Movie Recommender System

Andreas Makris
Department of Computer Science Durham University
Durham, England
andreas.makris@durham.ac.uk

***Abstract*—This report aims to outline a recommender system for the domain of movies. The dataset used for this project is the 20M MovieLens dataset. The recommendation technique involves a hybrid recommender known as Wide and Deep Recommender. The recommender system is then compared to its non-personalized version, with some evaluation metrics. We also include a user interface to get recommendations for a specific user.**

Keywords—wide and deep learning, hybrid recommender, recommender systems, movie recommender

I. Introduction

A. Domain of Application and Aim

Nowadays, recommendations have become the norm in our everyday life [1]. Their applications range in multiple domains including e-commerce, business, shopping, and learning [2]. A domain that is particularly interesting and that we apply our recommendation techniques in this project is the one that concerns movies.

Recently, deep neural networks have produced tremendous results in many applications like computer vision and natural language processing [3,4]. Their metric scores in movie recommender systems have been impressive [5]. Wide and Deep Learning [6] has shown great results on Google Play and its implementation on MovieLens 100K [7] is impressive. Therefore, we check if this recommender is appropriate for larger movie datasets in particular, MovieLens 20M.

II. Methods

A. Data Description

The dataset that we use is the common benchmark, MovieLens 20M [8]. It contains about 20 million ratings, 140,000 users and 27,000 movies. We know the timestamp of each rating and have the genre and the year of release for each movie. Additionally, we have tags that users assigned to movies and tag scores for each movie.

B. Data Preparation and Feature Extraction

Firstly, we remove some ratings of the dataset. The architecture of the model doesn't allow us to use more than 10,000 movies, as we run out of GPU memory and the training of the model is particularly slow (about 16 hours for one epoch, for 10,000 movies), so we choose to use the 3,000 most popular movies. This doesn't substantially reduce the size of the dataset, as we still have about 17.5 million ratings and the same number of users.

The dataset needed some cleaning as the genres and the year of release were not in an appropriate format. We also added a column for the tags, indicating whether the user had tagged that movie or not, as tags have been useful in many approaches [10,11].

We follow the idea from the 'surprise' package and turn the user and movie ids from 'raw' to 'inner' [13]. That means that we give a new id to each user and each movie (starting form 0 and incrementing 1 for each user/movie), so that we can process the data easier and, keep the relation between raw and inner ids in a dictionary for future retrieval.

Following the implementation of [7], we view the recommendation problem as a classification problem where we want to predict the next movie viewed on a specific moment in time. Therefore, we sort the dataset by users and by timestamp and split on a timestamp, so that the ratings before are in the training set and the ratings after are in the testing set (we use 20% of the data as testing, as in [7]).

Furthermore, we need some continuous features [6,7], thus we calculate the average rating and share of ratings for each genre for each user. We also need the history of each user, that is the movies that the user has already rated [7]. This history will later be transformed into sparse [14] and index formats.

Finally, we choose the 50 tags with the highest genome score and merge them with the main dataset [11]. This provides additional contextual data about the movies and battles the cold-start problem for new movies [15]. There are plenty algorithms to address the cold-start problem for users [21], but as we don't have any contextual user data, we just remove from the dataset the first five ratings of the users (which reduces the dataset only slightly) [7].

As the dataset is huge, we cannot process it all at once. Therefore, we process the dataset in smaller chunks and save the chunks. Then, we merge the chunks to get the final dataset.

C. Recommendation Techniques / Algorithms

We implement a deep model known as Wide and Deep Recommender [6]. This is a hybrid recommender [9], as it makes use of the item features (we would also use user features, but there are none on the MovieLens dataset) and the item

ratings (in a way that we learn about a user according to his similarity to other users).

In order to implement the Wide and Deep model, we first need the movie history for each entry in the rating data frame [7]. We translate the history in two formats;

- Sparse format: we indicate with 1 if the movie was rated and 0 if it wasn't in the appropriate location of an array of length 3000 (number of unique movies in the dataset).
- Index format: One that contains the movies rated, padded with 0s. We make sure that its length is the same as the highest number of movies rated by a user in the dataset.

