80%-20% split to get the training and validation sets respectively.

Repeatedly generate model in the training set by selecting different values for low rank k starting with some low rank value like 4 and compute RMSE values on the validation set.

Pick the value of k and the corresponding model which gives the minimum RMSE. This is one way of computing the optimal k

Use this model to compute the RMSE on the test set.

Solution 3)

a) Thus the prediction by user i for movie j is :

$$\widehat{r_{ij}} = o_i + p_j + \sum_{s=1}^{k} u_{is} \cdot v_{js}$$

b) Although this looks slightly different but it is almost the same thing. We just need to modify our U and V matrices. U is expanded as m*(k+2) matrix and V as n*(k+2) matrix. Now we need set the following values in the matrix:

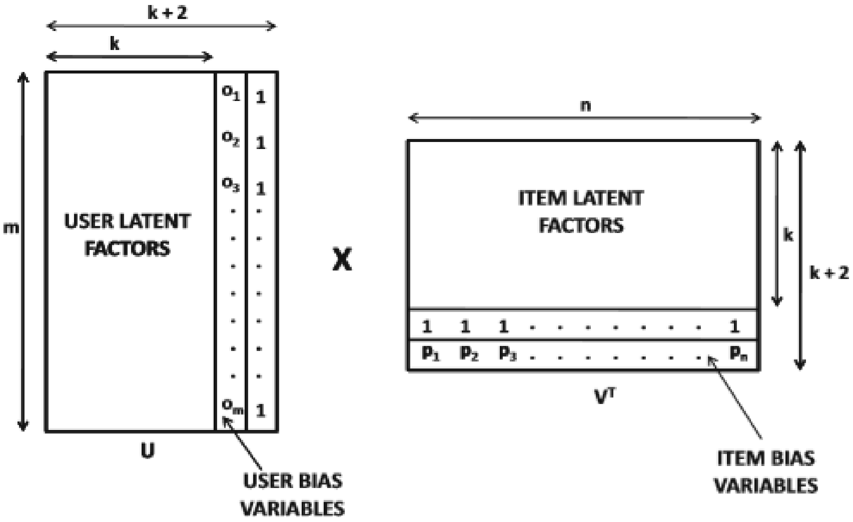
$u_{i,k+1} = o_i \quad \forall i \in \{1 \dots m\}$

$u_{i,k+2} = 1 \quad \forall i \in \{1 \dots m\}$

$v_{j,k+1} = 1 \quad \forall j \in \{1 \dots n\}$

$v_{j,k+2} = p_j \quad \forall i \in \{1 \dots n\}$

Thus, U and V matrices now look like:



c)

$$Minimize\ J = \frac{1}{2} \sum_{(i,j)\epsilon S} \left(r_{ij} - \sum_{s=1}^{k+2} u_{is} \cdot v_{js} \right)^2 + \frac{\lambda}{2} \sum_{s=1}^{k+2} \left(\sum_{i=1}^{m} u_{is}^2 + \sum_{j=1}^{n} v_{js}^2 \right)$$

Subject to:

(k+2)th column of U contains only 1s

(k+1)th column of V contains only 1s

d)

Since we want the (k+2)th column of U and k+1th column of V always set to 1 , hence, this is a constraint problem. This is also very simple to do. Just follow the previous update versions but after updating all values, specifically update these values back to 1. A more efficient method would be to keep a track of what we are updating and never update values of these columns.

e)

$$u_{iq} \Leftarrow u_{iq} + \alpha\left(e_{ij} \cdot v_{jq} - \lambda \cdot u_{iq}\right) \forall q \in \{1 \dots k+2\}$$

$$v_{jq} \Leftarrow v_{jq} + \alpha\left(e_{ij} \cdot u_{iq} - \lambda \cdot v_{jq}\right) \forall q \in \{1 \dots k+2\}$$

Reset perturbed entried in (k+2)th column of U and (k+1)th column of V to 1's

References

[1] Recommender Systems by Charu C. Aggarwal

[2] https://www.coursera.org/learn/machine-learning

[3] https://www.quora.com/What-is-the-Alternating-Least-Squares-method-in-recommendation-systems

[4] http://yifanhu.net/PUB/cf.pdf

[5] http://www.cs.cmu.edu/~akshaykr/files/matrix_norms.pdf

[6] https://pdfs.semanticscholar.org/2535/3bc78453da76e43e199a925ab54457e18ca5.pdf

[7] https://papers.nips.cc/paper/2655-maximum-margin-matrix-factorization.pdf

[8] https://stanford.edu/~rezab/papers/fastals.pdf

[9] http://www.aaai.org/Papers/ICML/2003/ICML03-094.pdf

[10] http://www.cs.rochester.edu/twiki/pub/Main/HarpSeminar/Factorization_Meets_the_Neighborhood-_a_Multifaceted_Collaborative_Filtering_Model.pdf

[11] https://www.coursera.org/learn/ml-foundations/lecture/ejs8C/recommendations-from-known-user-item-features

[12] GroupLens MovieLens 1M dataset - http://grouplens.org/datasets/movielens/.

[13] GroupLens MovieLens 1M dataset - http://files.grouplens.org/datasets/movielens/ ml-1m-README.txt.

All diagrams and formulae have been referenced from the following book -

Recommender Systems by Charu C. Aggarwal