Capstone1: MovieLens project

Louis Mono

March 2019

Contents

Abstract

Recommender systems are one of the most successful and widespread application of machine learning technologies in business. There are a subsclass of information filtering system that seek to predict the "rating" or "preference" a user would give to an item. One of the most famous success story of the recommender system is the Netflix competition launched on october 2006. In 2009, at the end of the challenge , Netflix awarded a one million dollar prize to a developer team for an algorithm that increased the accuracy of the company's recommendation engine by 10%. Many well-known recommendation algorithms, such as latent factor models, were popularized by the Netflix contest. The Netflix prize contest is become notable for its numerous contributions to the data science community. With the 10M version of movielens data , and following some ML techniques that went into the winning solution for the Netflix competition, our findings suggest that Linear regression model with regularized movie and user effects, and recommender engine using Matrix factorization with Stochastic Gradient descent lead us to the smallest Root Mean Squared Error(RMSE), 0.8648170 and 0.7826226 ,respectively.

1 Introduction

According to Aggarwal(2016) , the recommendation problem may be formulated in various ways, among which the two main are as follows: The first approach , the "prediction version of problem" aims to predict the rating value for a user-item combination. It is also referred to as "matrix completion problem". The second approach, the "ranking version of problem" seeks to recommend the top-k items for a particular user, or determine the top-k users to target for a particular item.

For the movielens project, we use the 10M version of the MovieLens dataset generated by the GroupLens research lab. We aim to create our own recommendation system using the "prediction version of problem". We are going to train our algorithms using the inputs in edx set to predict movie ratings in the validation set. RMSE will be used to evaluate how close our predictions are to the true values in the validation set.This approach will be studied and the features processed and analyzed with different Machine Learning techniques, focusing on regression models, ensemble methods (gradient boosting, random forest) and recommender engines (slopeOne, matrix factorization with gradient descent) following the main steps of a data mining problem as described in Ricci et al (2015).

2 Dataset and executive summary

2.1 Overview

```
 "edx set"

Observations: 9,000,055
Variables: 6
$ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
$ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
$ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
$ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
$ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
$ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|D...

 "validation set"

Observations: 999,999
Variables: 6
$ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
$ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
$ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
$ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
$ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
$ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Child...
```

The edx set is made of 6 features for a total of about 9,000,055 observations while the validation set which represents 10% of the 10M Movielens dataset contains the same features , but with a total of 999,999 occurences. we made sure that userId and movieId in edx set are also in validation set. Each row represents a rating given by one user to one movie. The column "rating" is the outcome we want to predict, y.

Taking into account both datasets, here are the features and their characteristics:

quantitative features

-*userId* : discrete, Unique ID for the user.

-*movieId*: discrete, Unique ID for the movie.

-*timestamp* : discrete , Date and time the rating was given.

qualitative features

-*title*: nominal , movie title (not unique)

-*genres*: nominal, genres associated with the movie.

outcome,y

-*rating* : continuous, a rating between 0 and 5 for the movie.

2.2 Data exploration

The Data exploration step disclosed the following findings :

- ***Outcome(rating)*** : the average user's activity reveals that no user gives 0 as rating and half star ratings are less common than whole star ratings. The top 5 ratings from most to least are : 4, 3, 5, 3.5 and 2.

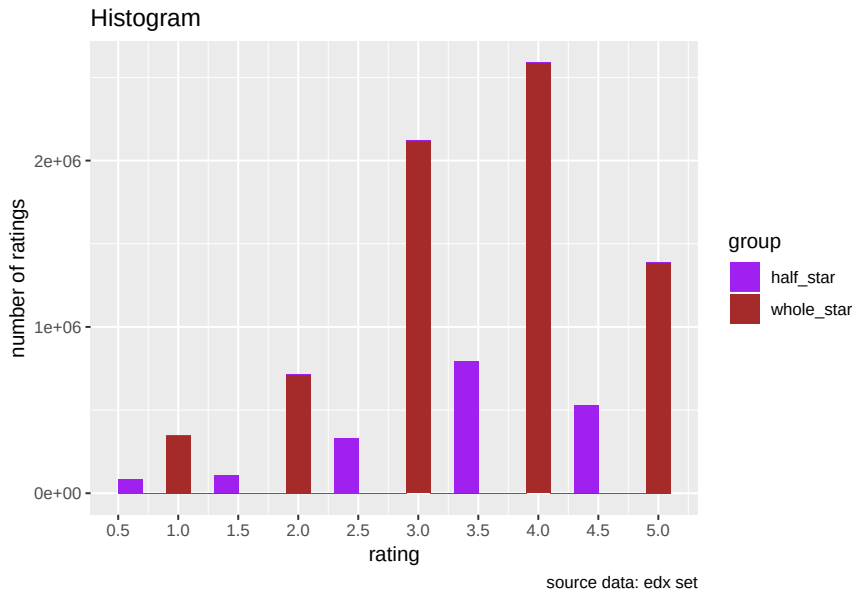

Figure 1: Number of ratings for each rating

- ***qualitative features(genres,title)*** : visual exploration shows a strong evidence of a genre effect (Figure 2).Movies which have the highest number of ratings are in the top genres categories : Drama, Comedy or Action (see details in the Rmd file report)

