MOVIE RECOMMENDER SYSTEM USING MATRIX FACTORIZATION AND DEEP LEARNING TECHNIQUES

Project report submitted in partial fulfillment of the requirement
for the degree of Bachelor of Technology

In

Computer Science and Engineering

By
Sparsh Mittal (181272)

Under the supervision of
Dr. Aman Sharma
Assistant Professor (SG)
to

Department of Computer Science &

Engineering and Information Technology

Jaypee University of Information Technology

Waknaghat, Solan -173234, Himachal Pradesh

certi

ackno.

CONTENTS

ABSTRACT

Recommendations drive so many of our decisions on a daily basis. Recommender systems help consumers to find new information, products and services tailored to their requirements. Recommendation engines use the feedback of users to find new relevant items for them or for others with the assumption that users who have made homogeneous choices in past are highly anticipated to make similar choices in forthcoming future. There are various types of recommendation systems like – non personalized recommender system, collaborative filtering based recommender systems, deep learning based recommender systems. The goal of our project is to predict the ratings that users may give to movies that they have not rated yet and to build and test various recommender systems and then finally minimize the root-mean-square-error between the projected user ratings and true ratings of the user using matrix factorization and Deep Learning techniques. The whole project is based on user movie ratings data, so we need to collect that data. We have collected the data from movielens website and then we have filtered and processed our data. The data is then split into training and test sets and the test set. To make our Matrix Factorization model, we have used SVD, SVD++ and related algorithms and also keras. For our Deep Learning models, we have built Autoencoders, Matrix Factorization and Residual Learning in keras. Finally, have calculated RMSEs for each of our recommender systems and compared them.

CHAPTER-1 INTRODUCTION

1.1 INTRODUCTION

Recommendations drive so many of our decisions on a daily basis. Be it obvious recommendations such as suggestions of new restaurants from friends, or a certain model of camera discussed in a blog, to less direct recommendations such as Netflix promoting shows you are likely to enjoy, or Amazon proposing other purchases that go well with what you are buying. Recommender systems help consumers to find new information, products and services tailored to their requirements. They are intended to recommend suitable items to users. Recommender systems are very important in a lot of domains because they can help produce huge revenues when they are effective and also become a way to stand ahead from their adversaries.

Recommendation engines target one specific kind of machine learning problem, they are designed to suggest a product, service, or entity to a user based on other users, and their own feedback. Let's take some examples. Making a suggestion as to what movie a user would like based on what genres of movies they have ranked highly in the past would be suited to a recommendation engine. Predicting whether that movie will do well in the box office on the other hand would be better suited to a different kind of statistical model.

Recommendation engines use the feedback of users to find new relevant items for them or for others with the assumption that users who have made homogeneous choices in past are highly anticipated to make similar choices in forthcoming future like the example here. Recommendation engines benefit from having a many to many match between the users giving the feedback, and the items receiving the feedback. In other words, a better recommendation can be made for an item that has been given a lot of feedback, and more personalized recommendations can be given for a user that has given a lot of feedback.

In a typical scenario users rate multiple items, and each item is rated by multiple users. This permits us to find other users with homogeneous choices. This is valuable as users who have made homogeneous choices in past are highly anticipated to make similar choices in forthcoming future.

In this project, we will build different recommender engines using matrix factorization and deep learning techniques i.e. Autoencoders and Residual Networks. For each of them we will test them using RMSE(Root Mean Squared Error).

1.2 PROBLEM STATEMENT

Given a user u_i and a movie I_j . The user has already watched I1, I7, I8 etc. The goal is to predict what rating u_i will give to $I_{j.}$ based on users prior history using Movielens 1Million Dataset and various machine learning and deep learning models.

The algorithms used are

-Matrix Factorization

-Deep Learning and Matrix Factorization

-Autoencoders

-KNN based models

The metrics used for our recommender engines is RMSE.

1.3 OBJECTIVE

A Recommendation System predicts the likelihood that whether a user may like a certain item. Based on user's previous interaction with the data source from which the system learns from (besides from other users' data, or previous trends), the system can make recommendations of certain item to a user.

The goal of our project is to predict the ratings that users may give to movies that they have not rated yet and to build and test various recommender systems and then finally minimize the root-mean-square-error between the projected user ratings and true ratings of the user using matrix factorization and Deep Learning techniques. Our goal is to find the best recommender engine for the prediction of movie ratings. The implementation of Matrix Factorization based algorithms are created using an open source library called Surprise. The Deep Learning models are made using Keras.

1.4 METHODOLOGY

In order to accomplish the objective of our project, the first step is to do sufficient background study, so the literature survey has been performed. We have learnt various methods to build recommender systems i.e. Non Personalized Recommender System, Collaborative Filtering based Recommender System, Machine Learning and Deep Learning based Recommender Systems. But not all can be applied for our use case. So we have chosen the ones that will work.

The whole project is based on user movie ratings data, so we need to collect that data. We have collected the data from movielens website and then we have filtered and processed our data and perform exploratory data analysis to find important information regarding the data. The data is utilized in 3 forms – Pandas DataFrame, Surprise Dataframe and Sparse matrix form.

The data is then split into training and test sets and the test set is also utilized as validation set.

To make our Matrix Factorization model, we have used SVD, SVD++ and related algorithms and also keras. For our Deep Learning models, we have built Autoencoders, Matrix Factorization and Residual Learning in keras.

Finally, have calculated RMSEs for each of our recommender systems and compared them.

1.5 REPORT STRUCTURE

CHAPTER 2 –

In chapter 2, we discuss about the theory behind recommender systems. This contains the details of various types of recommendation system engines and the mathematics behind them. Also we will discuss the pros and cons of these approaches

CHAPTER 3 –

In chapter 3, we have explained the system development. This includes the information regarding the dataset, data preprocessing, exploratory data analysis and the architecture of the models that we built.

CHAPTER 4 –

Chapter 4 contains the results that we achieved after training our recommendation engines on the dataset described in chapter 3.

CHAPTER 5 –

Chapter 5 contains the conclusion of our project.

CHAPTER-2 LITERATURE SURVEY

2.1 RECOMMENDATION SYSTEMS

A recommendation system is a kind of information filtering system. By inferring from large data sets, the system's algorithm can discover precise user preferences. Once we know users' preferences, we can recommend them new, relevant products. And that includes everything from movies and music, to online shopping.

Youtube, Netflix, Amazon and Spotify are examples of recommendation systems in effect. The systems provide users with suitable suggestions based on the choices they make. The advantages of adding a recommender system include- increment in sales due to user tailored offers , improved customer experience, increased time spent on the website and much more.

Epsilon in their study on market discovered that 90% of all users find personalization attractive. Also, some 80% say that they are more likely to do business with a company if a personalized experience is provided.

Recommendation engines use the feedback of users to find new relevant items for them or for others with the assumption that as users who have made homogeneous choices in past are highly anticipated to make similar choices in forthcoming future like the example here. Recommendation engines benefit from having a many to many match between the users giving the feedback, and the items receiving the feedback. In other words, a better recommendation can be made for an item that has been given a lot of feedback, and more personalized recommendations can be given for a user that has given a lot of feedback.

Goal of Recommender systems is to predict interests of users and propose items that more likely are captivating for them. They are one of the most powerful machine learning systems that online businesses utilize so as to increase their sales.

2.2 WHY USE RECOMMENDATION SYSTEMS

Companies are utilizing recommendation system engines targeting an increase in sales as a result of user tailored offers and an improved customer experience.

Recommendations often increase search speed and makes it simpler for users to get hold of the content they like, and entertain the users with suggestions they would have otherwise never looked for.

In addition, the companies are capable to increase customer base and hold on to existing customers by sharing out emails containing links to new products that connect with the interests of the customer, or suggestions of movies and TV series that match with their preferences.

The user begins feeling familiar and accepted and are therefore highly likely to purchase additional products or utilize additional content. By knowing what a user's needs, the company gains a competitive edge and the risk of losing a customer to adversaries is reduced.

Catering that extra value to users by incorporating recommendations into systems and products is attractive. In addition, it permits companies to stay ahead of their adversaries and ultimately increase their profits.

Epsilon in their study on market discovered that 90% of all users find personalization attractive. Also, some 80% say that they are more likely to do business with a company if a personalized experience is provided.

2.3 DATA FOR RECOMMENDATION SYSTEM

The data for recommender are generally represented in the form of a matrix. Let we have n number of users, m number of products. The ratings Matrix A of can be represented as

$$A = \left[\begin{array}{c|cccc} & I1 & ,\ I2 & \ldots & In \\ \hline U1 & & & & \\ U2 & & & & \\ \ldots. & & & Aij & \\ Um & & & & \end{array}\right]$$

Where Aij = rating given by user Ui on item Ij. If Ui has not interacted with Ij, Aij is null.

