Collaborative Filtering for Recommender Systems

Sqn Ldr Sahil Deep [*1]

Abstract

Recommender systems are tools for interacting with large and complex information spaces. They provide a personalized view of such spaces by prioritizing items likely to be of interest to the user. Personalized recom- mendations are an important part of many on-line e-commerce applications such as Amazon.com, Netflix etc. Major thrust in Recommender System has been on Collaborative Filtering methods based on its performance in NetFlix Prize competition. Collaborative filtering (CF) predicts user preferences in item selection based on the known user ratings of items. The most well-known algorithm of Collaborative Filtering is the matrix factorization. MF is favored by enterprises and academia, because it has advantages of accuracy, concision and parallelization. MF is based on the inner product of item factor vector and user factor vector which denotes users rating for a certain item. Another technique that has been employed in CF is Restricted Boltzmann Machine(RBM). It is basically employed as a network of stochastic units with undirected interactions between pairs of visible and hidden units. Learning weights of edges and biases for both the units i.e hidden and visible is used to subsequently predict the recommendation score for unrated item.

1. Introduction

Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to users preferences, interest, or observed behavior about item. Recommender systems (RS) which use data mining and information filtering techniques to provide products, services and information to potential customers have attracted a lot of attention. Broadly speaking, the recommender systems can be divided into three main branches according to different strategies - collaborative filtering (CF), content-based and the combination of both. Among of them, the collaborative filtering is the most widely used since it has the advantage that does not require the creation of explicit profiles. CF can be divided into two main branches: Nearest Neighbourhood based and Latent Fator model. Latent Factor model is the most commonly used Colaborative Filtering Technique which tries explain ratings by characterizing both users and items on say 5 or 10 factors inferred from ratings pattern. One of the most successful realization of latent factor model is based on matrix factorization. It characterizes both items and users by vectors of factors inferred from rating pattern. High correspondence between item and user factors leads to a recommendation. They have good scalability and predictive accuracy. In addition, they provide more flexibility in modeling real-life situations. Recommender systems research encompasses a wide variety of artificial intelligence techniques which include machine learning, data mining, user modeling, case-based reasoning and constraint satisfaction etc. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the users profile. Similarly on the lines of latent factor approach is the Restricted Boltzmann Machine which is based on two layers of neural networks that form a bipartite graph and the purpose of the model is to learn weights of connecting edges and biases of both hidden and visible units. It uses the concept of assigning probability to the item corresponding to a particular user and based on say (20) maximum probabilities provides recommendations.

Table 1. Data Sets: Each user has rated at least 20 movies

DATA SET	USERS	ITEMS	RATINGS	RATING SCALE
100K	610	9742	100836	[1-5]
1 M	6040	3952	1000209	[1-5]
10 M	71567	10681	10000054	[0.5-5]
20 M	138493	27278	20000263	[0.5-5]

2. Dataset

Datsets used for building the recommender system were taken from https://grouplens.org/datasets/movielens/. Of all these except 100K all else are benchmarked datasets that have been used for research publications and development of recommender systems. The details of the datsets are as depicted in Table 1. Various files in the

dataset are as follows:-
Ratings file <UserId::MovieId::Ratings::TimeStamp>
Movies file <MovieId::Movie Name::Genre>. A movie can have multiple genres.

3. Data Analysis

In order to have a deeper understanding of the datset. I took 20 M dataset and seperated the movies on the basis of their 7 prominent genres out of 14. The movies ratings have been generated from ratings given from January 09, 1995 and March 31, 2015 (approx 20 years of rating data). Subsequently, I have analyzed the trends of ratings per genre over the entire 20 year period. The plot given in Figure 1 shows that crime movies have higher average ratings and horror movies are least preferred. Additionally, I have also plotted average ratings of all the movies based on the release year. Figure 3 gives the average rating for each the movies over the entire rating period. Figure 4 shows that there are very few users that have given considerable amount of ratings i.e the user-movie matrix is sufficiently sparse.

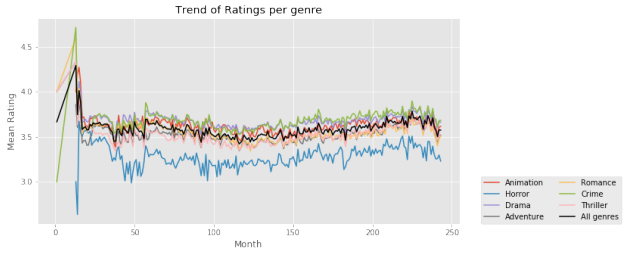

Figure 1.