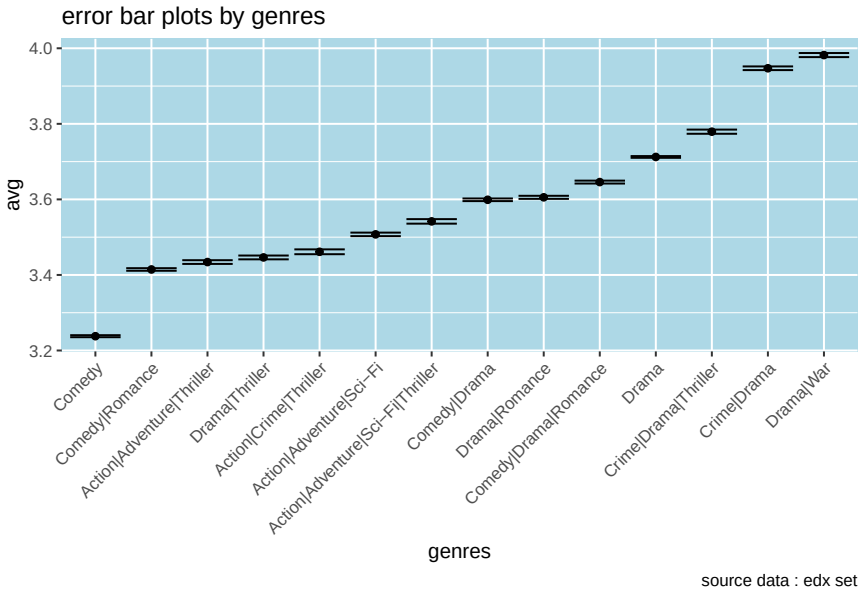

Figure 2: error bar plots by genres

- ***quantitative features(movieId,userId,timestamp)*** : There's presence of movies effects and users effects, but also time effects ; some movies get rated more than others, and some users are more active than others at rating movies (Figure 3). The trend of the average ratings versus the date(timestamp in the datetime format with weekly time unit) shows some evidence of time effect (Figure 3). Since the number of uniques values for the userId is 69878 and for the movieId 10677 (that means that there are less movies provided for ratings than users that rated them), if we think in terms of a large matrix with user on the rows and movies on the columns, the challenges we face are the sparsity of our matrix and the curse of dimensionality.

n	**distinct_values**
users	**69878**
movies	**10677**

Table 1: unique values for userId and movieId

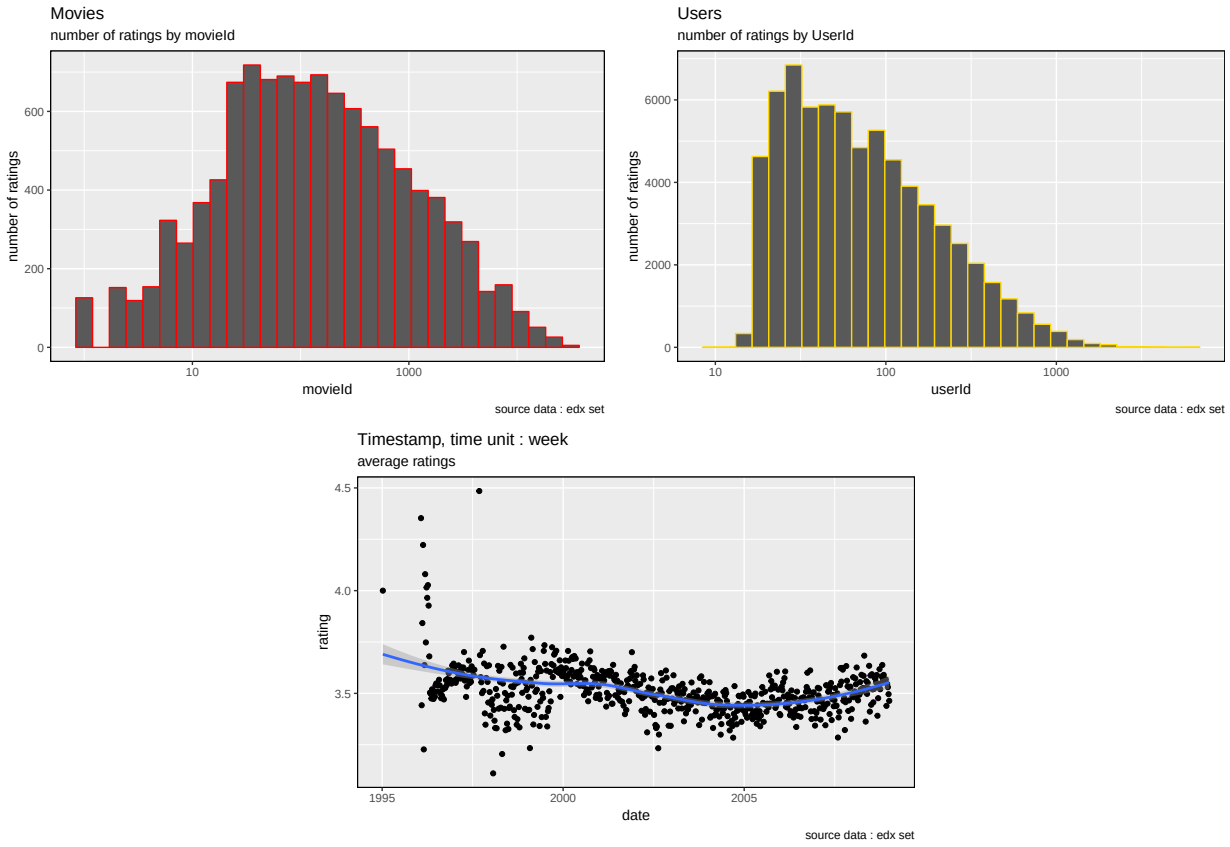

Figure 3: Movie,user and time effects

3 Data Preprocessing

Real-life data typically needs to be preprocessed (e.g. cleansed, filtered, transformed) in order to be used by the machine learning techniques in the analysis step.

In this section, we focused mainly on data preprocessing techniques that are of particular importance when designing a Recommender system. These techniques include similarity measures (such as Euclidean distance, Cosine distance,etc) , sampling methods , and dimensionality reduction (such as PCA or SVD). We will use them when necessary.

First avoid, we are going to create the rating matrix . In the Data exploration step, we already higlighted the sparsity problem when considering a large matrix with users on the rows and movies on the columns. Let's build effectively that matrix.

