Data Analytics and Artificial Intelligence
Vol: 1(3), 2021
REST Publisher
ISBN: 978-81-948459-4-2
Website: http://restpublisher.com/book-series/daai/

Deep Learning-Based Movie Recommendations

M. Geethanjali, P. Madhubala
St. Joseph's College of Arts and Science for Women, Hosur, Tamil Nadu, India.
Don Bosco College, Dharmapuri, Tamil Nadu, India.
Corresponding Author Email: geethanjalim26@gmail.com

Abstract
Especially in streaming services, recommendation systems play a critical role in item suggestions. Recommendation engines are vital for streaming video services like Netflix because they enable consumers to discover new films that they will like. In this research, we present a deep learning method based on autoencoders to generate a collaborative filtering system that anticipates user-submitted movie evaluations by utilizing a vast library of user ratings. We investigate the application of deep learning to predict users' ratings on new films, enabling movie recommendations, using the Movie Lens dataset. We contrast our deep learning method with two common collaborative filtering approaches, k-nearest-neighbor and matrix-factorization, to confirm its uniqueness and accuracy. The experimental results demonstrate that our recommendation system performs better than a user-based neighborhood baseline in a survey where people evaluate recommendations from both systems and in terms of root mean squared error on anticipated ratings.
Keywords: collaborative filtering, deep learning, and movie suggestion

1. Introduction

Customers are using movie streaming services like Netflix, Hulu, Amazon Prime, and others more frequently to watch videos. For instance, over 140 million hours were watched daily by all Netflix subscribers in 20171, and the company made over $11 billion in revenue that same year [5]. Approximately 80% of the hours streamed at Netflix were impacted by their in-house recommendation engine. The existence of movie streaming services has unquestionably become a necessary component of our modern video consumption habits, and the significance of movie recommendation algorithms is as important. In light of this, we suggest tackling the issue of collaborative filtering based on deep learning technique for movie suggestions. Recommendation engines play a critical role in assisting consumers of movie streaming services such as Netflix in finding new and entertaining material. Although much of the system's specifics are kept under wraps, we do know that it combines a number of distinct recommendation systems, including some methods that make use of cooperative filtering systems. Given this, we investigate the issue of collaborative filtering for movie recommendations. A recommendation system technique known as "collaborative filtering" uses both the individual user's ratings and the ratings of users who are similar to them. The core premise is that if we can anticipate movie ratings with any degree of accuracy, we should be able to suggest new films to consumers, including ones they might not have thought of previously. Consequently, collaborative filtering in the context of movie recommendation seeks to forecast unknown movie ratings for a specific user based on both the user's known ratings and the ratings of other users inside the system for that particular movie. When there are additional users who share the same preferences, collaborative filtering accounts for a wider range of tastes than content-based algorithms. By identifying comparable users, new goods can be suggested with the presumption that the user in question will enjoy items that similar people enjoy. Collaborative filtering can be done in a variety of ways, for example, by using user profiles and k-nearest neighbor clustering [2]. Many methods have been proposed for calculating similarity; however, a straightforward method is to utilize some measure of similarity (e.g., cosine similarity) between the vectors that constitute a user profile. Instead, using the theory that consumers who enjoy one item will also enjoy related items, the k-nearest-neighbor technique computes similarity between pairs of items [11]. Matrix factorization is another often used technique for collaborative filtering [8]. By using this method, a user-item matrix is factorized into two matrices, where some latent components are represented by the inner dimension. In order to recommend new items to users based on the latent factors, the resulting factorization reflects both users and items in terms of the latent factors. Deep learning has proven to be capable of handling recommendation problems recently. Deep learning algorithms have been gaining traction in recommender systems because of their cutting-edge performances and excellent recommendations. Deep learning offers a greater grasp of customer desires, item features, and history interactions than typical recommendation algorithms. This is how the remainder of the paper is structured. The most widely used techniques for collaborative filtering are covered in Section II. These techniques function by calculating the neighbourhoods of comparable persons or objects. On the other hand, we suggest a deep learning strategy for autoencoder-based collaborative filtering in Section III. In Section IV, we show that our method performs better than the neighbourhood-based baseline. In Section V, we offer a final observation.

