# Project report of Movie recommendation system

Sravanthi Kapu (skapu001@odu.edu)
Old Dominion University

## Introduction to Recommendation systems:

**Recommendation systems:** Recommendation systems can be termed as sub class of information filtering systems which are used to present information on items or products that are likely to be of interest to the user. Many ecommerce websites are using recommendation systems in order to predict recommendations and suggest user the items/ products which the user may be interested so they can increase their sales, so it has become part of social networking, retail and entertainment industries. Websites like Netflix, YouTube, shopping and music websites, twitter pages, etc. use recommendation systems.

**Types of Recommendations:** recommendations regarding the products user is interested in can be done in various ways. They are [1]:

1. Collaborative recommender system
2. Content based recommender system
3. Utility Based recommender system
4. Knowledge based recommender system
5. Hybrid recommender system

Recommendations are made based on the data related to users. The data used for recommendations can be categorized into two types:

1. **Implicit data:** is data we gather from the user's behaviour, with no ratings or specific actions needed. It could be what items a user purchased, number of times they played a song or watched a movie, how long they've spent reading a specific article etc.[2]
2. **Explicit data:** is data where we have some sort of rating. Here we know how much a user likes or dislikes an item. But here the user should provide few of the rating. Ex: 1 to 5 ratings from the Movie Lens or Netflix dataset, etc.[2]

## Project Idea and Implementation:

**Movie recommendation system:** The idea is to build a recommendation system which recommends movies to a user, based on other user's previous ratings. Implementation to find the movies which user might be interested will be done by using collaborative filtering using Alternate least Squares Algorithm.

**Collaborative Filtering:** based on the commonalities between the ratings of the users, new ratings are generated using inter-user comparisons. The new ratings are generated using matrix factorization.

**Matrix Factorization:** All the ratings of all the users is taken as, sparse, R matrix. Now it is considered as the product of two smaller matrices. By fixing the matrices, we get closer to the actual rating and thus how less the error value, that accurate the predictions. We can consider them a user feature matrix and movie feature matrix. This approximation is also going to smooth out the zeros and in the process give us our predicted ratings.

Matrix factorization can be done using many algorithms. In this project Alternate least Squares is used to estimate the latent factors.

**Alternate Least Squares Algorithm:**

The matrix R is considered to be the product of two matrices P and Q. So the matrix factorization is done alternatively i.e. by fixing P, Q value is estimated and similarly by fixing Q, P value is estimated. After enough number of iterations, we are aiming to reach a convergence point where either the matrices P and Q are no longer changing or the change is quite small and then the product of the two matrices gives the predicted ratings.

**Dataset:** Movielens data set is used. The files, ratings and movies are used.

The ratings.txt file (Size: 24.4 MB) contains ratings given by 6040 users for 3706 movies.

The movies.txt file contains the names and movie ids of the movies.

**Implementation details:**

**Step 1:** the files ratings.txt is opened and then the data is stored in a RDD. The data is then converted into matrix format.

**Step 2:** Two new matrices "userfmat" and "moviefmat" are generated with size M*K and N*K. where M is number of users i.e. 6040, N is number of movies i.e. 3706 and K is hidden latent factor, here K=15.

The two matrices are stored as RDD's of vectors.

**Step 3:** The matrices are estimated by using alternate least squares algorithm. By fixing one matrix, the other is estimated and vice versa.

The equation to find either of them is: (If we consider two matrices moviefmat and userfmat be X and Y) [3]:

We assume Y is fixed and solve for X: $X_u = (Y^T Y + \lambda I)^{-1} Y^T r_u$

Then, we assume X is fixed and solve for Y: $Y_u = (X^T X + \lambda I)^{-1} X^T r_u$

LAMBDA (λ) is regularization constant, it's assumed as 0.001.

The functions are called accordingly to estimate the matrices inside the loop

Since the operations in calculating the matrix are independent, there is possibility of parallelization.

**Parallelization:** the elements of the collection are copied to form a distributed dataset that can be operated on in parallel [4].

The matrix which is to be estimated is parallelized and the other matrix which is assumed to be fixed is broadcasted.