3.1 Data transformation

We used the SparseMatrix function of the Matrix package to consider that large matrix with users on the rows and movies on the column . In order to perform our transformation, we needed in a first step to encode

the movieId and userId vectors as categories with the `as.factor()` function , and in a second step to convert userId & movieId into `numeric vectors` . These transformations were made on an edx.copy set since we wanted to keep unchanged our original training(edx) set (see code details in the rmd file report). The rating matrix() is a 69878 x 10677 rating matrix of class 'realRatingMatrix' with 9,000,055 ratings. Here , a look on the first 10 users of our sparse matrix.

```
10 x 10 sparse Matrix of class "dgCMatrix"

u1  . .   . . . . . . . .
u2  . .   . . . . . . . .
u3  . .   . . . . . . . .
u4  . .   . . . . . . . .
u5  1 .   . . . . 3 . . .
u6  . .   . . . . . . . .
u7  . .   . . . . . . . .
u8  . 2.5 . . 3 4 . . . .
u9  . .   . . . . . . . .
u10 . .   . . . . 3 . . .
```

3.2 Similarity measures

More often, data mining techniques that are used for the modelling of different recommender systems approaches(collaborative filtering, content based, hybrid methods) are highly dependent on defining an appropriate similarity or distance measure.

To measure the similarity between users or between items, it is possible to use the following:

```
+ Minkowski Distance
+ Mahalanobis distance
+ Pearson correlation
+ Cosine similarity
```

we will use the cosine similarity which is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. According to Ricci et al(2015), it is defined as follow :

$$\cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}|| \cdot ||\boldsymbol{y}||}$$

where $\cdot$ indicates vector dot product and ||x|| is the norm of vector x .

The main advantages using this distance measure , as developed in Saranya et al (2016) are :

- Solves the problem of sparsity, scalability and cold start and it is more robust to noise.
- It improves prediction accuracy and consistency
- The Cosine similarity can still be calculated even though the matrix has many missing elements.
- As the dimensional space becomes large, this still works well.
- The low Computational complexity , especially for sparse vectors.

Because of our huge amount of data, we calculate our similarity on the first 50 users for the visualization.

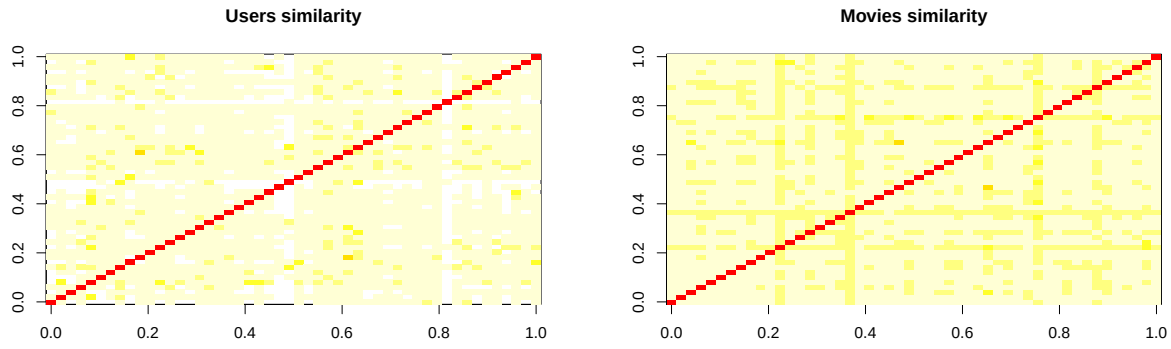

Figure 4: Users and Movies similarity

In the given matrices, each row and each column corresponds to a user, and each cell corresponds to the similarity between two users.The more red the cell is, the more similar two users are. Note that the diagonal is red, since it's comparing each user with itself.

When we observe the two similarity matrices, they leave out the following plausible analysis : since there are more similar ratings between certain users than others, and more similar ratings between certain movies than others, we can evidence the existence of a group of users pattern or a group of movies pattern.

3.3 Dimension reduction

The analysis of the previous similarity matrices leads to the thought that it can exists users with similar ratings patterns and movies with similar rating patterns. However Sparsity and the curse of dimensionality remain a recurent problem, and we have to deal with many NA too. Dimensionality reduction techniques such as "pca" and "svd" can help overcome these problem by transforming the original high-dimensional space into a lower-dimensionality.

To face the RAM memory problem, we are going to use the Irlba package, which it is a fast and memory-efficient way to compute a partial SVD. The augmented implicitly restarted Lanczos bidiagonalization algorithm (IRLBA) finds a few approximate largest (or, optionally, smallest) singular values and corresponding singular vectors of a sparse or dense matrix using a method of Baglama and Reichel

According to Irizarry,R 2018 *Matrix factorization*,github page,accessed 15 January 2019, https://rafalab.github.io/dsbook/matrix-factorization.html , the SVD tells us that we can decompose a $N * P$ matrix Y with $P < N$ as follow :

$$Y = UDV^T$$

, where

```
+ U : orthogonal matrix of dimensions N x m
+ D : diagonal matrix containing the singular values of the original matrix, m x m
+ V : orthogonal matrix of dimensions m x P
```

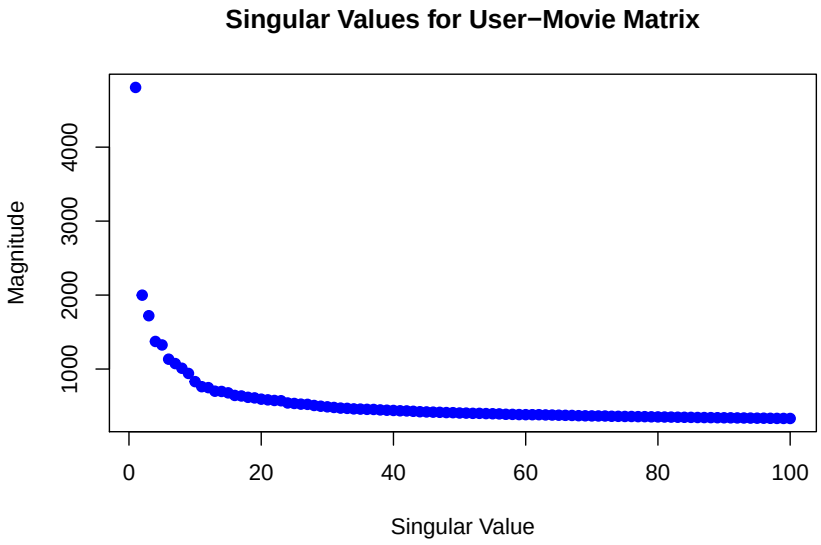

Figure 5: Singular values for User-Movie Matrix

singular values	**variability described***
first6	**0.6187623**
first12	**0.7052297**
first20	**0.7646435**

Table 2: variability described by first 6, 12, and 20 singular values
Note: * is obtained as sum of squares of the specific singular value over sum of squares of all singular values

First six singular values explain more than half of the variability of the imputed ratings matrix, with the first dozen explaining nearly 70% and the first twenty explaining more than 75%. However,the goal is to identify the first k singular values whose squares sum to at least 90% of the total of the sums of the squares of all of the singular values.

