MATRIX FACTORISATION ALGORITHMS FOR HIGHLY SCALABLE RECOMMENDER SYSTEMS

ROHAN CHOPRA , 1001780925
ROHAN.CHOPRA@MAVS.UTA.EDU
UNIVERSITY OF TEXAS AT ARLINGTON

ABSTRACT. We investigate the use of matrix factorisation to compare them for recommender systems. Recommender systems apply knowledge discovery techniques to provide personalised product recommendations. The tremendous growth of customers and products in recent years poses some challenges for these systems like computing quick and good recommendations. Another key challenge is of processing on large data sets of high dimensionality. Matrix decomposition is one of most commonly used technique to solve these issues. In this paper, we will compare and validate different types of matrix factorisation techniques. Movie Recommending system is one of the most commonly used recommender system that we see in everyday life. So, we have used SVD , QR and Non Negative Factorization methods on a real life Movielens data set to train a movie recommender system.

1. INTRODUCTION

The ongoing advancement of computational hardware has created a demand for highly automated software. These software are often supervised and require multiple large batches of data for their functioning. With the growing size of data, storage, and processing are the two primary problems associated with it. This paper focuses on recommender systems. They have evolved in the interactive world of the web, having copious applications. They apply machine learning techniques to the problem of recommending a user which movie to watch based on their previous liking. These systems, especially collaborative filtering [8] ones, are rapidly being more utilized, especially in industries like E-commerce and Media systems. Nowadays, as discussed above, a large volume of customer data is stressing these systems out. This poses two main challenges for these recommender systems, improving the quality of the recommendations, and improving the scalability of the algorithms. Firstly, recommender systems are required to make correct and significant recommendations for the user. The quality of the recommendation would decide whether the user trusts the system or not. Secondly, they pose a scalability issue where it is hard to use them with millions of data.

To some extent, these two challenges conflict. To make an algorithm perform faster we trade it off with the quality of the recommendation. So the less time an algorithm takes to do some computation, the more scalable it will be but the quality will be equally bad. So we need to tackle both the challenges concurrently. For this reason, newer technologies are required for collaborative based recommender systems. Many such techniques were generalized after the Netflix movie recommendation challenge of 2007, 2008, and 2009 [3]. This challenge was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users of films, i.e. without users or films being identified except by numbers assigned for the contest. Netflix provided a training data set of 100,480,507 ratings that 480,189 users gave to 17,770 movies [1]. The participants were asked to predict the grades/ratings on an unlabeled test set and reduce the RMSE error against the true grades (Known by the judges). Matrix factorization was one of the most popular and highly efficient algorithms that were theorized by many researchers after this challenge. The winning team's algorithm used a form matrix factorization, SVD-based approach, and bested Netflix's algorithm for predicting ratings by 10.06%.

The goal of Collaborative filtering [8] is to eliminate the use of multiple movie attributes and simplify the process of recommendations. Collaborative filtering [6, 8] does not require any movie attributes like movie genre, year, director, etc. It uses the user's historical preference on a set of items. Because it's based on historical data, the core assumption is that if the user liked the previous trends would also agree to the same in the future. It can be of either two categories implicit or explicit rating. Implicit ratings are indirect user preferences like views, clicks, and so on. For this paper, we are using Explicit ratings, which are the ratings directly given by the user.

Collaborative Filtering is always implemented by feeding the weight matrix to a machine learning algorithm. For this paper, we will be using the standard method for collaborative filtering known as Nearest Neighborhood Algorithm [5]. The process is to calculate the similarities and the distances between all the items, then selecting items that are closest to each other.

In this paper, we experiment and compare different type of matrix factorization techniques for making recommendations for a recommender system using collaborative filtering. In particular, we will be using the SVD, QR decomposition, and Non Negative Factorization on a rating/ weight matrix. The rating matrix is constructed using the MovieLens data set, which has more than 10 Million movie ratings for different users. The columns signify users, rows signify movies, and their corresponding values are the ratings by the user for that movie. This rating matrix is of high dimensionality with a lot of zero values for missing ratings. Finally, we apply the K nearest model to find the recommendations and draw comparisons on the experimental results for different factorization methods.

The rest of the paper is structured as follows. The next section of this paper formulates the problem statement and experiments that we ran on the data. Section 5 collects all my experimental results to draw out the comparison between different factorization methods. The final section includes concluding results and giving future directions.