The matrix A of ratings is very sparse. This is explained below

Let us suppose we have 1million users and 10K items. So the total cells of our matrix = 1M*10K = 10Billion.

Let us consider an average user rates 50 items. So the total number of ratings =

1M*50 = 50 Million

Sparsity of A = Number of ratings/Total number of cells

= 50Million/10Billion

= 0.005

2.4 NON-PERSONALIZED RECOMMENDER SYSTEMS

The first type of recommendations are called non-personalized recommendations. They are called this as they are made to all users, without taking their preferences into account. The common theme of these methods is that it doesn't matter who you are.

Some examples are Reddit, Hacker News, Google Page Rank and Amazon.

When we perform Google Search, we see the same thing as someone else performing the same search. Other example is recommending the items most frequently seen together like you can see here on Amazon. This might not select the 'best' items or items that are most suited to you, but there is a good chance you will not hate them as they are so common. This is known as stereotyped recommendation system.

Problems might arise if we try to sort by average rating naively. The problem is this doesn't take into account the number of ratings that an item has received . For example -an item that has one five star rating is probably not that better than an item that has 1000 ratings with an average of four stars.

To overcome this challenge, we damp the overall mean using features like minimum number of ratings. We then cut down our data by creating a mask of only items that have been reviewed more a fixed times in our dataset. By combining the initial work of counting occurrences with mean ratings, we can get very useful recommendations.

Apart from Stereotyped Recommendation System, another class of non-personalized recommender system use product association to recommend products. These are known as Product Association Learning based Recommenders.

In Product Association Recommenders, although not personalized, uses the knowledge of the products that a consumer is looking for and using this knowledge, it provides product recommendations. One way of providing these recommendations is to go through all the historical transactions and see what other people have most frequently bought along with the given product at the same time(like Amazon). For example, apriori based systems .

2.5 CONTENT BASED RECOMMENDATION SYSTEMS

The recommendations made by finding items with similar attributes are called content-based recommendations. For example, if a user likes movie A, and we calculate that movie A and movie B are similar, we believe the user will like book B. We can do so by comparing the attributes of our items. A big advantage of using an item's attributes over user feedback is that you can make recommendations for any items you have attribute data on. This includes even brand new items that users have not seen yet. Content-based models require us to use any available attributes to build profiles of items in a way that allows us to mathematically compare between them. This allows us for example to find the most similar items and recommend them. This is best done by encoding each item as a vector.

Example -

Let us suppose we have a MOVIE vector consisting of attributes as follows-

GENRE	DECADE	ACTOR	DIRECTOR	ETC.

And we have a USER vector consisting of attributes as follows –

GENDER	AGE	LOCATION	PAST GENRE	ETC.

We can get feature representation using these 2 vectors. Once we get feature representation, we can use classification/regression algorithm for predictions.

So, our combined vector X becomes -

GENRE	DECADE	ACTOR	DIR,		GENDER	AGE	PAST..	LOCATION	

Movie Vec. User Vec.

So, we are using meta info to predict and hence this is called content based recommendation

2.6 COLLABORATIVE FILTERING BASED RECOMMENDATION SYSTEMS

A collaborative filtering based recommendation system explores the similarity between either users or items interactions. It works around the premise that person A has similar tastes to person B and C. Once the system calculates the similarities, it provides user recommendations. Generally, users are recommended the items that similar users liked.

We can understand the user item interactions using a matrix, where each cell (i,j) represents the interaction between user i and item j.

Eg. Following is a matrix of users and the movies they like –

U1	M1	M2	M3
U2	M1	M3	M4
U3	MI		

Since U3 LIKES M1 which is also liked by U1 and U2 and U1 and U2 both like M3 also, so we will recommend M3 to U3 as there are high chances that U3 will like M3. The core idea encompassing this is - Users who agreed in the past are likely to agree in future.

There are two types of collaborative filtering systems -

1 – User user similarity based Collaborative Filtering

2 – Item item similarity based Collaborative Filtering

2.6.1 User User based Collaborative Filtering

User based recommendations are when we use the mean of the ratings the k most similar users gave an item, to suggest what rating the target user would give. This method is said to be “user-centred” as it represent users based on their interactions with items and evaluate distances between users.

Let us say we have user vectors Ui and Uj which contain ratings of users on items I1, I2 etc.

U =

I1	I2	I3	I4	I5	I6	…….

The first step of user user based collaborative filtering is to find users with homogeneous taste. Similarity between Ui and Uj can be calculated using similarity formulas. Similarity measure is such that the users with similar reactions on the same products should be considered as being near to each other. Eg. Cosine Similarity which is given by –

$$\text{Cosine (Ui, Uj)} = \frac{(\mathrm{UiT})(\mathrm{Uj})}{|Ui||Uj|}$$

We can Calculate the similarity between every pair of users in our dataset. Lets recommend new item to user Ui . We will calculate the similar users to Ui and filter out the ones with large similarities. Let U1,U2 and U3 are the most similar users to Ui. We will pick the items that are liked by U1,U2,U3 and are not yet watched by Ui. These items are the recommended items for Ui.

2.6.2 Item-Item Collaborative Filtering

To make a new recommendation to a user, the idea of item-item method is to find items similar to the ones the user already "positively" interacted with. Item-based collaborative-filtering is different, it assumes users will like items that are similar with items that the user liked before. Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be item-centred as it represent items based on interactions users had with them and evaluate distances between those items. This method is popularized and used by large companies like Amazon.

Let us consider we have item vectors Ii and Ij which contains ratings of items by users U1, U2 etc.

I =

U1	U2	U3	U4	U5	U6	…….

The first step of item based collaborative filtering is to find items with similar ratings by users. Similarity between Ii and Ij are calculated with the help of similarity formulas. Similarity measure is such that two items with similar ratings by same users should be considered as being close. Eg. Cosine Similarity which is given by –

$$\text{Cosine (Ii, Ij)} = \frac{(\text{IiT})(\text{Ij})}{|Ii||Ij|}$$

We can Calculate the similarity between every pair of movies in our dataset. Lets recommend new item to user Ui . We will calculate the similar items to Ii and filter out the ones with large similarities. Let I1,I2 and I3 are the most similar item to Ii. We will pick the items that and are not yet watched by Ui. These items are the recommended items for Ui.

2.6.3 Problems With Collaborative Filtering

- Cold start: We should have enough information (user-item interactions) for the system to work. If we setup a new e-commerce site, we cannot give recommendations until users have interacted with a significant number of items

- Adding new users/items to the system: whether it is a new user or item, we have no prior information about them since they don’t have existing interactions

- For User based collaborative filtering, the problem is that the user’s preferences may change over time and there is no way for the similarity matrix to check for the same.

- Another problem is that computing similarity scores for millions of users and products can be computationally expensive and since they need to be updated regularly, it is a major issue.

- These are not appropriate for global recommendations as the users and items end to have similar ratings regionally.

2.7 MATRIX FACTORIZATION

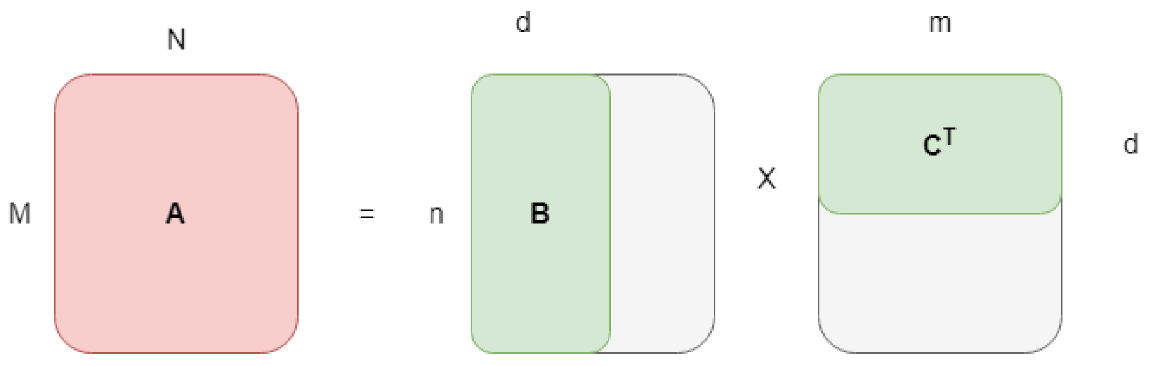
Recommender system can be represented as Matrix Completion problem. As we have seen in Section 2.3 , the data is represented in the form of sparse matrix with many empty cells. So, Matrix completion says – Given a matrix A where some values are given and many are empty, our job is to fill up empty cells with reasonable values based on values in non empty cells. So the idea behind Matrix Factorization is that we want to express the Matrix A in terms of a product of two smaller matrices.