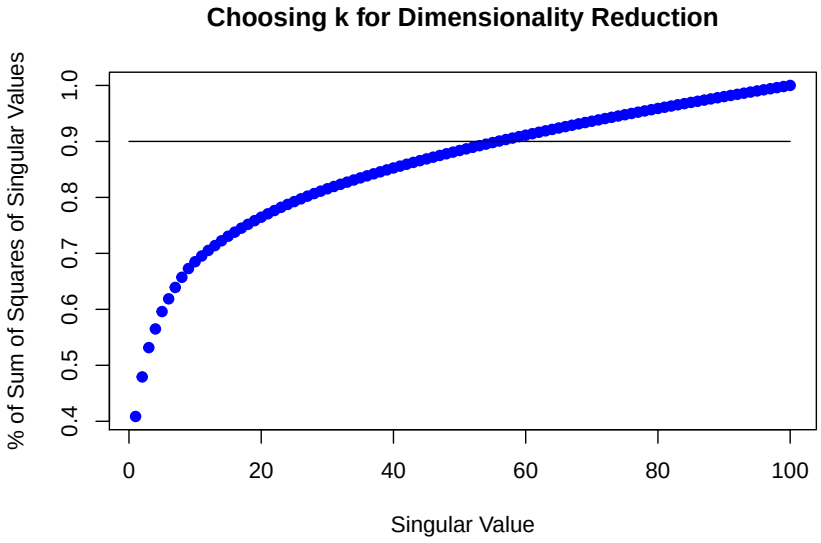

Figure 6: best k for dimensionality reduction

A plot of a running sum of squares for the singular values shows that the 90% hurdle is achieved using somewhere between 50 and 60 singular values. After calculating the length of the vector that remains from our running sum of squares after excluding any items within that vector that exceed 0.90 (see code details in the rmd file report), we noticed that k=55 will retain 90% of variability.

Y_Decomposition	**matrix_dimension**
U_k	**69878*55**
D_k	**55*55**
V_k	**55*10677**

Table 3: SVD decomposition of Y

As we can see above (table 3), we now have a matrix D_k of size 55 x 55, a matrix U_k of size 69878 x 55, and a matrix V_k of size 55 x 10677. Therefore, the total number of numeric values required to house these component matrices is $(69878 * 55) + (55 * 55) + (55 * 10677) = 4,433,550$. This represents an approximately 50.7% decrease in required storage relative to the original 9,000,055 entries.

Reducing the dimensionality, the RAM memory problem persisted. Thats why we needed to go further with another reduction technique. We selected the relevant data using the whole rating matrix.

3.4 Relevant Data

We know that some users saw more movies than the others. So, instead of displaying some random users and movies, we should select the most relevant users and movies. Thus we visualize only the users who have seen many movies and the movies that have been seen by many users.To identify and select the most relevant users and movies, we follow these steps:

a. Determine the minimum number of movies per user.
b. Determine the minimum number of users per movie.
c. Select the users and movies matching these criteria.

```
 "min_n_movies"

90%
301

 "min_n_users"

   90%
2150.2

 "matching criteria: ratings_movies"

6978 x 1068 rating matrix of class 'realRatingMatrix' with 2313148 ratings.
```

we can notice that now, we have a rating matrix of 6978 distinct users x 1068 distinct movies, with 2,313,148 ratings .

The data preprocessing phase is usually not definitive because it requires a lot of attention and subsequent various explorations on the variables. It must be aimed at obtaining better predictive results and in this sense, the further phases of model evaluations can help us to understand which particular preprocessing approaches are actually indispensable or useful for a specific model purpose.

4 Methods and Analysis

In this section, we are going to explain the methodology over differents Machine Learning algorithms we used , and present the metric for the model performance evaluation.

4.1 Evaluated Algorithms

4.1.1 Regression Models

- **Modelling effects**

As in Irizarry,R 2018 *Recommender systems*,github page,accessed 5 January 2019, https://rafalab.github.io/dsbook/recommendation-systems.html, we followed the same approach to build our linear regression models as the simplest possible recommendation systems. We started from considering the same rating for all movies and users with all the differences explained by random variation $Y_{u,i} = \mu + \varepsilon_{u,i}$ and thus, modelling successively the different effects.

movie effects : since we know that some movies are generally rated higher than others, we can augment our previous model by adding the term b_i to represent average ranking for movie i:

$$Y_{u,i} = \mu + b_i + \varepsilon_{u,i} (1)$$

where :

°μ the "true" rating for all movies

°b_i bs effects or bias , movie-specific effect.

$^{\circ}\varepsilon_{u,i}$ independent errors sampled from the same distribution centered at 0

movie + user effects : We also know that some users are more active than others at rating movies. This implies that an additional improvement to our model may be :

$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i} (2)$$

where :

$^{\circ}$ μ , b_i , $\varepsilon_{u,i}$ are defined as in (1)

$^{\circ}$ b_u user-specific effect

movie + user + time effects . As in data exploration we showed some evidence of time effect, if we define with $d_{u,i}$ as the day for user's u rating of movie i the new model is the following :

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i} (3)$$

with f a smooth function of $d_{u,i}$. A further and detailed explanations of time effects are developped in Koren(2009). For convenience in our study , we just converted timestamp to a datetime object and rounded it to the weekly unit.

movie + user + genre effects : The same approach with the genre effect leads to the following model ,

$$Y_{u,i} = \mu + b_i + b_u + \sum k = 1^K x_{u,i}\beta_k + \varepsilon_{u,i} (4)$$

where $g_{u,i}$ is defined as the genre for user's u and $x^k_{u,i} = 1$ if $g_{u,i}$ is genre k. We just present the model here, but we didn't perform it in our analysis.