2. Problem Statement

In this section, we formulate the problem associated with collaborative filtering and explain the algorithm and the matrix decomposition techniques that we apply to the data set. Then, we briefly describe the algorithms and steps we used for various factorization methods.

The collaborative filtering methods are typically worked out using a weight/rating matrix. The weight/rating matrix is typically very sparse, huge, and has removed values. A weight matrix R is a $n \times m$ matrix, with item $p_i, i = 1....n$ and user $u_j, j = 1...m$, where $R(i, j)$ is nothing the but Explicit rating the user p_j for item p_i.

Here, $R \in \mathbb{R}^{n\times m}$ is the weight matrix where rows are items and columns users.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots \\ r_{21} & r_{22} & r_{23} & \dots \\ r_{31} & r_{32} & r_{33} & \dots \\ \vdots & \vdots & \vdots & \dots \end{bmatrix}$$

The weight matrix is of size $n \times m$ and it can become extremely dense and huge if the value of n and m are large. Due to the tremendous increase in customers and products, the values of n and m are large for all real-life applications. The large size of this matrix requires more memory to store and large computational power to efficiently apply a machine learning model to predict good results. As seen in the Netflix prize of 2007 [4], Matrix factorization outperformed every other alternative solution to these problems. In this paper, We am solving the same problem by using multiple Matrix factorization methods on a real-life Movie data set.

2.1. Data set.

The data set We have used for this problem is from MovieLens [7]. MovieLens is a 25M movie ratings, stable benchmark data set. It has a million ratings and one million genre tags applied to 62,000 movies by 162,000 users. For running all my experiments, We have used a sample of this data due to resource limitations. We have used a data set with 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. For running my experiments , We used 6 types of Sub data sets. The largest one is described below :

The weight matrix created was of size , $9,066 \times 671$, It has $100,004$ non zero values.

If R is total number of ratings , P total number of items and U total number of users , Then Sparsity is given by,

$$\text{Sparsity} = \text{S} \;= 1 - \frac{R}{(P * U)}$$

.

For this dataset, R = 100,004 , P = 9,066 , and U = 671 . So sparsity is 0.9835 or 98.35% , which tells us that the data has a lot of zeros values .

2.2. **Algorithms and Pseudocodes.** My movie recommendation algorithm goes through three phases. The first phase is reading the data and converting it into a pivot/weight matrix. The second phase is applying matrix factorization to data and account for the missing values. The final step is to feed the factorized matrix to the k nearest neighbors model.

Algorithm 1 Find Recommendations

```
Require: $A$ is a matrix with userid , movie id and ratings, where. n is number of movies and m is the number of users and $n > m$
  //First part of the algorithm , create a weight matrix
  for i in 1..n do
    for j in 1..m do

      if A[i][j] != 0 then
      R[i][j] $\leftarrow a[i][j]$
      else
      R[i][j] $\leftarrow 0$
      end if
    end for
  end for
  //Second part of the algorithm , is to apply matrix factorization method
  array$A \leftarrow matrixfactorfunc(W)$
  //matrixfactorfunc is the matrix factorization function like SVD , QR
  for e in array$A$ do
    R = R * e // R is a matrix and e is also a computed matrix/vector // we perform matrix multiplication.
  end for
  //Third part of the algorithm , is to apply k nearest neighbor method
  recommendations = nearestneighborsfor(W , k = 20)
  print(recommendations)
```

The above algorithm completes the rating matrix using a matrix factorization method. It works in three different phases . The first part of this algorithm is create a weight/ratings matrix from the data set. It uses two for loops for number of movies and number of users to map the rating into a new weight matrix called R. The matrix R is the crux to second part of the algorithm and is further fed into a matrix factorization function . This function returns an array of all the computed components by factorization. Finally , we use a for loop to multiply all the computed components using matrix multiplication since they are all matrices and vectors . This result is the new completed rating matrix which is fed into a KNN classifier for the recommendations . The classifier finds the nearest neighbors by computing the distance for all movies.

The final result given is $N_k(p_i, u_j)$, where k is the number of neighbors . This notation is used to signify a set of k ratings which are predicted by the KNN classifier and are present at the i and j index in the rating matrix.

2.3. **Experiments.** This section covers in detail about my implementations of various types of matrix factorization techniques. The methods are applied to a rating matrix.

2.3.1. *Single Value Decomposition.* The Singular Value Decomposition (SVD)[9, 13], a method from linear algebra was popularized by the Netflix Prize for matrix factorisation. It has been generally used as a dimensionality reduction technique in machine learning. It is a popular way of doing collaborative filtering.

