Deep Movie Recommender Systems

Abby Yangzi Deng
MSc Computer Science

Echo Liu
BSc Computer Science

Yudong Luo
MSc Computer Science

Tatyana Mozgacheva
MSc Computer Science

MoHan Zhang
MSc Computer Science

Abstract

Movie recommender systems are a crucial part of online streaming services, as they effectively help users find videos that they might enjoy watching. Many deep-learning based recommendation systems have been proposed in recent years. In this report we investigate and compare two recommender systems that utilize deep learning. We compare the performance of these models using Precision and Hit Rate. We also implement a few non-deep learning baseline models to compare with the more complex models. All models are run and evaluated using the Movielens 1-Million Dataset.

1 Introduction

With the rapid development of steaming technology, online streaming service giants, such as Netflix, Youtube, and Amazon, are increasingly used by users to enjoy video contents. In fact, it is estimated that in 2017, Netflix users collectively watched more than one billion hours of videos per week[1]. Gomez-Uribe *et al*. [1] have shown that roughly 80% of hours streamed at Netflix are directly or indirectly influenced by the recommended videos. Thus, recommender systems have become an integral part of streaming services, and they directly impact how video content in today's world. Recommender Systems are mainly divided into three classes: collaborative filtering, content-based and hybrid [2]. Due to the success of deep learning, many deep-learning based movie recommender systems have been proposed for each class in recent years. For example, Lund, Ng and Lee propose a deep learning approach for collaborative filtering using Autoencoders, Linear Boltzmann Machines, Variational Autoencoder Recurrent neural networks [3]. In this report we investigate a few deep learning frameworks and compare their performances in terms of Precision and Hit Rate. We also implement a set of non-deep learning baseline models to compare with the deep learning models. The rest of the report will be structured as follows: in Section 2, we provide a quantitative analysis of the dataset that we use; in Section 3, we discuss each of our model in detail; in Section 4, we describe our error metrics and present our quantitative findings; in Section 5, we describe some additional methods we tried that are not compared with the deep learning frameworks; in Section 6, we discuss our findings and some future directions of research in this area.

2 Data analysis

MovieLens was created in 1997 by GroupLens Research. It is a web-based recommender system that recommends movies for its users to watch. The MovieLens Dataset is real data gathered from MovieLens website for research purposes. We use MovieLens 1-Million Dataset[2] in this project

[1] https://techcrunch.com/2017/12/11/netflix-users-collectively-watched-1-billion-hours-of-content-per-week-in-2017/

[2] http://files.grouplens.org/datasets/movielens/ml-1m.zip

for model construction and comparison. This dataset consists of 1,000,209 ratings by 6,040 users across 3,952 movies. The ratings are on a 5-point scale, starting from 0.5 and going to 5. All users had rated at least 20 movies and the following demographic information is included: gender, age, occupation. The dataset has additional information about the movies in the form of genre. The dataset was split into two partitions: training and test by sampling randomly in the ratios 80%, 20% respectively. The bigger existing dataset presented significant scalability challenges. Cedar was employed to attempt to resolve scalability issues. However, due the project time limit, this project deals with above described dataset.

Before any modeling, the Exploratory Data Analysis has been performed in order to understand the data assumption free. Figure 1 shows that users are quite generous in their ratings. The mean rating is 3.58. Half the movies have a rating of 4 and 5. Further analysis reveals that all users have different rating styles. For example, some users are lenient in their ratings, whereas others gives lower ratings to almost all movies. Empirical Cumulative Distribution Function (ECDF) plot, in Figure 2, represents the distributions of number of ratings by user and by movie. In general, it shows that a large fraction of movies and users have few ratings associated with them, but a few movies and users have many more ratings. This means that the matrix of possible ratings is quite sparse, and that it is necessary to have models that deal with this lack of data and incorporate additional information such as movie name, genres, age, occupation, gender, not just ratings.

