Project- Recommender Systems

Company- Iken Solution Private Limited, Mumbai.

Summer Internship report submitted by-

Srinivas JK Mathur

15PH20031

Department of Physics

Indian Institute of Technology Kharagpur 721 302, India

**Index**

*1) Introduction*

*2) Machine learning Models*

*a.) Memory Based*

*b.) Model Based*

*3) Deep learning Models*

*a.) Restricted Boltzmann Machines (RBM)*

*b.) Stacked Auto Encoders*

*4) Analysis of user-movie data*

*5)Implementation of Models: Parameters and Results*

*6) Conclusion*

Introduction-

Recommender systems are one of the most successful and widespread application of machine learning technologies in business. From E-commerce to online advertisement, all such companies leverage recommender systems to suggest items to the users according to their taste and preference to improve user experience. For example, Companies like Amazon, Netflix, Spotify have built smart and intelligent recommender systems to recommend products and items to the users which he/she is most likely to buy or use. So, knowing in advance the items which a user is likely to buy can generate huge amount of income.

What are Recommender systems?

Recommender systems or Recommendation engines are algorithms which provide most relevant and accurate items to the user by filtering out items from a huge pool of information base. It learns from the data of user behavior towards products and produces outcomes that relates best to his/her past interests.

Two most common types of recommender systems are **Content-Based** and **Collaborative Filtering (CF).**

1. Collaborative filtering produces recommendations based on the knowledge of users’ attitude to items, that is it uses the "wisdom of the crowd" to recommend items.
2. Content-based recommender systems focus on the attributes of the items and give you recommendations based on the similarity between them.

In general, Collaborative filtering (CF) is more commonly used than content-based systems because it usually gives better results and is relatively easy to understand (from an overall implementation perspective). The algorithm has the ability to do feature learning on its own, which means that it can start to learn for itself what features to use.

CF can be divided into **Memory-Based Collaborative Filtering** and **Model-Based Collaborative filtering**.

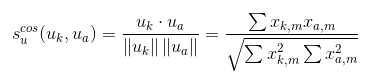
**Machine learning Models**

**A) Memory-Based Collaborative Filtering:**

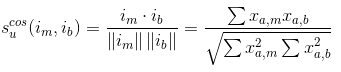
Memory-Based Collaborative Filtering approaches can be broadly divided into two main sections: **user-item filtering** and **item-item filtering**.

1. A user-item filtering will take a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked.
2. Item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations.

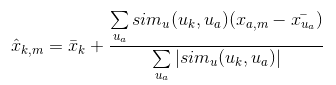
A distance metric commonly used in recommender systems is *cosine similarity*, where the ratings are seen as vectors in n-dimensional space and the similarity is calculated based on the angle between these vectors. Cosine similarity for users *a\* and \*k* can be calculated using the formula below, where dot product of the user vector 𝑢𝑘 and the user vector 𝑢𝑎 is taken and divided by multiplication of the Euclidean lengths of the vectors.



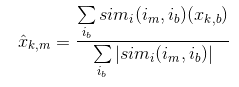
To calculate similarity between items *m\* and \*b,* use the formula:



Predicted ratings for user-item filtering are given by,



And for item-item filtering,



B) Model Based Collaborative filtering:

Model-based Collaborative Filtering is based on **matrix factorization (MF)** which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based CF. The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items. When the rating is very sparse matrix, with a lot of dimensions, by doing matrix factorization one can restructure the user-item matrix into low-rank structure, and represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector. The predicted ratings for new movies are obtained by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix.

A well-known matrix factorization method is **Singular value decomposition (SVD)**. Collaborative Filtering can be formulated by approximating a matrix  X  by using singular value decomposition.

 The general equation can be expressed as follows: https://latex.codecogs.com/gif.latex?X=USV%5eT

Given m x n matrix X:

* *U* is an *(m x r)* orthogonal matrix
* *S* is an *(r x r)* diagonal matrix with non-negative real numbers on the diagonal
* *V^T* is an *(r x n)* orthogonal matrix

Elements on the diagonal in S are known as *singular values of X*.