The dimensionality reduction approach in SVD can be very useful for the collaborative filtering process. SVD produces a set of uncorrelated eigenvectors. Each customer and product is represented by its corresponding eigenvector. The process of dimensionality reduction may help customers who rated similar products (but not exactly the same products) to be mapped into the space spanned by the same eigenvectors.

Suppose $A \in \mathbb{R}^{n\times m}$ then SVD is given by ,

$$A = U\Sigma V^T$$

,

$$U \in \mathbb{R}^{n\times k}, \Sigma \in \mathbb{R}^{k\times k}, V \in \mathbb{R}^{k\times m}$$

.

Since A is a low rank matrix, we have used the compact SVD for this project.

SVD is a matrix factorization technique, which reduces the number of features of a data set by reducing the space dimension from $n \times m$ to three sub matrices. Any real matrix A can be decomposed into three matrices U, Σ, and V. Here A is a $n \times m$ rating matrix, U is a $n \times k$ orthogonal left singular matrix, which represents the relationship between users and latent factors, is a $k \times k$ diagonal matrix, which describes the strength of each latent factor and V is a $k \times m$ right singular matrix, which indicates the similarity between items and latent factors.

The SVD decreases the dimension of the weight matrix A by extracting its latent features. The latent features here are the features of the items, for example, the ratings of the movies. It maps each user and each item into a k-dimensional latent space. This mapping promotes a clear representation of relationships between users and items. SVD also offers an unique advantage over all other factorization methods, called folding of SVD [13]. Folding enables adding new items to the already computed SVD. SVD is an expensive task when the data set is large. For a recommender system, the data set is always expanding. Suppose, new matrix C, $r \times m$ matrix is added to the original rating matrix. V is a $k \times m$ right singular matrix, so VC^T is used to find the new latent features. This result is appended to the old U.

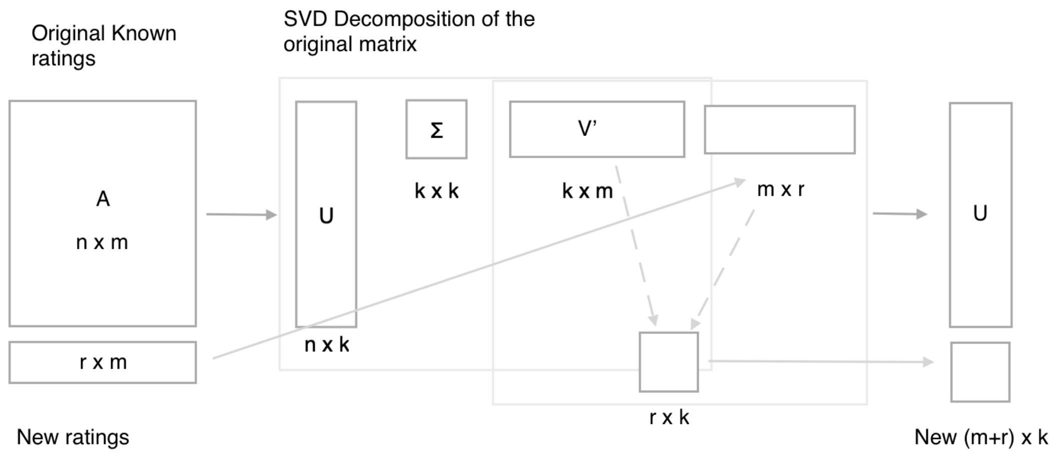

FIGURE 1. Schematic Diagram of SVD folding technique

Prediction generation By SVD

Let each item be represented by a vector x_i and each user is represented by a vector y_u . The expected rating by a user on an item can be given as,

$$\hat{r_{ui}} = x_i^T y_u$$

.

Here, is a form of factorization in singular value decomposition. The x_i and y_u can be obtained in a manner that the square error difference between their matrix; product and the expected rating in the user-item matrix is minimum. We can create an objective function (that we want to minimize) with respect to x and y, which are (m,k) and (k,n) matrices.It is expressed as,

$$min(x,y) \sum_{(u,i)\in k} (r_{ui} - x_i^T y_u)^2$$

.

Here, K is a set for all users u and movies i pairs for which ratings r_{ui} are known .

In order to let the model generalise well and not over fit the training data, a regularisation term is added as a penalty to the above formula,

$$min(x,y) \sum_{(u,i)\in k} (r_{ui} - x_i^T y_u)^2 + \lambda(||x_i||^2 + ||y_u||^2)$$

.