For instance, a user that has rated only two movies with movie ids 2 and 505 would have the following history arrays:

Fig. 1. History formats

Before passing the features from the Wide and Deep model, we use batch normalization in order to bring the features to the desired scale [23].

The Wide and Deep model has two components: a wide one and a deep one. The wide component is a simple linear model and follows the equation $y = \mathbf{w}^T * \mathbf{x} + b$,where $\mathbf{w}$ are the model parameters, $\mathbf{x}$ are the input features and b is some bias [6].

The Deep part of the architecture is a feed-forward deep neural network. Each layer (blue and yellow block in figure 2) performs the following computation $\alpha^{l+1} = f(W^{(l)}\alpha^{(l)} + b^{(l)})$ where $\alpha^{(l)}$ is the activation, W are the model weights and b is the bias at l-th layer. $f(.)$ is the activation function, in our case we use ReLU.

In our implementation, for each rating in our dataset we pass the history in index format through a simple embedding and take the mean across all movies in the history to reduce its dimension. Then, we concatenate the result with the features of that user-movie-rating pair. Subsequently, we pass the output through a deep neural network that consists of 3 linear-ReLU layers. Finally, we concatenate the output with the user history in sparse format and pass it through a final linear model.

The advantage of this architecture (both combining Wide and Deep components) is that it combines the powers of both models [6]. Wide linear models are great in memorization, while deep neural networks are excellent in generalization.

Note that this is not an ensemble hybrid recommender, but the two models are jointly trained. The weights of both models are simultaneously optimized using gradient descent and backpropagation [6].

We use Adam optimizer with a learning rate of 1e-4, cross entropy as the loss function, a batch size of 64 and train for 3 epochs (which takes about 16 hours) [7].

Moreover, we tried to change the target of the model and instead of predicting the next movie, we considered the problem as a classifier that classifies ratings (10 classes for numbers between 0.5 and 5.0, with 0.5 increments) [16].

In order to train the models, we utilize the PyTorch class 'Iterable Dataset' which allows for loading only a small amount of data every time, as loading the entire dataset at once is not possible.

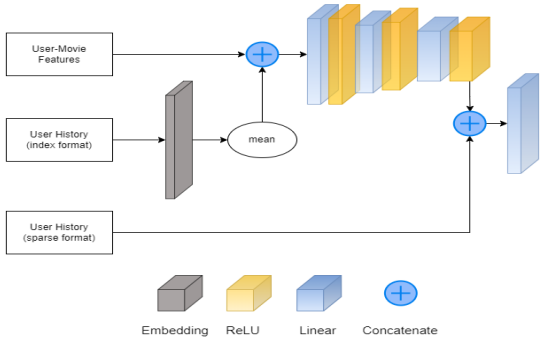

Fig. 2. Model architecture

D. *Evaluation Methods*

The accuracy metric that we choose to evaluate our recommenders on is the root mean squared error (RMSE), as it is the most common accuracy metric for different MovieLens datasets [17,18,19]. The following equation shows how RMSE works:

$$RMSE = \sqrt{\frac{1}{n}\sum_{u,i}\left(p_{u,i} - r_{u,i}\right)^2}$$

where $r_{u,i}$ is the rating of user u on item (movie) i and $p_{u,i}$ is the respective prediction of the recommender system.

On top of that we implement the mean reciprocal rank (MRR). This metric is simple, but useful as it explains how accurate the first recommendation that we make is, according to its rank [20]:

$$\text{MRR} = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{\text{rank}_i}$$

where $|Q|$ is the number of users and $rank_i$ is the true rank of the movie with the highest predicted rating for user i.

We also use prediction coverage [22] to evaluate our recommenders. Prediction coverage gives us the percentage of movies being recommended out of all the movies in the dataset:

$$\text{Prediction coverage} = \frac{|I_p|}{|I|}$$

where $|I|_p$ is the number of movies that can be recommended and $|I|$ is the total number of movies in the dataset.

III. Implementation

A. Input Interface

Recommendations can be produced through a command line interface. The users that we can recommend movies for are the ones in the testing dataset i.e., the ones with id between 111,629 and 138,493.). The command line doesn't run the recommender, but instead keeps the results in a csv file and produces them according to the user. The idea is that the model updates the datasets regularly, but as the recommendation algorithms take a long time to run, they are not being run every time a user needs recommendations. The users first input their user id in the input interface and then choose an algorithm to gain recommendations (or to go back in case they chose a wrong user id). We let the user know what type of data we collect, in order to make predictions and that they can choose between two recommenders, depending on the diversity of the recommendations that they want.

```
Enter your User Id:114789
We use data from your rating/tagging history to produce recommendations!

Please enter the type of recommender you want ("Popularity" to receive p
opular movies / "WnD" to receive more diverse recommendations) or enter
"back" to choose a different User Id:
```

B. Output Interface

The default number of recommendations we provide are 5. We don't want a very small number of recommendations (as it would provide no novelty), or a very large number (as many movies would be irrelevant to the user). The recommendations are ranked (from 1st to 5th) and include the title, year, and genres of the movies, in order to provide the user some contextual information about their recommendations. Note than in case that we don't have enough information about a user, we may recommend less than 5 movies.

```
You chose the Wide and Deep recommender
Loading recommendations...
                                        title                       genres
1                          Men of Honor (2000)                        Drama
2          Legend of Bagger Vance, The (2000)                Drama|Romance
3                       Jurassic Park (1993)  Action|Adventure|Sci-Fi|Thriller
4  Star Wars: Episode VI - Return of the Jedi (1983)          Action|Adventure|Sci-Fi
5          Independence Day (a.k.a. ID4) (1996)  Action|Adventure|Sci-Fi|Thriller
```

TABLE I. EVALUATION METRICS

	Evaluation Metrics		
	RMSE	***MRR***	***Coverage***
Popularity recommender	0.869	0.687	53.2%
Wide and Deep	1.104	0.444	58.5%
Wide and Deep (ratings classification)	1.230	0.444	58.5%

IV. Evaluation

A. Evaluation Metrics' Results

The Wide and Deep architecture achieves a RMSE score of 1.104, an MRR score of 0.444 and covers 58.5% of all the movies in the dataset.

The Wide and Deep architecture using the ratings classification, instead of the movie classification achieves a RMSE score of 1.230 and has the same MRR and coverage as the original Wide and Deep model.

As we can see from figure 3, the loss in the Wide and Deep model is increasing instead of decreasing. This is likely happening due to the wide and deep architecture not being deep enough to capture the complex interactions between users and movies.

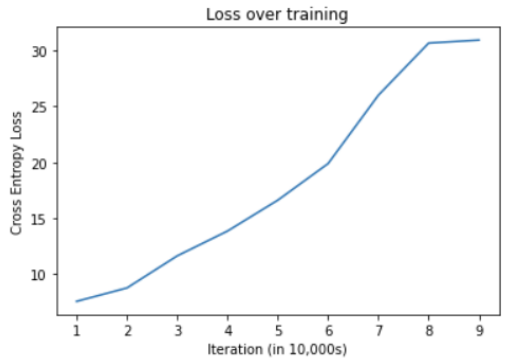

Fig. 3. Wide and Deep, Loss per Iteration

B. Comparison

The RMSE and MRR from the Wide and Deep architecture are worse than the baseline results. On the other hand, the Wide and Deep models recommend more novel and diverse movies, as shown in their coverage scores.

V. Conclusion

The Wide and Deep architecture has many limitations and is not appropriate for recommending movies in our case. The model that we used is a classification task with 3000 movies, therefore the simple architecture used here cannot capture the complex interactions between the users and the movies. In addition, training a model of this magnitude is extremely slow (classification of 3000 movies), the movie history being kept in memory demands a lot of memory and we haven't shuffled the dataset. Running and experimenting with this class of models on large scale datasets is extremely difficult with the limited resources that we had (computational power and memory).

