Movie Recommendation System

Ashish Sirohi
1211009485
asirohi1@asu.edu

Dhrumil Parmar
1213179381
dmparmar@asu.edu

Jayanth Kumar Melinavolagerehalli Jayaramaiah
1212387590
jmelinav@asu.edu

Naitik Shah
1213166628
nrshah11@asu.edu

Prit Sheth
1213203392
prsheth1@asu.edu

Priyekant Aghi
1211211206
paghi@asu.edu

Shaishavkumar Jogani
1212392985
sjagoni@asu.edu

Swaraj Patel
1213099067
srpate19@asu.edu

Abstract

Through this paper, we have implemented and compared various models for the movie recommendation systems. Our project is inspired from the Netflix Prize, an open coding challenge organized by Netflix (an online DVD-rental and video streaming service) in 2009, to come up with a best collaborative filtering algorithm which can predict the rating for a movie just based on the previous rating and without any other information of the user or the movie. In our paper, we have discussed 4 different approaches i.e. A baseline Model, Item-Item Collaborative Filtering, Matrix Factorization (Latent Factor Model) and Neural Collaborative Filtering. We have used the 20M as well as 100K Movie lens dataset for the training and the testing our models and comparing their results to see how to stack up against each other.

1 Introduction

We are living in an era where Internet is loaded with a lots of content ranging from media to products to service and it has become important to push the content back of queue if it is not relevant to a particular user in order to improve the user experience. Each of the product/service offering organizations wants to customize the service experience for each user so that they can lure the user to come back to their platform and re-use the service. For e.g. Amazon uses a recommendation system which tries to recommend the products based on some recent purchase or based on some other activities of a particular user. Same way, Netflix uses a recommendation system [4] which recommends the movies to an user based on his past interest and several other factors. All these major organization collects lots of data for each of their user so that they can use this collected data to make the user experience more personalize. In this paper, we have discussed four such models/approaches: Baseline, Item-Item Collaborative Filtering [8], Latent Factor Model [5], Neural Collaborative Filtering [2], which use such collected data and try to come up with some recommendations for a particular user. We have used 20M Movie Lens [1] dataset for our result comparison across fours different

models. The rest of the paper is organized as follows: Problem Statement, Dataset, Solution to the above discussed problem, Results and comparison between different models, Conclusion & Future work and References along with the list of people who contributed to this project.

2 Problem Statement

Personalization is a hot keyword these days and almost all companies whether it's a service offering company or a product offering organization, to present their customers with the most personalized experience by recommending them the service and the products which are most relevant to them. Recommendation systems are playing huge role in the growth of an organization as a good recommendation system can help in selling of more services/products but at the same time a bad recommendation system can flee the users from their platform. In this project we will do a comparative study and implementation on Movie Recommendation System to explore the following existing algorithms:

- Baseline Model
- Item-Item Collaborative Filtering
- Matrix Factorization (Latent Factor Model)
- Neural Collaborative Filtering

3 Dataset

For experiments we are using Movie lens 20M dataset which is considered to be a stable benchmark dataset for movie recommendation model testing. It contains 20000263 user ratings and 465564 tag applications across 27278 movies. These data was created by 138493 users between January 09, 1995 and March 31, 2015. All selected users had rated at least 20 movies. The format of the dataset is a set of comma saperated files(csv) : movies.csv, rating.csv, tags.csv

- **rating.csv**: This file contains ratings of user for the movies and the timestamp of when it was rated. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:

  $$userId, movieId, rating, timestamp$$

  The lines within this file are ordered first by userId, then, within user, by movieId. Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars).
- **movies.csv**: This file contains list of movies, its ID and genres.

  $$movieId, title, genres$$
- **tags.csv**: Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user. Each line of this file after the header row represents one tag applied to one movie by one user, and has the following format:

  $$userId, movieId, tag, timestamp$$

  The lines within this file are ordered first by userId, then, within user, by movieId.