4. MATRIX FACTORIZATION

Matrix factorization models map both users and items to a joint latent factor space of dimensionality f such that user-item interations are modeled as inner products in that space. Each item i is associated with a vector $q_i \in \mathbb{R}^f$ and each user is associated with vector $p_u \in \mathbb{R}^f$. p_u and q_i measure the extent to which users and items possess those factors(+ve or -ve). Modeling directly the observed ratings while avoiding overfitting through regularized model. Observed rating is broken into its four components: global average, item bias, user bias, and user-item interaction. This allows each component to explain only the part of a signal relevant to it. The system learns by minimizing the squared error function to learn the factor vectors(p_u and q_i) the system minimizes the squared error on the set of known ratings.

4.1. Implementation Details

Entire implementation was done using TensorFlow for both local and Distributed implementation. The path of enire code and data in Turing cluster is available at /home/sahildeep/PROJECT/code and /home/sahildeep/PROJECT/data. Salient features that were added in the implementation are as follows:

- Hyper Parameters :- Regularization Coefficient=0.02 Learning Rate =0.001 , Latent Factors=5. All the above hyperparameters are tunable from command line itself.
- Initially I was using a dense matrix and that approach was very very slow and didnt' really use the fact that the matrix is sparse. So I started using the concepst enumerated in [4] in our code for performance. This resulted in training time of around 20 min for 20M datset and similarly for other datasets.
- Used 80 : 20 split for train/test and this is randomized spliting at each instantiation of the code.
- Initially, I started with GradientDescentOptimizer but realized that the number of epochs required was much more and hence switched to AdamOptimizer, which reduced the loss function faster and hence we required lesser number of epochs for convergence.

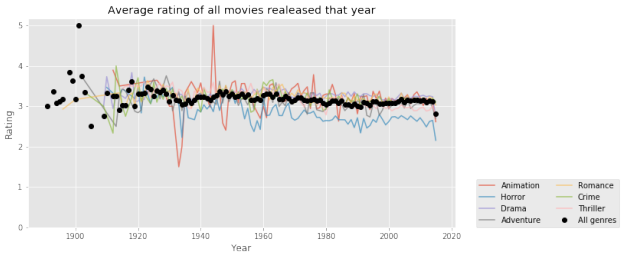

Figure 2.

$$min_{p^*,q^*,b^*} \sum_{u,i \in \mathbb{K}} (r_{ui} - \mu - b_u - b_i - p_i^T q_i) + \lambda(||p_u||^2 + ||q_i||^2 + b_i^2 + b_u^2)$$

Here $\mathbb{K}$ is the set of (u, i) pairs for which r_{ui} is known i.e training set.

4.2. Stochastic Gradient Descent(SGD)

Algorithm loops through all ratings in the training set. For each training case system predicts r_{ui} and computes the associated prediction error

$$e_{ui} = r_{ui} - q_i^t p_u$$

Subsequently it modifies the parameters by a magnitude propotional to γ in opposite direction of the gradient yeilding

$$q_i \leftarrow q_i + \gamma(e_{ui}.p_u - \lambda.q_i)$$
$$p_u \leftarrow p_u + \gamma(e_{ui}.q_i - \lambda.p_u)$$
$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda.b_u)$$
$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda.b_u)$$

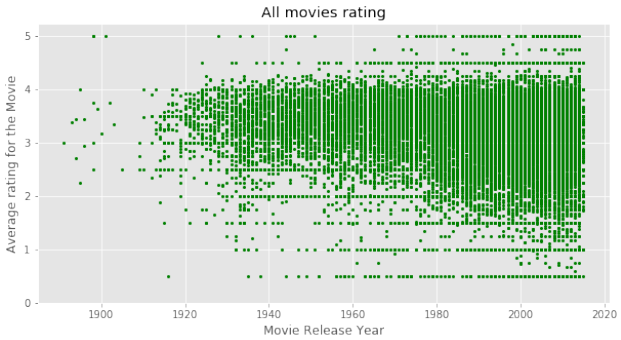

Figure 3.