Related Work: The most popular technique for collaborative filtering is to have participants use the k-nearest-neighbor method [2]. This method begins with a user-item matrix R, where the value 0 denotes the absence of a certain rating and Ri,j provides the rating of user i for item j. Utilising R · RT, one can calculate the user-user similarity matrix S from R, where Si,j represents the similarity between users i and j. It should be noted that populating S with alternative distance metrics, such as the cosine similarity or correlation similarity measure, works just as well. After computing S, we may estimate user i's rating for item j by calculating RT · Si, which is the mean of other users' ratings for item j weighted by how similar they are to user i.j, To forecast the rating for item j, we can also consider the k users who are most similar to user i. Empirically, this performs better than the weighted average across all users; nevertheless, computing the k nearest neighbors at test time necessitates additional work. This method is predicated on the idea that if two people gave the same item a similar rating, they probably would give other goods a similar rating as well. Local neighbors between user profiles can be computed more quickly at scale using data structures like ball trees and k-d trees, which are a binary space division tree in k-dimensions. Instead of using user similarity, the alternative k-nearest-neighbor strategy considers item similarity between pairs, assuming that consumers who enjoy one item will also enjoy related items [11]. Using this method, we calculate an item-to-item similarity matrix I as RT ·R. Just like previously, we can populate I with additional similarly metrics. We may compute Ri·Ij, which yields an average of user i's ratings weighted by how similar those things are to item j, to forecast user i's rating on item j. User-user collaborative filtering can be more performant in recommender systems because there are typically many more users than items. Matrix factorization is another often used technique for collaborative filtering [8]. This method uses techniques like singular value decomposition (SVD) to factorize a user-item matrix into two matrices, with the inner dimension indicating some latent factors. Users and items are both usefully represented in terms of the latent factors by the resulting factorization to suggest fresh products. Our initial studies on movie ratings show that user-user neighborhood approaches outperform matrix factorization, much like item-item neighborhood approaches do. Natural language processing is one of the many computer science disciplines that deep learning has transformed [9]. In spite of this, deep learning is still relatively new and has not gotten much attention in the recommender system field [18]. That being stated, Wang et al. [15] introduce a collaborative deep learning (CDL) model that combines collaborative filtering for the ratings matrix with deep representation learning for the content information. Our approach is different from CDL's since we do not rely on content information, while the latter does. A deep learning recommendation system based on user-provided search queries and web browsing history is introduced by Elkahky et al. [3]. By mapping users and items to a latent space, they maximize the similarity between users and their favorite items. This method is limited by the requirement for users' search queries and browser history, both of which are not always available. A deep neural network model that collects item content information for the purpose of predicting ratings for cold start items is developed by Wei et al. [16]. We have a different recommendation algorithm because we don't handle user material.

Our Proposed Recommendation System: Deep learning may learn complex decision boundaries for classification or sophisticated non-linear regressions. Deep learning is really just deep artificial neural networks. Deep neural networks are able to learn complex functions by learning to extract numerous low-level properties from the data and compose them in practical non-linear combinations. This is achieved by stacking a high number of hidden layers in these networks.

Architecture of Networks: Although every computable function, such as the mapping of user profiles to movie ratings, can be approximated by neural networks in theory, choosing a neural network's architecture requires careful consideration in real-world applications. Although our network's extracted structure is flexible, there are a few good places to start. Contributions. Two n-dimensional vectors, where n is the number of films in a movie dataset, like the Movie Lens database, are the inputs used in our network architecture. A single vector represents a specific user profile, where each dimension represents the rating, the user assigned to a certain movie (or a zero to signify no rating). One-hot encoding of a specific movie is represented by the other vector, which has all other values set to zero and only on
"hot" dimension set to 1. With these two vectors, the network is asked to forecast a user's rating for a given movie.

Input: With this input format, we can eliminate the need for a single rating from a known user profile and utilise the for the item that was withheld as an example with a label. Consequently, every single one of the 26,000,000 individual evaluations represents a train example, even though we only have 270,000 user profiles.

Hidden Layers: There are numerous approaches to organising a basic feed-forward neural network. Several of the common fully-connected layers are where we start. But we also explore with other topologies, such ResNets [7], which achieve state-of-the-art performance in other domains like image recognition at the moment.