If we decompose a matrix into product of other matrices, it is known as Matrix Factorization. Let us suppose we have our ratings matrix A, then

A = B. C   (B and C are decomposed matrices) or A = B.C.D

From mathematics point of view PCA is also matrix factorization. This means that just like PCA, we can represent the users and item vectors using latent features. The model discovers these features on its own and the users and items are mapped to the dimension with the features such that they retain the maximal information. As they are learned and not given, extracted features taken individually have a mathematical meaning but no intuitive interpretation.

The product of the user and item in latent dimension give us the rating given by the user on that item.



2.7.1 Intuition Behind Matrix Factorization

Let us consider our ratings matrix

$$R = \left[\begin{array}{c|c|c|c|c} & I1 & , I2 & \dots & In \\ \hline U1 & & & & \\ U2 & & & & \\ \dots. & & & Aij & \\ Um & & & & \end{array}\right]$$

No. of users = n , No. of items = m

$$W = \left[\begin{array}{cc} U1 & \text{----------d------------} \\ U2 & \text{-----------d-----------} \\ \dots. & \text{-----------d-----------} \\ Un & \text{-----------d-----------} \end{array}\right] n*d \qquad U = \left[\begin{array}{cc} I1 & \text{-----------d-----------} \\ I2 & \text{-----------d-----------} \\ \dots. & \text{-----------d-----------} \\ Im & \text{------------d----------} \end{array}\right] m*d$$

Rij is a cell in matrix that contains ratings on movie Ij by use Ui. Matrix R is very sparse.

Imagine $R_{n*m} = W_{n*d}. U_{d*m}{}^{T}$.

Where R = n * m matrix

W = n * d matrix

U = m * d matrix

d>=0 and d<min(m,n) (Latent dimensions)

So, $R_{ij} \sim U_i * W_j{}^{T}$ (matrix multiplication)

Our goal is to find B and C such that whenever we have a non-empty Rij , $(Rij – W_j U_i{}^{T})^2$ is minimum . So -

$$R \approx \hat{R} = W U^T$$

So, our matrix factorization has become a simple optimization problem where we have to minimize the loss between actual R and R̂

So, our cost function becomes –

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2 = \sum_{i,j \in \Omega} (r_{ij} - {w_i}^T u_j)^2$$

After adding bias, our new equation becomes

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2$$

$$\hat{r}_{ij} = {w_i}^T u_j + b_i + c_j + \mu$$

Where b_i = User bias

c_i = Movie bias

μ = Global average

Using Stochastic Gradient descent, we can minimize the loss. So, after taking derivatives for gradient descent, we get –

$$w_i = \left(\sum_{j \in \Psi_i} u_j {u_j}^T \right)^{-1} \sum_{j \in \Psi_i} (r_{ij} - b_i - c_j - \mu) u_j$$

$$u_j = \left(\sum_{i \in \Omega_j} w_i {w_i}^T \right)^{-1} \sum_{i \in \Omega_j} (r_{ij} - b_i - c_j - \mu) w_i$$

$$b_i = \frac{1}{|\Psi_i|} \sum_{j \in \Psi_i} (r_{ij} - {w_i}^T u_j - c_j - \mu)$$

$$c_j = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} (r_{ij} - {w_i}^T u_j - b_i - \mu)$$

After adding regularization parameters, our equation becomes –

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2 + \lambda(\|W\|_F{}^2 + \|U\|_F{}^2 + \|b\|_2{}^2 + \|c\|_2{}^2)$$

Thus by using SGD, we can solve Matrix Factorization.

2.7.2 Singular Value Decomposition

SVD is a matrix Factorization technique that is related to PCA.

Let us suppose we have a matrix X_{n*d} . So according to SVD, we can split this matrix as

$X_{n*d} = U_{n*n} \cdot \sum_{n*d} \cdot V_{d*d}^T$.

The columns of U are known as left singular vectors of X . The ith column of U is the ith eigen vector of XX^T

The columns of V are known as right singular vectors of X. The ith column of V is the ith eigen vector of X^TX

$\sum$ is a diagonal matrix with s1,s2,s3…..sd as diagonal elements. And these diagonal elements are known as Singular Values of X.

$$\sum = \begin{bmatrix} s1 & & & 0 \\ & s2 & & \\ & & \ldots & \\ & & & sd \\ 0 & & & 0 \end{bmatrix}$$

As we know PCA makes dimensionality reduction on the basis of eigen values. There is a relation between singular values and eigen values.

$(S_i)^2/(n-1)$ = eigen value (lambda)

The singular values that form the central matrix of SVD are roughly square root of eigen values of X

2.7.3 Matrix Factorization For Feature Engineering

Suppose we have a data set which contains the items ratings given by various users. This is a typical recommender systems problem where your job is do recommend new items to the ith user based on the previous items ith user has rated.

Let’s take n : number of users, m : number of items then our rating Matrix will be of the order of (nxm). After applying Matrix Factorization we get two matrices, user matrix of shape (nxd) and item matrix of shape (dxm).

ith row in user matrix is the ith user vector, ith column in item matrix is the ith item vector. All the vectors are Real numbers of dimension d

So we user vector Ui =

L1	L2	L3	L4	L5	..	..	..	LD

And item vector Ij =

l1	l2	l3	l4	l5	..	..	..	lD

We can concat these 2 vectors and thus our vector X becomes

L1	L2	…	…	LD	l1	l2	..	lD

Also let y = Aij (Rating by user Ui on movie Ij

So, we have X and Y and we can solve our recommendation problem using classification or regression techniques based on output.

2.7.4 Hyperparameter Tuning W.R.T Matrix Factorization

Hyperparameters are the parameters chosen by the ML engineer himself in order to recommend with effective predictions. Finding optimal hyperparameters is called hyperparameter tuning.

Let $A = B * C^T$

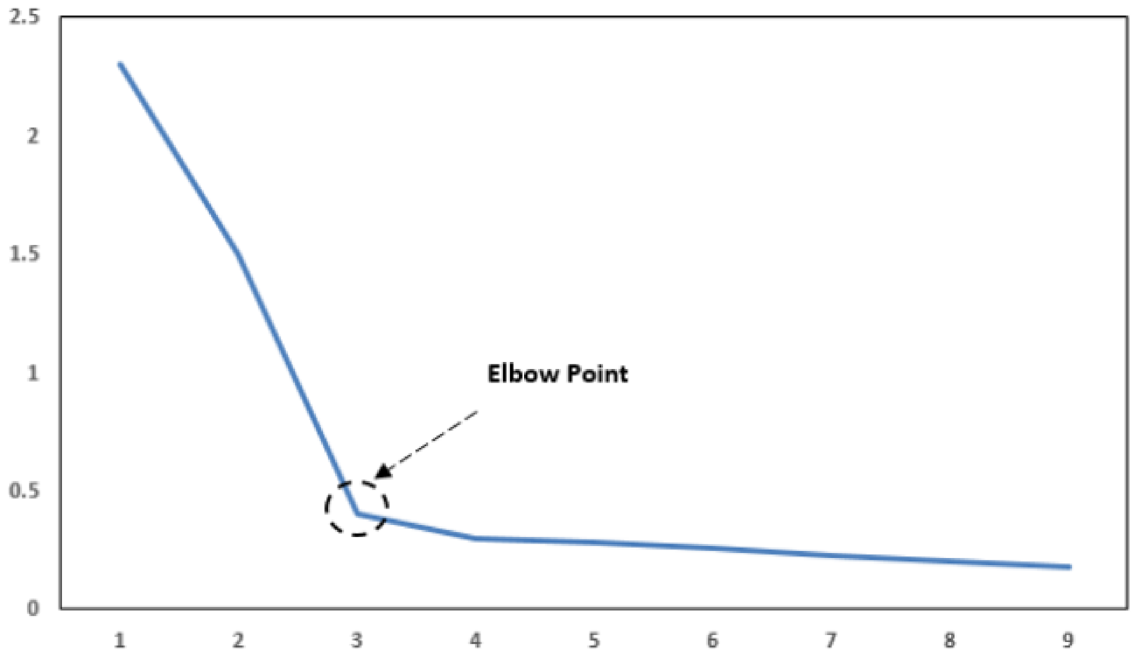
Where $A = n*m$

$B = n*d$

$C = m*d$

In case of MF, Hyperparameters are the number of latent features to extract (d).

This can be done using elbow method. A typical graph of error vs d for MF looks like –

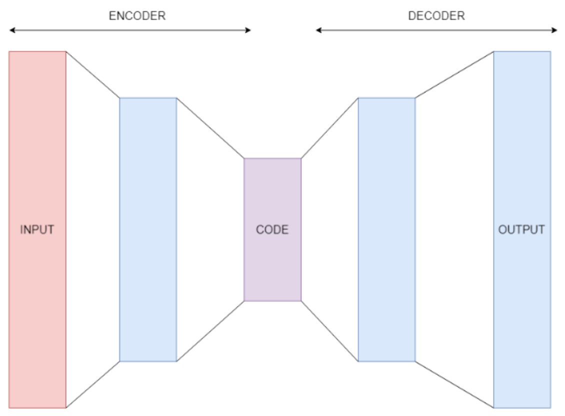


As d increases, the error decreases but after a certain point, the rate of change of error drops significantly. This is known as elbow point or inflection point.