4 Solution

4.1 Baseline Model

This is the first model we used which is simplest yet effective one. The Baseline Estimate is the process where the user's inclination to rate movies will be considered based on the global mean of the movies. The prediction of the new movie is based on three major factors, i.e. users inclination for the movie (rating) compared to the global mean, global mean sum, and the mean of all other users rating compared to the global mean.

$$r_{xj} = b_{xi} = \mu + b_x + b_i$$

where

- r_{xi} = Rating of user x on item i
- b_{xi} = Global baseline estimate on user x and item i
- μ = Overall mean movie rating
- b_x = Rating deviation of user x
  User x deviation= (avg. rating of user x across all movies user x has rated) - μ
- b_i Average movie rating given by all user.
  b_i = (avg. rating of movie i) - μ

As we can see in the above equation that b_x measures how the user is deviated on giving ratings. This will check if the user is more lenient or critique while giving the rating. Next, b_i measures how the movie is performing among all the users. It takes the mean of ratings given by all users on the movie.

4.2 Item-Item Collaborative Filtering

So far the basic model explained above uses the current user ranked items to predict the rank of the new movie by a particular user. This model aims at finding the users rating by using the similarity between movies. As user tags describe the movies better than the genres of the movie, we are considering user tag for calculating similarities. For a given user, the model will predict the users rating based on the similarities between movies user has rated and the movie to be rated. The below section explains how this is achieved.

4.2.1 Term Frequency

Term Frequency (TF) is used for ranking the most relevant documents for a query string using the terms that appear in the doucment [9]. For example; let's say the string being searched for is "the harry potter", the document retrieval using simple term frequency will rank the document with the words in the query string, appearing more number of times in them as the most relevant document. The raw count of the words is the most basic term frequency measure, however there are other variation of it. The one that is used in the project is raw count with total words in the document as a normalizing factor. Below is the formula for Term frequency.

$$TF(t,d) = \frac{C(t,d)}{\sum_{t' \in d} C(t',d)}, \text{ where } C(t,d) \text{ is count of term t appears in document d}$$

4.2.2 Inverse Document Frequency

There are certain words that appear in all the document, for example the words like 'the', and these certain words are not relevant for the query. For e.g. 'The Harry Potter', since word 'the' appears many times, they will be given higher ranking than they should get. To avoid this, inverse document fequency helps in identifying these common terms and assigns low weightage to it. If a term appears only once in a document, it will be given highest value inferring that specific word discriminates the document from the others. Below is the formula for inverse document frequency:

$$IDF(t,D) = log \frac{N}{|\{d \in D : t \in d\}|},$$

where,

- N is total number of document
- t is the term in document d
- D is the set of all documents

4.2.3 Calculating Item-item similarity

For calculating movie-movie similarity, all the user tags and movie genres has been considered. We follow following steps.

1. For calculating term frequency and inverse document frequency, the user tags for the movies are used as the terms of a document.
2. As a first step in the process, the data cleaning is done to consider the genres as all movies may not have tags.
3. The data format contains movies and genres in a csv and movies and corresponding user tags in another csv file and for movies which belong to multiple genres, genres are concatenated using a delimiter.
4. To join tags and genres, first, if a movie has multiple genres, multiple rows are created for a movie with each genres and tags, and those multiple rows are appended to the same csv.
5. To calculate term frequency, a counter column is added, and group by movie_id and tag is executed to get the counter for each tag for a movie.
6. The data frame is pivoted to form movies as rows and tags for the movies as column. After this step Cell(i,j) contains count for tag j for movie i.
7. From this data frame, term frequency is calculated using the formula mentioned above.
8. Inverse document frequency is calculated using the formula mentioned above.
9. Term frequency (TF) and Inverse document frequency (IDF) are multiplied to form tf-idf values for each movie tag pair in the data frame.
10. To remove the noisy labels from the data, Singular value decomposition is applied on the data.
11. On this data, the cosine similarity between the movies is calculated using the formula

$$s_{ij} = \cos(\theta) = \frac{\sum_{i=1}^{n} A_i * B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