Output: Our network's output can go in two different directions. The first approach is to approach this problem as a classification problem, where the five start ratings found in the data are represented by five distinct classes. Using cross entropy as our loss function, we represent the five network outputs as unnormalized log probabilities under this architecture. We discuss the deep learning architecture that is suggested as an alternative to the user-based neighbourhood strategy, using the fundamental neural network architecture that was previously introduced. First, we take into account the neural network's input and output dimensions. A training example is a user profile (i.e., a row from the user-item matrix R) with one rating withheld. This allows us to feed the network as much training data as possible. It is necessary to calculate the network's loss on the training example in relation to the single withheld rating. This has the effect of tying each rating in the training set—rather than each user—to a specific training example. We select to utilise root mean squared error (RMSE) with regard to known ratings as our loss function since we are interested in what is essentially a regression. The root mean squared error more severely penalises forecasts that are off than the mean absolute error. We rationalise that

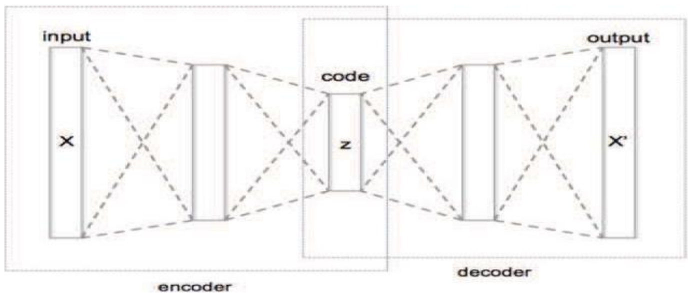

FIGURE 1. Three fully connected hidden layers in an autoencoder

A neural network that has been trained to replicate its input to its output is called an autoencoder. Its usual use is in dimension reduction—that is, in lowering the total number of random variables that must be taken into account. It has an encoder function that allows it to build a hidden layer—or several hidden layers—with code that describes the input. An input reconstruction from the hidden layer is produced by a decoder. Then, an autoencoder can be made useful by having a hidden layer that is smaller than the input layer, which forces the autoencoder to learn correlations in the data and build a compressed representation of the data in the hidden layer. This autoencoder is an example of unsupervised learning; in contrast to input-output pairings, an autoencoder simply requires unlabeled data, which is a set of input data. For linear reconstructions, the autoencoder learns a function to minimize the root mean square difference using an unsupervised learning technique.

Autoencoder: The Deep Neural Network (DNN) model is one of the current deep learning models. A Multi-Layer Perceptron (MLP) model with numerous hidden layers is called a DNN. DNN's superior parameter initialization approaches and greater number of hidden units make it unique. Greater modelling power may be achieved by a DNN model with a large number of hidden units. Even though the DNN model's learnt parameters are a local optimum, meaning they need more training data and processing resources, they can still outperform models with fewer hidden units. One particular kind of DNN is Deep Auto Encoder.

Multilayer Perceptron: Initially, the input from the row of the user-item matrix R with the rating for some item j withheld and a one-hot encoded query indicating the network should forecast the rating for user i on item j comprise the design of our recommender system. Regrettably, training this architecture has proven challenging since the network needs to learn not just about user profiles but also about how those profiles interact with query inputs. Regarding the squared root mean mistake on the training set, we were never able to obtain a loss smaller than 1.2 using this particular architecture.