To fit these models and get least square estimates of $\hat{b}_i$, $\hat{b}_u$, $\hat{b}_t$, using the **lm()** function of the stats package will be very slow since there are more than 10,000 of b_i effects and more that 60,000 b_u effects. For the different models (1),(2),(3) we get estimates of least squares as follow :

$^{\circ}$ $\hat{b}_i$ as the average of $y_{u,i} - \hat{\mu}$ for each movie i and the predicted ratings are : $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$ $(1')$

$^{\circ}$ $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$ and the predicted ratings are : $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u$ $(2')$

$^{\circ}$ $\hat{b}_t$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i - \hat{b}_u$ and the predicted ratings are : $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i + \hat{b}_u + \hat{b}_t$ $(3')$

- **Regularization**

Regularization permits us to penalize large estimates that come from small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions. The general idea is to add a penalty for large values of bi, bu to the sum of squares equation that we minimize. So having many large bi or bu makes it harder to minimize.

A more accurate estimation of bu and bi will treat them symmetrically, by solving the least squares problem

$$\frac{1}{N}\sum_{u,i}(y_{i,u} - \mu - b_i - b_u)^2 + \lambda\left(\sum_i b_i^2 + \sum_u b_u^2\right) (5)$$

where the first term , $\frac{1}{N}\sum_{u,i}(y_{i,u} - \mu - b_i - b_u)^2$, strives to find bu's and bi's that fit the given ratings. The regularizing term, $\lambda\left(\sum_i b_i^2 + \sum_u b_u^2\right)$, avoids overfitting by penalizing the magnitudes of the parameters. This least square problem can be solved fairly efficiently by the method of stochastic gradient descent that will be object in the matrix factorization recommender engine.

At this step, we used cross validation to pick the optimal λ (which corresponds to the minimum RMSE) and using calculus we can show that the values of b_i and b_u that minimize this equation are :

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

$$\hat{b}_u(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} \left(Y_{u,i} - \hat{\mu} - \hat{b}_i\right)$$

where n_i is the number of ratings made for movie i.

When n_i is very large, which will give us a stable estimate, then λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0 .The larger λ, the more we shrink.

4.1.2 Recommender engines

- **Slope One**

Slope One was introduced in a 2005 paper by Daniel Lemire and Anna Maclachlan , 'Slope One Predictors for Online Rating-Based Collaborative Filtering'. This algorithm is one of the simplest way to perform collaborative filtering based on items' similarity. This makes it very easy to implement and use, and accuracy of this algorithm equals to the accuracy of more complicated and resource-intensive algorithms. Slope One method operates with average differences of ratings between each item and makes predictions based on their weighted value. More details are given in the said paper.

To perform the SlopeOne method , we need to perform a specific pre-processing approach. We followed the different steps :

i) create a copy of edx and validation set (edx.copy and valid.copy) ,retaining only userId, movieId and rating columns.
ii) change names and types of variables (to characters) for edx.copy and valid.copy dataset to make them suitable for the SlopeOne package.
iii) Package SlopeOne works with data.table objects. It is pretty fast, but for huge dataset it needs a lot of RAM. Thats why we need to sample our edx.copy dataset. we create a Data partition index (a random sampling without replacement, with p = 0.1) to produce a small training set.
iv) The rest of the steps consist to normalize the ratings, build the Slope one model and finally make prediction on our validation set.

- **Matrix Factorization with stochastic gradient Descent**

Matrix Factorization is a popular technique to solve recommender system problem. The main idea is to approximate the matrix $R_{m\times n}$ by the product of two matrixes of lower dimension, $P_{k\times m}$ and $Q_{k\times n}$ such that :

$$R \approx P'Q$$

. Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by user u on item v would be predicted as $p'_u q_v$. A typical solution for P and Q is given by the following regularization problem as defined in Chin et al(2015) :

$$\min_{P,Q} \sum_{(u,v)\in R} \left[f(p_u, q_v; r_{u,v}) + \mu_P ||p_u||_1 + \mu_Q ||q_v||_1 + \frac{\lambda_P}{2} ||p_u||_2^2 + \frac{\lambda_Q}{2} ||q_v||_2^2 \right]$$

where (u, v) are locations of observed entries in R, $r_{u,v}$ is the observed rating, f is the loss function, and μ_P,μ_Q,λ_P,λ_Q are penalty parameters to avoid overfitting.

Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item . So, to find rating for item i by user u we simply

need to compute two vectors: P[,u]' x Q[,i]. Further descriptions of this technique and the recosystem package are available here.

To perform our recommender system using parallel Matrix factorization with stochastic gradient descent, we followed the different steps:

i) As in the SlopeOne method, we created an identical copy of edx and validation set (edx.copy and valid.copy) , selecting only userId, movieId and rating columns. However, with the recosystem package, the data file for training set needs to be arranged in sparse matrix triplet form, i.e., each line in the file contains three numbers "user_index", "item_index", "rating".

ii) No RAM problem : Unlike most other R packages for statistical modeling that store the whole dataset and model object in memory, recosystem can significantly reduce memory use, for instance the constructed model that contains information for prediction can be stored in the hard disk, and output result can also be directly written into a file rather than be kept in memory. Thats why we simply use our whole edx.copy as train set (9,000,055 occurences) and valid.copy as validation set (999,999 occurences).

iii) Finally, we create a model object by calling Reco() , call the tune() method to select best tuning parameters along a set of candidate values , train the model by calling the train() method and use the predict() method to compute predicted values.