Furthermore, this model predicts the next movie viewed for a user on a particular time. If our goal is to provide ratings the conversion between the probabilities for each movie from the model is another limitation.

Finally, this dataset suffers from lack of contextual information about the users. We have limited resources in tackling the cold start problem for new users. A simple idea to battle this is to simply recommend the most popular movies to new users.

Future work is required with higher computational power in order to determine the full capabilities of this model on recommending movies on large scale datasets.

References

[1] Lops, P., Jannach, D., Musto, C. et al. Trends in content-based recommendation. User Model User-Adap Inter 29, 239–249 (2019). https://doi.org/10.1007/s11257-019-09231-w

[2] Recommender Systems, Jie Lu (University of Technology Sydney, Australia), Qian Zhang (University of Technology Sydney, Australia), and Guangquan Zhang (University of Technology Sydney, Australia)

[3] Ashish Vaswani et al., Attention Is All You Need, http://arxiv.org/abs/1706.03762, 2021

[4] Kaiming He et al., Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385, 2015

[5] Xiangnan He et al., Neural Collaborative Filtering, https://doi.org/10.48550/arXiv.1708.05031.

[6] Cheng et al., Wide and Deep Learning for Recommender Systems, https://arxiv.org/abs/1606.07792, 2016.

[7] Pavel Salikov, Wide & Deep Learning for RecSys with Pytorch, https://www.kaggle.com/code/matanivanov/wide-deep-learning-for-recsys-with-pytorch/notebook, 2022.

[8] 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, http://dx.doi.org/10.1145/2827872.

[9] Blanca Li et al., Train Wide & Deep Recommender, https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/train-wide-and-deep-recommender, 2022

[10] Duong Tan, Nghia & Vuong, Tuan & Nguyen, Duc & Dang, Quang Hieu. (2020). Utilizing an Autoencoder-Generated Item Representation in Hybrid Recommendation System. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.2989408.

[11] Johnson Kuan, How to Build a Simple Movie Recommender System with Tags, https://towardsdatascience.com/how-to-build-a-simple-movie-recommender-system-with-tags-b9ab5cb3b616, 2019

[12] Vig, Jesse & Sen, Shilad & Riedl, John. (2023). Computing the tag genome.

[13] What are raw and inner ids, https://surprise.readthedocs.io/en/stable/FAQ.html#what-are-raw-and-inner-ids.

[14] Machine Learning & Simulation, Sparse Matrices | Coordinate (COO) Format | Intro & Implementation in C, https://www.youtube.com/watch?v=BbbCVzJt1Xk, 2021

[15] Gabriella Vas, How Recommendation Systems Tackle the Cold Start Problem, https://www.yusp.com/blog-posts/cold-start-problem/, 2021

[16] Sebastian Poliak, 1 to 5 Star Ratings – Classification or Regression?, https://towardsdatascience.com/1-to-5-star-ratings-classification-or-regression-b0462708a4df, 2020.

[17] Papers with Code, https://paperswithcode.com/sota/collaborative-filtering-on-movielens-10m.

[18] Papers with Code, https://paperswithcode.com/sota/collaborative-filtering-on-movielens-1m.

[19] Papers with Code, https://paperswithcode.com/sota/collaborative-filtering-on-movielens-100k.

[20] Muffaddal Qutbuddin, An Exhaustive List of Methods to Evaluate Recommender Systems, https://towardsdatascience.com/an-exhaustive-list-of-methods-to-evaluate-recommender-systems-a70c05e121de, 2020.

[21] Le Hoang Son, Dealing with the new user cold-start problem in recommender systems: A comparative review, Information Systems, Volume 58, 2016, Pages 87-104, ISSN 0306-4379, https://doi.org/10.1016/j.is.2014.10.001.

[22] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. 2010. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In Proceedings of the fourth ACM conference on Recommender systems (RecSys '10). Association for Computing Machinery, New York, NY, USA, 257–260. https://doi.org/10.1145/1864708.1864761.

[23] Sergey Ioffe, Christian Shift, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, http://arxiv.org/abs/1502.03167, 2018