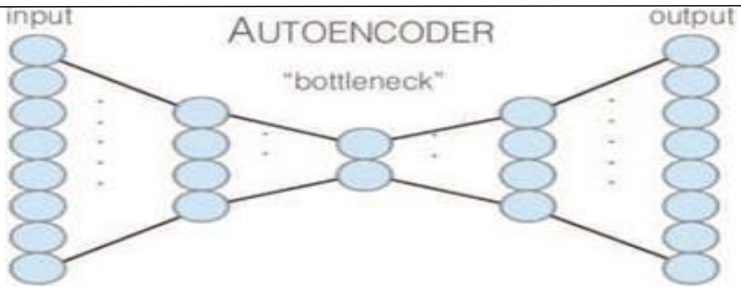

FIGURE 2. An overview of our recommender's network architecture

Rather, we base the architecture of our neural network on the idea of an autoencoder (refer to Figure 2). An input is connected to a number of completely connected hidden layers in this straightforward architecture, one of which is a hidden layer known as the "bottleneck" that is significantly lower in dimensions than the input. Next, the network's output is re-expanded until it has the same number of dimensions as the input. The network is then trained to learn the identity function with the understanding that it needs to learn a dense representation of the input in order to compute the identity function through the bottleneck. As such, one could think of the autoencoder as a kind of dimensionality reduction method. It is also possible that the bottleneck layer picks up some valuable knowledge about the input's underlying structure. A neuron located in the bottleneck layer could, for instance, represent a concept associated with a certain movie genre or set of related movies. Keep in mind that our objective is to anticipate missing ratings, not to replicate the zeros in the input vectors, therefore learning how to compute an identity function is not something we are interested in learning. Consequently, the network is actually trained using a loss function for regression (i.e., RMSE) with the goal of learning to predict missing ratings, even if our final network topology resembles an autoencoder with the bottleneck hidden layers and the matching dimensions on input and output. To be more precise, user profiles with one rating withheld serve as the network's training instances, and the output is the anticipated ratings for every movie in the dataset. We only know the response for the single withheld rating, even though the network is supposed to forecast ratings for every film based on user profiles. As such, when we learn from the training example, we simply propagate loss for the missing rating.3

The Deep Learning Recommender System: Withholding ratings does have the regrettable effect of limiting the deep learning model's ability to learn ratings for films that the user has actually seen. This is because the output on unrelated films does not directly impact the loss function. The model must generalise to some extent because of the bottleneck layer, although it can struggle. for films that diverge significantly from the films the user gave their actual ratings. It may be challenging for the model to forecast ratings for entirely unrelated films, even while consumers do occasionally see films that they score poorly and rarely rate more than a few hundred items. The fact that our network might not be able to output ratings for entirely unrelated movies does not appear to affect the test loss for the purposes of our loss function, which is root mean squared error on known ratings. This is likely because the test data's movies are sufficiently related for the patterns learned from the training data to generalise to the test data's ratings. It could be useful to add a regularisation term (explained in detail in Section III-B) to the loss because, of course, it might alter the rankings. After establishing this fundamental architecture, we experimented with multiple iterations of this architecture, varying the bottleneck layer's size and the number of levels. The most intriguing option was the small-est bottleneck layer's size. We experimented with a range of numbers before deciding on 512 as the bottleneck size. After that, we experimented with various numbers of fully connected layers, always adjusting the dimensionality by a power of two. Seven fully connected hidden layers with size [4096, 2048, 1024, 512, 1024, 2048, 4096] make up the final network structure. Every layer made use of a The non-linear activation function is the corrected linear unit 4. Using Xavier initialization [4], the biases were set to zeros and the linking weights of the hidden layers were initialized.

Clustering: Using the network's least bottleneck layer as a natural clustering mechanism has been discussed. It follows that the model has to pick up some knowledge about the fundamental structure of the input data in order to force the input into such a limited dimensional space. It was hypothesized that we could visualize that structure by displaying the movies that trigger each cluster by fixing one neuron in the bottleneck layer, zeroing out the remaining neurons in the bottleneck layer, and then optimizing the input space for this specific activation. For instance, we anticipate that there may be a single neuron or a small group of neurons that fire in response to different movie genres or cinematography techniques. An illustration of such a "cluster" is provided in Table I, where a single bottleneck neuron is triggered by optimizing the input. There is a recurring motif in these films. This network obviously has to learn some kind of structure in order to be able to predict movie ratings with any degree of accuracy. This structure is, however, more dispersed than anticipated throughout the bottleneck layer. Including a regularization term in the loss function to promote sparsity in the bottleneck layer is one possible way to address this issue. The definition of the rectified linear unit, or ReLU, is max(0; x). Despite being straightforward, it is the most advanced activation function for DNN at the moment.

