Journal of Machine Learning Research 1 (2021) 1-8 Submitted 12/18; Published 12/18

Recommendation System for Movies Recommendation using Machine Learning approaches

Wenjuan Qi WJQI@UWATERLOO.CA
Department of Statistics
University of Waterloo
Waterloo, ON N2L 3G1, Canada

Editor: Wenjuan Qi

Abstract

This paper utilizes three machine learning models to construct a recommendation system in Movie dataset. The advantages and disadvantages are discussed in this paper. We describe one of common issue in recommendation system: sparsity of data. We present efficient algorithms to tackle this problem. We also discuss other algorithms to mitigate this problem in the same application as well. Empirical results demonstrate the performance of the models in movie ratings prediction.

Keywords: Recommendation system, Autoencoder, KNN,

1. Introduction

Nowadays, explosive growth of information from e-commerce to online advertisements on various products makes people difficult to make a choice. For this reason, recommendation systems have been developed to overcome the problem of over-choices of purchasers. In general a recommendation system is an algorithm which utilizes item information, past information of users to give suggestions to user. Examples could be a movie recommended from Netflix to watch, a nice restaurant selected on Yelp and a birthday gift recommended from Amazon. Companies like *Netflix* and *Youtube* makes use of recommendation system not only facilitate people's life but also attain a huge profit.

There two types of recommendations systems are commonly used in major companies: content-based recommendation system and collaborative filtering systems. The first type aims to find degree of similarity based on item description and user's past buying history and output the next item with the highest degree of similarity with content of items which purchased by the same user. We call this type as user-to-item recommendation. Thus, content-based recommendation system do not make use of other user's information, however, collaborative filtering system utilizes user's ratings and predict one user's preference as a weighted linear function of other users' preference. We call this type as user-to-user recommendation. Both classes have their limitations: they need a large data set on user's preference and an accurate ratings data for items. Problem arises when we do not have enough data from users but we have large number of products. For example, some features of data will be sparse such as ratings. For this reason, I will use machine learning models to tackle this problem.

To predict ratings, item-user matrix is constructed as training data set, we will be concerned with two problems: time and memory. When number of user and number of movies are large, it will not only make computation inefficient but also cause out-of-memory issue.

The project we only focus on Collaborative filtering (CF) system to make use of both users and items. Collaborative Filtering system consists of two types: user-based approach and item-based approach. There are many differences between these two concepts: user-based approach takes input from a user, most of time it would be ratings for several movies. Our database contains millions of movies, so vector of movie ratings with the respect to whole database could be very sparse. We then utilize this vector and the rest user's ratings to find similarities to further infer what ratings should be for other movies that the same user has not seen before. On the other hand, item-based approach is on the opposite way, it utilize movie information and find similar movies based on different distance measures based on ratings. This project will visit both approaches using different models: for user-based approach I will illustrate using *Autorec* model which is based on *Autoencoder* paradigm. *LSTM* model based on tags from users are also implemented to predict ratings as output. For content-based approach, I will implement unsupervised K-Nearest Neighbour approach to find similar movies and output top N movies to users based on user input movie name. Our result shows that *LSTM* performed the best with lowest Mean Square Error and then follows by *Autorec* but unsupervised KNN is the easiest to implement with least run time.

As a final product of three collaborative filtering approaches users need to specify, different inputs required for three models: for *KNN* how many movies they would like to watch say K need to specify, then For *Autorec*, rating of all movies in database will be predicted, top K movies which achieve the highest rating will be the output. For *unsupervised K-Nearest-Neighbor*, top K movies with smallest distance to user input movie will be the output. For *LSTM*, ratings will be predicted if user give tag or comment about a movie.

2. Dataset and Preprocessing of Data

Dataset we will used in this project is "Movielens" data. This data sets were collected by the GroupLens Research Project at the University of Minnesota. It can be downloaded from `https://grouplens.org/datasets/movielens/latest/`. The dataset

This data set consists of:

- 27,000,000 ratings (1-5) and 1,100,000 tag from 280,000 users on 58,000 movies.
- This dataset (ml-latest) describes 5-star rating and free-text tagging activity from "MovieLens" website.
- Data contains only users who rated at least one movie.