12. The Movie-Movie similarity matrix will be used to predicting user ranks.

4.2.4 Rank Prediction

1. To predict ranking of a movie by a User u, a movie is picked from the test data set.
2. For this movie top ten similar movies will be picked from the similarity matrix.
3. The top ten movies picked should also satisfy a condition that its been ranked by the user u.
4. Each similarity value of these ten movies is weighed by the difference between the global baseline value (Baseline estimate of the rank to this movie) and user u's rank for the movie, this weighted sum is normalized by sum of the similarities.
5. The value calculated in the step 4 is added to the baseline estimate.
6. The following formula describes the ranking prediction.

$$r_{xj} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \times (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

where

- r_{xi} is rank of movie i give by user x
- b_{xi} is global baseline estimate for movie i of user x
- r_{xj} is rank of movie j give by user x
- b_{xj} is global baseline estimate for movie j of user x
- $N(i;x)$ is a set of movies rated by x which are similar to movie i
- s_{ij} is cosine similarity between movie i and movie j

4.3 Matrix Factorization (Latent Factor Model)

So far we have discussed two different models and have seen a significant improvement in the recommendation model as CF takes many other things into consideration while recommending rather than just average across users and across movie which is the case for base line model. In this section, we will be discussing our third approach i.e. Matrix Factorization also known as Latent Factor Model [5]. As the name suggests, matrix factorization is a method to factorize a matrix into two or more sub matrices. One such factorization method is Singular Value Decomposition(SVD) where we factorize the matrix into three sub matrices. The idea behind our approach is that we can discover some latent features underlying the interactions between two different entities(in our case, it's users and items). For example, if two users gives high ratings to a certain movie if they both like the actors or actresses in the movie, or if the movie is an comedy movie, which is a genre preferred by both users. So, if we can discover these latent features, they can play a big role in the prediction of a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item. So, Matrix factorization makes sense for our dataset and that's why we have chosen this approach as one of our model for comparison. We have used a modified version of SVD for the factorization i.e. decomposition of the user-item matrix (let's say matrix R, also known as utility Matrix (Fig 1) of size $|U| \times |I|$ where U is the set of users and I is the set of items) into two separate components, let's say P (of size $|U| \times |K|$) & Q (of size $|I| \times |K|$ where K is the no of latent feature) using Singular Value Decomposition(SVD) approach rather than into traditional three components. Next, we will fill the missing values in the matrix as 0 and optimize the Q & P components using RMSE and gradient descent until convergence.

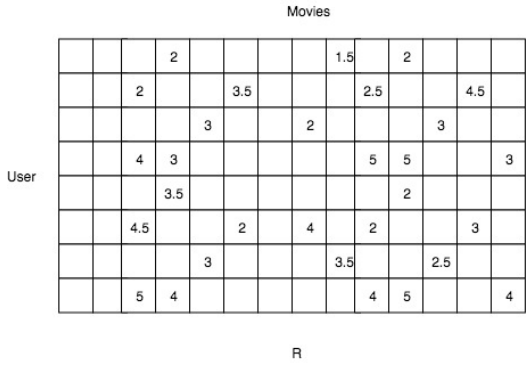

Figure 1: Utility Matrix

We know SVD produces three components i.e.

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

In our modified SVD version, we can think of U as Q and ΣV^T as P^T i.e.