TABLE 1. A Cluster When Optimizing The Input To Trigger A Single Bottleneck Neuron

Jules and Jim (1961) / Jules et Jim
Knight, the first (1995)
Final Cut of Urban Legends (2000)
The Paradine Case (1947)
The Deadly Wake (1997) ...
Cyborg 2087 (1966)

Regularization: Regularization in deep learning, and in machine learning in general, is an important concept which solves the overſtting problem. It is very important to implement the regularization while training a good model, since it is a technique used in an attempt to solve the overſtting problem. As mentioned earlier, regularization is an attempt to correct for model overſtting by introducing additional information to the cost function.

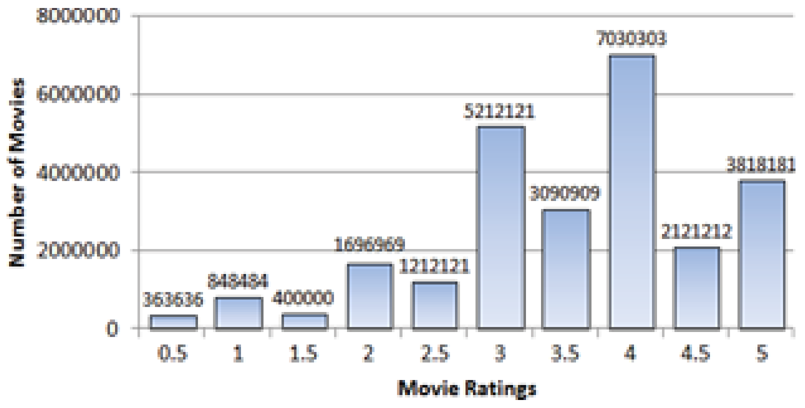

FIGURE 3. Rating distribution throughout the entire Movie Lens dataset

Experimental Results: We examine the dataset used for the empirical investigation and the experimental setting before giving the experimental outcomes of our recommendation system. After providing a brief explanation of the baseline model that was used as a point of comparison, we go on to present the Movie Lens dataset.
Movie Lens Data: The Netflix reward data made available via Google is arguably the second most well-known movie ratings dataset in academia, after the Movie Lens dataset [6].5 For our recommendation engine, we use the most recent version of the Movie Lens dataset, which is the one that is advised for development and instruction. A total of 270,896 individuals have contributed ratings for 45,115 distinct films to the Movie Lens dataset. The collection includes 26,024,289 distinct movie ratings in total. Users are able to rate films in half-star increments from one to five stars for each rating. The distribution of the ratings in the data is displayed in Figure 3. A time stamp is also included with each evaluation. We divided the data into training and test sets using these time stamps because the dataset does not have a typical train/test split. The oldest 90% of the data that comprised the most recent training set 10% of the test set's total data. We purposefully mimicked the issue that the real-world movie encountered. Recommendation systems that have access to all available data up to a specific moment in time must forecast future movie ratings.

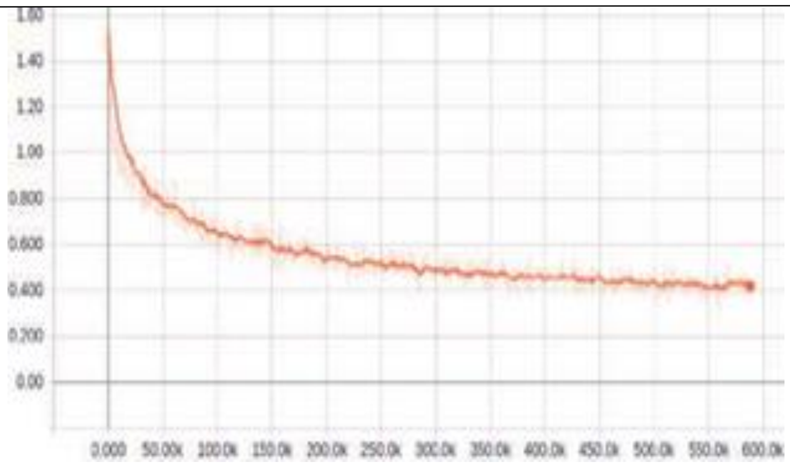

FIGURE 4. A graph demonstrating the decline in loss (root mean squared error) with time.