In order to reduce the error between the value predicted by the model and the actual value, the algorithm uses a bias term.

$$min(x,y,b_i,b_u) \sum_{(u,i)\in k} (r_{ui} - x_i^T y_u - -b_W e - b_u)^2 + \lambda(||x_i||^2 + ||y_u||^2) + b_i^2 + b_u^2$$

,

is the final equation after adding the regularisation term and bias. Let for a user-item pair (u, i), μ is the average rating of all items, b_i is the average rating of item We minus μ and b_u is the average rating given by user $u - \mu$

The above equations are the main components of the algorithm which works for singular value decomposition based recommendation system.

Once the $m \times n$ rating matrix R is decomposed and reduced into three SVD component matrices with k features, prediction can be generated from it by computing the matrix product between the three SVD components. The rating matrix is set to have 0 values for movies i , with no ratings for the user j .The prediction generation process involves only a matrix multiplication , which takes O(nm) time since k is a constant. The rating matrix might have negative values, but since we are passing it to the K nearest neighbor (calculating cosine similarity) the negative values would help in finding better recommendations.

2.3.2. *LS problem using the QR factorization.* QR decomposition [14], in linear algebra, is a decomposition of a matrix A into a product of an orthogonal matrix Q and an upper triangular matrix R. QR decomposition is often used to solve the linear least-squares problem and is the basis for a particular eigenvalue algorithm.

Suppose $A \in \mathbb{R}^{n\times m}$ then, QR-decomposition is an orthogonal matrix Q and an upper-triangular matrix R, such that ,

$$A = QR$$

,

$$QQ^T = Q^T Q = We \implies Q^T = Q^{-1}$$

.

There are multiple ways to find the "best fit" for the data.The least-square problem [11] states that finding a linear relationship between variables is extremely unlikely . QR decomposition is one of the most used methods for finding this relationship.

Suppose you have an invertible matrix A and vector b. Consider the task of a vector v such that,

$$Av = b$$

.

The problem is that we cannot find a value of v because we have more observations than variable A.

So we try to find the value of v that approximately is the best fit to the data with little error as possible. One of the most common ways to do this is called least squares.

We minimize the sum of squared derivations. We take derivations with respect to V and setting it lead to the normal equations. This finally gives us the solution. But this whole process is tedious so we use QR matrix decomposition to solve this problem

For a matrix A , we find its QR decomposition,

$$Av = QRv = b$$

.

Since both Q and R are invertible we can write the solution in the form ,

$$v = R^{-1}Q^{T}b$$

.

Movie Recommendations using QR Factorization

In order for the movie recommender system to work, We use QR factorization [6] to predict the undetermined rating r_{ij} . We will be using an item-oriented approach since an item here is a movie. For a given set of neighbors of K neighbors ,$N_k(p_i, u_j)$ We find a weight matrix as $w_q|p_q \in N_k(p_i, u_j)$ [6]. such that ,

$$r_{ij} = \Sigma_{p_q \in N_k(p_i,u_j)} w_{qj} r_{ij}$$

.

Suppose we have A weight matrix W of 6×7 order with item $P_i, i = 1....7$ and user $U_j, j = 1...9$, where $W(i, j)$ is nothing the but Explicit rating the user U_j for item P_i.

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} & w_{16} & w_{17} \\ w_{21} & w_{22} & w_{23} & w_{24} & w_{25} & w_{26} & w_{27} \\ w_{31} & w_{32} & w_{33} & w_{34} & w_{35} & w_{36} & w_{37} \\ w_{41} & w_{42} & w_{43} & w_{44} & w_{45} & w_{46} & w_{47} \\ w_{51} & w_{52} & w_{53} & w_{54} & w_{55} & w_{56} & w_{57} \\ w_{61} & w_{62} & w_{63} & w_{64} & w_{65} & w_{66} & w_{67} \end{bmatrix}$$

We need to predict the rating for r_{23}, which is the 2nd item given by the 3rd user. We need to find the K nearest neighbors, here we say 3rd user rated $p_1, p_3, p_4,$and p_5 and among these the most similar ones to $r_2$3 are $p_1, p_3$and p_4. So the prediction rules say ,

$$p_q \in N_3(q_3, u_5)$$

,

$$r_{23} = \Sigma_{p_q \in N_3(q_2,u_3)} w_{q2} r_{q3} = w_{12}r_{13} + w_{32}r_{33} + w_{42}r_{43}$$

.

To find the weight matrix W_q We create a Least square problem and use QR decomposition to solve them. The weight matrix can be used to fill out the missing values in the rating matrix. Suppose, we take the same matrix W from the above example. We know , $N_3(q_3, u_5) = p_1, p_3$ and p_4 and the users who rated are u_1, u_4 and u_5. A submatrix can be created using the intersection of the entries of the set u_1, u_4 and u_5 and p_1, p_3 and p_4 . Then we write the equation as,

$$Bw = C$$

.

$$\implies \begin{bmatrix} r_{11} & r_{31} & r_{41} \\ r_{14} & r_{34} & r_{44} \\ r_{15} & r_{35} & r_{45} \end{bmatrix} * \begin{bmatrix} w_{13} \\ w_{31} \\ w_{43} \end{bmatrix} = \begin{bmatrix} r_{13} \\ r_{43} \\ r_{53} \end{bmatrix}$$

Then the least square solution of this equation becomes $B^T Bw = B^T c$ it's identical to the solution $Aw = b$. To guarantee the numerical stability one can use the QR decomposition as a solution.

The QR decomposition can also be applied to an $n \times m$ rectangular matrix with $n > m$ as,

$$B = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

.

The bottom (m-n) rows of the right- hand side of the matrix R consists of all zeros .Using the QR decomposition the solution of the corresponding equation becomes,

$$Bw = C$$

, and

$$\implies w = B^{-1}C = R^{-1}Q^{T}C$$

.

Now that We have finally filled the gaps in the rating matrix. We can pass this completed matrix to the nearest neighbor algorithm. Less missing values means better recommendations. The QR matrix decomposition is a quick algorithm, which makes it easier and faster to complete the ratings matrix

2.3.3. *Non-negative matrix Factorization.* Non-negative matrix factorization (NMF)[2, 15] has become a widely used tool for the analysis of high-dimensional data as it automatically extracts sparse and meaningful features from a set of non-negative data vectors.

In NMF , we decompose the matrix A into two sub matrices W and H. Dimensions of the arrays are defined by the number of components we set for the algorithm. If A has n rows and m columns and we want to decompose it to k components, then W is $n \times k$ columns and H is $k \times m$.

Suppose $A \in \mathbb{R}^{n\times m}$ then,

$$A = WH$$

$$A \approx V - WH$$

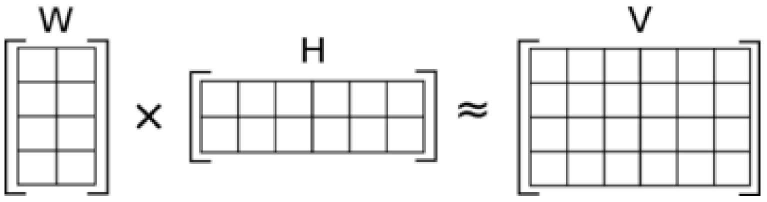

FIGURE 2. Non-negative matrix Factorization

It finds a decomposition of samples X into two matrices W and H of non-negative elements, by optimizing the distance d between X and the matrix product of WH .The most widely used distance function is the squared Frobenius norm, which is an obvious extension of the Euclidean norm to matrices,

$$d_F(X,Y) = \frac{1}{2}||X-Y||_F^2 = \frac{1}{2}\Sigma_{i,j}(X_{i,j} - Y_{i,j})^2$$

.

The reason why NMF has become so popular is because of its ability to automatically extract sparse and easily interpretable factors.

Movie Recommendations using NMF

To use NMF for recommender systems, we find the approximate rating matrix with no zero values. We calculate an NMF model for finding the W and H components for the given rating matrix A. We use the sklearn library to find these components. This implementation uses the Frobenius norm as a distance function. We use 50 components for this implementation. This means the size of W here is $n \times 50$ and and for H is $50 \times m$. Finally We create an approximation matrix V which is the matrix multiplication of W and H .

$$A \approx V = WH$$

. This approximation can be fed into the nearest neighbor algorithm for finding recommendations.

3. Results

This section covers all the results from successfully implementing all the above experiments on the movieLens data set using a core i5 6th generation with 16 GB of memory running Macintosh. We used certain evaluation metrics to compare the results from the above experiments. It covers various evaluation metrics [12, 10] like accuracy, the time taken and quality of recommendations to evaluate the performance of the matrix factorization techniques.

Some of the evaluation metrics We used for this paper are :

3.1. **Prediction Accuracy.**

The prediction accuracy metric measures the recommender system's predictions that are close to the actual true ratings. There are various metrics available in Sklearn's library. We used three different types of metrics.

3.1.1. *Mean square Error.* The mean squared error tells you how close to a collection of points a regression line is. It does this by taking the distances from the points to the line of regression (the "errors" are these distances) and squaring them. There is a need for the square to eliminate any negative signs. It also gives larger variations more weight. As you find an average of a set of errors, it's called the mean squared error.

$$MSE = \frac{1}{N}||y - \hat{y}||^2$$

.

$\hat{y}$: is the predicted rating for movie.
y : is the predicted rating for movie .
N : It is total number of movies rated by users.

3.1.2. *Root Mean square Error.* The standard deviation of the residuals (prediction errors) is root mean square error (RMSE). Residuals are a measure of how far data points are away from the regression line; RMSE is a measure of how these residuals are spread out. It tells you, in other words, how concentrated the data is along the best fit line. To check experimental findings, root mean square error is widely used in climatology, forecasting, and regression analysis.

$$RMSE = \frac{1}{N^{\frac{1}{2}}}||y - \hat{y}||_2$$

.

$\hat{y}$: is the predicted rating for movie.
y : is the predicted rating for movie .
N : It is total number of movies rated by users.

3.1.3. *R2 score Error.* It reflects the proportion of variance of y that the independent variables in the model have clarified. It offers an indicator of fit quality and thus a measure of how well unseen samples, through the proportion of explained variance, are likely to be predicted by the model.

If $\hat{y}_i$ is the predicted value of the $i-th$ sample and y_i is the corresponding true value for total n samples, the estimated R^2 is defined as,

$$R^2(y, \hat{y}) = 1 - \frac{||y - \hat{y}||^2}{||y - \bar{y}||^2}$$

Where ,

$$\bar{y} = \frac{1}{2}\sum_{i=1}^{n} y_i$$

.

3.1.4. *Mean squared logarithmic error.* The mean squared log error function calculates a probability metric that corresponds to the squared logarithmic (quadratic) error or loss expected value.

$$\text{MSLE}(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{n}(\ln(1 + y_i) - \ln(1 + \hat{y}_i))^2.$$

Here , $\hat{y}$ is the predicted value of the i^{th} sample . N is the total number of samples .

3.2. **Time Complexity.** Time complexity is the time taken by the Matrix Decomposition technique to work on the training data set. Calculating quick recommendations is one of the main ideas behind using matrix decomposition techniques for recommender systems. So, there are some of the valid metric for comparing different methods.

3.3. **Experimental Results.** This part of the section reports all the experimental results We got when We performed the experiments on the dataset. We have used six different datasets with different sizes .

- Dataset1 : Consists of 100004 movies and 671 user.
- Dataset2 : Consists of 3108 movies and 940 users.
- Dataset3 : Consists of 1375 movies and 940 users.
- Dataset4 : Consists of 1375 movies and 940 users
- Dataset5 : Consists of 1375 movies and 940 users
- Dataset6 : Consists of 1375 movies and 940 users

3.3.1. *SVD.* For my implementation of SVD on the data set, We used the scipy library. SVD function of the scipy library is a normal implementation of SVD using ARPACK or LOBPCG as an eigensolver on $A.H * A$ or $A * A.H$, depending on which one is more efficient. We ran the SVD function using $k = 50$ number of factors on 6 different data sets. We calculated RMSE, MSE, MSLE, R2 Error, and Time complexity for each data set. We have reported my results below.

Results For SVD					
Dataset	MSE	RMSE	MSLE	R2 Error	Time(Seconds)
Dataset1	12.398	3.521	1.804	-15.297	1.71
Dataset2	9.936	3.152	1.188	-7.706	0.658
Dataset3	13.236	3.638	1.907	-102.190	0.422
Dataset4	13.083	3.617	1.892	-90.455	1.038
Dataset5	13.176	3.629	1.896	-84.277	0.885
Dataset6	13.157	3.627	1.899	-79.004	0.731

TABLE 1. This table summarizes the results we got when we ran SVD matrix factorization for all sub data sets. We report the value we observed for the performance metrics discussed above .

In 1 we have summarised all the results that we got for SVD by running it for all sub data set. The table gives individual metric results. We can see average MSE error for SVD is 13%, average RMSE error is 3.5% and average time taken is $1sec$.

3.3.2. *Recommendations using QR Factorization.* For my implementation of QR decomposition on the dataset, We used the NumPy library. QR function of the NumPy library is a generic implementation of QR. My algorithm applied QR decomposition to the dataset to find a weight matrix W by constructing a least square problem. We take the dot product between the weight matrix and the original rating matrix, A to complete the rating matrix. Finally, We apply the K nearest neighbor to find the recommendations. We calculate all the performance metrics and reported them below.

In 2 we have summarised all the results that we got for QR factorisation by running it for all sub data set. The table gives individual metric results. We can see average MSE error for SVD is 13.3%, average RMSE error is 3.5% and average time taken is $0.5sec$.

3.3.3. *Non-negative matrix Factorization.* For my implementation of Non-negative Matrix Factorization decomposition on the dataset, We used the sklearn library. We used k = 50, the number of components for constructing W and H matrices. The dot product between W and H gave an approximation to the original rating matrix but with less sparsity. We calculate all the performance metrics and reported them below.

In 3 we have summarised all the results that we got for NMF by running it for all sub data set. The table gives individual metric results. We can see average MSE error for SVD is 12%, average RMSE error is 3.5% and average time taken is $6sec$.

Results For QR Factorization					
Dataset	MSE	RMSE	MSLE	R2 Error	Time(Seconds)
Dataset1	12.532	3.540	2.060	-9.850	1.429
Dataset2	11.677	3.417	1.885	-4.830	0.627
Dataset3	13.431	3.664	2.181	-41.136	0.484
Dataset4	13.265	3.642	2.171	-37.1126	0.452
Dataset5	13.389	3.659	2.191	-37.747	0.400
Dataset6	13.365	3.655	2.196	-35.652	0.398

TABLE 2. This table summarizes the results we got when we ran QR factorization for all sub data sets. We report the value we observed for the performance metrics discussed above .

Results For NMF					
Dataset	MSE	RMSE	MSLE	R2 Error	Time(Seconds)
Dataset1	12.274	3.503	1.926	-15.644	12.708
Dataset2	9.386	3.063	1.240	-8.520	4.696
Dataset3	13.0218	3.608	2.023	-112.6880	4.1419
Dataset4	12.859	3.585	1.999	-111.977	3.683
Dataset5	12.903	3.592	2.005	-96.413	4.755
Dataset6	12.923	3.594	2.012	-101.340	3.811

TABLE 3. This table summarizes the results we got when we ran NMF factorization for all sub data sets. We report the value we observed for the performance metrics discussed above .

For comparing my results, We chose the two of the most important features for evaluating collaborative filtering algorithm [10]. Time complexity and RMSE are two of the most commonly used comparison metrics.

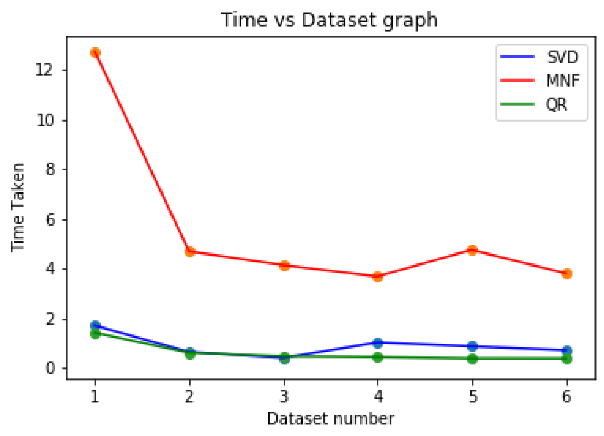

FIGURE 3. Time vs Data set graph for different Algorithms

The 3 is a scatter plot for all three algorithms, visualizing the time taken to run an algorithm for a given data set. The graph is used to visualize the run time for all different matrix factorization algorithms we have used in out project . The x-axis of this plot is the data set number for which the observation belongs to. The y-axis corresponds to the time in seconds. The time column reported in the table 1 ,2 and 3 are being used here to create this plot . This plot can be used to make conclusions on how fast an algorithm performs . It can be observed that QR factorization is the fastest algorithm when compared with SVD and

NMF. SVD is the runner up with being almost as fast as QR factorization for some data sets. MNF takes the most time to run on all 6 data sets.