In our study, before to perform SlopeOne and Matrix factorization, we also presented recommender algorithms of the recommenderlab package(UBCF, IBCF , Popular). However, we will not develop them here since they were not relevant for the evaluation through the root mean squared error(RMSE) on the validation set.Short description of these methods are available here or in this kernel.

4.1.3 Ensemble Methods

- **GBDT: Gradient Boosting decision trees**

Gradient Boosting Machine (for Regression and Classification) is a forward learning ensemble method. The guiding heuristic is that good predictive results can be obtained through increasingly refined approximations. H2O's GBM sequentially builds regression trees on all the features of the dataset in a fully distributed way - each tree is built in parallel.

H2O's Gradient Boosting Algorithms follow the algorithm specified by Hastie et al (2001):

Initialize $f_{k0} = 0, k = 1, 2, ., K$

For $m = 1$ to M:

1.set $p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, k = 1, 2, ., K$

2.For $k = 1$ to K:

a. compute $r_{ikm} = y_{ik} - p_k(x_i), i = 1, 2, ., N$
b. Fit a regression tree to the targets $r_{ikm}, i = 1, 2, ., N$, giving terminal regions $R_{jim}, j = 1, 2, ., J_m$
c. compute $\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} (r_{ikm})}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1-|r_{ikm})}, j = 1, 2, ., J_m$
d. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$

Output $\hat{f}_k(x) = f_{kM}(x), k = 1, 2, ., K$

The gbm h2o function has many parameters. For example, nfolds, ntrees, max_depth or learn rate which control cross-validation , max number of trees , maximum tree depth and learning ratecan help to overcome overfitting. Instead, GBM's parallel performance is strongly determined by the max_depth, nbins, nbins_cats parameters. In general, for datasets over 10GB, it makes sense to use 2 to 4 nodes; for datasets over 100GB, it makes sense to use over 10 nodes, and so on. Since GBDTs have a built-in ability to apply different methods to different slices of the data, we added in some predictors that help the trees make useful clusterings: not only factors vectors of users and movies, but also number of movies each user rated (**n.movies_byUser**) and number of users that rated each movie (**n.users_bymovie**).

After creating a copy of edx set with all features(same for the validation set), with the mutate function we added the n_m and n_u attributes, converted usersId and movieId into factors, split the data intro train and test , perform the gbm algorithm with some tunes on parameters, and finally make prediction on the validation set.

The next steps consisted to build :

- random forest model on an h2oFrame (using the same training set as in the gbm method)
- a super learner or stacked ensemble using the best previous gbm and random forest models.

For more details on Machine learning h2o algorithms (Gradient boosting, random forest,etc) and the h2o package, follow h2oalgorithms and h2o package, respectively.

4.2 Model performance evaluation

To assess our model performance , we seek to evaluate how close our predictions are to the true rating values in the validation set. For this, we take into account the Root Mean Square Error (RMSE).

To construct the RMSE, you first need to determine the residuals. Residuals are the difference between the actual values and the predicted values. I denoted them by $\hat{y}_{u,i} - y_{u,i}$, where $y_{u,i}$ is the observed value for the ith observation and $\hat{y}_{u,i}$ is the predicted value.

They can be positive or negative as the predicted value under or over estimates the actual value. Squaring the residuals, averaging the squares, and taking the square root gives us the RMSE. We then use the RMSE as a measure of the spread of the y values about the predicted y value.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i} - y_{u,i}\right)^2}$$

5 Results

5.1 Identifying optimal model

5.1.1 Regression Models

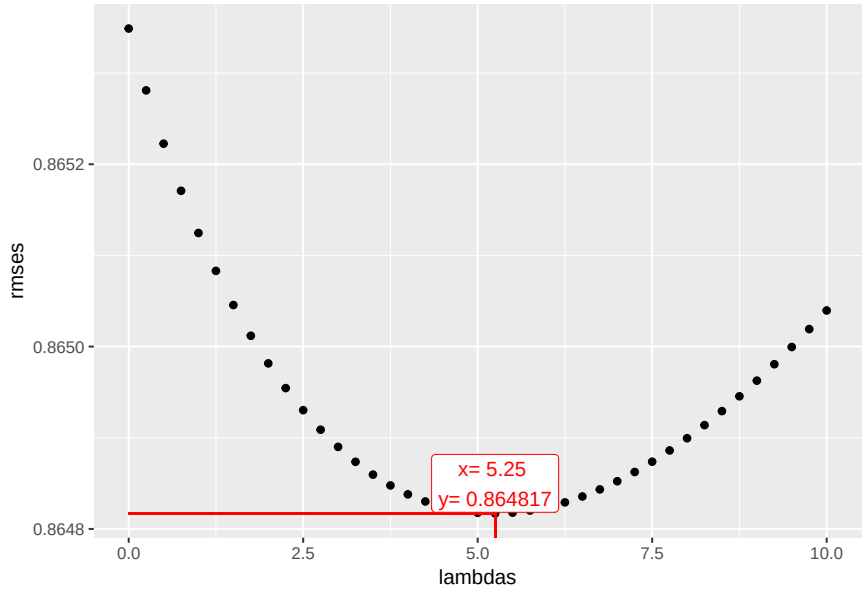

Figure 7: Choosing the optimal lambda for regularization

we can notice that with the movie and user effects combined, our RMSE decreased by almost 10% with respect to the only movie effect. The improvement when we add the time effect is not significant, (about a decrease of 0.011%). Then, we performed regularization (as explained in Methods and analysis section) using only the movie and user effects . With the optimal $\lambda = 5.25$ (figure 7) we observed that regularization gets down the RMSE's value to 0.8648170. RMSE's values for linear regression models with different adding effects and regularization are summarized in table 4.

methods	rmse
movie effects	**0.9439087**
movie + user effects	**0.8653488**
movie + user + time effects	**0.8652511**
Regularized Movie + User effects Model	**0.8648170**

Table 4: RMSE values- Linear regression models

5.1.2 Recommender engines

We obtained the following rmse's values for Popular,ubcf and ibcf methods of the recommenderlab package :

```
"Popular method"
```

```
     RMSE
0.8482917

 "ubcf method"

     RMSE
0.8589153

 "ibcf method"

    RMSE
0.963769
```

We observed that user-based CF gives an ability to achieve lower RMSE on test set than Item-based CF and Popular methods. However, these methods don't fill with the scope of our study since the rmse evaluation is made on a test set after partitioning. We want to predict ratings for the 999999 rows of the validation set and then evaluate with RMSE how close these predictions are with respect to the true ratings values in validation set. Moreover, the new data in the predict function should be of a class "ratingMatrix". Validation set doesn't have to be modified.

We then moved to the SlopeOne and Matrix factorization methods.

Performing the Matrix factorization with GD recommender method, we achieved a RMSE definitely lower than 0.8. There's an improvement (a decrease) of more than 23% with respect to the slopeOne method. (see results in table 5) . Note that the high rmse value for the SlopeOne method is also due to the small training set we used to face the RAM problem(see Methods and Analysis section - SlopeOne method). Instead, by having a larger training set, we give our algorithm a better chance of understanding the patterns in the set, rather than just learning to identify specific examples from the training set. This could save our time with validation in the long run, since we won't be dealing with quite so much overfitting.

```
 "Matrix Factorization : tuning parameters with nfolds=5,dim=c(10,20,30),\nlrate=c(0.1,0.2),nthread=1"

 "Penalty parameters to avoid overfitting : costp_l1 = 0,costq_l1 = 0\ncostp_l2 = 0.01,costq_l2 = 0.1"

 30

 0

 0.01

 0

 0.1

$min$lrate
[1] 0.1

 0.797987


$res
   dim costp_l1 costp_l2 costq_l1 costq_l2 lrate  loss_fun
1   10        0     0.01        0     0.01   0.1 0.8255899
```

```
2   20        0     0.01        0     0.01   0.1 0.8070605
3   30        0     0.01        0     0.01   0.1 0.8157530
4   10        0     0.10        0     0.01   0.1 0.8279932
5   20        0     0.10        0     0.01   0.1 0.8019461
6   30        0     0.10        0     0.01   0.1 0.8016862
7   10        0     0.01        0     0.10   0.1 0.8284874
8   20        0     0.01        0     0.10   0.1 0.8018076
9   30        0     0.01        0     0.10   0.1 0.7979870
10  10        0     0.10        0     0.10   0.1 0.8372085
11  20        0     0.10        0     0.10   0.1 0.8315828
12  30        0     0.10        0     0.10   0.1 0.8274714
13  10        0     0.01        0     0.01   0.2 0.8162612
14  20        0     0.01        0     0.01   0.2 0.9162043
15  30        0     0.01        0     0.01   0.2 0.8809001
16  10        0     0.10        0     0.01   0.2 0.8173864
17  20        0     0.10        0     0.01   0.2 0.8599332
18  30        0     0.10        0     0.01   0.2 0.9107871
19  10        0     0.01        0     0.10   0.2 0.8222024
20  20        0     0.01        0     0.10   0.2 0.8004464
21  30        0     0.01        0     0.10   0.2 0.7995085
22  10        0     0.10        0     0.10   0.2 0.8410999
23  20        0     0.10        0     0.10   0.2 0.8264263
24  30        0     0.10        0     0.10   0.2 0.8259178

 " Matrix Factorization : trains the recommender model "

iter      tr_rmse          obj
   0       0.9737  1.1979e+007
   1       0.8708  9.7956e+006
   2       0.8382  9.0926e+006
   3       0.8175  8.6875e+006
   4       0.8021  8.4172e+006
   5       0.7900  8.2171e+006
   6       0.7801  8.0667e+006
   7       0.7719  7.9424e+006
   8       0.7650  7.8477e+006
   9       0.7590  7.7642e+006
  10       0.7539  7.7000e+006
  11       0.7493  7.6431e+006
  12       0.7451  7.5922e+006
  13       0.7413  7.5483e+006
  14       0.7379  7.5110e+006
  15       0.7347  7.4773e+006
  16       0.7318  7.4464e+006
  17       0.7290  7.4151e+006
  18       0.7266  7.3907e+006
  19       0.7243  7.3692e+006

 "Matrix Factorization : show first 10 predicted values"

 4.26273 5.17404 4.73851 3.44047 4.45806 2.81567 4.10844 4.34959
 4.25262 3.45201
```

methods	rmse
SlopeOne	**1.0296368**
Matrix factorization with GD	**0.7826226**

Table 5: RMSE values- recommender engines

5.1.3 Ensemble Methods

We observed that the best gbdt and rf models (gbdt_3 and rf_3) have the lowest Root Mean Squared errors on the training set and produced rmse values on the validation set equals to 0.9839035 and 0.9543947, respectively . For the gbm model this happened when with 50 trees and max depth equals to 5, we only take into account the movieId and UserId (factor vectors) features, and with 3 folds cross validation .For the rf model , this happened when with 3 folds cross validation we trained 50 trees with max depth equals to 20 for the movieId and UserId (factor vectors) features. However, performance appears slower in RF than in GBM since RF can go to depth 20, which can lead to up to 1+2+4+8+.+2^19 ~ 1M nodes to be split, and for every one of them, mtries=sqrt(4600)=67 columns need to be considered for splitting. Instead, GBM can go to depth 5, so only 1+2+4+8+16 = 31 nodes to be split, and for every one of them, all 4600 columns need to be considered. The rmse's values on validation set for the gbdt , rf and stacked ensemble models are summarized in table 6.

```
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%

RMSE for h2o gbdt model on the validation set:
we used the best gbdt model on training set (lowest rmse) -> gbdt_3;
ntrees = 50, max depth = 5, learn rate = 0.1, nfolds = 3, fold_assignment = Random,
 keep_cross_validation_predictions = TRUE,
(see model details on rmd file report)

RMSE for h2o rf model on the validation set:
we used the best rf model on training set (lowest rmse) -> rf_3;
ntrees = 50, max.depth = 20 , nfolds = 3, nbins=20 , fold_assignment = Random,
 keep_cross_validation_predictions = TRUE,
(see model details on rmd file report)
```

methods	rmse
gradient Boosting	**0.9839035**
random forest	**0.9543947**
stacked ensemble	**0.9493815**

Table 6: RMSE values- Ensemble methods

5.2 Increasing model performance

Based on the differents RMSE's values we found for Linear regression models, recommenders algorithms and Ensemble models, we choose the two candidates which best predict ratings on the validation set : Linear

regression model with Regularized movie + user effect (i) and Matrix factorization with stochastic gradient (ii).