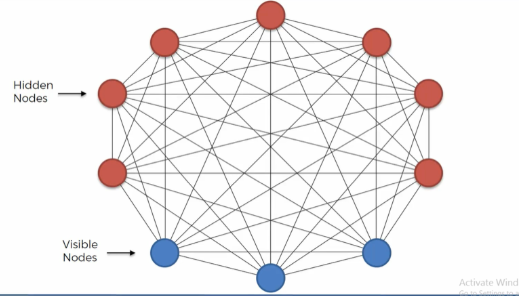
Matrix *X* can be factorized to *U*, *S* and *V*. The *U* matrix represents the feature vectors corresponding to the users in the hidden feature space and the *V* matrix represents the feature vectors corresponding to the items in the hidden feature space.

The value of ‘r’ in the dimension of matrices decides the number of latent features of the user and the movie to consider. Lower value of r will simply output popular movies with high ratings without much personalization. Choosing higher value of r will help capture more features of the user and suggest more personalized content based on user’s taste.

Deep Learning based Models-

A) Restricted Boltzmann Machines:

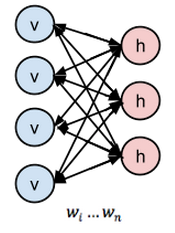
Boltzmann Machines (BM) are bi-directionally connected neural networks with stochastic processing units. A BM can be used to learn important aspects of an unknown probability distribution based on samples from this distribution. In general, this learning process is difficult and time-consuming. However, the learning problem can be simplified by imposing restrictions on the network topology, which leads us to *restricted Boltzmann machines (RBM).*



Architecture of BM

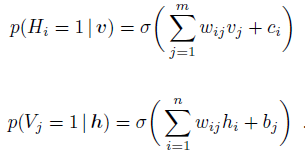
Architecture of RBM

RBM has two layers, i.e. A hidden layer and a visible layer. A full Boltzmann Machine has every neuron connected to every other neuron. While in a RBM, the neurons in each layer communicate with neurons in the other layer but not with neurons in the same layer, there is no intra-layer communication among the neurons.

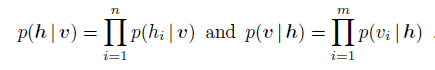


Training RBM

* For the movie-recommender system, we will have a *m* x *n* matrix with *m* users and *n* movies. We pass a batch of *k* users with their *n* movie ratings into the RBM neural network and train for a certain number of *epochs*.
* The visible layer will have X neurons, where X is the number of movies in the dataset. Each neuron will have a normalized rating value from zero to one, where zero means the user has not seen the movie. The closer the normalized rating value is to one, the more the user likes the movie represented by the neuron.
* The neurons from the visible layer communicate to the neurons from the hidden layer, and then the hidden layer passes back information to the visible layer. RBMs perform this communication passing back and forth several times between the visible and hidden layer to develop a generative model which tries to learn the underlying, latent features that characterize the user-movie preferences such that the reconstructions from the outputs of the hidden layer are similar to the original inputs.
* Each input *v0* is multiplied by its respective weight *W*. The weights are learned by the connections from the visible layer to the hidden layer. Then a bias vector is added at the hidden layer called *hb*. The bias ensures that at least some of the neurons fire. This *W\*v0+hb* result is passed through a sigmoid activation function.



* After this, A sample of the outputs generated via a process is taken known as *Gibbs sampling*. In other words, the activation of the hidden layer results in final outputs that are generated stochastically. This level of randomness helps build a better-performing and more robust generative model.



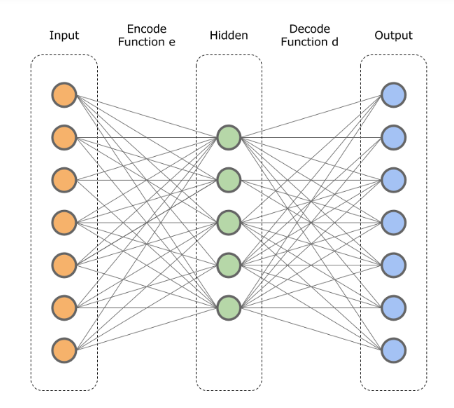
Sampling hidden nodes given the visible nodes.