$$R \approx Q \times P^T$$

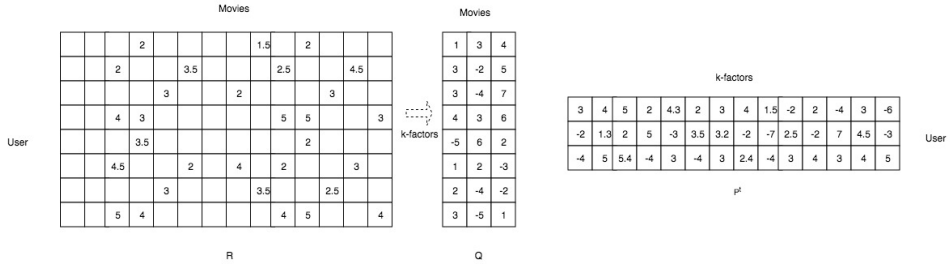

Figure 2: $R \approx Q \times P^T$

From Fig 2, we can see that there are lots of empty cells i.e. some particular user hasn't given any rating to some particular movie. As SVD doesn't work on matrices with empty cells, we need to fill those empty cells somehow and then apply the SVD on it. We have filled the missing values in our utility matrix R with 0 but for our optimization problem we will only consider the known rating instead of all the cells in the matrix. So, at each iteration we will find the error by comparing the actual rating and the predicted rating in that iteration and will try optimize the value of Q and P by iteratively minimizing the error between the predicted rating and the actual rating for a particular cell (Only known cells) in the matrix. Once Q and P are optimized, we can use these matrices to predict the values of unknown ratings in the test set. Here is the optimization formula:

$$\text{J(P, Q)} = min_{P,Q} \sum_{i,j \in R} (r_{ij} - \hat{r}_{ij})^2$$

where,

$$\hat{r}_{ij} = p_i^T \times q_i = \sum_{k=1}^{k} p_{ik} q_{kj}$$

In Fig. 3, We are trying to show how our Latent Factor Model works in a nutshell. We took the utility matrix R and extracted the list of all the movies and users from the data and mapped them to a 2 - dimensional space. The axis of this latent subspace are called factors which we find through SVD. The idea behind this is that the users and movies are mapped to the axis and these axis are the axis of variation and every movie and user is a data point in this space. In this representation some of the users and movies are closer to each other whereas some other movies and users are far apart. The users who are closer to a certain movie liked that movie and if the user is far from a movie he/she didn't like the move that much.

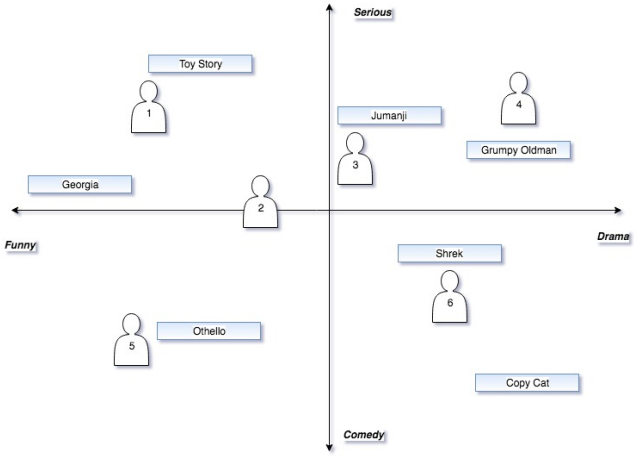

Figure 3: Latent factor explanation[5]

Now that we have our optimization function, we need to determine a way to find the P and Q to minimize the error. One such way is, initialize the P and Q randomly and get their product in some matrix M and find the difference from the original matrix R, try to minimize this difference (Let's say error e) in each iteration until convergence. We call this method gradient descent in which we try to find the local minimum of the difference i.e.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2$$

In order the minimize the above error and obtain P and Q at each iteration, we need to differentiate the above equation, once with respect to P and then with respect to Q i.e.

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2\ (r_{ij} - \hat{r}_{ij})(q_{kj}) = -2\ e_{ij} q_{kj}$$

$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2\ (r_{ij} - \hat{r}_{ij})(p_{ik}) = -2\ e_{ij} p_{ik}$$

Now that we have the gradients for both $p_{ij}/$ and q_{kj}, we will update them using the following rules:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij} p_{ik}$$

Here α is a learning rate which determines the rate of approaching the local minimum. We will use the above update rule and will keep the parameters updating our parameter iteratively until we reach the local minimum and converged. The overall error can be calculated as:

$$\mathrm{E} = \sum_{i,j \in R} (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2$$

One general problem with the learning algorithms is over-fitting i.e. we need to determine a way such that it does not over fit the data and stops at the local minimum. One such approach to avoid over fitting is regularization. It is done by adding a parameter β and modifying the squared error:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{k} (|| P ||^2 + || Q ||^2)$$

Here parameter β controls the magnitude of the user-feature and item-feature vectors and provides a good approximation for the original matrix R. Based on the new squared error formula, the new update rules will be:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2 e_{ij} q_{kj} - \beta p_{ik})$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2 e_{ij} p_{ik} - \beta q_{kj})$$

Using the new update rules, we will keep on updating our parameters in each iteration until it converges.