Full Dataset Versus BaseLine:
There are 1,000 training examples in each step. In terms of root mean squared prediction error, the user-user neighbourhood technique that uses a neighbourhood size of five and cosine similarity works well. We employed all of them on the entire MovieLens dataset in our empirical investigation. In order to fully vectorize these methods, we allotted enough RAM. For example, we created a user-user similarity matrix, which required about 600 GB of RAM, in order to process the vectorized version of the user-user closest neighbour approach. It took almost a week to complete the method in its non-vectorized brute force form. As an alternative, you can use the Baseline dataset, a condensed version of the Movie Lens dataset with 1,682 films and 943 users, as a development dataset. By dividing the Baseline database into a train and test set, we can calculate the root mean squared error of each suggested baseline algorithm's predictions.

Results: We trained the architecture given in Section III-A using 90% of the full Movie Lens dataset. Using a Titan X GPU, it took about 4 days to make 30 runs over the entire data before the training loss stabilized. We talk about our model's performance on the test set and contrast its outcomes with those of the neighbourhood models based on users.
Root Mean Squared Error:
Table II presents the findings of our model-based approach against the neighbourhood baseline based on user input. Our method is stable at 0.42 when applied to training data. Since the neighbourhood strategy just uses the training data itself to generate predictions, it has acquired parameters. There is therefore nothing to report regarding training loss.

TABLE 2. Root Mean Squared Error (Rmse) For Our User-Based Neighborhood Baseline and Autoencoder Inspired By Our Model-Based Approach

Methods	User-User KNN	Model-based
Train	N/A	0.4209
Test	11.6715	0.3544

Our deep learning model-based solution performs significantly better on test data than the neighborhood approach.It should be mentioned, though, that we don't really care about the error when it comes to suggesting films. Rather, the list of the top five highest rated films is what matters to us. While it is logical to assume that a better ranking algorithm will also have a lower root mean squared error, it is also feasible that the top-ranked films from the model-based approach yield better suggestions even with larger errors. This is particularly true when we take into account that our algorithm does not automatically discover really disparate films.

Comparing Our Movie Recommendation Systems with Others: In addition to utilising the RMSE as indicated in Table II, we also compare our deep learning movie recommendation model with a number of other well-known movie recommenders. These existing movie recommenders were picked because, firstly, they simply rely on user ratings rather than contents alone, and secondly, they achieve high accuracy in movie recommendations based on their unique models.

MF. Matrix factorization (MF) is used by Yu et al. [17] and Singh et al. [13] to forecast movie ratings. MF can be used to solve large-scale collaborative filtering challenges. Yu et al. create a non-parametric matrix factorization (NPMF) technique that successfully takes use of data sparsity and produces predicted rankings on items that are on par with or even better than the capabilities of the most advanced low-rank matrix factorization techniques. Singh et al. present a relational learning-based collaborative matrix factorization (CMF) approach that uses a given database of entities and observed relations among entities to predict user ratings on items based on the items' genres and role players, which are treated as unknown values of a relation between entities of a particular item. Several stochastic optimisation techniques are presented by Singh et al. in order to effectively handle sparse and large-scale data sets using relational frameworks. They have shown that processing relational domains with hundreds of thousands of entities using their methodology is feasible.