The data was collected through the MovieLens web site (movielens.umn.edu) between January 09, 1995 and September 26, 2018.

In order to run in Google Colab without issue of lacking memory, the data set we work with is filtered to contain only 5% of users (14161) and movies which are rated less than 50 times are removed from our data and we left with 3838 movies. Dataset are transformed into 1. users-movies sparse matrix for user-based approach using pivot function in Python.

2. movies-users sparse matrix for movie-based approach. Tags dataset is used to train LSTM model to predict rating based on tages. Ratings data is used for KNN and Autorec. Tag dataset mainly contains phrases, so we use tensors to represent each word based on letters and symbols. Sparse matrix is imputed by "0" if an entry is missing.

3. Techniques and Models

To tackle sparsity problem, normally *Matrix Factorization* method is applied. *Matrix Factorization* is an advanced algorithm to reduce dimension of sparse matrix to only retain latent factors/features via decomposition of the sparse matrix (Bokde et al., 2014). To explain intuitively, a set of latent features could be genres of movies. To factorize an user rating matrix is basically finding among the movies rated high by users what categories of movies belongs to. The way of factorization of matrix is via Singular Value Decomposition: a sparse matrix R is approached by condensed matrix (not sparse) A by finding what matrix U and V should be constructed so their multiplication matrix A can approach rating matrix R as close as possible (Bokde et al., 2014). The way of finding matrices U and V is minimizing objective function with L_2 norm to prevent overfitting. However this approach faces cold start problem. Another commonly used approach is Recurrent Neural Network (RNN) which treat rating as a sequence of prediction. This approach suggests not only a set of movies we are interested in but also the order that customer consumed. The idea behind RNN is that each movie is regarded as a word, and catalog of movies as the whole vocabulary, and the movie sequence will be historic movies watched by user before (Devooght and Bersini, 2017). The state-of-art RNN used in recommendation system is "gated" RNNs such as "LSTM" and "GRU". The loss function used here is *categorical cross entropy loss* where it is correct when next item is as predicted (Devooght and Bersini, 2017). In addition, logistic regression could also be used in recommendation system. Using similarity measures, similarity between user and item can be calculated and then we could get K highest similarities namely k movies to recommend for user i (Wang, 2016). Next for each user-item pair, we apply logistic regression model to see if the item is worth to recommend to user using features from users and items (Wang, 2016).

The techniques I used in this project are the following:

- Unsupervised K-Nearest Neighbour (KNN)
- Autorec
- Long short-term memory

3.1 Unsupervised KNN

Neighborhood approach are one of the standard approach for recommendation system is K-nearest neighbour. We use KNN to illustrate the first case: item-based recommendation system. First, user inputs a movie name and using a string match algorithm a list of similar movie names are given. The string matching algorithm is called Fuzzy String Matching in Python. Next, based on the most similar movie name, we find it in our movie-user matrix as a vector of rating. Finally we find similar movies with the respect to this row based on different similarity measures. Here, we main use two measures:

Cosine Similarity is a gauge to measure how two vectors are similar to each other independent of their vector sizes. The smaller the anger the higher the similarity. Cosine Similarity is bounded between 0 and 1. A higher cosine value of the angle implies two movies are more similar.

Jaccard Similarity computes percentage of overlap between two sets of data. The difference between Jaccard Similarity and Cosine similarity is Jaccard Similarity takes into account the number of movies rated by both users but not the actual rates, however Cosine Similarity takes into account the actual rates but not the number of movies rated by both users. In sparse movie-user matrix case, Cosine Similarity is preferred because it is not always can find overlapping between rating from different users.