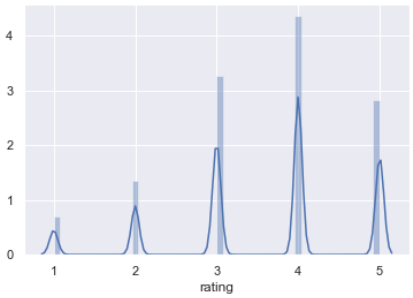

Figure 1: The Rating Distribution

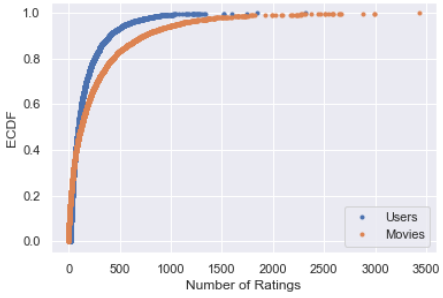

Figure 2: Empirical Cumulative Distribution Function Plot

Speaking of genre, the popularity of genres are shown in Figure 3 and reveals that Drama and Comedy are the main genres in cinematography. Figure 4 shows a Word Cloud which identifying words that frequently appear in movie titles. According to the Word Cloud, There are a lot of movie franchises in this dataset (e.g. II and III) and apparently, fourth title in a franchise appears to be rare.

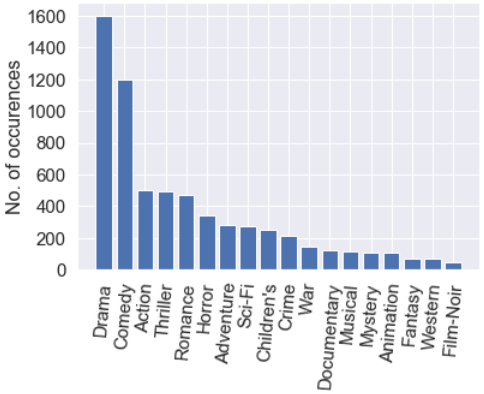

Figure 3: Number of Movies per Genre

Figure 4: Movie Title Word Cloud.

3 Models

3.1 Baseline Models: Random and Content-Based Filtering

We implement a Random Algorithm that randomly recommends 10 movies for each user, and Content-Based Filtering using Cosine Similarity [4].

The idea behind content-based filtering is to recommend movies to a user based only on other movies that user has watched. We first compute the Term Frequency-Inverse Document Frequency (TF-IDF) (which is frequency of a word occurring in a movie, down-weighted by the number of movies in which it occurs) for each movie. We then compute the Cosine Similarity score between all pairs of movies based on the TF-IDF Score. The formula for computing Cosine Similarity is defined as follows:

$$Cosine\ Similarity(V, W) = \frac{\sum_{i=1}^{n} V_i * W_i}{\sqrt{\sum_{i=1}^{n} V_i^2} * \sqrt{\sum_{i=1}^{n} W_i^2}}$$

, where V and W are two vectors. Finally we take the last watched movie of a user, and recommend top 10 (unwatched) most similar movies to the last watched movie.

3.2 YOUTUBE-NET

The first deep learning model we use is YOUTUBE-NET. This neural network framework is proposed by Paul Covington *et al*. [5] for YouTube video recommendations. The framework consists of two parts, one for candidate generation and one for ranking. As the scale of YouTube video corpus is really large, first generating candidate and then ranking could well tackle this issue. However, MovieLens is a lightweight dataset for building recommendation systems. Hence we just adopt part of the framework and add new features (e.g. titles) to build our own model with less complexity.

The overall structure of our YOUTUBE-NET is illustrated in the left figure of Figure 5. The intuitive idea is that we approach recommendations to be an extreme multi-class classification problem, where the prediction problem for one person becomes classifying a certain movie watch w_t at time t among all movies (classes) i given user info U and context C, where C contains movie info and view history.

$$P(w_t = i|U, C) = \frac{exp(m_i e)}{\sum_{j \in M} exp(m_j e)}$$

where $e \in R^N$ represent the high-level embedding of U and C, m_j represents the embedding of each movie. The goal of this neural network is to learn the high-level embedding of U and C to discriminate among movies with a softmax.

The user info and watched movie ID along with genres are mapped to a dense vector representation via embedding, then TF-IDF feature of movie title is concatenated. The high level embedding of U and C are learned jointly using two layers of fully connected Rectified Linear Units (ReLU) [6].

3.3 EUM-CT NET

The structure of our EUM-CT (**E**mbedding **U**ser **M**ovie info along with **C**onvolutional **T**itle info) NET is illustrated in the right one of Figure 5. The difference between EUM-CT and YOUTUBE NET (YTB-NET) is that we deploy a Convolutional Neural Network (CNN) to engineer features from movie titles and the user and movie vectors are seperate features whereas in the YTB-NET model they are learnt in tandem. Further, the top-k movies are selected by nearest neighbor, whereas we deploy class probabilities for YTB-NET.