5. Model and Evaluation Metrics

Matrix Factorization

$$min_{q^*,p_u^*,b^*} \sum_{r_{ui} \in \mathbb{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 +$$
$$\lambda(\sum ||q_i||^2 + \sum ||p_u||^2 + \sum b_u^2 + \sum b_i^2)$$

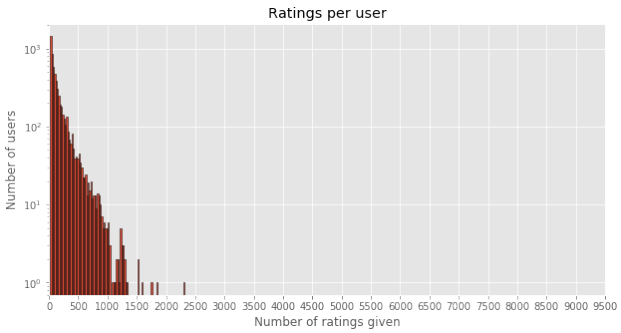

Figure 4.

Evaluation Metrics

$$\text{Root Mean Square Error}(RMSE) = \sqrt{\frac{1}{|T|} \sum_{r_{ui} \in \mathbb{K}} |r_{ui} - \hat{r}_{ui}|^2}$$

$$\text{Mean Absolute Error}(MAE) = \frac{1}{|T|} \sum_{r_{ui} \in \mathbb{K}} |r_{ui} - \hat{r}_{ui}|$$

Table 2. Average rating(μ) over the entire datset

DATA SET	USERS
100K	3.549
1 M	3.58
10 M	3.59
20 M	3.62

6. Local Results

On implementation of the basic factorization model the results obtained are as enumerated in Table 2 and Figures 4,5 and 6 respectively. RMSE results are quite healthy for all datasets as case in point is the fact that the best matrix factorization model that has been implemented in on 20 M dataset (largest used for our implementation) has a RMSE of 0.81 [3]. Figure 5 gives us the training loss over number of epochs for all the datasets and the baseline(similiar to Cinematch i.e without matrix factorization). Figure 6 gives the time in seconds for training, testing time for training and test data for the 4 datasets i.e shows us the scalability of the model as data grows. Figure 7 clearly show that as expected, the model performs better on training dataset as compared to test datset for both RMSE and MAE as desired.

7. Matrix Factorization: Distributed Implementation

2. Distributed Matrix Factorization using Parameter Server:

- **Framework**: Code was written in python and run on Turing clusters different worker nodes using Distributed Tensorflow framework. This framework was chosen as dis- tributed graph learning can be done iteratively on Tensor- flow with various requirements that is synchronous and asynchronously.
- **Data Preparation** Data preparation is same as mentioned in section 4.1.
- **Hyperparameters**: Batch size 500/50000 depending upon the dataset used to increase the speed of convergence.
- Training epochs 20
- Constant learning Rate 0.002

The hyperparameters are same for all distributed SGD techniques. For all strategies I have maintained two parameter servers and two worker nodes. I have used Between-graph replication for distributed training. In this approach, there is a separate client for each /job:worker task, typically in the

same process as the worker task. The cluster specification for distributed training was done in the following manner:-

- parameter servers = [10.24.32.203:2225,]
- workers = [10.24.32.204:2225, 10.24.32.205:2225]
- cluster = tf.train.ClusterSpec(ps:parameter worker:workers) servers,

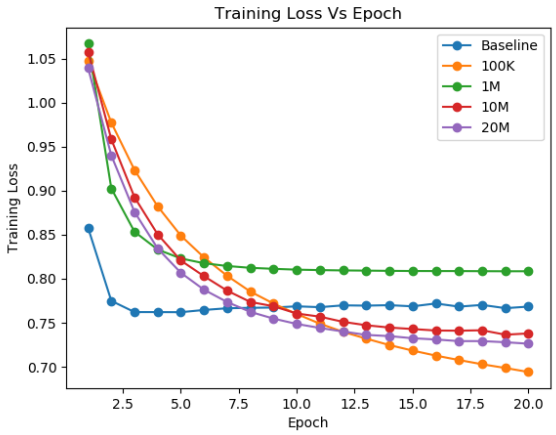

Figure 5.

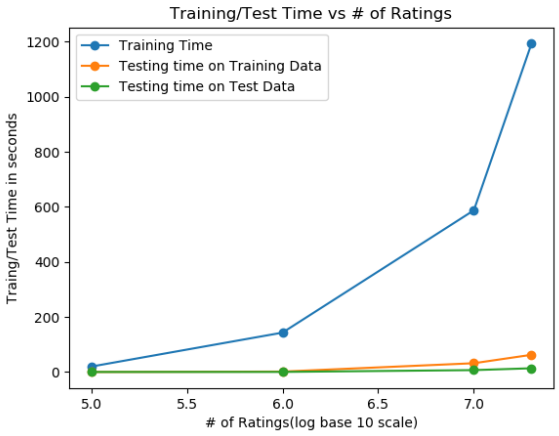

Figure 6.