ML. In addition to matrix factorization techniques, probabilistic frameworks for rating predictions have been presented. A combined matrix factorization model is put forth by Shi et al. [12] to create context-aware item recommendations. In addition to factorising the user-item rating matrix, the matrix factorization model created by Shi et al. also takes the items' contextual information into account. Like in a traditional collaborative filtering model, the model can learn from the user-item matrix while also using contextual data in the recommendation process. But a key distinction between Shi et al.'s MF model and other MF techniques is that the former's contextual information is predicated on the mood of the film, whereas other MF models provide suggestions based on the cinematic context. MudRecS [10] is a recommendation engine that suggests works of literature, films, music, and art that are comparable to one another in terms of content to those that a user of MudRecS has expressed interest in. To complete the recommendation task, MudRecS does not rely on user access patterns/histories, condition information taken from social networking sites, cooperative filtering techniques, or user traits (like gender/age). It just takes into account the ratings, genres, role players (writers or artists), and reviews that consumers have left for various multimedia products. To provide recommendations, MudRecS forecasts the ratings of multimedia content that aligns with a user's interests. Netflix. Using MudRecS, we indirectly evaluate our deep learning recommendation system to the 20 systems that took part in the 2008 Netflix competition [10]. The best recommendation algorithm with the lowest RMSE score in predicting user ratings on films based on past ratings won the Netflix Prize in this open competition hosted by the online DVD rental business Netflix. The main prize of one million dollars was awarded on September 21, 2009. The Netflix website provides the RMSE scores attained by each of the twenty systems along with in-depth analyses of each rating prediction algorithms.7 Using the MovieLens dataset, Figure 5 displays the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) scores of our deep learning movie recommender and other recommendation systems. Two performance metrics that are frequently used to assess rating predictions on multimedia data are RMSE and MAE [1]. The average magnitude of mistake, or average prediction error, on improperly assigned ratings is measured by both RMSE and MAE. Whereas MAE is a linear score—that is, the absolute values of each individual difference in an inaccurate assignment are weighted equally in the average—RMSE error values are squared before they are summed and averaged, giving comparatively high weight to errors of large magnitude. Additionally, Belkor and Ensemble take into consideration temporal effects, or how a user's preference may change over time and result in varying ratings for the same film. However, the constraints of the temporal effect mean that it is not applicable to all users and that a larger proportion of training data is needed to provide consistent findings. MudRecS performs much better than 17 recommendation systems when a 95% confidence interval is taken into account, while none of the other 20 systems performs significantly worse. MudRecS outperforms Netflix's recommender CineMatch, with an RMSE score of 0.9514 on the Netflix dataset. Our deep learning renderer system was tested using the Netflix dataset, and although the results are not statistically significant, it managed to obtain a lower RMSE score (0.782) than MudRecS.

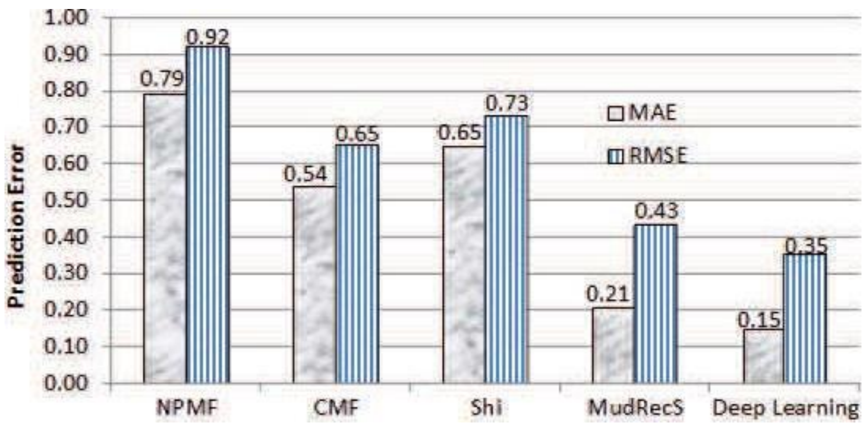

FIGURE 5. Based on the Movie Lens dataset, the MAE and RMSE ratings for several movie recommendation systems

User Evaluation: To determine the efficacy of our deep learning method for movie recommendation, we carried out user research wherein users, acting as appraisers, were given the opportunity to assess the movies that our system and the user-based neighbourhood (KNN) approach had advised. Assessors were presented with a user profile that included all of the films and ratings that each relevant user had given. After that, two potential suggestions were given to each appraiser: one based on our system and the other on the user-based neighbourhood approach. The movies that had previously received a user rating were not included in the recommendation process; instead, the movie with the highest predicted rating from either system was selected. The two potential proposals were presented in a random order. Assessors were asked to select the recommendation that they felt was more appropriate for the particular user profile (a study example is shown in Figure 6).