* Next, the output after Gibbs sampling known as *h0* is passed back to the visible layer in the opposite direction also known as *backward pass*. In the backward pass, the activations in the *forward pass* after Gibbs sampling are fed into the hidden layer and multiplied by the same weights *W* as before. We then add a new bias vector at the visible layer called *vb*.
* This *W\_h0+vb* is passed through an activation function, and then we perform Gibbs sampling. The output of this is *v1*, which is then passed as the new input into the visible layer and through the neural network as another forward pass.
* The RBM goes several forward and backward passes like this to learn the optimal weights as it attempts to build a robust generative model.
* Iteratively the weights of the neural net are adjusted in such a way that the RBM can find the relationships among input features and then determines which features are relevant, the RBM learns to approximate the original data as best as possible. This can now be used to predict ratings for never-before-seen data.

B) Stacked Autoencoders:

Autoencoder is an Artificial Neural Network which learns to reconstruct the input at the output layer by using encoding for dimensionality reduction (compression) and decoding for reproducing the original input. The fact that it tries to reproduce and best match to the input rather than some output corresponding to certain input, makes it an Unsupervised Learning algorithm.

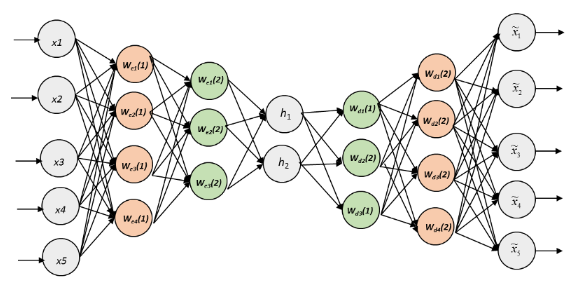
Apart from Recommender systems, Autoencoders finds its use in image compression, image reconstruction, image denoising, feature extraction etc.



Autoencoders

Architecture of Stacked Autoencoders

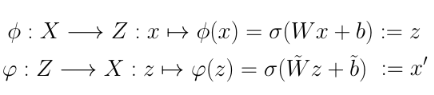
Autoencoders generally contain input layer, a hidden layer and output layer. To improve its ability to learn more complex features, several hidden layers are added, this is known as Stacked or Deep AutoEncoders. The first layer of the Deep Autoencoder may learn first-order features in the raw input (such as edges in an image). The second layer may learn second-order features corresponding to patterns in the appearance of first-order features (e.g., in terms of what edges tend to occur together — for example, to form contour or corner detectors). Deeper layers of the Deep Autoencoder tend to learn even higher-order features.



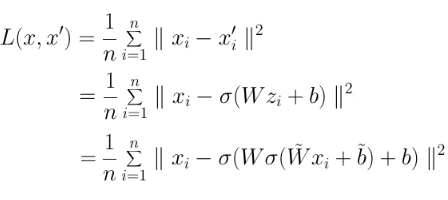
Stacked Autoencoders

Training Stacked Autoencoders:

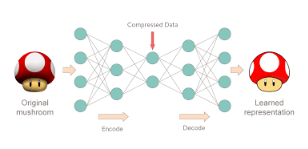
* The transition from the input to the hidden layer is called the encoding step and the transition from the hidden to the output layer is called the decoding step. During training, the encoder takes an input data sample**x** and maps it to the so called hidden or latent representation **z.**Then the decoder maps **z**to the outputvector **x’**which is (in the best case scenario) the exact representation of the input data **x**. We can also define these transitions mathematically as a mapping:



* In the first pass Random weights and biases are initialized using Gaussian distribution and the output **x’ is calculated.** Having the output **x’**the training consists of applying stochastic gradient descent to minimize a predefined loss such as a mean squared error.



* Mean squared error is calculated in every pass and the weights and biases are updated each time in order to minimize the error.
* This process is repeated for several times on complete dataset to train the model (called Epoch). It should be noted that after certain number of epochs the performance may start sliding down owing to overfitting, therefore an optimal number of epochs must be chosen.
* After the model has been trained on the training data, it is now ready to predict ratings for the unseen movies based on the latent feature learned and recommend movies which happen to have high predicted ratings.