7.1. Results :Distributed Stochastic Gradient Descent

Figure 8 shows that training loss was maximum on BSP and least on Async, with SSP-10 close to Async. However, eventually over a period of 20 epochs, especially after 5 epochs, their training loss catches up with each other and

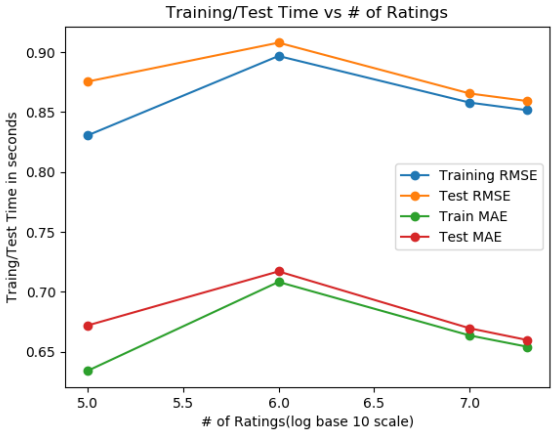

Figure 7.

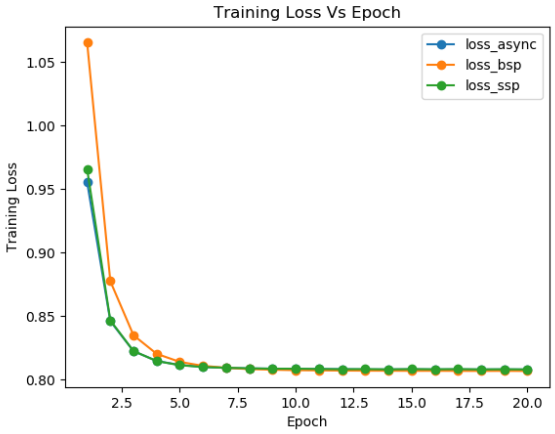

Figure 8.

SSP-10 has a least loss eventually. Even after increasing the number of workers, I could not see a substantial decrease in the training time in distributed setting over a single machine. The only decrease observed was for the 20M dataset and that was only for was in the range of 50 - 60 seconds. For all 3 configurations the results for the metrics RMSE and MAE were similar upto 3 decimal places.

8. Restricted Boltzmann Machine

The Restricted Boltzmann Machine model has two layers of neurons, one of which is what we call a visible input layer and the other is called a hidden layer. The hidden layer is used to learn features from the information fed through the input layer. For our model, the input is going to contain X

Table 3. Data Sets: Each user has rated at least 20 movies

DATA SET	SPARSITY	TEST(RMSE)	TEST(MAE)
1 M	4.26%	0.857	0.717
10 M	1.33%	0.857	0.663
20 M	0.52%	0.857	0.658

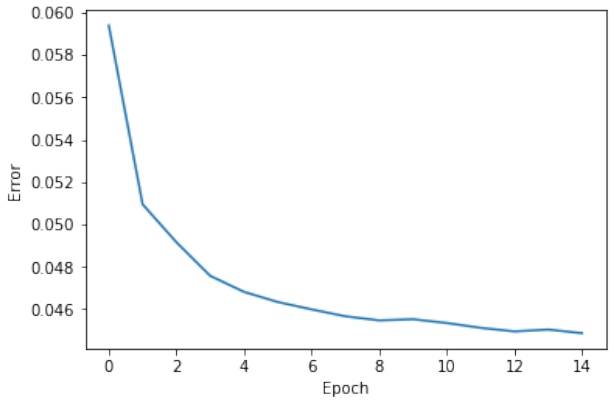

Figure 9.

neurons, where X is the amount of movies in our dataset. Each of these neurons will possess a normalized rating value varying from 0 to 1, where 0 meaning that a user has not watched that movie and the closer the value is to 1, the more the user likes the movie that neuron's representing. These normalized values, of course, will be extracted and normalized from the ratings dataset.

After passing in the input, we train the RBM on it and have the hidden layer learn its features. These features are what we use to reconstruct the input, which in our case, will predict the ratings for movies that user hasn't watched, which is exactly what we can use to recommend movies.