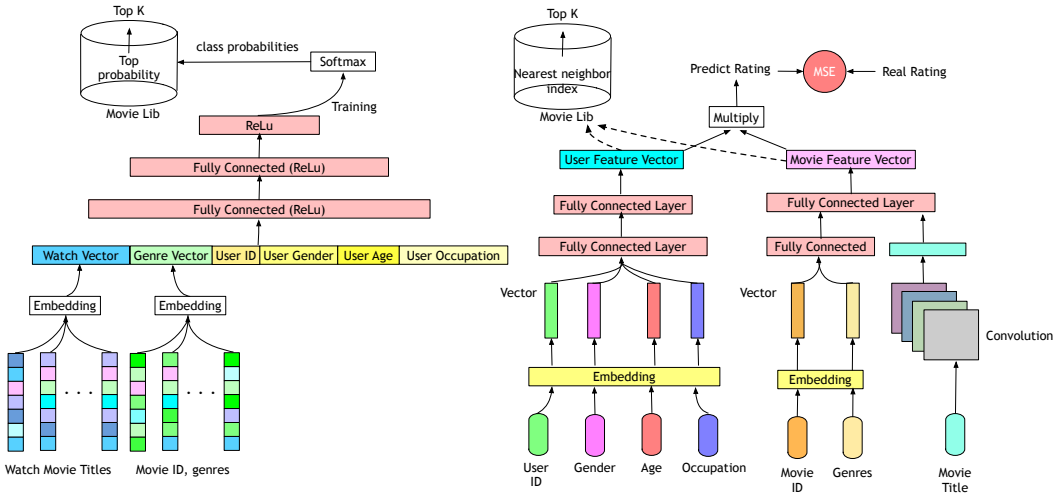

Figure 5: YOUTUBE NET and EUM-CT NET Architecture

We use the CNN proposed by Yoon Kim [7] to engineer features based on movie titles. The CNN was trained with one CNN layer on top of pre-trained word vectors obtained from an unsupervised neural language model. The first layer of CNN is a matrix composed of concatenated vectors extracting features for each word. Secondly, the convolutional layer applies a kernel over words to produce a feature map. We then apply a max pooling operation over the feature map and take the maximum value as the feature corresponding to this particular filter. The idea is to capture the most important feature which is the max of each feature.

The details of this architecture are shown in Figure 6.

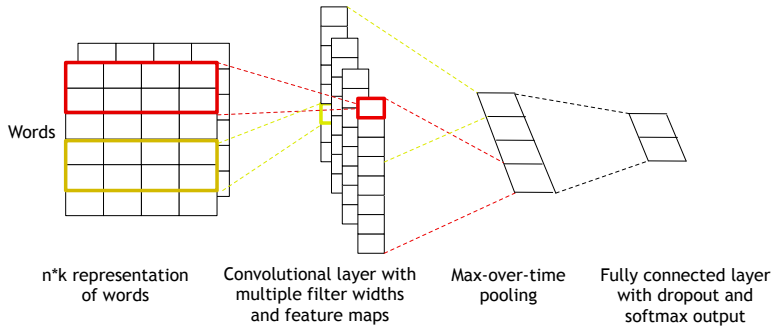

Figure 6: Convolutional Neural Network for Title Feature Extraction

After the extraction of title features, the features are combined with movie id and genre embedding to feed into a fully connected layer. The user info embedding is fed into another two fully connected layers to get a high level user feature vector. Finally, the user and movie feature vectors are learned separately to predict the rating score for a certain movie. The loss function for this model is Root Mean Squared Error between the predicted ratings and the actual ratings. The top-K movies are predicted by finding the nearest neighbours to each user using a combination of User and Movie Feature Vectors.

4 Experiments and Results

All the models are evaluated on dataset that was described in Data analysis section. Usually, the recommendation given by a recommender system is based on the user's view history. We first align

all the movie records along timeline using timestamps. Then the dated records serve as the training set and the newer records serve as the validating and testing set.

4.1 YOUTUBE NET

We build our YOUTUBE NET with the Tensorflow framework. The number of neurons for the two fully connected layers are chosen by searching from [50, 100, 200, 400] and we use the number 50 for final model setup. The activation function for each layer is ReLU [6]. The learning rate is chosen by searching from [0.1, 0.01, 0.001, 0.0001] and we use learning rate 0.01 for final model setup.