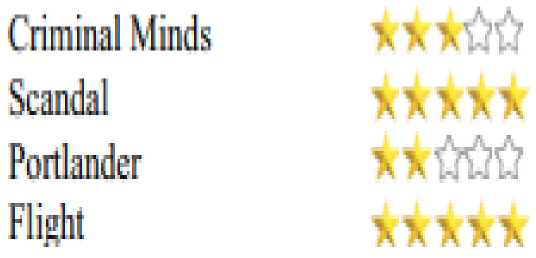

FIGURE 6. An illustration of the kinds of questions appraisers were required to respond to throughout the user assessment process for both our user-based KNN technique and our deep learning-based system. The study included one hundred participants, all of whom were students at the authors' university. Every user, who has been tasked with rating fifteen randomly selected recommendations as an appraiser. This survey's promising finding is that appraisers favoured our deep learning approach's advice 71.67% of the time above the baseline approach's recommendation. It is evident, of course, that a tiny sample size was used in this survey. Furthermore, the majority of the appraisers admitted that they had not seen the majority of the films mentioned in the poll. Aware of this issue beforehand, we stated in the survey that they were permitted to consult databases such as Google9 and IMDBa10 in order to inform their decisions.

2. Conclusion

For collaborative filtering, we have suggested a straightforward neural network model that achieves good results in terms of root mean squared error. This adds to the body of research that indicates deep learning can be an effective solution for a range of information retrieval issues [18]. Ultimately, this effort improves the ability to forecast user ratings and suggest films. Regularisation is used by our recommender system to further reduce prediction mistakes. Furthermore, our approach produced better movie suggestions and easily outperformed the neighbourhood-based baseline. Our deep learning approach has the extra benefit of being significantly more scalable during testing.

References

1. T. Chai and R. Draxler. Root Mean Square Error (RMSE) or Mean Absolute Error (MAE)? Geoscientiſc Model Development Discussions, 7(1):1525–1534, 2014.
2. W. Croft, D. Metzler, and T. Strohman. Search Engines: Information Retrieval in Practice. Addison Wesley, 2010.
3. A. Elkahky, Y. Song, and X. He. A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems. In WWW, pages 278–288, 2015.
4. X. Glorot and Y. Bengio. Understanding the Difſculty of Training Deep Feed-Forward Neural Networks. In AISTATS, pages 249–256, 2010. 9https://www.google.com 10www.imdb.com
5. C. Gomez-Uribe and N. Hunt. The Netƀix Recommender System: Algorithms, Business Value, and Innovation. ACM TMIS, 6(4):Article 13, 2016.
6. F. Harper and J. Konstan. The Movielens Datasets: History and Context. ACM TiiS, 5(4):Article 19, 2016.
7. K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In IEEE CVPR, pages 770–778, 2016.
8. Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. Computer, 42(8):30– 37, 2009.
9. W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. Alsaadi. A Survey of Deep Neural Network Architectures and Their Applications. Neurocomputing, 234:11–26, 2017.
10. R. Qumsiyeh and Y.-K. Ng. Predicting the Ratings of Multimedia Items for Making Personalized Recommendations. In ACM SIGIR, pages 475–484, 2012.
11. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Itembased Collaborative Filtering Recommendation Algorithms. In WWW, pages 285–295, 2001.
12. Y. Shi, M. Larson, and A. Hanjalic. Mining Mood-Speciſc Movie Similarity with Matrix Factorization for Context-Aware Recommendation. In Context-Aware Movie Recommendation, pages 34–40, 2010.
13. A. Singh and G. Gordon. Relational Learning via Collective Matrix Factorization. In SIGKDD, pages 650–658, 2008.
14. M. Smucker, J. Allan, and B. Carterette. Agreement Among Statistical Signiſcance Tests for Information Retrieval Evaluation at Varying Sample Sizes. In SIGIR, pages 630–631, 2009.
15. H. Wang, N. Wang, and D.-Y. Yeung. Collaborative Deep Learning for Recommender Systems. In KDD, pages 1235– 1244, 2015.
16. J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang. Collaborative Filtering and Deep Learning Based Recommendation System for Cold Start Items. Expert Systems with Applications, 69:29– 39, 2017.
17. K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast Nonparametric Matrix Factorization for Large-Scale Collaborative Filtering. In ACM SIGIR, pages 211–218, 2009. [18] S. Zhang, L. Yao, and A. Sun. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM JOCCH, 1(1): Article 35, 2017.