8.1. Training Algorithm for RBM

Restricted Boltzmann machines are trained to maximize the product of probabilities assigned to some training set $\mathbf{V}$ (a matrix, each row of which is treated as a visible vector $\mathbf{v}$)

$$\underset{\mathbf{W}}{\operatorname{argmax}} \Pi_{v \in \mathbf{V}} P(v)$$

or equivalently, to maximize the expected log probability of a training sample v selected randomly from $\mathbf{V}$. The algorithm used to train RBMs i.e. to optimize the weight vector $\mathbf{W}$ is the contrastive divergence (CD) algorithm:-

- Take a training sample v, compute the probabilities of the hidden units and sample a hidden activation vector h from this probability distribution.
- Compute the outer product of v and h and call this the positive gradient.
- From h, sample a reconstruction v' of the visible units, then resample the hidden activations h' from this.
- Compute the outer product of v' and h' and call this the negative gradient.
- Let the update to the weight matrix $\mathbf{W}$ be the positive gradient minus the negative gradient, times some learning rate $\Delta \mathbf{W} = \epsilon(vh^T - v'h'^T)$.
- Update the biases a and b as $\Delta a = \epsilon(v - v')$, $\Delta b = \epsilon(h - h')$.

	MovieID	Title	Genres	RecommendationScore	UserID	Rating	Timestamp
253	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	0.995626	340.0	3.0	976340754.0
1106	1196	Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Drama|Sci-Fi|War	0.722396	340.0	3.0	976342179.0
1025	1097	E.T. the Extra-Terrestrial (1982)	Children's|Drama|Fantasy|Sci-Fi	0.717542	340.0	3.0	976340754.0
1178	1270	Back to the Future (1985)	Comedy|Sci-Fi	0.617930	340.0	3.0	976340681.0
1215	1307	When Harry Met Sally... (1989)	Comedy|Romance	0.574277	NaN	NaN	NaN
576	590	Dances with Wolves (1990)	Adventure|Drama|Western	0.566590	NaN	NaN	NaN
1120	1210	Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Romance|Sci-Fi|War	0.539140	340.0	3.0	976340465.0
1537	1674	Witness (1985)	Drama|Romance|Thriller	0.537937	NaN	NaN	NaN
851	912	Casablanca (1942)	Drama|Romance|War	0.534754	340.0	5.0	976341157.0
858	919	Wizard of Oz, The (1939)	Adventure|Children's|Drama|Musical	0.530972	340.0	5.0	976341003.0
1108	1198	Raiders of the Lost Ark (1981)	Action|Adventure	0.522815	NaN	NaN	NaN
3238	3471	Close Encounters of the Third Kind (1977)	Drama|Sci-Fi	0.520835	NaN	NaN	NaN
1107	1197	Princess Bride, The (1987)	Action|Adventure|Comedy|Romance	0.508558	340.0	5.0	976340681.0
1135	1225	Amadeus (1984)	Drama	0.486574	NaN	NaN	NaN
863	924	2001: A Space Odyssey (1968)	Drama|Mystery|Sci-Fi|Thriller	0.486087	NaN	NaN	NaN
908	969	African Queen, The (1951)	Action|Adventure|Romance|War	0.468208	340.0	4.0	976341312.0
513	527	Schindler's List (1993)	Drama|War	0.463906	NaN	NaN	NaN

Figure 10.

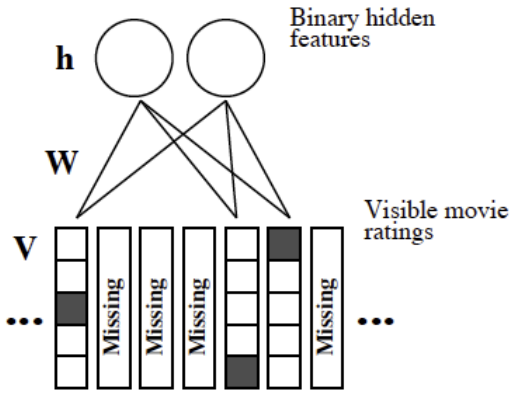

Figure 11.

8.2. RBM Results

We see that that there is a high correlation $r = 0.72$ for movies which have received a high rating from a user and for which our model predicted a high recommendation score ≥ 0.8 for that user.

References

[1]Yehuda Koren, Robert Bell, Chris Volinsky:Matrix factorization techniques for Recommender Systems IEEE Publication 2009/8/1 Computer Journal Issue 8 Pages 30-37

[2] 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

[3] https://support.treasuredata.com/hc/en-us/articles/360001260847-MovieLens-20m-Rating-\Prediction-using-Factorization-Machine.

[4]https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html

[5] https://en.wikipedia.org/wiki/Restricted_Boltzmann_machine