**Broadcasting**: Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner [4].

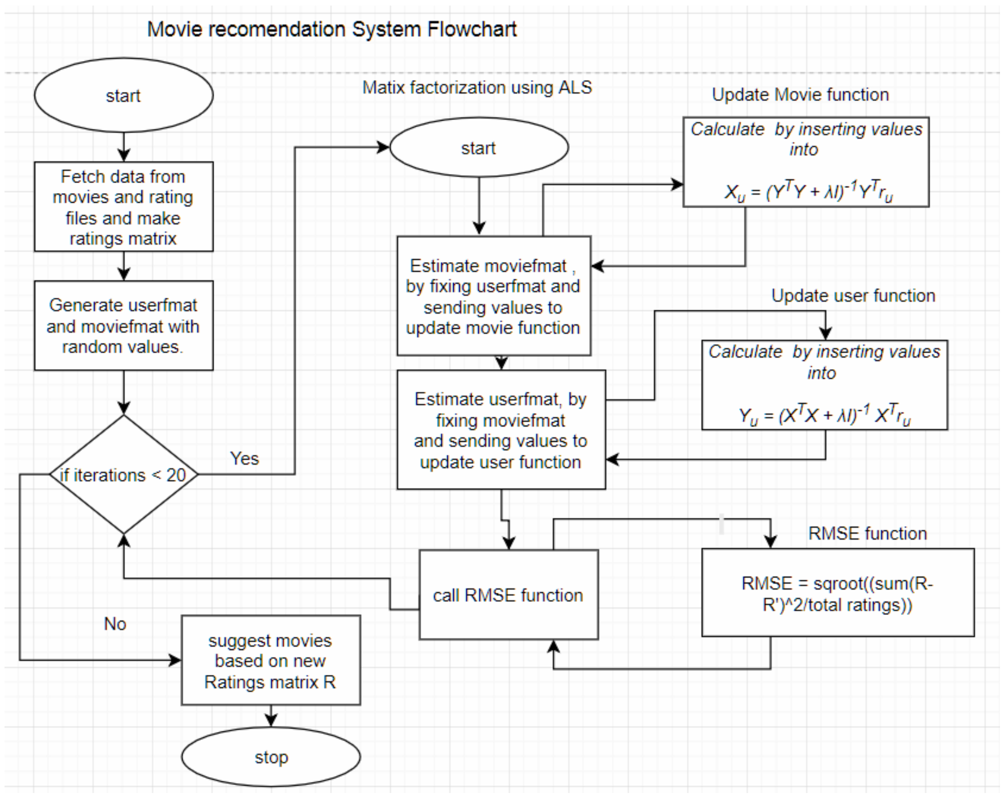
Thus, this reduces the time of execution.

**Step 4:** Calculate the RMSE value. The Root mean Square error is calculated by RMSE formula [5]: $\sqrt{(\sum(R-R')^2)/total\ ratings}$

The steps 3 and 4 are repeated for 20 iterations, until we get less RMSE value.

**Step 4:** Now, once the two matrices are estimated, the product of the two matrices is calculated and that is our new ratings matrix. By comparing the zero terms in the actual matrix and the ratings which are above 3 are predicted to the user. The results are written to an output file recommendations.txt