2.8 AUTO ENCODERS

An auto encoder is just a feed forward neural network that predicts its own input . Its objective, therefore, is to minimize the error between its output and its own input. So the only difference between a general neural network and an auto encoder is that instead of passing in some target, why the target is just input itself.

Auto-encoder is a type of neural network suited for unsupervised learning tasks, including generative modeling, dimensionality reduction. It has shown its superiority in learning underlying feature representation in many domains, including computer vision, speech recognition, and language modeling.



A vanilla auto-encoder consists of an input layer, a hidden layer, and an output layer. The input data is passed into the input layer. The input layer and the hidden layer constructs an encoder. The hidden layer and the output layer constructs a decoder. The output data comes out of the output layer.

For encoding operation –

$Z = \sigma(Wx + b)$

For decoding operation –

$X' = \sigma'(W'Z + b')$

One of the applications of auto encoders is denoising auto encoders. Basically , the idea is to pass in a noisy version of our input. The job of autoencoder is to reconstruct the whole input, even though it is noisy.

So we have to do exactly the same thing for our recommender systems. Here as our noisy input, we have missing ratings. The job of autoencoder is to predict these missing ratings so as to make our input complete.

Here, each user is a sample and each feature is a movie rating. To make autoencoder more predictable, we can add more noise to the data by deleting some ratings so that the neural network will learn how to predict on actual missing ratings. This is also known as dropout regularization. The idea is to force the hidden layer to acquire more robust features

Increasing the number of hidden neurons or the number of layers improves model performance. This makes sense as expanding the dimensionality of the hidden layer allows Autoencoder to have more capacity to simulate the input features. Adding more layers to formulate a deep network can lead to slight improvement.

CHAPTER-3 SYSTEM DEVELOPMENT

3.1 ABOUT THE DATASET

“We have used Movielens 1M dataset by grouplens for our project. The GroupLens Research Project is a researchers group in University of Minnesota. Members of the GroupLens Research Project participate in many research projects synonymous to the areas of information filtering, collaborative filtering, and recommender systems. The project is headed by professors John Riedl and Joseph Konstan. The project started exploring automated collaborative filtering in 1992, but is mostly popular for its world wide trial of an automated collaborative filtering system for Usenet news in 1996. Since then the project has expanded its scope to research overall information filtering solutions, integrating in content-based methods as well as improving current collaborative filtering technology.”

The dataset comprises of 1,000,209 ratings of roughly around 3,900 movies made by 6,040 Movie Lens users who joined Movie Lens in 2000.

The ratings are stored in the file "ratings.dat" and follow the following format:

“UserID :: MovieID :: Rating :: Timestamp

- UserIDs range between 1 and 6040

- MovieIDs range between 1 and 3952

- Ratings are made on a 5-star scale (including partial ratings)

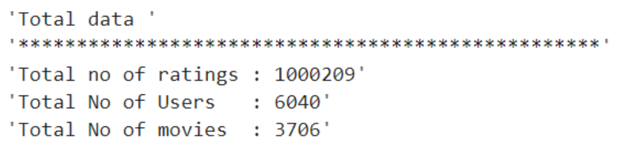
- Timestamp is represented in seconds since the epoch as returned by time(2)

- Each user has at least 20 ratings”

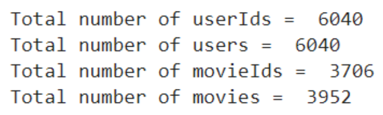
3.2 EXPLORATORY DATA ANALYSIS

The .dat file is loaded into our python notebook with the help of pd.read_table.

After loading the data, we get following numbers –



```
'Total data '
'**************************************************'
'Total no of ratings : 1000209'
'Total No of Users   : 6040'
'Total No of movies  : 3706'
```



```
Total number of userIds =  6040
Total number of users =  6040
Total number of movieIds =  3706
Total number of movies =  3952
```

This shows that there are some movies which are not present in the ratings data.

3.2.1 Checking For Nan And Duplicate Values

After checking for nan and duplicates, we find 0 nan and duplicate values

3.2.2 Information About Ratings Column

We got the following information after describing ratings column –

1 – The total no. of ratings = 1000209

2 – The mean of all ratings = 3.58

3 – The minimum rating is 1.0 and the maximum rating is 5.0

4 – The standard deviation is 1.11 and the median is 4.0

3.2.3 Information About Ratings With Respect To Users

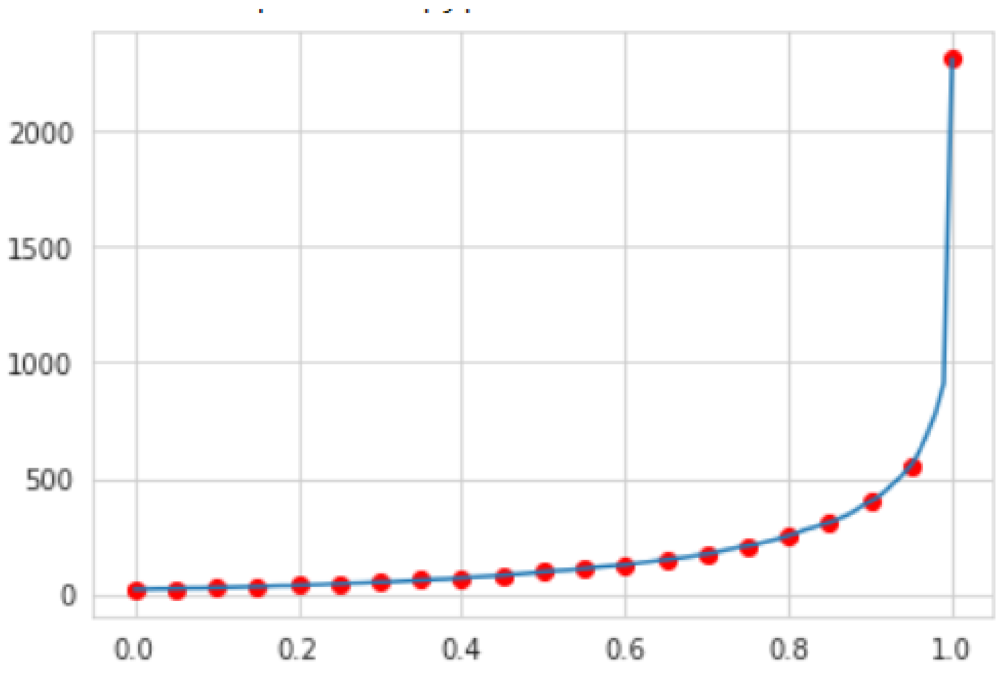
We got the following information after calculating ratings with respect to users

1 – The total number of users = 6040

2 – The average number of movies a user rates = 165.59

3 – Minimum number of movies a user rates= 20 and maximum number of movies a user rates = 2314

4 – The median number of rating by any user = 96



This is the plot of ratings vs quantitle. It is evident from the graph that there is a sharp jump in the number of movies rated by a user after 0.95. So , looking at them closely we get

```
0.95     556
0.96     616
0.97     694
0.98     785
0.99     909
1.00    2314
```

3.2.3 Information About Ratings With Respect To Movies

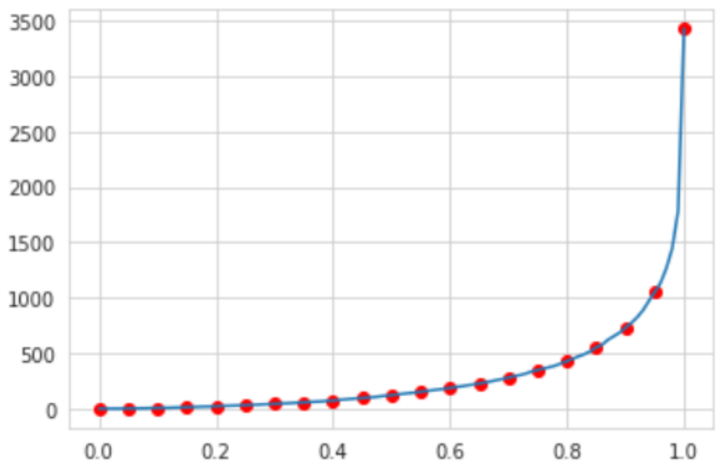
We got the following information after calculating ratings with respect to movies

1 – The total no. of movies = 3706

2 – The average no. of user who rate a movie = 269.88

3 – Minimum number of ratings a movie received= 1 and max number of ratings a movie received = 3428

4 – The median number of rating of any movie = 123



This is the plot of ratings vs quantitle. It is evident from the graph that there is a sharp jump in the number of movies rated after 0.9. So , looking at them closely we get

```
0.90     730
0.91     774
0.92     825
0.93     888
0.94     972
0.95    1052
0.96    1135
0.97    1269
0.98    1448
0.99    1785
1.00    3428
```

A few movies (the famous ones) is rated by more users.