4.2 EUM-CT NET

The EUM-CT NET is also built with the Tensorflow framework. The number of neurons for the two fully connected layers of user info part are 128 and 200, respectively. The number of neurons for the two layers of movie info part are 64 and 200, respectively. These neuron numbers are got based on our initial experiments. The activation functions for all layers are ReLU [6]. The convolutional neural network has 8 filters and the filter stride is [2, 3, 4, 5]. The learning rate is set to be 0.0001 by searching from [0.1, 0.01, 0.001, 0.0001].

The training error of EUM-CT NET is shown in Figure 7 and the test error is shown in Figure 8. We can see that the training loss converges and stays stable after thousands of epoch. The test error fluctuates at some small value with descending tendency if we do linear regression.

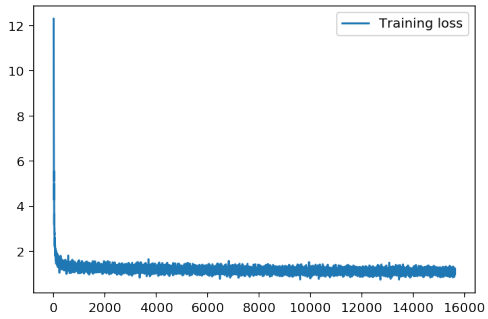

Figure 7: Training Loss of EUM-CT NET

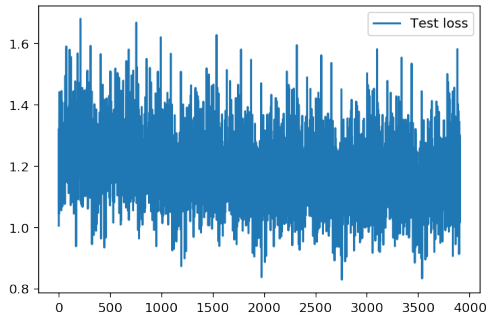

Figure 8: Test Error of EUM-CT NET

4.3 Error Metrics

We use Precision and Hit Rate to evaluate our models.

Precision is defined as the fraction of test users that watched at least one movie from our predicted list of movies. That is, we deem a recommendation successful if the user watched some movies returned by our models. Hit Rate is defined as the fraction of the predicted movies that users actually watched. We use this to check that among the list of movies predicted, what percentage of them are actually successful.

4.4 Results

We let each model recommends 10 movies to the user. The appropriate results for the precision and recall are listed below:

Methods	EUM-CT Net	YTB Net	KNN	Random
Precision	**0.774**	0.623	0.170	0.083
Hit Rate	**0.137**	0.113	0.006	0.0027

Table 1: Precision and Hit Rate of the Four Different Methods

As seen from the results both deep learning models significantly outperform the baseline models. This is the expected behaviour, as the networks capture much more features than the baseline models. The EUM-CT Model outperforms the YOUTUBE model. This is also expected, since EUM-CT employs a more complex structure. Also, since EUM-CT uses a CNN to process movie titles instead of TF-IDF used by YOUTUBE NET, we believe that EUM-CT is better able to capture the semantic meanings of the movie titles. However, EUM-CT still does not achieve a very high accuracy. We think the reason is that the dataset is not large enough to utilize the structure of EUM-CT Net to its full extent. Namely, EUM-CT is too complex for a dataset as small as Movielens 1-Million. Thus we hypothesize that if the dataset was larger in scope then the test error will be smaller than what we have achieved here.

5 Trial

This section covers models that were implemented, however they don't fall under the proper experiments since they have different set of features or output and can not be compared with models from Models section.

5.1 Bi-Directional Recurrent Neural Network

For the recommendation N-top Movie problem, RNNs can taking the evolution of users taste into account [8]. BiRNN consists of forward and backward RNN structure (GRU cell). In the forward RNN, the input sequence is sorted in ascending order by timestamps (chronological order) for each user. The backward RNN takes the input sequence in the reverse order. To compute the final prediction. The output of both GRU is fed into the last (softmax) output layer. We recommend the N-movies that have the highest values in the output layer.

5.2 Rating Prediction Models

5.2.1 Neighbourhood-Based Collaborative Filtering

We implement Neighbourhood-Based Collaborative Filtering (NBCF), a method intended for rating prediction, to predict movies. In NBCF, we first compute a similarity matrix between all pairs of users using Pearson Correlation [9] based on the ratings that the users gave. The similarity between user x and user y is given by the formula

$$\frac{\sum_{m \in Movies}(R_{x,m} - \overline{R}_x) * (R_{y,m} - \overline{R}_y)}{\sqrt{\sum_{m \in Movies}(R_{x,m} - \overline{R}_x)^2 * (R_{y,m} - \overline{R}_y)^2}}$$

where $R_{i,m}$ is the rating that user i gave to movie m and $\overline{R}_i$ is mean rating that user i gave. To predict ratings for a particular user u for a particular movie m, we take the K users with the highest correlation score and compute a weighted combination of the K users' ratings for movie m, using the formula below

$$R_{u,m} = \frac{\sum_{k \in K \ Selected \ Users}(R_{k,m} - \overline{R}_k) * P_{u,k}}{\sum_{k \in K \ Selected \ Users} P_{u,k}}$$

where $P_{u,k}$ is the Pearson Correlation between users u and k. Finally we scale round the rating to one of $0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5$ to simulate the real rating. After measuring the precision and hit rate of this method, we found that it performs worse than random, and thus we conclude that NBCF is not suitable for the task of movie prediction.

5.2.2 Variational Autoencoders

A variational autoencoder [10] is identical to a neural network except the output is identical to its' input. The procedure to deploying autoencoders for recommendations involves turning the users and their respective ratings into embedding vectors to feed as input into the variational autoencoder. The variational autoencoder uses a two layer encoder and two layer decoder network layer.

6 Conclusion and Future Work

In this paper, we developed two models to outperform baseline models for recommender systems. Furthermore, in the process we implemented two models from scratch and devised new architectures and deployed hybridized architectures by manipulating embeddings. In the future, we plan to test our models against a more modern data set. It will be interesting to observe whether the trends in movie preferences have changed over the years, by applying our models to newer datasets. Investigating the scalability of the models is another direction for future work. The ability to scale to large datasets is crucial for the success of recommender systems. Some methods for increasing scalability include fine tuning the GPU parallelization feature in modern frameworks and distributed computing frameworks, such as Spark or Hadoop, on compute clusters. To improve on the model themselves, we aim to test more approaches such as applying Bayesian Networks and Social Network Analysis to correlate social network data from each user and predict what movies they would prefer based on social media data.

7 Contributions

Echo Liu: Implemented Variational AutoEncoder. Assisted in basic troubleshooting of EUM-CT NET. Conclusion section. Experiment and Results Sections.
Abby Deng: Implemented EUM-CT NET. Model, Experiment and Results Sections.
Yudong Luo: Implemented YOUTUBE NET. Model, Experiment and Results Sections.
Tatyana Mozgacheva: Project management (The task assignment, meetings, setting goals, etc.), Data analysis Section, Implemented BiRNN.
MoHan Zhang: Implemented Random, Content-Based Filtering and Collaborative Filtering algorithms. Abstract, Introduction and Models Sections.

References

[1] Gomez-Uribe, Carlos A., and Neil Hunt. "The netflix recommender system: Algorithms, business value, and innovation." ACM Transactions on Management Information Systems (TMIS) 6.4 (2016): 13.

[2] Zhang, Shuai, Lina Yao, and Aixin Sun. "Deep learning based recommender system: A survey and new perspectives." arXiv preprint arXiv:1707.07435 (2017).

[3] Lund, Jeffrey, and Yiu-Kai Ng. "Movie Recommendations Using the Deep Learning Approach." 2018 IEEE International Conference on Information Reuse and Integration (IRI). IEEE, 2018.

[4] Huang, Anna. "Similarity measures for text document clustering." Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand. 2008.

[5] Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016.

[6] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011.

[7] Asela G., Guy Shani. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. Journal of Machine Learning Research 10 (2009) 2935-2962. Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

[8] Devooght, Robin, and Hugues Bersini. "Collaborative filtering with recurrent neural networks." arXiv preprint arXiv:1608.07400 (2016).

[9] Gunawardana, Asela, and Guy Shani. "A survey of accuracy evaluation metrics of recommendation tasks." Journal of Machine Learning Research 10.Dec (2009): 2935-2962.

[10] Masci, Jonathan, et al. "Stacked convolutional auto-encoders for hierarchical feature extraction." International Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 2011.