**Flow chart:**

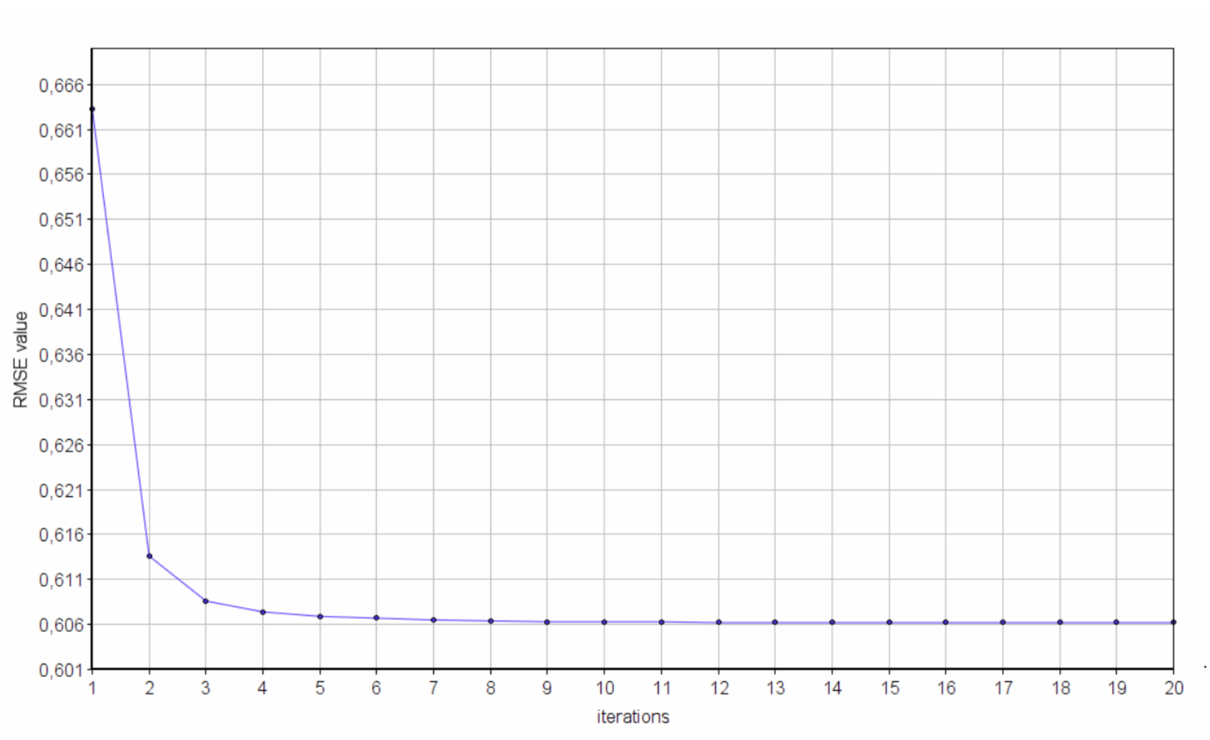

Movie recomendation System Flowchart
start
Fetch data from movies and rating files and make ratings matrix
Generate userfmat and moviefmat with random values.
if iterations < 20
Yes
No
suggest movies based on new Ratings matrix R
stop
Matix factorization using ALS
start
Estimate moviefmat , by fixing userfmat and sending values to update movie function
Estimate userfmat, by fixing moviefmat and sending values to update user function
call RMSE function
Update Movie function
Calculate by inserting values into
$X_u = (Y^TY + \lambda I)^{-1}Y^Tr_u$
Update user function
Calculate by inserting values into
$Y_u = (X^TX + \lambda I)^{-1} X^Tr_u$
RMSE function
RMSE = sqroot((sum(R-R')^2/total ratings))

**Discussion of results:**

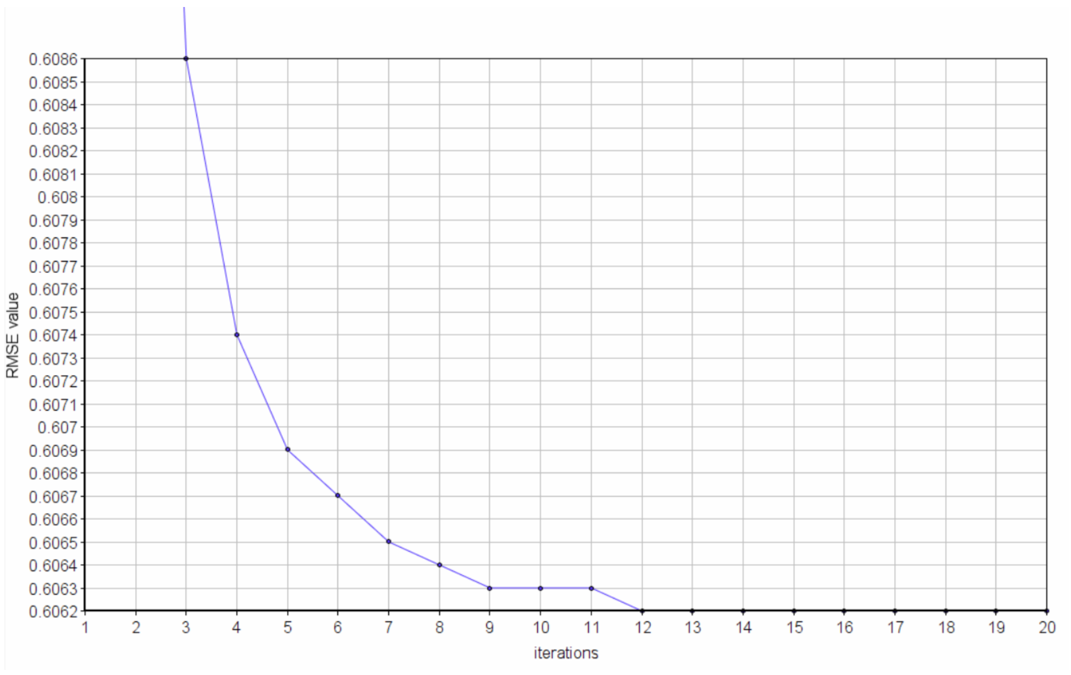
**Graph:** RMSE value Vs. Iterations

Below is the graph, the RMSE values plotted for the 20 iterations on a sample run.

As the RMSE value is reducing, it indicates the two matrices are converging close to generate Actual rating.



Zoomed graph for the iterations 3 to 20:



1. The running time of the program is 7.44 min (best of 3 runs).

2. The predicted ratings for every user are written to recoommendations.txt file. The output is in the format: Movie title, Movie ID and predicted rating for every user.

3. The operations performed on RDD's are parallelized, so the program can be used for large data and when used in a large cluster, by changing the partition's number intelligently, faster execution can be achieved for large datasets also.

**References:**

1. https://www.bluepiit.com/blog/classifying-recommender-systems/
2. https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe
3. http://danielnee.com/2016/09/collaborative-filtering-using-alternating-least-squares/
4. https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html#parallelized-collections
5. https://www.coursera.org/learn/recommender-metrics/lecture/LGLb3/prediction-accuracy-metrics
6. https://www.codementor.io/jadianes/building-a-recommender-with-apache-spark-python-example-app-part1-du1083qbw
7. https://cognitiveclass.ai/blog/nested-lists-multidimensional-numpy-arrays/
8. https://stackoverflow.com/questions/17028329/python-create-a-pivot-table?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa