International Research Journal of Engineering and Technology (IRJET) e-ISSN:2395-0056
Volume: 09 Issue: 05 | May 2022 www.irjet.net p-ISSN:2395-0072

Latent Factor Model For Collaborative Filtering

Naveen Kumar Kateghar[1]

[1]*B.Tech,Computer Science and Engineering, TKR college of Engineering and Technology, Telangana, India*

Abstract - *The restrictions of neighborhood-based Collaborative Filtering (CF) methods including scalability and inadequate information present impediments to efficient recommendation systems. These strategies result in less precision, accuracy and consume a huge amount of time in recommending items. Model-based matrix factorization is an effective approach used to overcome the previously mentioned limitations of CF. In this paper, we are going to discuss a matrix factorization technique called singular value decomposition, which would help us model our recommendation system and result in good performance.*

Key Words: Recommendation systems, Collaborative Filtering, Matrix factorization, Latent Factors, Singular value decomposition

1. INTRODUCTION

A recommender system, also known as a recommendation engine or platform, is a type of information filtering system that attempts to forecast a user's "rating" or "preference" for an item.These algorithms are used to recommend products to online customers in online advertising ,ott and e-commerce platforms. When you look at a product on an e-commerce site, the recommender system may offer additional products that are similar to the one you're looking at.In general, there are two sorts of recommendation system approaches, content-based and collaborative filtering based systems.

1.1 Content Based Recommendation

The content filtering approach creates a profile for each user or product to characterize its nature. For example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, and so forth.User profiles might include demographic information or answers provided on a suitable questionnaire. The profile allows us to associate users with matching products. Of course, content-based strategies require gathering external information that might not be available or easy to collect.

1.2 Collaborative Filtering

The limitation of content-based recommendation systems in collecting various attributes and characteristics of users and items is addressed by collaborative filtering. This method attempts to account for interactions between users and items. There are two different kinds of collaborative filtering systems.

(i) Neighborhood/Memory Based: This method takes into account a user-item matrix including user ratings on items and calculates the similarity between user-user or item-item. To suggest items to a user using a user-user similarity-based strategy, the system detects users who are similar to the target user and recommends items that are rated highly by those users. Because user interests vary over time, this strategy does not work in real world circumstances. An item-item-based system tries to recommend an item to a user based on previous items that the user liked. Sparsity is one of the primary concerns with this technique. Because the user-item matrix is sparse, computing similarities would be inaccurate.

(ii) Model Based: The model based approach intends to build a recommender system using machine learning techniques and poses the recommendation system as an optimization problem , where the algorithm learns the parameters to best predict the ratings . This way, we can overcome the problems faced by memory based techniques.

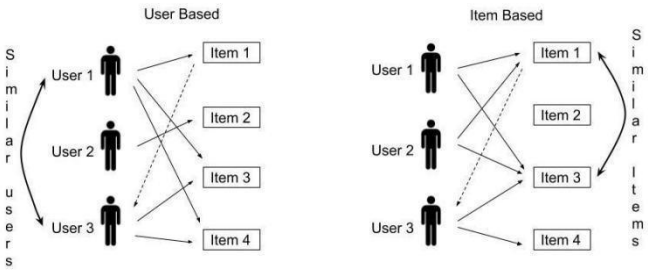

Fig -1: User-User vs Item-Item Model

2. LATENT FACTOR MODEL

The Latent Factor model explains the ratings by characterizing both items and users according to many factors inferred from the rating pattern. This method represents the user and item profiles in k-dimension space .This is known as the k- rank approximation of a matrix. Although this method is highly inspired by Singular Value Decomposition (SVD) , we approach it by formulating a minimization problem because SVD is not defined when the matrix has missing values.

2.1 Eigenvalue Decomposition

Eigenvalue decomposition (EVD) is a method of breaking a square matrix into its components. factorization of a matrix. It can be thought of as an analogy for factoring an integer .

EVD states that every square matrix $\mathbf{A_{nxn}}$ can be decomposed as product of three matrices namely $\mathbf{U\,\Lambda\,U^{-1}}$ where **U** is a matrix containing eigenvectors of **A** as its columns and **Λ** is diagonal matrix containing eigenvalues of A as its elements.

Let u1, u2, . . . , un be the eigenvectors of a matrix A and let λ1, λ2, . . . , λn be the corresponding eigenvalues. Consider a matrix U whose columns are u1, u2, . . . , un.

By definition of Eigenvectors we can write,

$$Au_i = \lambda_i u_i \quad \forall\ 1 \le i \le n$$

$$AU = A\begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \dots & u_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ Au_1 & Au_2 & \dots & Au_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

$$= \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \lambda_1 u_1 & \lambda_2 u_2 & \dots & \lambda_n u_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

$$= \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ u_1 & u_2 & \dots & u_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & \lambda_n \end{bmatrix} = U\Lambda$$

$$AU = U\Lambda$$

Now , If $\mathbf{U^{-1}}$ exists, then we can write,

$$AUU^{-1} = U\Lambda U^{-1}$$

$A = U\Lambda U^{-1}$ ($\because U^{-1}U = I$) [eigenvalue decomposition]

and

$$U^{-1}AU = U^{-1}U\Lambda$$

$U^{-1}AU = \Lambda$ ($\because U^{-1}U = I$) [diagonalization of A]

Note that $\mathbf{U^{-1}}$ only exists if the columns of U are linearly independent and since the U contains eigenvectors corresponding to different eigenvalues of A, they are linearly independent.

Furthermore, if the matrix ***A*** is symmetric and the columns of U are normalized to unit length we can easily prove that ,

$$U^T U = I \quad (\mathbf{I} \text{ is an identity matrix})$$

this means that $U^T = U^{-1}$ and the EVD of A can be rewritten as,

$$\mathbf{A = U\,\Lambda\,U^T}$$

2.2 Singular Value Decomposition

One of the major limitations of Eigenvalue decomposition includes that EVD exists only for a square matrix because eigenvalues aren't defined for rectangular matrices, so EVD for a non-square or rectangular matrix does not exist. Singular Value Decomposition(SVD) overcomes this problem and provides a decomposition of any matrix into its components.

Note that we were able to decompose a matrix because of this equation, $\mathbf{Au_i = \lambda_i u_i}$. This gave us an Eigenvalue decomposition for a square matrix. In the earlier scenario, while we were dealing with square matrices we selected u1, u2, . . . ,un to be eigenvectors and also note that they also form a basis for the space $\mathbf{R^n}$. Since we are dealing with rectangular matrices , they perform linear transformations between two different spaces.

Let $\mathbf{A_{mxn}}$ is a matrix which performs transformations from n-dimension space to m-dimension space and let $v_1, v_2, . , v_r , . , v_n$ be the basis of input space i.e., $\mathbf{R^n}$ and $u_1, u_2, . , u_r , . , u_m$ be the basis of output space i.e., $\mathbf{R^m}$.Now our goal is to find these basis such that,

$$\mathbf{Av_i = \sigma_i u_i}$$

We have to select first **r** v_i's such that it spans the row space of A and is also orthogonal (where **r** is the rank of matrix A). We then have to select the remaining **n-r** v_i's such that they span the null space of A. Since the null space of A is orthogonal to the row space of A, the v_i's we selected would act as a basis for the $\mathbf{R^n}$. Now for $1 \le i \le k$ define u_i to be a unit vector parallel to Av_i and extend this basis such that it spans the left null space i.e., null space of A^T.Since null space of A^T is orthogonal to the column space of A, the u_i's we selected would act as a basis for the $\mathbf{R^m}$.

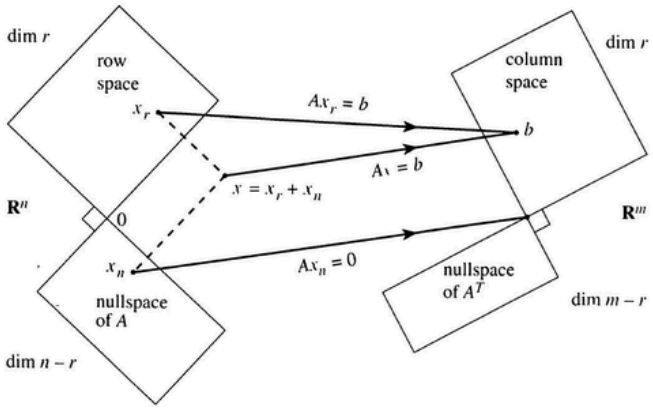

Fig -2: Strang diagram of four fundamental subspaces

Now let's see what would be our matrix decomposition after selecting such a basis . Let $\mathbf{U_{mxr}}$ be a matrix containing $u_1, u_2,..., u_r$ as its columns where u_i is the basis vector in $\mathbf{R^m}$ and $\mathbf{V_{nxr}}$ is a matrix containing $v_1, v_2, \ldots, v_r$ as its columns where v_i is the basis vector in $\mathbf{R^n}$. Σ is a diagonal matrix with the singular values.

we know that $\mathbf{Av_i = \sigma_i u_i}$ so,

$$A_{m\times n}V_{n\times r} = A_{m\times n}\begin{bmatrix}\uparrow & \uparrow & & \uparrow\\ v_1 & v_2 & \dots & v_r\\ \downarrow & \downarrow & & \downarrow\end{bmatrix}_{n\times r} = \begin{bmatrix}\uparrow & \uparrow & & \uparrow\\ Av_1 & Av_2 & \dots & Av_r\\ \downarrow & \downarrow & & \downarrow\end{bmatrix}_{m\times r}$$

$$= \begin{bmatrix}\uparrow & \uparrow & & \uparrow\\ \sigma_1 u_1 & \sigma_2 u_2 & \dots & \sigma_r u_r\\ \downarrow & \downarrow & & \downarrow\end{bmatrix}_{m\times r}$$

$$= \begin{bmatrix}\uparrow & \uparrow & & \uparrow\\ u_1 & u_2 & \dots & u_r\\ \downarrow & \downarrow & & \downarrow\end{bmatrix}_{m\times r}\begin{bmatrix}\sigma_1 & 0 & \dots & 0\\ 0 & \sigma_2 & & \vdots\\ \vdots & & \ddots & 0\\ 0 & \dots & 0 & \sigma_r\end{bmatrix}_{r\times r} = U_{m\times r}\Sigma_{r\times r}$$

$$A_{m\,x\,n}\,V_{n\,x\,r} = U_{m\,x\,r}\,\Sigma_{r\,x\,r}$$

We can extend the matrices U and V from n x r and m x r to n x n and m x m respectively by putting in the remaining n-r and m-r basis vectors.

Hence, the equation becomes ,

$$A_{m\,x\,n}\,V_{n\,x\,n} = U_{m\,x\,m}\,\Sigma_{m\,x\,n}$$

Observe that the matrix Σ dimensions now would be m x n wherein the last m-r rows and n-r columns would be zeros.

Now,

$$A_{m\,x\,n}\,V_{n\,x\,n}\,V^{-1}{}_{n\,x\,n} = U_{m\,x\,m}\,\Sigma_{m\,x\,n}\,V^{-1}{}_{n\,x\,n}$$

Let's assume matrix V's columns are normalized to unit length and, since we know U and V are orthogonal (by the way we have constructed U and V).

$$A_{m\,x\,n} = U_{m\,x\,m}\,\Sigma_{m\,x\,n}\,V^{-1}{}_{n\,x\,n}$$
$$(\because VV^{-1} = I)$$

$$A_{m\,x\,n} = U_{m\,x\,m}\,\Sigma_{m\,x\,n}\,V^{T}{}_{n\,x\,n}$$
$$(\because V^{-1} = V^T)$$

Hence , the SVD decomposition of any matrix $A = U\Sigma V^T$.

Till now, we have discussed how to select a basis so that $Av_i = \sigma_i u_i$, and what decomposition would we get if such a basis were selected .Now let's see how to actually compute U,V and Σ .

we know that $A = U\Sigma V^T$. so ,

$$A^TA = (U\Sigma V^T)^T U\Sigma V^T$$
$$= V\Sigma^T U^T U\Sigma V^T \quad \because (AB)^T = B^T A^T$$
$$= V\Sigma^2 V^T \because U^T U = I$$

This is the eigenvalue decomposition of A^TA. Thus, matrix V contains the eigenvectors of A^TA. Similarly, $AA^T = U\Sigma^2U^T$, thus matrix U contains the eigenvectors of AA^T. We know that AA^T is a square symmetric matrix hence, A^TA and AA^T have the same eigenvalues. Therefore, Σ is a diagonal matrix with its elements as $\sqrt{\boldsymbol{\lambda_i}}$ where each $\boldsymbol{\lambda_i}$ is the eigenvalue of A^TA / AA^T .

We can formulate the singular value decomposition of a matrix A in below manner ,

$$\begin{bmatrix} & A & \end{bmatrix}_{m\times n} = \begin{bmatrix}\uparrow & \cdots & \uparrow\\ u_1 & \cdots & u_k\\ \downarrow & \cdots & \downarrow\end{bmatrix}_{m\times k}\begin{bmatrix}\sigma_1 & & \\ & \ddots & \\ & & \sigma_k\end{bmatrix}_{k\times k}\begin{bmatrix}\leftarrow & v_1 & \rightarrow\\ & \vdots & \\ \leftarrow & v_k & \rightarrow\end{bmatrix}_{k\times n}$$
$$= \sum_{i=1}^{k}\sigma_i u_i v_i^T$$

We order decomposition in such a way that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$. we can prove that $\boldsymbol{\sigma_1 u_1 v_1^T}$ is a rank 1 matrix and $\boldsymbol{\sigma_1 u_1 v_1^T}$ will be the best rank-1 approximation of the original matrix A because σ_1 is the largest σ and it corresponds to the largest $\sigma_i u_i v_i^T$ term. Hence ,best rank-1 approximation. Similarly, $\sum_{i=1}^{k}$ $\boldsymbol{\sigma_i u_i v_i^T}$ is the best rank-k approximation of matrix A.

3. IMPLEMENTATION

The way we have defined the SVD of a matrix, we realize that the SVD is not defined if the matrix does not contain any value. In real world scenarios, the user-item matrix, wherein the rows represent the ratings of a user on various items and the columns represent the ratings of various users on an item, is largely sparse. Hence, we cannot perform SVD on the user-item matrix. Hence, we change our approach and try to formulate a minimization problem and find the optimal solution to it.

3.1 Formulating Optimization Problem

Let $A_{m\,x\,n}$ be the user-item rating matrix . We know that this matrix can be decomposed into $U_{mxk}\,V^T{}_{nxk}$ ($\because \Sigma$ can be interpreted as a scalar on the U matrix i.e, $U\Sigma = U$) where U contains the basis for rows of the user-item matrix i.e The users(rows) can be formed by the matrix U

and similarly V contains the basis for columns of the user-item matrix i.e., the items(columns) can be formed by the matrix V.

$$\therefore A \approx U_{mxk} V^{T}_{nxk}$$

Now, each rating $\mathbf{r_{ij}}$ of matrix A can be computed as,

$$\mathbf{r_{ij}} \approx \sum_{k=0}^{K} u_{ik} v^{T}_{jk}$$

Where K is the dimension of latent factors in which we want our decomposition to be represented.

Our objective is to minimize the sum of squared errors between known ratings in the matrix and computed ratings for these ratings. i.e.,

$$\min_{\{u_i\}_{i=1}^{m},\{v_j\}_{j=1}^{n}} \sum_{i,j\in\mathcal{X}} \sum_{k} (r_{ij} - u_{ik} v_{jk}^{T})^2$$

where $\mathbf{r_{ij}}$ is the actual rating of the user i on item j and $u_{ik}v^{T}_{jk}$ is the computed or predicted rating for the user i on item j. In the above equation X, is the set of known ratings. This means we only perform the summation over the known ratings.We want to find such ui 's and vj 's that minimize the above function.

We have overcome the problem of sparsity with this approach as we are iterating only through the known ratings of the user-item matrix. Now since we know that there would be only a few such cells in the matrix where ratings are present, this leads to overfitting, i.e., if we optimize for the above equation, we won't be able to generalize the predicted ratings for unknown data. To solve this limitation, we introduce regularization. This allows us not to make the parameters of the minimization equation complex, which leads to overfitting.

The modified optimization equation would be,

$$\min_{\{u_i\}_{i=1}^{m},\{v_j\}_{j=1}^{n}} \sum_{i,j\in\mathcal{X}} \sum_{k} (r_{ij} - u_{ik} v_{jk}^{T})^2 + \lambda \left(\sum_{i} \sum_{k} u_{ik}^2 + \sum_{j} \sum_{k} v_{jk}^2 \right)$$

Here, λ is the regularization strength , which controls the trade off between underfitting and overfitting , we can estimate this variable using techniques like cross-validation etc.,

Finally, we can also account for global effects. This includes the average rating of the entire user-item rating matrix, the average deviation of a movie's rating (movie bias) compared to other movies, and the average deviation of a user's rating (user bias) compared to other users. This part allows us to capture various aspects of the data, which improves the performance of our model.

The predicted rating of a user i on movie j can be computed as ,

$$\mathbf{r_{ij}} \approx \mu + b_i + c_j + \sum_{k=0}^{K} u_{ik} v^{T}_{jk}$$

where ,
μ = average rating of all users on all items.
b_i = standard deviation of u_i
c_j = standard deviation of v_j^T

We can calculate μ directly from the matrix and include b_i ,c_j into the optimization equation and learn from the data.The modified equation would be,

$$\min_{b,c,\{u_i\}_{i=1}^{m},\{v_j\}_{j=1}^{n}} \sum_{i,j\in\mathcal{X}} \sum_{k} (r_{ij} - \mu - b_i - c_j - u_{ik} v_{jk}^{T})^2 + \lambda \left(\sum_{i} \sum_{k} u_{ik}^2 + \sum_{j} \sum_{k} v_{jk}^2 + \sum_{i} b_i^2 + \sum_{j} c_i^2 \right)$$

This is the final problem formulation. In the next section, we'll see how to solve for the equation parameters.

3.2 Solving Optimization Problem

To solve the optimization problem in the above section, we can choose any variant of gradient descent algorithms. In our case, we are going to implement the Stochastic Gradient Descent version as it performs better and is also time-efficient. I am going to demonstrate implementation in python.

The algorithm for SGD is given as :

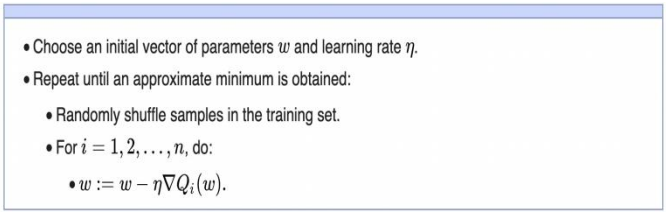

Fig -3: SGD Algorithm

The dataset is being collected here . It contains a large number of data points of 26 million users and ratings (on a scale of 5) of each user on movies. Only 100K data points are being considered for demonstration purposes. Here is a sample of the dataset.

	user_id	item_id	rating
0	1	110	1.0
1	1	147	4.5
2	1	858	5.0
3	1	1221	5.0
4	1	1246	5.0

Fig -4: Sample Data

We would split the complete dataset into train and test sets so as to measure the goodness of our recommendation system.

The first step is to initialize the parameters of the minimization equation and decide on the dimension of latent factors.

```
#initialize parameters of optimization
def initialize(dim):
    '''In this function, we will initialize bias value 'b','c' and matrices U and V '''

    vector=np.zeros(dim)
    return list(vector)

#dim=  gives the number of dimensions for u_bias
dim=train_adjacency_matrix.shape[0]
u_bias=initialize(dim)

#dim= gives the number of dimensions for m_bias
dim=train_adjacency_matrix.shape[1]
m_bias=initialize(dim)

latent_factors = 10

u = np.zeros((train_adjacency_matrix.shape[0],latent_factors))
v = np.zeros((train_adjacency_matrix.shape[1],latent_factors))
```

Fig -5: Code for initializing parameters

In the above piece of code, train_adjacency_matrix is the user-movie matrix formed from the train data set.

Next, we will calculate the derivative of the minimization equation w.r.t u_bias (user bias), m_bias (movie bias), ui (user i's row in matrix U) and v_j (movie j's row in matrix V^T).

```
def derivative_db(u_id,m_id,rating,u,v,global_mean,lamda):
    '''In this function, we will compute dL/db_i'''

    e_ij = rating-global_mean-u_bias[u_id]-m_bias[m_id]-np.dot(u[u_id],v[m_id].T)
    #dL_db= (2*lamda*(u_bias[u_id]))-(2*(e_ij))
    dL_db= ((lamda * (u_bias[u_id]))- e_ij )


    return dL_db
```

Fig -5: Derivative of mininization eqn w.r.t user bias

```
def derivative_dc(u_id,m_id,rating,u,v,global_mean,lamda):
    '''In this function, we will compute dL/dc'''

    e_ij = rating-global_mean-u_bias[u_id]-m_bias[m_id]-np.dot(u[u_id] ,v[m_id].T)
    #dL_dc= (2*lamda*(m_bias[m_id]))-(2*(e_ij))
    dL_dc= (lamda * (m_bias[m_id])) - (e_ij)

    return dL_dc
```

Fig -6: Derivative of mininization eqn w.r.t movie bias

```
def derivative_du(u_id,m_id,rating,u,v,global_mean,lamda):
    '''In this function, we will compute dL/du'''

    e_ij = rating-global_mean-u_bias[u_id]-m_bias[m_id]-np.dot(u[u_id],v[m_id].T)
    #dL_du= (2*lamda*(m_bias[m_id]))-(2*(e_ij) * (v[m_id]))
    dL_du= (lamda* m_bias[m_id] ) - ( e_ij * v[m_id])

    return dL_du
```

Fig -7: Derivative of mininization eqn w.r.t user vector

```
def derivative_dv(u_id,m_id,rating,u,v,global_mean,lamda):
    '''In this function, we will compute dL/dv'''

    e_ij = rating-global_mean-u_bias[u_id]-m_bias[m_id]-np.dot(u[u_id] ,v[m_id].T)
    #dL_dv= (2*lamda*(m_bias[m_id]))-(2*(e_ij) * (u[u_id]))
    dL_dv= (lamda*m_bias[m_id])- (e_ij * u[u_id])
    return dL_dv
```

Fig -8: Derivative of mininization eqn w.r.t movie vector

Following that, we define values for alpha (learning rate), reg_strength (regularization strength), and the number of epochs for converging the minimization equation, and we run stochastic gradient descent training on our train dataset, calculating the test and train MSE (mean squared error), which is the evaluation criterion for goodness of fit.

After iterating for all the epochs, we get the final best parameters of the equation, which are b, c, U, and V.

```
train_mse=[]
test_mse=[]

alpha = 0.01
reg_strength = 0.1
n_epochs = 10

for epoch in range(n_epochs):
    for x,y,z in zip(list(x_train['user_id']),list(x_train['item_id']),list(y_train)):

        #updating bias and latent factor for users and items
        u_bias[x]=u_bias[x]-( alpha * derivative_db(x,y,z,u,v,global_mean,reg_strength))
        m_bias[y]=m_bias[y]-( alpha * derivative_dc(x,y,z,u,v,global_mean,reg_strength))

        u[x] = u[x] - ( alpha * derivative_du(x,y,z,u,v,global_mean,reg_strength))
        v[y] = v[y] - ( alpha * derivative_dv(x,y,z,u,v,global_mean,reg_strength))


    #Getting Predictions
    y_train_pred=[]
    for p,q in zip(list(x_train['user_id']),list(x_train['item_id'])):
      y_train_pred.append(global_mean+u_bias[p]+m_bias[q]+np.dot(u[p],v[q]))

    y_test_pred=[]
    for p,q in zip(list(x_test['user_id']),list(x_test['item_id'])):
      y_test_pred.append(global_mean+u_bias[p]+m_bias[q]+np.dot(u[p],v[q]))

    #calculating train MSE
    train_mse_val=mean_squared_error(np.array(y_train),np.array(y_train_pred))
    train_mse.append(train_mse_val)

    #calculating test MSE
    test_mse_val=mean_squared_error(np.array(y_test),np.array(y_test_pred))
    test_mse.append(test_mse_val)

    print("Test MSE OF Epoch ",epoch+1," is ===>. ",test_mse_val)
    print("================================================")
```

Fig -9: Training SGD on our data

4. RESULT ANALYSIS

Results for each epoch are :

```
Test MSE OF Epoch  1  is ===>.  0.8526279289758775
================================================
Test MSE OF Epoch  2  is ===>.  0.8337790233535978
================================================
Test MSE OF Epoch  3  is ===>.  0.8243775598736407
================================================
Test MSE OF Epoch  4  is ===>.  0.8178408444011503
================================================
Test MSE OF Epoch  5  is ===>.  0.8122584373713904
================================================
Test MSE OF Epoch  6  is ===>.  0.8070870771999097
================================================
Test MSE OF Epoch  7  is ===>.  0.8025880430436362
================================================
Test MSE OF Epoch  8  is ===>.  0.7988370824985829
================================================
Test MSE OF Epoch  9  is ===>.  0.7957293328933156
================================================
Test MSE OF Epoch  10  is ===>.  0.7931390998082614
================================================
```

Fig -10: Results on each epoch

Observe that the MSE for the last epoch is 0.74, which means, on an average, we are mistaken by 0.74 in predicting ratings for unknown user-item pairs.

We were still able to achieve a good MSE for random selections of various parameters. We can use more techniques like cross validation to select the best learning rate, regularization parameter, and also the number of latent factors. There are also many advanced convergence algorithms like AdaGrad, Adam Optimizer, etc. Since we have initialized all the parameters of the equation with zeros, we could be more mindful and use better initialization strategies.

Given below is the graph which depicts the change in MSE with an increase in the number of epochs. This gives us an idea of whether our model is overfitting or underfitting. If the train error is decreasing while the test error is increasing with an increase in the number of epochs, then the model is said to be overfitted. If both the train and test errors increase with an increase in the number of epochs, then the model is said to be underfitted. In our case, both the above scenarios do not seem to occur, so our model is able to generalize the predictions made on ratings.

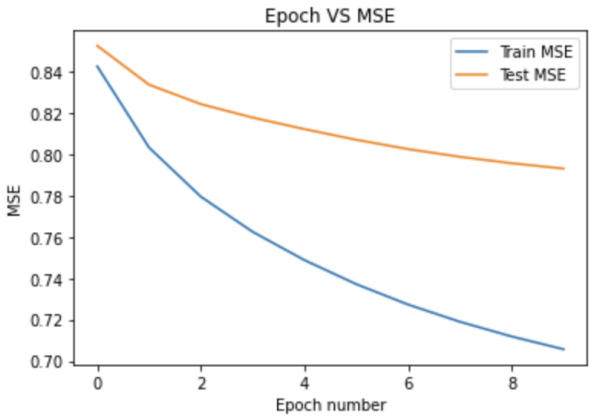

Fig -11: Train and Test Error VS Number of epochs

5. CONCLUSION

In this paper , we have discussed the theory of latent factor models which uses matrix factorization techniques and also practically implemented the truncated svd method for matrix factorization. We have also briefly discussed various types of recommender systems present and the limitations faced by them. Latent factor model is matrix factorization technique in which the model finds the relationships between the user and an item by representing the user and item vectors in latent dimensions. This technique employed recommendation system overcomes many limitations such as sparsity, scalability, computation time , memory etc,.

REFERENCES

[1] Sandeep Kumar Raghuwanshi, Rajesh Kumar Pateriya, Accelerated Singular Value Decomposition (ASVD) using momentum based Gradient Descent Optimization, https://doi.org/10.1016/j.jksuci.2018.03.012. (https://www.sciencedirect.com/science/article/pii/S1319157818300636)

[2] Dan Kalman .The American University Washington, DC 20016,"A Singularly Valuable Decomposition: The SVD of Matrix" (https://www-users.cse.umn.edu/~lerman/math5467/svd.pdf)

[3] Nicolas Hug , "Understanding matrix factorization for recommendation" (http://nicolas-hug.com/blog/matrix_facto_1)

[4] Yehuda Koren, Yahoo Research" MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS "

[5] (https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf)

[6] https://en.wikipedia.org/wiki/Stochastic_gradient_descent

[7] https://en.wikipedia.org/wiki/Recommender_system

[8] Aggarwal, C.C., 2016. Model-based collaborative filtering. In: Recommender Systems, Springer International Publishing, Cham, pp. 71–138. https://doi.org/10.1007/978-3-319-29659-3_3.

[9] Bell, R., Koren, Y., Volinsky, C., 2007. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In: Proc. 13th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. – KDD '07 95. https://doi.org/10.1145/1281192.1281206.

[10] Koren, Y., Bell, R., Volinsky, C., 2009. Matrix factorization techniques for recommender systems. Computer (Long. Beach. Calif). 42, 30–37. https://doi.org/10.1109/MC.2009.263.

[11] Gilbert Strang, Linear Algebra and its Applications, 2nd edition, Academic Press, New York, 1980.

BIOGRAPHIES

K.Naveen Kumar
B.Tech , CSE
TKR College Of Engineering and Technology