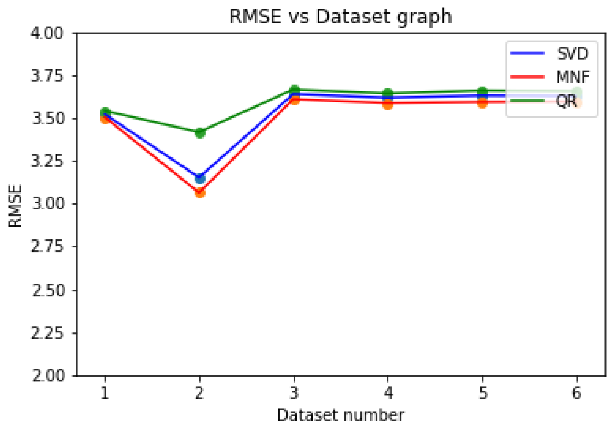

FIGURE 4. RMSE vs Data set graph for different Algorithms

The 4 is also a scatter plot for all three algorithms, visualizing the RMSE for all the given datasets. The graph is used to visualize the RMSE for all different matrix factorization algorithms we have used in out project . The x-axis of this plot is the data set number for which the observation belongs to. The y-axis corresponds to the RMSE error in % The RMSE column reported in the table 1 ,2 and 3 are being used here to create this plot . This plot can be used to make conclusions on how well an algorithm performs. It can be observed that MNF is the most accurate algorithm. With SVD being the runner up and QR factorization being the least accurate.

4. CONCLUSION

After closely looking at my results, We believe that both SVD and QR matrix decomposition methods are effective when it comes to recommender systems and undetermined ratings. We can conclude that QR decomposition is the fastest algorithm with highest error while NMF is the slowest with least error. But as we discussed in the paper above we need a system that is fast and accurate. Therefore , we choose SVD as the ideal algorithm for a recommender system. It hold a middle ground when it comes to effectiveness and would work ideally for large data sets.

But if the matrix is rank deficient we would use QR decomposition for the solution, otherwise, we use the singular value decomposition.

Some of the challenged We faced during this project were sole because of missing ratings and low computational power. The fact that the rating matrix was very sparse, finding a logical way to calculate the undetermined ratings was a challenge. Pivot matrices are very memory intensive so running these computations on my machine took a lot of memory and time.

Even if all the algorithms performed well and logically found the undetermined ratings, many other problems have been missed. The accuracy of the prediction is not dependent on Neither the set of neighborhoods nor the approaches are used. It cannot be solely trusted because of many reasons. Temporal influence is one of them and could be the key one, e.g. products served at different times may be judged changeably by consumers of similar taste. The future aspect of this project is to improve on the current algorithm. Mainly focusing on the pitfalls of the current implementation.

We compared some of the different methods of matrix factorisation that are available and can be used for recommender systems.

REFERENCES

[1] Netflix prize, Oct 2020.
[2] Non-negative matrix factorization, Nov 2020.

[3] Robert M Bell and Yehuda Koren. Lessons from the netflix prize challenge. Acm Sigkdd Explorations Newsletter, 9(2):75–79, 2007.

[4] James Bennett, Stan Lanning, et al. The netflix prize. In Proceedings of KDD cup and workshop, volume 2007, page 35. New York, 2007.

[5] Kittipong Chomboon, Pasapitch Chujai, Pongsakorn Teerarassamee, Kittisak Kerdprasop, and Nittaya Kerdprasop. An empirical study of distance metrics for k-nearest neighbor algorithm. In Proceedings of the 3rd international conference on industrial application engineering, pages 280–285, 2015.

[6] O Bora Fikir, Ilker O Yaz, and Tansel Özyer. A movie rating prediction algorithm with collaborative filtering. In 2010 International Conference on Advances in Social Networks Analysis and Mining, pages 321–325. IEEE, 2010.

[7] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. Acm transactions on interactive intelligent systems (tiis), 5(4):1–19, 2015.

[8] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '99, page 230–237, New York, NY, USA, 1999. Association for Computing Machinery.

[9] Dr. Vaibhav Kumar. Singular value decomposition (svd) in recommender system, Oct 2020.

[10] Joonseok Lee, Mingxuan Sun, and Guy Lebanon. A comparative study of collaborative filtering algorithms. arXiv preprint arXiv:1205.3193, 2012.

[11] Steven J Miller. The method of least squares. Mathematics Department Brown University, 8:1–7, 2006.

[12] Tumas Rackaitis. Evaluating recommender systems: Root means squared error or mean absolute error?, May 2019.

[13] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In Fifth international conference on computer and information science, volume 1, pages 27–8. Citeseer, 2002.

[14] Ben Denis Shaffer. Qr matrix factorization, Feb 2020.

[15] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In Proceedings of the 2006 SIAM international conference on data mining, pages 549–553. SIAM, 2006.