3.3 DATA PREPARATION

Original data file was a .dat format file with fields separated with “::”.

Step 1 – Read the .dat into pandas dataframe using pd.read_table

Step 2 – Since our movieIds are non continuous, we will map them to new concurrent movieIds and store the mapping as key value pairs in a dictionary

Step 3 – Drop the timestamp column

Step 4 – Split the dataset into training and test set with 80:20 ratio split

3.3.1 Creating Sparse Matrices Of Our Data

We have used csr representation to create sparse matrix for our training and test set.

Train Matrix -

Shape of the Train matrix is : (user, movie) : (6039, 3883)

Sparsity Of Train matrix : 96.58773470766057 %

Test Matrix –

Shape of the Test matrix is : (user, movie) : (6039, 3883)

Sparsity Of Test matrix : 99.14693047854412 %

3.5 MODELS

3.5.1 Surprise Baseline Model

This is a linear model and was first proposed by Yehuda Koren in his paper "*Factor in the Neighbors: Scalable and Accurate Collaborative Filtering*". The algorithm predicts the baseline estimate for given user and item. In a real world scenario, there is a probability that certain users are more optimistic and tend to give higher ratings to movies while others are pessimistic or critiques and tend to give lower ratings to movies. This is called user bias. Also some items tend to receive higher ratings than others. This may be due to brand image or endorsements etc. This is known as item bias. So by taking these into consideration the algorithm predicting the baseline estimate for given user and item is–

$$\hat{r} = b_{ui} = \mu + b_U + b_i$$

Where-

μ : Mean of all the ratings in our train dataset.

b_u : Bias of user

b_i : Item bias (here movie biases)

Optimization function (Least Squares Problem)

The above equation is a simple optimizing problem and may be interpreted as Least square problem using SGD -

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right).$$

Where λ is regularization parameter.

3.5.2 Surprise KNNBaseline Model

This model was also proposed by Yehuda Koren in his paper "*Factor in the Neighbors: Scalable and Accurate Collaborative Filtering*". It is a basic collaborative filtering algorithm built further to our Baseline model baseline rating model. Our objective is to predict rating given by user U_i on item I_j . Using similarity measures, we can find k similar users or items. The new ratings can be predicted by taking weighed averages of these ratings. The KNN can be built using item item similarity or user user similarity.

Algorithm for KNNBaseline using User User similarity –

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - b_{vi})}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

Algorithm for KNNBaseline using Item Item similarity –

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - b_{uj})}{\sum\limits_{j \in N_u^k(j)} \text{sim}(i, j)}$$

Where

b_{ui} : Baseline estimate for user 'u' on item 'i'.

$N_i^k(u)$: Is a set containing k most similar users of user 'u' who rated the movie 'I'

sim (u, v) : Similarity among the users u and v

This can be solved as simple Optimization Problem

3.5.3 Singular Value Decomposition (SVD)

This is a type of Matrix Factorization technique. Our goal is to factorize the users matrix so as to find the latent features explaining the ratings.

The equation behind this is –

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

Where

q_i : Representing the latent features of item(movie) vector

p_u; Representing the latent features of user vector

Optimization Function with regularization:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2 \right)$$

3.5.4 SVD With Implicit Feedback From User (SVD++)

This model adds implicit feedback of users to SVD model. Lets suppose a user watches a movie. Now, irrespective of the rating he/she gave, the very fact that the user went on to see the movie shows that the target user is intrigued by similar movies. This is known as implicit feedback. On the other hand, explicit feedback of a user are the ratings. So, the SVD++ takes into account these implicit feedbacks.

The equation for SVD++ is :

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$$

Where I_u : is a set containing all the items that user u has rated

3.5.5 Matrix Factorization Using Keras

In this model we will implement Matrix Factorization using Keras. Keras is a deep learning library. For matrix factorization, both gradient descent and alternating least squares are valid training algorithms. The userIds and movieIds are discrete numerical values. We can embed every userId and movieId to a corresponding feature vector.

We do this using Keras Embedding layers –

User embedding layer = Embedding(N,K)

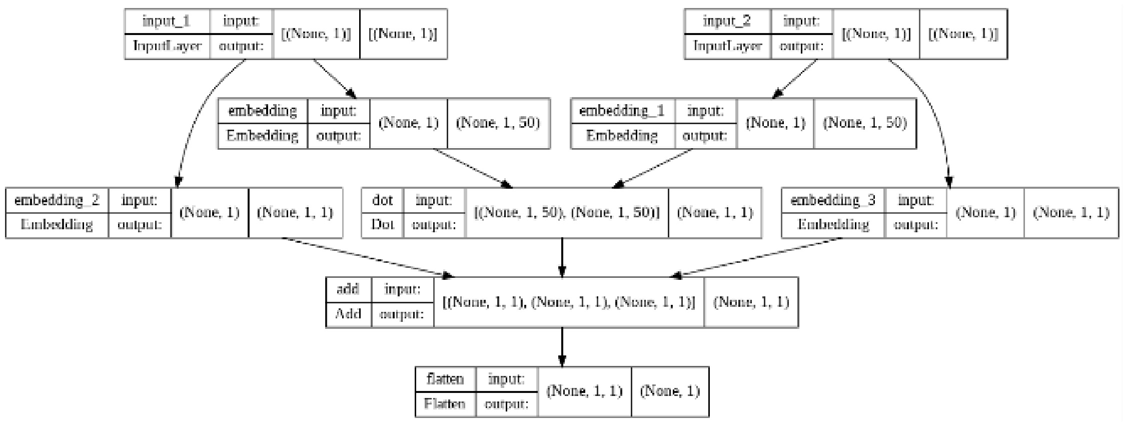
Movie embedding layer = Embedding(M,k)

We then dot product these to get rating given by a user on a movie.

We also add bias terms to this rating to get more accurate results

Overall we are calculating – $r_{ij} = dot(w_i,u_{j)} + b_i + c_j + \mu$

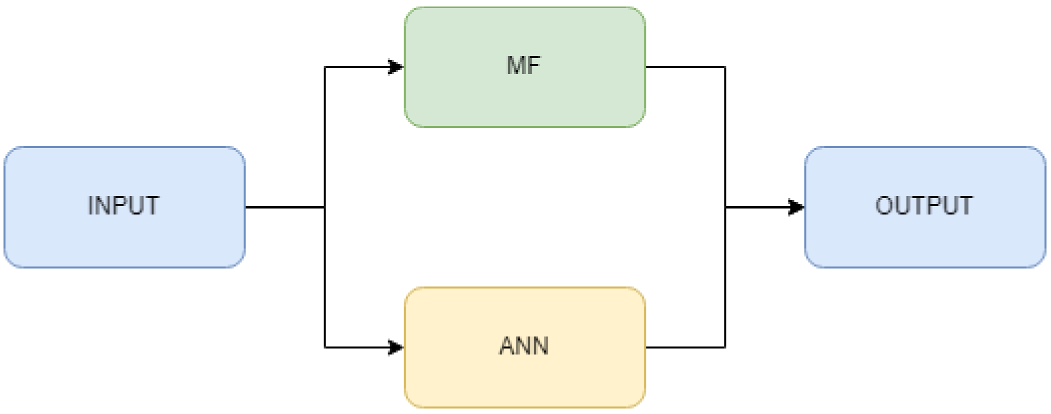
So, the architecture of our model is -


input_1 | InputLayer | input: | output: | [(None, 1)] | [(None, 1)]
input_2 | InputLayer | input: | output: | [(None, 1)] | [(None, 1)]
embedding | Embedding | input: | output: | (None, 1) | (None, 1, 50)
embedding_1 | Embedding | input: | output: | (None, 1) | (None, 1, 50)
embedding_2 | Embedding | input: | output: | (None, 1) | (None, 1, 1)
dot | Dot | input: | output: | [(None, 1, 50), (None, 1, 50)] | (None, 1, 1)
embedding_3 | Embedding | input: | output: | (None, 1) | (None, 1, 1)
add | Add | input: | output: | [(None, 1, 1), (None, 1, 1), (None, 1, 1)] | (None, 1, 1)
flatten | Flatten | input: | output: | (None, 1, 1) | (None, 1)

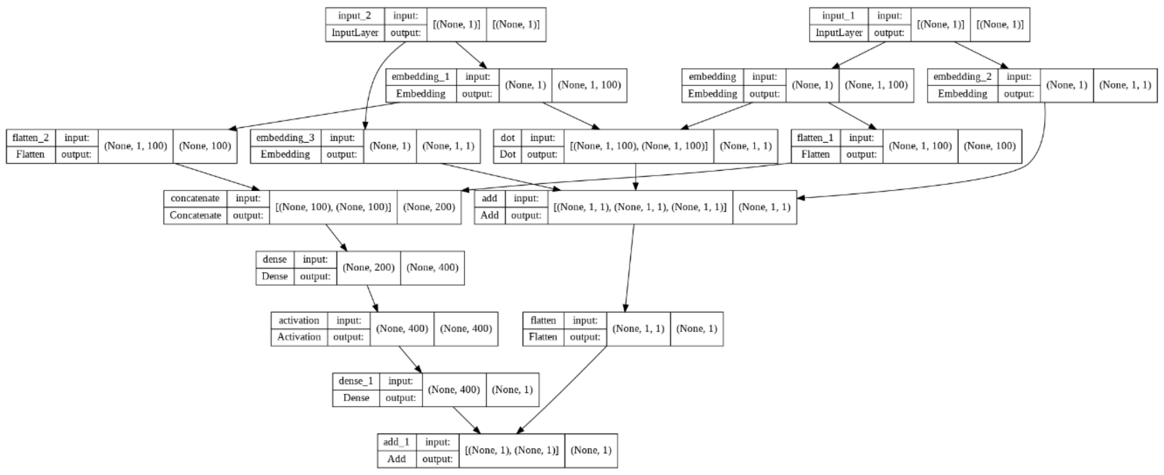
We have used Keras Callbacks method to save the model at best perfoming epoch and stop the training is model starts overfitting.

3.5.6 Residual Network For Recommendation

The idea behind using residual network for recommendation is – The Matrix Factorization model that we built is linear which means it wont learn non linear pattern. So we built branches in out neural network such that MF and Deep Neural Network work in parallel. We will then add the 2 parts using ADD() layer.



The final architecture of our model is as follows –

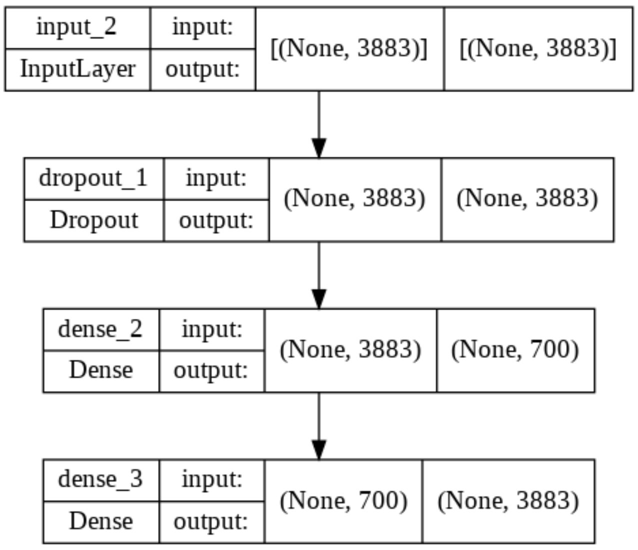

input_2 | InputLayer | input: | output: | [(None, 1)] | [(None, 1)]
input_1 | InputLayer | input: | output: | [(None, 1)] | [(None, 1)]
embedding_1 | Embedding | input: | output: | (None, 1) | (None, 1, 100)
embedding | Embedding | input: | output: | (None, 1) | (None, 1, 100)
embedding_2 | Embedding | input: | output: | (None, 1) | (None, 1, 1)
flatten_2 | Flatten | input: | output: | (None, 1, 100) | (None, 100)
embedding_3 | Embedding | input: | output: | (None, 1) | (None, 1, 1)
dot | Dot | input: | output: | [(None, 1, 100), (None, 1, 100)] | (None, 1, 1)
flatten_1 | Flatten | input: | output: | (None, 1, 100) | (None, 100)
concatenate | Concatenate | input: | output: | [(None, 100), (None, 100)] | (None, 200)
add | Add | input: | output: | [(None, 1, 1), (None, 1, 1), (None, 1, 1)] | (None, 1, 1)
dense | Dense | input: | output: | (None, 200) | (None, 400)
activation | Activation | input: | output: | (None, 400) | (None, 400)
flatten | Flatten | input: | output: | (None, 1, 1) | (None, 1)
dense_1 | Dense | input: | output: | (None, 400) | (None, 1)
add_1 | Add | input: | output: | [(None, 1), (None, 1)] | (None, 1)

3.5.7 Autoencoder For Recommendations Using Keras.

Autoencoders are models that reproduce the input itself. The error metrics is such that the difference between input and regenerated input is minimum. First we create a sparse matrix using scipy sparse matrix for our ratings data such that each user corresponds to a sample and each movie rating corresponds to a feature. This is faster than previous deep learning techniques because earlier we were treating each rating as a unique sample. But, here we are taking each user as a sample. Finally we add noise to our input using Dropout() Layer.

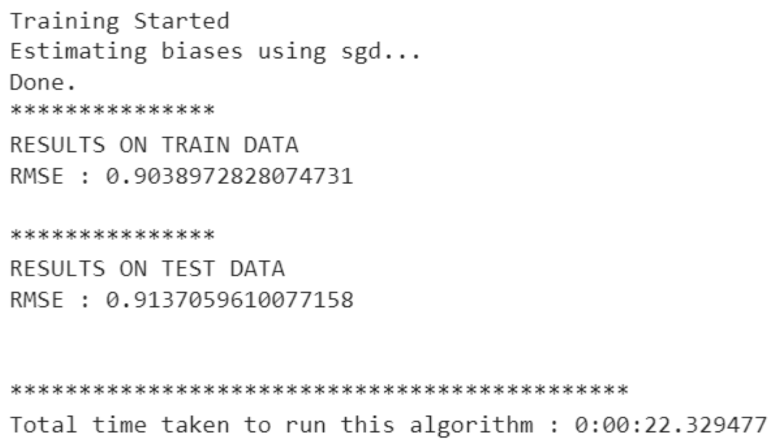
Keras does not recognize sparse matrix so we will generate batches of input from sparse matrix separately.

The final architecture of our model is as follows –



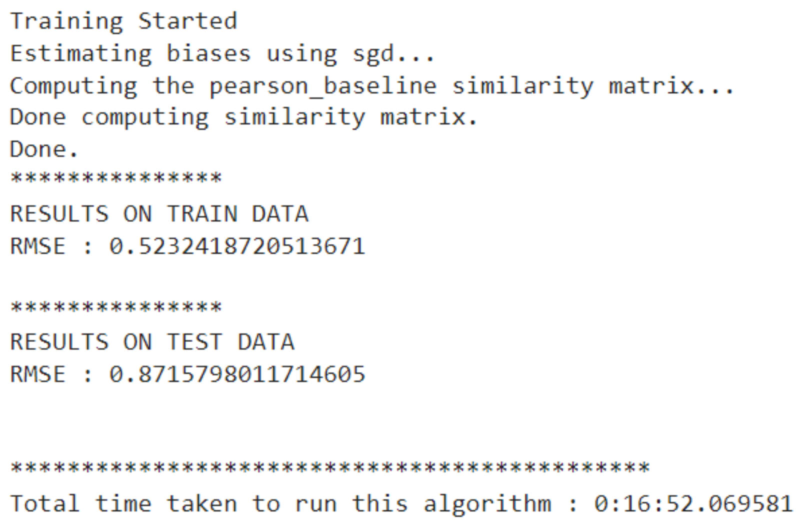
CHAPTER-4 PERFORMANCE ANALYSIS

4.1 MODEL-1 SURPRISE BASELINE PREDICTOR



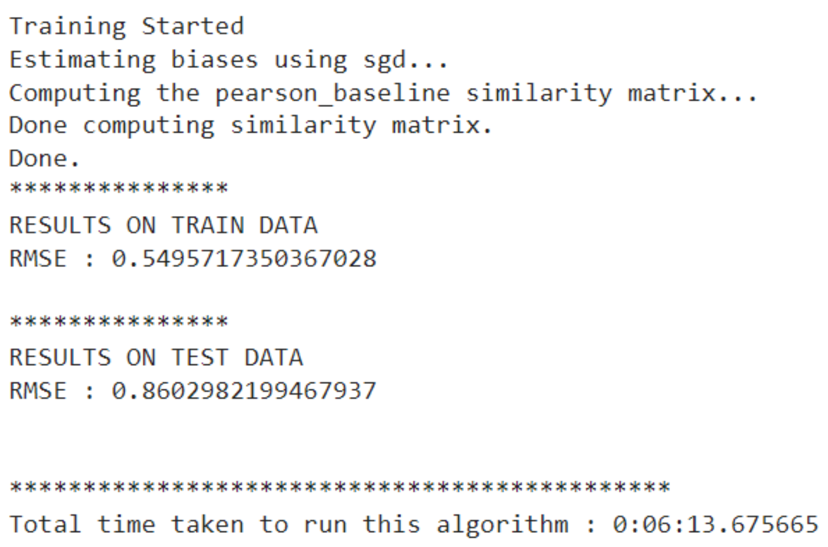
```
Training Started
Estimating biases using sgd...
Done.
***************
RESULTS ON TRAIN DATA
RMSE : 0.9038972828074731

***************
RESULTS ON TEST DATA
RMSE : 0.9137059610077158


*********************************************
Total time taken to run this algorithm : 0:00:22.329477
```

4.2 SURPRISE KNNBASELINE USING USER-USER SIMILAR



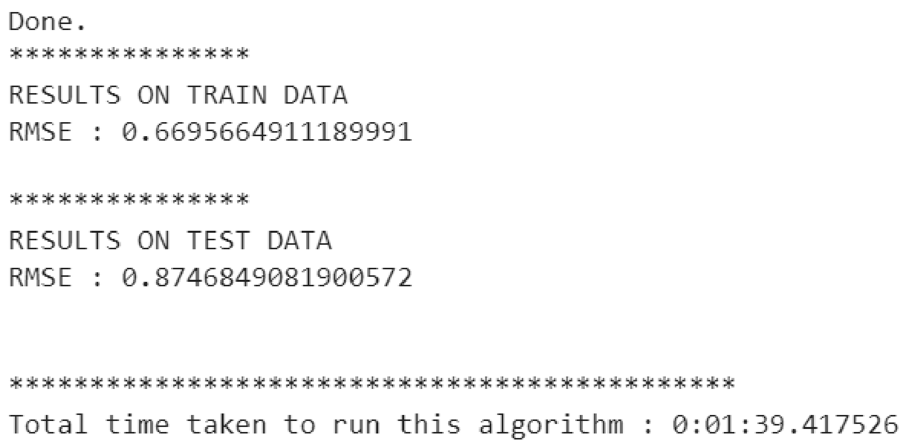
```
Training Started
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done.
***************
RESULTS ON TRAIN DATA
RMSE : 0.5232418720513671

***************
RESULTS ON TEST DATA
RMSE : 0.8715798011714605


*********************************************
Total time taken to run this algorithm : 0:16:52.069581
```

4.3 SURPRISE KNNBASELINE USING MOVIE-MOVIE SIMILAR



```
Training Started
Estimating biases using sgd...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Done.
***************
RESULTS ON TRAIN DATA
RMSE : 0.5495717350367028

***************
RESULTS ON TEST DATA
RMSE : 0.8602982199467937


*********************************************
Total time taken to run this algorithm : 0:06:13.675665
```

4.4 SURPRISE SVD MODEL



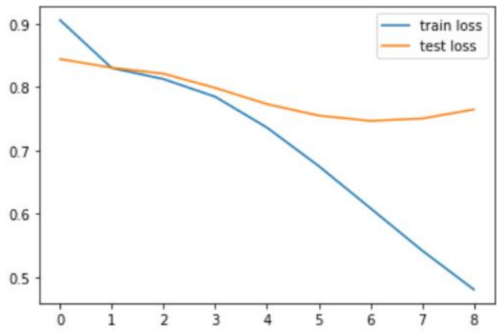
```
Done.
***************
RESULTS ON TRAIN DATA
RMSE : 0.6695664911189991

***************
RESULTS ON TEST DATA
RMSE : 0.8746849081900572


*********************************************
Total time taken to run this algorithm : 0:01:39.417526
```

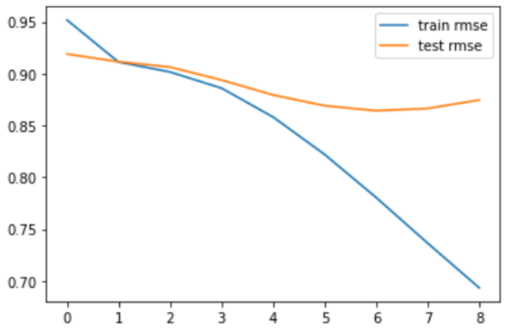
4.5 SURPRISE SVD++ MODEL

```
Done.
***************
RESULTS ON TRAIN DATA
RMSE : 0.6791759920424655

***************
RESULTS ON TEST DATA
RMSE : 0.8688528325768125


*********************************************
Total time taken to run this algorithm : 1:45:05.251886
```

4.6 MATRIX FACTORIZATION USING KERAS

```
Epoch 1/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.9059 - root_mean_squared_error: 0.9518 - val_loss: 0.8445 - val_root_mean_squared_error: 0.9189
Epoch 2/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.8303 - root_mean_squared_error: 0.9112 - val_loss: 0.8308 - val_root_mean_squared_error: 0.9115
Epoch 3/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.8129 - root_mean_squared_error: 0.9016 - val_loss: 0.8214 - val_root_mean_squared_error: 0.9063
Epoch 4/15
6252/6252 [==============================] - 22s 4ms/step - loss: 0.7850 - root_mean_squared_error: 0.8860 - val_loss: 0.7989 - val_root_mean_squared_error: 0.8938
Epoch 5/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.7364 - root_mean_squared_error: 0.8581 - val_loss: 0.7734 - val_root_mean_squared_error: 0.8794
Epoch 6/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.6757 - root_mean_squared_error: 0.8220 - val_loss: 0.7552 - val_root_mean_squared_error: 0.8690
Epoch 7/15
6252/6252 [==============================] - 21s 3ms/step - loss: 0.6091 - root_mean_squared_error: 0.7805 - val_loss: 0.7469 - val_root_mean_squared_error: 0.8642
Epoch 8/15
6252/6252 [==============================] - 20s 3ms/step - loss: 0.5423 - root_mean_squared_error: 0.7364 - val_loss: 0.7506 - val_root_mean_squared_error: 0.8664
Epoch 9/15
6252/6252 [==============================] - 20s 3ms/step - loss: 0.4807 - root_mean_squared_error: 0.6933 - val_loss: 0.7649 - val_root_mean_squared_error: 0.8746
```

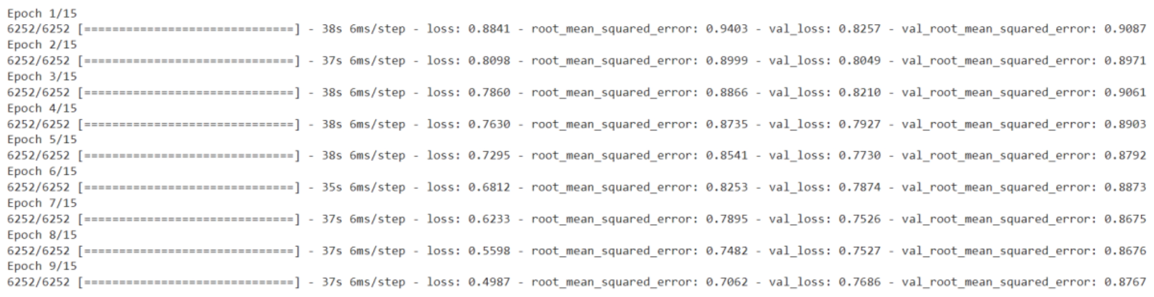




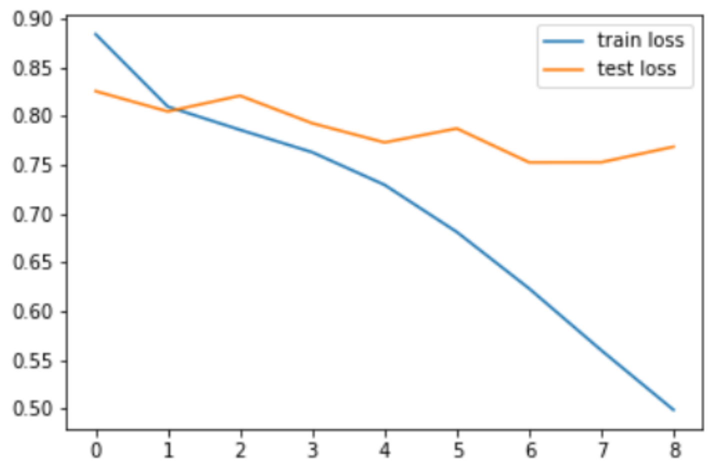
Test Data Score

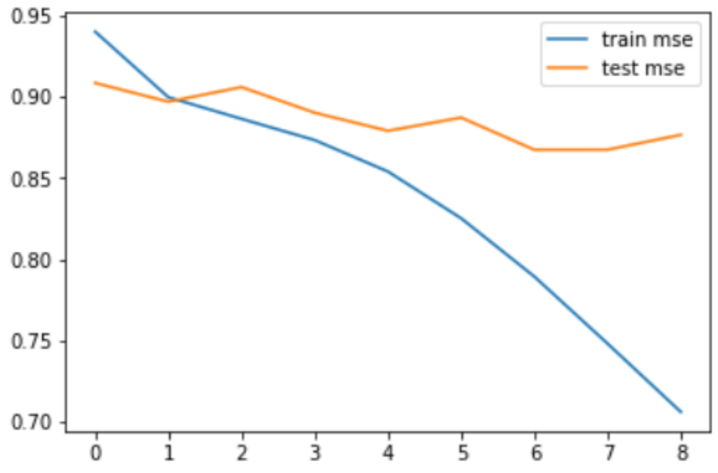
```
6252/6252 [==============================] - 13s 2ms/step - loss: 0.7469 - root_mean_squared_error: 0.8642
[0.7468881011009216, 0.8642268776893616]
```