4.4 Neural Collaborative Filtering

In this section, we describe our final approach that we developed for our project. This was one of the most challenging approaches as usually neural networks are not preferred when we are developing some recommendation system especially movie recommendation system since it usually does not perform well on the sparse input data. Neural network performs very well in some of the other fields like speech recognition and Natural Language Processing. There have been attempts at using Deep neural networks for music recommendation[7], So we tried exploring the neural networks by trying to predict how a user would rate a certain movie.

The neural network developed is using Keras with tensorflow in the backend. The Keras API is a neural network API, which runs over tensorflow. It allowed us to run the neural network effectively.

Describing about our architecture, we have three major sections, that includes the Input Layer, Hidden Layers and Output Layer. Complete architecture of our model is displayed in fig. 4. This architecture is developed based on the research conducted in the research paper neural collaborating filtering[2]. We used the one hot encoding process which converts into a form consisting of categorical variables that is suitable for neural networks and performs better prediction. We have created a list of unique movieIds and userIds which are used by the one hot encoder. The input layer thus consists of one hot encoding of MovieId and one hot encoding of UserId combined together to feed the layer one of the neural network. Using this approach we tried to understand that whether a neural network can find the similarity between movies and users during training as we are passing

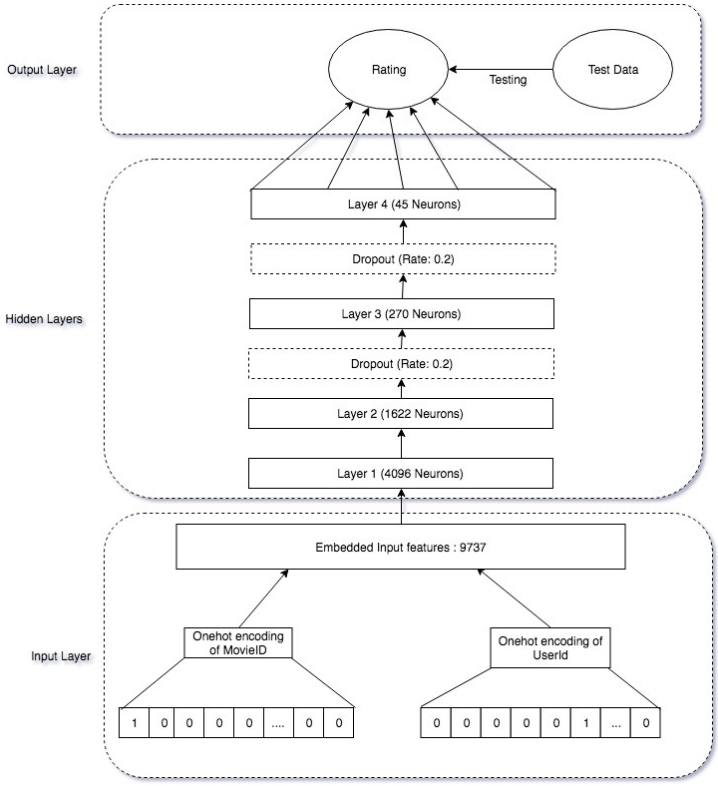

Figure 4: Architecture of Neural Network.

one-hot embedded vector as an input feature. Further, we have 4 hidden layers with 4096, 1622, 270, 45 Neurons respectively. Between hidden layer 2 and 3, and layer 3 and 4, we used the dropout technique to avoid the overfitting by our model. Here, we choose dropout rate = 0.2 which means we randomly drop 0.2 of neurons during training. At the output layer we have one feature which is the rating of movie given by user. The activation function used at each layer is ReLu function. It is given by:

ReLu function: (f(x)) = max(0,x)

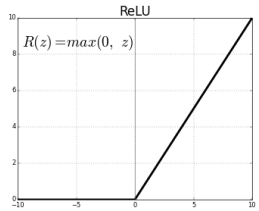

Figure 5: ReLU function

The ReLU function is an effective activation function as it increases the speed of training. The computation of gradient is either 1 or 0 based on the sign of x. The negative elements are set to 0 by the ReLU function. It also reduces the likelihood of the gradient. The optimizer used is ADAM[3], which is a technique based on first-order gradient of stochastic function that estimates lower order moments. This optimizer allows to define various hyper parameters while developing the model. This is used when we train the model for faster operation of the algorithm. During the backpropogation we calculate the loss function Mean Squared Error to update the weights. Mean Square Error is given by:

$$J = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_\theta(x_i))^2$$

Another hyper parameter that we defined is the learning rate. It tells the optimizer about how the weights must be moved and in which direction of the gradient. Based on the learning rate, we can get to know if the training is reliable or not and will the optimization make sense. If the learning rate is less, it will take a high amount of time to run as steps of the loss function are small but the training would be reliable. Else, if the learning rate is more, then we wont be sure if the training will take place correctly as there would be huge difference in the weights, thus the training might not converge or diverge. We had experimentally tried different learning rate on a small dataset to find the optimal value that could best be applied to the model. According to the theories, learning rate should be initially large since the weights are random and not optimal, finally we can decrease the learning rate and decide the one which provides optimal value. We finally kept the learning rate as 0.01.

Thus the final parameters that we feed include the learning rate (0.01), ADAM optimizer, loss function as MSE using the command 'model.compile' and 'model.fit' to train. Once the model is trained the testing can be performed using the 'model.predict' function which gives the outputs the loss of the model and MSE.

The output layer consists of a single neuron which gives the value of the rating of that movie and is in the range of 0 to 5.

Please note, due to computational limitation, we were not able to run the model on the original 20 Million dataset. The ANN we designed requires very high dimensional input feature vector. Our original movie data set has around 20 million movie ratings, 138000 users and 27000 movies. If we consider all this then our NN's input feature vector becomes of size (27000+138000) = 165000 and we train/test them on 20 million samples which is very large. Currently, we don't have any computational unit that can handles this data-set in our NN. We tried to reduce the input feature factor but it degraded the performance our model. So we ran this model on small dataset where we had 100,000 user ratings, which gave us some good results.

5 Evaluation measure and Result

A traditional approach to evaluate recommendation systems is to give a list of movies to the users and then check how they react to it. However, this process is very time consuming, costly and not reliable. To overcome this issue we developed our models such that instead of listing out recommended movies to a user, it will predict the rating of a movie given userId and movieId. This approach really helps us understand the performance of our models. Further, we can easily leverage this technique to transform it into a traditional way by running a model on all movies and choose a highest rated movie to recommend.

	100k dataset	**20 million**
Baseline	0.9424	0.8824
Item-Item	0.9340	0.8509
Latent Factor	0.9020	0.8603
Neural Network	0.9159	N/A

Figure 6: RMSE values of four models on 100K and 20 million datatset

We did not use the accuracy metrics to measure the performance of our models because the distance between prediction and actual rating is a lost. For e.g., if the actual movie rating is 2.5 and our first model predicts 3.0 rating and the second model predicts 5.0 rating. In this case, if we choose the accuracy as a measure then both the models are wrong. However, we can see that model 1 is more close to the actual rating so it is better than model 2. Accuracy percentage can't measure this. Here, comes the root mean square into the picture. It takes the difference between two values into account to give the better result. It is the square root of a sum of the differences between values predicted by a model or an estimator and the true values. We use the following formula to calculate the RMSE:

$$\text{Root-mean-square-error } (RMSE) = \sqrt{\frac{1}{N}\sum_{x,i\in N}(r_{xi} - \hat{r_{xi}}))^2}$$

- r_{xi} : Actual rating of movie i given by user x
- $\hat{r_{xi}}$: Predicted rating of movie i given by user x
- N : Total test cases

Now we have defined the performance measure metric we test our all models on the same dataset and find the RMSE value as given in the figure 6. As we can see in the that Latent factor model performs best on the small dataset where the models trained on 80,000 ratings and tested on 20,000 ratings. However, given more data for training, Item-Item Collaborative filter has improved its performance. Whereas, baseline gives least accurate results in both dataset.

6 Conclusion and Future Work

With the increase in the number of movies released in the entertainment industry, it is really important to improve the personalized recommendations given to a user based on his habits, interests and likes. Running efficient Machine Learning Algorithms can help to improve these recommendations. As expected the basseline model performs worst among other models because it does not consider any realtionship between movies or items. On the other hand Item-Item collabrative filtering stands out on a bigger dataset. The reason behind is that since we have more data, strong relationship has been established between different items i.e. movies. As this was an improvement on the gloabal baseline, the rating was updated based on the similarity weighted sum of top ten similar movie to the movie for which the rating is being predicted.

For future work, apart from making the neural network model more efficient in terms of training with larger dataset along with more features realted to movies and users, we will perform exploration of other approaches that can provide us some other key insights. Some of the other models that could be experimented for a movie recommendation system includes Support Vector Machine, Neighbor based Collaborative Filtering, Ridge Regression Collaborative Filtering etc.

7 Acknowledgement

We would like to thank Professor Jingrui He and the TAs Lawrence Luo and Ruocheng Guo for assisting us throughout the project. We would also like to mentioned individual work done by our all team members, which is shown in fig 7.

Team members:

No.	Name	ASU ID	Roles
1	Shaishavkumar Jogani	1212392985	Proposal, Data Preprocessing, BaseLine, Neural Collaborative Filtering, Presentation, Project Report
2	Naitik Shah	1213166628	Proposal, Item-Item Collaborative Filtering, Neural Collaborative Filtering, Presentation, Project Report
3	Swaraj Patel	1213099067	Proposal, Data Preprocessing, Neural Collaborative Filtering, Testing, Comparative Analysis, Project Report.
4	Dhrumil Parmar	1213179381	Proposal, Latent Factor Model, Presentation, Comparative Analysis, Project Report
5	Priyekant Aghi	1211211206	Proposal, Item Item Collaborative filtering, Latent Factor Model, Comparative Analysis, Presentation, Project Report
6	Prit Sheth	1213203392	Proposal, Data Preprocessing, Neural Collaborative Filtering, Testing, Comparative Analysis, Project Report
7	Ashish Sirohi	1211009485	Proposal, Latent Factor Model, Project Report
8	Jayanth Kumar Melinavolagerehalli Jayaramaiah	1212387590	Proposal, Item Item Collaborative Filtering, Testing, Comparative Analysis, Project Report

Figure 7: Contributions by individual team members.

8 References

1. https://grouplens.org/datasets/movielens/
2. Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua. Neural Collaborative Filtering. In WWW, pages 173-182, 2017.
3. Diederik P. Kingma, Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, ICLR 2015
4. https://en.wikipedia.org/wiki/Netflix_Prize
5. http://snap.stanford.edu/class/cs246-2013/slides/08-recsys2.pdf
6. Gbor Takcs, Istvn Pilszy, Bottyn Nmeth, Domonkos Tikk. Matrix factorization and neighbor based algorithms for the netflix prize problem. Proceedings of the 2008 ACM conference on Recommender systems Pages 267-274
7. A. Van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In NIPS,pages 26432651, 2013
8. Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-Based Collaborative Filtering Recommendation Algorithms. In WWW10, Pages 285-295, 2001.
9. Juan Ramos, Using TF-IDF to Determine Word Relevance in Document Queries.
10. Yifan Hu, Yehuda Koren, Chris Volinsky, Collaborative Filtering for Implicit Feedback Datasets