Representation of learned image

Analysis of the User-Movie data

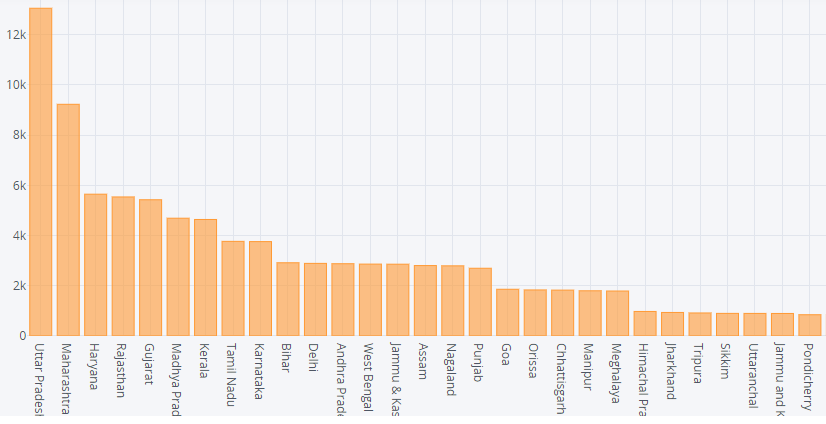
* We were provided with data containing information of the user, content and session.

1) Session data contained information of watching watched by an user along with the start and the end time, region, city etc.

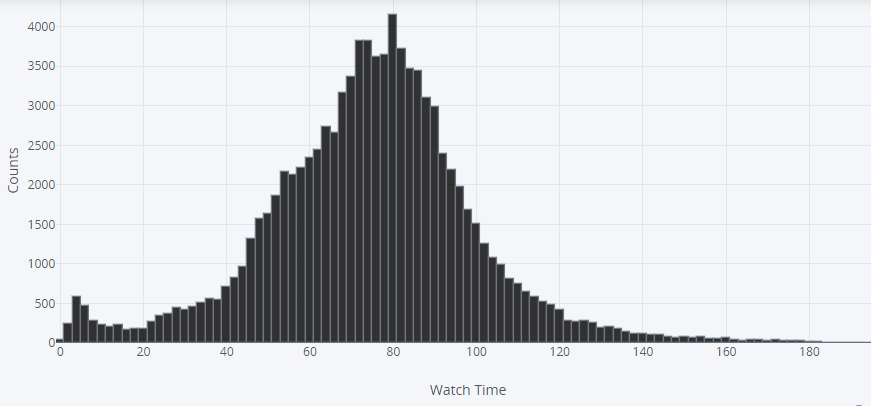
2) User file had all the details of the user like age, gender, region, city, subscription etc.

3) Content file contained the information about the movie, genre, language, release date etc.

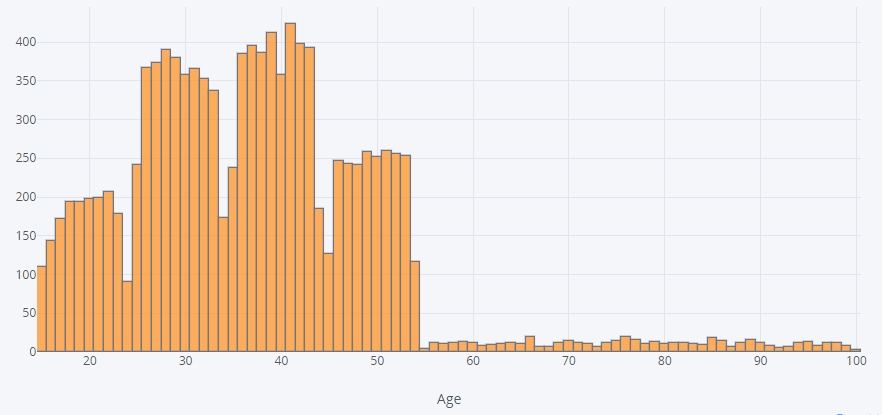
* State-wise distribution of user session.