3.2 Autorec

Autorec is an advanced method which is in Autoencoder framework and apply to Collaborative Filtering. This approach can outperform most state-of-art models such as deep neural network, biased Matrix Factorization (Xiaoyu et al., 2015). This approach works fine for sparse user-movie matrix because it can project a complex matrix into lower dimensional latent features which is less sparse and contain high-level summary of rating information between users and movies. After we obtain hidden state h to reconstruct it into original high dimensional space but this time missing ratings will be imputed. Autorec is different from Autoencoder: we only consider observed ratings and we apply regularization in objective function to prevent overfitting in observed ratings. To construct the model, I firstly put all rating matrix as input, and then concatenate it with a bias node. Then I initialized a weight matrix, multiply this matrix with the first concatenated input layer and then apply a Sigmoid function to the product. This hidden layer will be applied again the previous procedure but we don't apply activation function so we have the same dimension as input. This is our reconstructed ratings. Error is calculated as square distance between true input and reconstructed input. The paper which proposed *Autorec* suggests that this approach can find nonlinear latent representation where as Matrix Factorization can only capture linear latent representation (Xiaoyu et al., 2015).

3.3 Long Short-Term Memory

LSTM is one type of Recurrent Neural Network and it is used learning long-term dependencies. There are many tags corresponding to each rating and there are ten different ratings in total. It is necessary to choose a model which can remember long sets of comments.

LSTM is applied to tags dataset which is already filtered if there are any missing comments from users. Comments are tokenized into words, and tensors are constructed to represent each word according to relative frequency of each work. First step is to construct a corpus which contains all vocabulary from training set and test set. Word2Vec is applied to learn words embeddings. After we obtain tensor for each word, we then apply LSTM with 128 nodes for a single layer. Finally $ReLU$ activation function is applied at output layer to obtain single node which is rating.

4. Empirical Evaluation

Firstly, we compare results in terms of KNN between Jaccard similarity and Cosine Similarity. Here is one example output of inputed movie "Sense and Sensibility" using Jaccard similarity:

1. Sense and Sensibility (1995), with distance of 0.0
2. Leaving Las Vegas (1995), with distance of 0.7370666666666666
3. Dead Man Walking (1995), with distance of 0.7412140575079872
4. Four Weddings and a Funeral (1994), with distance of 0.7484946734599351
5. Mr. Holland's Opus (1995), with distance of 0.7577873254564984
6. Birdcage, The (1996), with distance of 0.7616161616161616
7. Sabrina (1995), with distance of 0.7765079365079365
8. Piano, The (1993), with distance of 0.7766019988242211
9. Remains of the Day, The (1993), with distance of 0.781227946365561
10. Much Ado About Nothing (1993), with distance of 0.7813484562066793

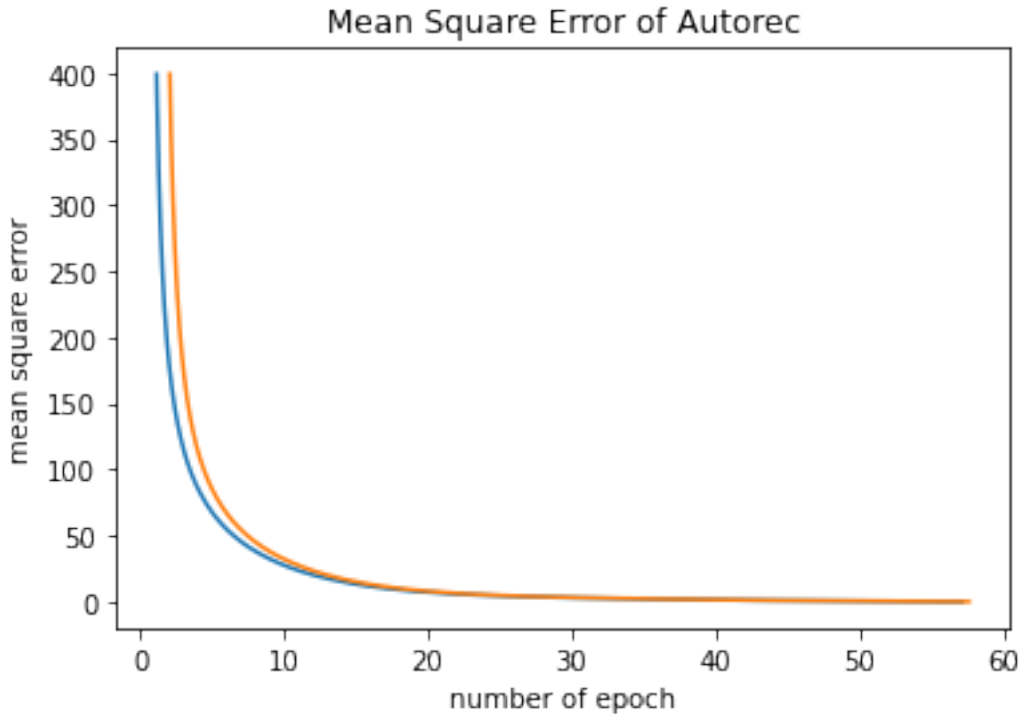
We can see movies related to "Sense and Sensibility" are belongs to the similar categories of the input. The reason could be people who enjoys love and drama will find similar movies to watch and give ratings, so users are overlapping in these films. Now, we see what is the case for Cosine Similarity:

1. Sense and Sensibility (1995), with distance of 0.0
2. Emma (1996), with distance of 0.5791304952271418
3. Remains of the Day, The (1993), with distance of 0.5838098067612358
4. Much Ado About Nothing (1993), with distance of 0.5963549432874296
5. Four Weddings and a Funeral (1994), with distance of 0.5974020349501747
6. Dead Man Walking (1995), with distance of 0.6086935510552152
7. Leaving Las Vegas (1995), with distance of 0.613800451603949
8. Mr. Holland's Opus (1995), with distance of 0.621809162729051
9. Postman, The (Postino, Il) (1994), with distance of 0.62514639294666
10. English Patient, The (1996), with distance of 0.6275259157398636

Both distance measures give the similar recommendations. Cosine similarity gives more reasonable results. We can see the first two movies based on the books written by Jane Austen.

In terms of time complexity, algorithm I used for Cosine Similarity is *brute force*. This approach find all pairs of points in dataset, suppose our movie-user matrix has D users or features, N movies or samples, this approach scales as O(DN). It takes 0.4 seconds to run. For Jaccard Similarity, I used *ball-Tree*, the algorithm has time complexity as $O(Dlog(N)$. This algorithm takes about 0.21 seconds to run.

For *Autorec*, for metric Mean Square Error, the approach achieves good result as number of epochs increase. Here is the plot for training and validation error where training is in 'orange' and validation is in 'blue':



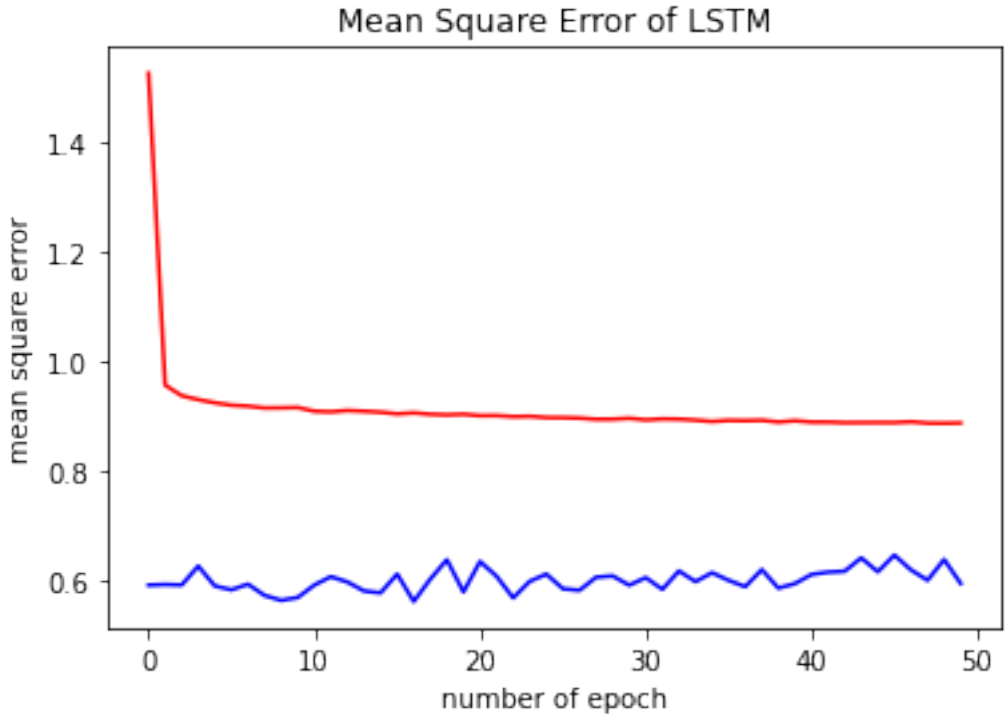
As training converges, the final Mean Square Error is around 2 for both validation and training set. The error depends on percentage of data I used for this model. By using all dataset of MovieLens-latest, the Mean Square Error could be 0.831 if using same activation function sigmoid at hidden layer h and identity link at output layer (Xiaoyu et al., 2015). The method takes about 1 minute to run. Here is one example of predicted ratings and recommended movies for an user who watched action and drama movies before :

1. Shawshank Redemption, The (1994), with rating of 4.704380989074707
2. Forrest Gump (1994), with rating of 4.326220989227295
3. Pulp Fiction (1994), with rating of 4.288584232330322
4. Silence of the Lambs, The (1991), with rating of 3.926417350769043

5. Matrix, The (1999), with rating of 3.648271083831787
6. Star Wars: Episode IV - A New Hope (1977), with rating of 3.3442983627319336
7. Schindler's List (1993), with rating of 2.977856397628784
8. Jurassic Park (1993), with rating of 2.83685564994812
9. Braveheart (1995), with rating of 2.8178184032440186

We can see that the movies recommended to users are popular movies received good rating from other users. With low MSE and reasonable predictions from *Autorec*, we can see this approach is a reliable recommendation system.

For *LSTM*, the Mean Square error for training set, validation set and test set are 1.01, 0.64, 0.72. Compared with MSE from *Autorec*, this approach can achieve even lower MSE even though we used different datasets to make prediction. The plot of training and validation error is shown in the graph where training error is in 'red, validation is in 'blue':



Both neural network approaches show no sign of overfitting and both approaches gives lowe MSE and reasonable rating in terms of input. The time complexity for LSTM for each weight and each time step is $O(1)$ (Sak, 2014). It is $O(W)$ for each time step for all parameters,suppose W is total number of parameters, in our project the number of parameters are 188,033, and total time steps (epochs) are 50. Therefore, time complexity in total is $O(50 * W)$ or $O(9401650)$.

In terms of ease of use, unsupervised KNN is the easiest to implement and obtain top k recommendation. The input is just a name of a movie, whereas for LSTM input will be a list of rating of movies watched so far and requires a padding of 0's for all unseen movies.

For Autorec the input is comments about a movie but we still need to give more movies' comments in order to give top k list of recommendation since input comment and output rating is one-to-one.

5. Conclusion

KNN is the baseline recommendation system with quick search capability if an user is watched several movies and hope to recommend the same genres of movies. Users need to give more ratings in order to get more types of movies recommended because recommendation system faces sparsity problem. However, if users hope to see what other genres of movies which also rated good by other users, they can go for recommendation system modelled by *Autorec*. This method can use the problem of sparsity and impute missing rating by generate new ratings. *LSTM* is used for movies if comments are given by users and movies with top predicted ratings will be given back to users.

In the future, both ratings and tags data set could be combined together and apply ensembled learning method to give more precise prediction of ratings. On top of this, unsupervised KNN could be used at first to group users in clusters and within the clusters *Autorec* and *LSTM* could be applied to give cluster-specific recommendations in terms of user characteristics and movie characteristics.

References

Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhaya. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 2014.

Robin Devooght and Hugues Bersini. Collaborative filtering with recurrent neural network. *arXiv*, 1(1), 2017.

Hasim.et al. Sak. Long short-term memory based recurrent neural network architectures for large vocabulary speech. *arXiv*, 2014.

Yaozheng et al Wang. A mobile recommendation system based on logistic regression and gradient boosting decision trees. *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.

Zhang Xiaoyu, Dai CHaofan, and Zhang Yanpeng. Logistic recommendation algorithm based on collaborative filtering. *2nd International Workshop on Materials Engineering and Computer Sciences*, 2015.