4.7 RESIDUAL NETWORK USING KERAS

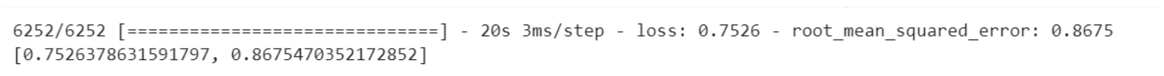


```
Epoch 1/15
6252/6252 [==============================] - 38s 6ms/step - loss: 0.8841 - root_mean_squared_error: 0.9403 - val_loss: 0.8257 - val_root_mean_squared_error: 0.9087
Epoch 2/15
6252/6252 [==============================] - 37s 6ms/step - loss: 0.8098 - root_mean_squared_error: 0.8999 - val_loss: 0.8049 - val_root_mean_squared_error: 0.8971
Epoch 3/15
6252/6252 [==============================] - 38s 6ms/step - loss: 0.7860 - root_mean_squared_error: 0.8866 - val_loss: 0.8210 - val_root_mean_squared_error: 0.9061
Epoch 4/15
6252/6252 [==============================] - 38s 6ms/step - loss: 0.7630 - root_mean_squared_error: 0.8735 - val_loss: 0.7927 - val_root_mean_squared_error: 0.8903
Epoch 5/15
6252/6252 [==============================] - 38s 6ms/step - loss: 0.7295 - root_mean_squared_error: 0.8541 - val_loss: 0.7730 - val_root_mean_squared_error: 0.8792
Epoch 6/15
6252/6252 [==============================] - 35s 6ms/step - loss: 0.6812 - root_mean_squared_error: 0.8253 - val_loss: 0.7874 - val_root_mean_squared_error: 0.8873
Epoch 7/15
6252/6252 [==============================] - 37s 6ms/step - loss: 0.6233 - root_mean_squared_error: 0.7895 - val_loss: 0.7526 - val_root_mean_squared_error: 0.8675
Epoch 8/15
6252/6252 [==============================] - 37s 6ms/step - loss: 0.5598 - root_mean_squared_error: 0.7482 - val_loss: 0.7527 - val_root_mean_squared_error: 0.8676
Epoch 9/15
6252/6252 [==============================] - 37s 6ms/step - loss: 0.4987 - root_mean_squared_error: 0.7062 - val_loss: 0.7686 - val_root_mean_squared_error: 0.8767
```





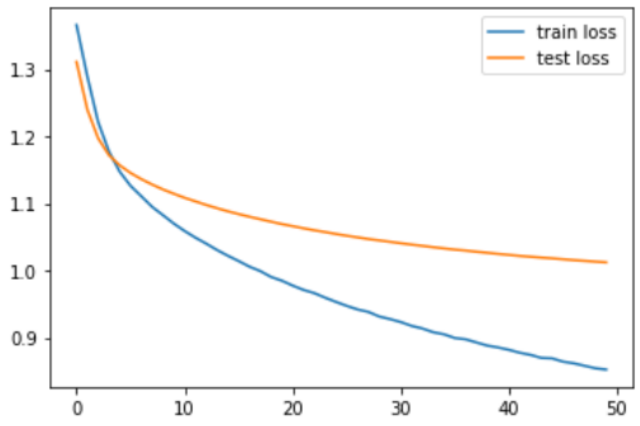
Test Data Scores –

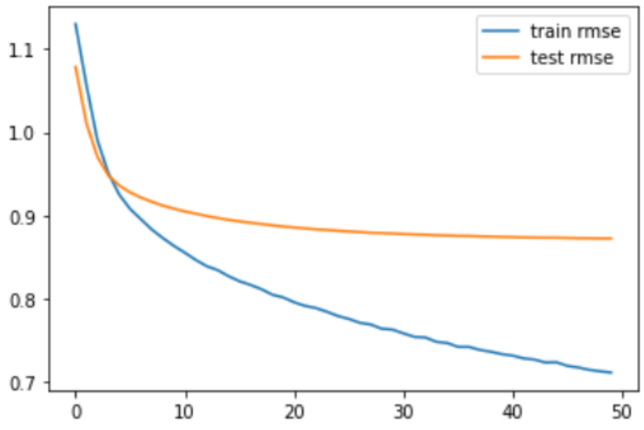


```
6252/6252 [==============================] - 20s 3ms/step - loss: 0.7526 - root_mean_squared_error: 0.8675
[0.7526378631591797, 0.8675470352172852]
```

4.8 AUTOENCODER RECOMMENDER SYSTEM USING KERAS

```
Epoch 26/30
48/48 [==============================] - 1s 22ms/step - loss: 0.9482 - custom_loss: 0.7758 - val_loss: 1.0516 - val_custom_loss: 0.8807
Epoch 27/30
48/48 [==============================] - 1s 25ms/step - loss: 0.9426 - custom_loss: 0.7729 - val_loss: 1.0493 - val_custom_loss: 0.8802
Epoch 28/30
48/48 [==============================] - 1s 25ms/step - loss: 0.9371 - custom_loss: 0.7687 - val_loss: 1.0470 - val_custom_loss: 0.8796
Epoch 29/30
48/48 [==============================] - 1s 22ms/step - loss: 0.9319 - custom_loss: 0.7666 - val_loss: 1.0448 - val_custom_loss: 0.8790
Epoch 30/30
48/48 [==============================] - 1s 24ms/step - loss: 0.9269 - custom_loss: 0.7618 - val_loss: 1.0426 - val_custom_loss: 0.8783
dict_keys(['loss', 'custom_loss', 'val_loss', 'val_custom_loss'])
```





Test Data Score –

```
48/48 [==============================] - 1s 12ms/step - loss: 1.0426 - custom_loss: 0.8783
[1.0426123142242432, 0.8783197402954102]
```

4.9 COMPARISION OF MODELS

MODEL	TEST RMSE
SURPRISE BASELINE MODEL	**0.913**
SURPRISE KNNBASELINE USING USER-USER SIMILAR	**0.871**
SURPRISE KNNBASELINE USING MOVIE-MOVIE SIMILAR	**0.860**
SURPRISE SVD MODEL	**0.874**
SURPRISE SVD++ MODEL	**0.868**
MATRIX FACTORISATION USING KERAS	**0.864**
RESIDUAL NETWORK USING KERAS	**0.867**
AUTOENCODER USING KERAS	**0.878**

So, the best performing model is Surprise KNN Baseline with movie movie similarity with RMSE = 0.860

CHAPTER-5 CONCLUSION

Recommender systems are very powerful tools that are very important for todays businesses. They help businesses by personalizing choices of customers. Also, since the number of products are increasing at an exponential rate, we need to recommend only the best ones.

In this project , we learned about various types of recommender system engines and their pros and cons. We learnt about Non personalized recommender systems, Collaborative filtering based approaches, Matrix Factorization, Surprise Library and Deep Learning based recommender systems. Not only this, we learnt about Auto Encoders which are another powerful models and can be applied in various fields Eg Computer Vision. We used Movielens 1M for training and testing. We collected the data, cleaned the data, did some EDA and finally built 8 models using Matrix Factorization, Surprise Library and Deep Learning. We achieved the best RMSE of 0.860 with Surprise KNN Baseline Predictor with User User Similarities.

These concepts can be applied to build other recommender systems with different products ad different rating methods.

REFERENCES

[1] Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi: 10.1109/MC.2009.263.

[2] Koren, Y. 2010. Factor in the neighbors: Scalable and accurate collaborative filtering. ACM Trans. Knowl. Discov. Data. 4, 1, Article 1 (January 2010), 24 pages

[3] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. AutoRec: Autoencoders Meet Collaborative Filtering. In Proceedings of the 24th International Conference on World Wide Web (WWW '15 Companion). Association for Computing Machinery, New York, NY, USA, 111–112.

[4] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted Boltzmann machines for collaborative filtering. In ICML, 2007.

[5] Netflix Recommendations: Beyond the 5 stars (Part 1) by Xavier Amatriain and Justin Basilico

[6] Netflix Recommendations: Beyond the 5 stars (Part 2) by Xavier Amatriain and Justin Basilico

[7] http://surprise.readthedocs.io/en/stable/getting_started.html

[8] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. 1999. 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). Association for Computing Machinery, New York, NY, USA, 230–237. https://doi.org/10.1145/312624.312682

[9] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872