* Distribution of movie watch time

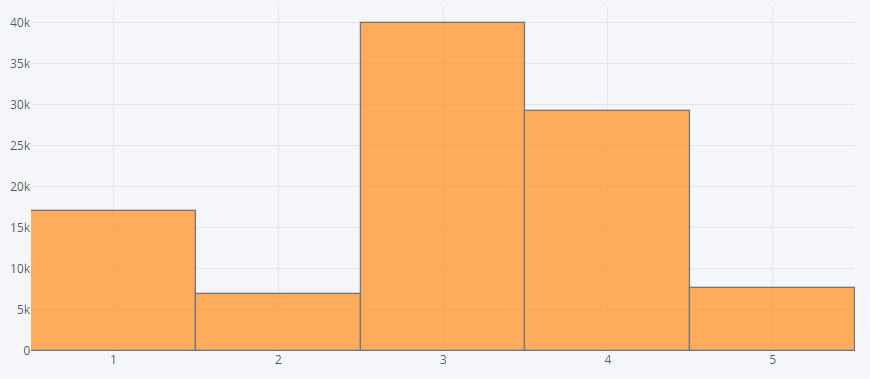


* Distribution of age of users.

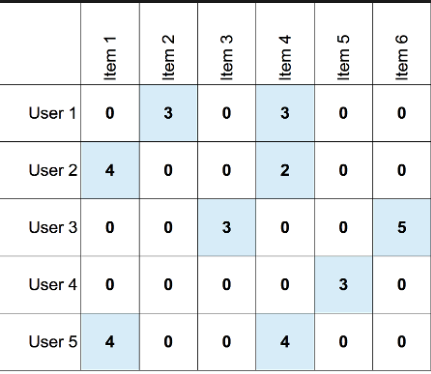


Data Preprocessing

* The movie ratings by the user were missing, therefore some logic had to be devised to get an idea about how did the user like the movie. For this purpose, the watch time of the user was compared with the full movie length and the median watch time of that particular movie. If a user quits the session before completing the movie it is very likely that the user didn’t like the movie. Also if the watch time of the user is lesser than the mean watch time of the movie, he/she didn’t like the movie and vice-versa.



* All the user ratings corresponding to the movies were converted into a matrix, with rows as users and columns as movies. The movies which were not watched by the user were rated as 0. There were 10483 users and 17218 movies. The matrix is 99.9% sparse i.e. 99.9% entries are 0.



* The dataset was split into training set and test set in 75:25 ratios.

Implementation of Models: Parameters and Results

1) Memory based collaborative filtering:

1.1) Item-item filtering:

* Libraries used : Numpy, Scipy
* Parameters : N/A
* Error on the test set: RMSE =3.42

1.2) User-item filtering:

* Libraries used : Numpy, Scipy
* Parameters : N/A
* Error on the test set: RMSE =3.35

2) SVD (Model based collaborative filtering):

* Libraries used : Scipy , Numpy
* Parameters used : k=40
* Error on the test set: RMSE = 3.21

3) Restricted Boltzmann Machines (RBM):

The movie ratings were converted to binary numbers 0 and 1 i.e. liked or disliked. All the movies with 3 or more ratings were labeled 1, less than 3 as 0 and unseen movies as -1.

* Libraries used : Numpy, Tensorflow, pytorch
* Parameters used :

No. of users =2000

visible node= 17206

hidden nodes=100

Epochs=50

Batch size=200

* Error on the test set: RMSE = 0.26

4) Stacked Autoencoders:

* Libraries used : Numpy, Tensorflow, pytorch
* Parameters used :

No. of users =2000

visible layer = 17206 nodes

hidden layer 1 = 20 nodes

hidden layer 2 = 10 nodes

hidden layer 3 = 20 nodes

Output layer = 17206 nodes

Epochs=100

* Error on the test set: RMSE = 1.97

Conclusion-

In this report, we explored some widely used machine learning models and some more advanced deep learning models for collaborating filtering which help users view more personalized content rather than trending and popular contents.

On comparing, we see that deep learning models outperform the traditional machine learning algorithms like SVD and user-item filtering and make more accurate predictions on the unseen data set. This is because Artificial Neural Networks (ANN) are capable of learning more complex and higher order features which machine learning models fail to learn. Also ANN offers more flexibility in terms of choosing parameters like the number of hidden layers, number of nodes in each layer, batch size and epochs. While SVD or user-item filtering offer no to very limited number of hyper parameters to tune and adjust the model. The performance of the model can be further improved by introducing regularization term in SVD, RBM and Autoencoders to prevent overfitting and trying different hyperparameters.