We focused on optimizing the performance of (ii) using 10 fold cross validation (nfolds = 10). This consisted to understand the behavior of the rmse value for the Matrix Factorization recommender engine when modifying values of number of latent factors (dim), gradient descend step rate (lrate), penalty parameters to avoid overfitting (cost) and number of threads for parallel computing (nthread).Figure 8 shows the number of latent factors vs. cross-validation RMSE . We noticed that the training model just needed 15 latent factors to reach a rmse value lower than 0.8. To sum up, we observed that even with these optimal criteria, the RMSE value for the Matrix factorization method is still below 0.8.(see value of rmse_mf_opt)

Matrix factorization with stochastic gradient descent : Increasing performance

optimization (in r$tune): nfolds = 10 ,niter = 50 ,lrate = 0.05 ,dim =c(1:20), nthread = 4

optimization (in r$train): niter = 100, nthread = 4

Penalty parameters(cost) to avoid overfitting remained unchanged

```
"Matrix Factorization : show first 10 predicted values"

4.20536 4.96643 4.84370 3.71277 5.10587 2.76322 4.33095 4.26656
4.22649 3.45355

"rmse_mf_opt"

0.7885491
```

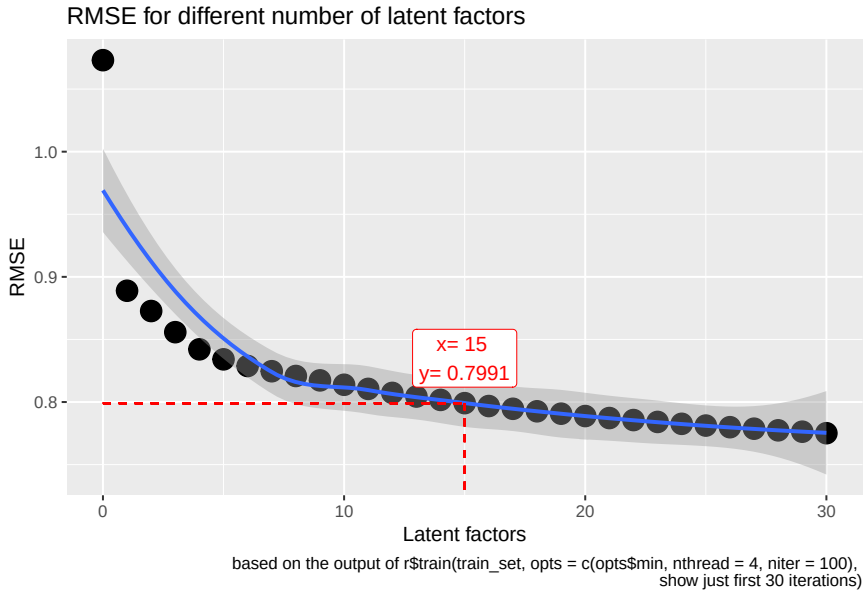

Figure 8: number of latent factors vs. cross-validation RMSE

6 Conclusion and suggestions

This MovieLens project has examined the potential best recommender system algorithm to predict movie ratings for the 10M version of the Movielens data. Using the provided training set (edx) and validation set, we successively trained different linear regression models, recommender engines and Ensemble methods. The model evaluation performance through the RMSE (root mean squared error) showed that the Linear regression model with regularized effects on users and movies, and the Matrix Factorization with Stochastic gradient descent are the two appropriate recommender systems to predict ratings on the validation set. Using 10 fold cross-validation to increase the performance of the Matrix Factorization method , in the sense of understanding the behavior of the rmse value with changing optimal criteria(dim, nthread, lrate, lim), we winded up a rmse value always less than 0.8. Future work includes further investigations on Ensemble methods in order to get lower RMSE values than the 0.9 order we got in Ensemble methods section. GBDT combine some advantages like including an ability to find non-linear transformations, ability to handle skewed variables without requiring transformations, computational robustness (e.g., highly collinear variables are not an issue) and high scalability. They also naturally lend themselves to parallelization. This has made them a good choice for several large scale practical problems such as ranking results of a search engine (Bellkor, 2009). Following this Bellkor winning solution, to get a rmse value less than 0.88 on Ensemble methods we should combine over 500 models adding not only the clustering effects of Number of movies each user rated, Number of users that rated each movie, but also hidden units of a restricted Boltzmann Machine.

References

- Aggarwal, C. C. (2016). Recommender systems (pp. 1-28). Cham: Springer International Publishing.
- Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender systems: introduction and challenges. In Recommender systems handbook (pp. 1-34). Springer, Boston, MA.
- Irizzary,R., 2018,*Introduction to Data Science*,github page,https://rafalab.github.io/dsbook/
- Shani, G., & Gunawardana, A. (2011). Evaluating recommendation systems. In Recommender systems handbook (pp. 257-297). Springer, Boston, MA.
- Saranya, K. G., Sadasivam, G. S., & Chandralekha, M. (2016). Performance comparison of different similarity measures for collaborative filtering technique. Indian J. Sci. Technol, 9(29), 1-8.
- Koren, Y. (2009). The bellkor solution to the netflix grand prize. Netflix prize documentation, 81, 1-10.
- Chin, W. S., Zhuang, Y., Juan, Y. C., & Lin, C. J. (2015). A fast parallel stochastic gradient method for matrix factorization in shared memory systems. ACM Transactions on Intelligent Systems and Technology (TIST), 6(1), 2.
- Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." Annals of Statistics (2001): 1189-1232.
- Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning (Vol. 1, No. 10). New York: Springer series in statistics.