Reel Recommendations: Leveraging Reinforcement Learning for Tailored LetterBoxd Recommendations

Luke Benham
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
`lnb6grp@virginia.edu`

Michael Fatemi
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
`gsk6me@virginia.edu`

Kasra Lekan
Department of Computer Science
University of Virginia
Charlottesville, VA 22903
`kl5sq@virginia.edu`

Abstract

We use offline reinforcement learning and language modeling to provide movie recommendations based on user data from LetterBoxd, popular movie review and cataloging website, and movie data from TMDB. We implement DeepFM, a framework for converting sparse features to dense embeddings. This approach tended to overfit the training data, reaching about 57% accuracy on test data. Additionally, we formulate the recommendation task as a language modeling task by fine-tuning GPT-2 to predict movie ratings based on a sequence of previously rated movies. Our implementation can be found at `https://github.com/anrath/ReelRecommendations`.

1 Introduction

Recommender systems play a crucial role in various domains, including entertainment, e-commerce, and social media, by providing personalized suggestions tailored to individual users' preferences. In the realm of movie recommendations, the task becomes particularly challenging due to the subjective nature of movie preferences, the abundance of available movies, and the dynamic nature of user tastes.

Letterboxd is a popular movie review and cataloging website among cinephiles (similar to GoodReads for books). The large number of reviews, the user social graph, and the long-form text reviews that people leave for their favorite movies provide a rich source of data for developing effective recommendation systems. Because we have access to an existing dataset, our learning algorithms will operate in an offline reinforcement learning setting.

Traditional recommender systems often rely on collaborative filtering techniques, which leverage user-item interaction data, such as ratings or purchase histories, to identify similar users or items. However, these methods may struggle to capture the nuances of user preferences, particularly in domains with sparse data or limited explicit feedback. Recent advances in natural language processing (NLP) and representation learning have opened up new avenues for incorporating rich textual data, such as movie descriptions, reviews, and user profiles, into recommender systems.

In this research, we explore the use of natural language descriptions of users and movies, employing techniques like SentenceTransformers to embed them into dense vector representations. Additionally, we incorporate explicit feature vectors for each movie, such as genre and length, to enhance the recommendation process. We consider methods that learn user embeddings, movie embeddings, or a combination of both, such as DeepFM, which blends these approaches while primarily focusing on movie embeddings. Furthermore, we formulate the recommendation task as a language modeling problem by fine-tuning GPT-2, a powerful language model, to predict movie ratings based on a sequence of previously rated movies.

By leveraging the rich data available on Letterboxd and combining techniques from offline reinforcement learning, language modeling, and representation learning, our research aims to develop a comprehensive and effective movie recommendation system that can capture the nuances of user preferences and provide personalized suggestions to enhance the overall user experience.

2 Related Work

Recommender systems have been an active area of research, with various approaches explored, including reinforcement learning (RL) and deep reinforcement learning (DRL) techniques. Afsar et al. [2023] provide a comprehensive survey of RL-based recommender systems, proposing a framework with components like state representation, policy optimization, reward formulation, and environment modeling. They highlight the shift towards formulating recommendation as a sequential decision problem to capture dynamic user-system interactions.

Liu et al. [2019] introduce a novel DRL-based recommendation framework that explicitly models user-item interactions. Their "Actor-Critic" mechanism aims to capture user preference dynamics and the sequential decision-making nature of recommendations, demonstrating improved performance over existing methods.

In the domain of sentiment analysis for movie reviews, Qaisar [2020] employs Long Short-Term Memory (LSTM) networks, achieving an 89.9% accuracy on IMDb data. Naeem et al. [2022] investigate machine learning models like SVM, Naive Bayes, and ensemble methods, finding SVM with TF-IDF features to be most accurate. Similarly, Mehra and Choudhury [2018] compare SVM and Naive Bayes, with SVM slightly outperforming on IMDb and Cornell datasets.

Unsupervised learning techniques like clustering have also been explored for movie recommendations. Aditya et al. [2018] compare K-means, Agglomerative, Birch, and Mean-shift algorithms, identifying K-means and Birch as particularly effective while emphasizing feature selection and cluster count importance.

Our work aims to leverage these advancements in RL-based recommenders, sentiment analysis, and clustering techniques to develop a comprehensive movie recommendation system tailored to user preferences and interactions.

3 Preliminaries

Overview. We use the DeepFM framework [Liu et al., 2019], which allows for efficient use of feature vectors. We create “user vectors" and “movie vectors", which jointly are used as a context to predict a reward value, $\hat{R}$. Because there is no intrinsic user information present in the dataset, we must use information about their past interactions with movies to estimate their future preferences. For now, we encode this by making user vectors trainable parameters (i.e., when creating the loss function for $\hat{R}$, the user vector can be optimized to represent learning their preferences).

Summary of the DeepFM Framework. DeepFM uses sparse features (such as movie and user metadata) and constructs “dense features" from those. DeepFM can be conceptually viewed as an extension of Linear Contextualized Bandits, but due to additional complexity, it is optimized with stochastic gradient descent rather than direct linear regression. Several factors are linearly combined to generate a final output logit, which can represent a rating prediction or an action. The input context is a user vector concatenated with a movie vector, and the action is represented as a single output logit, representing whether the movie should be recommended to the user.

The context vector is projected into a set of low-dimensional “dense features". These dense features are then converted into a compatibility vector by taking pairwise inner products between them to measure similarity. The results from these inner products are added to the final output logit.

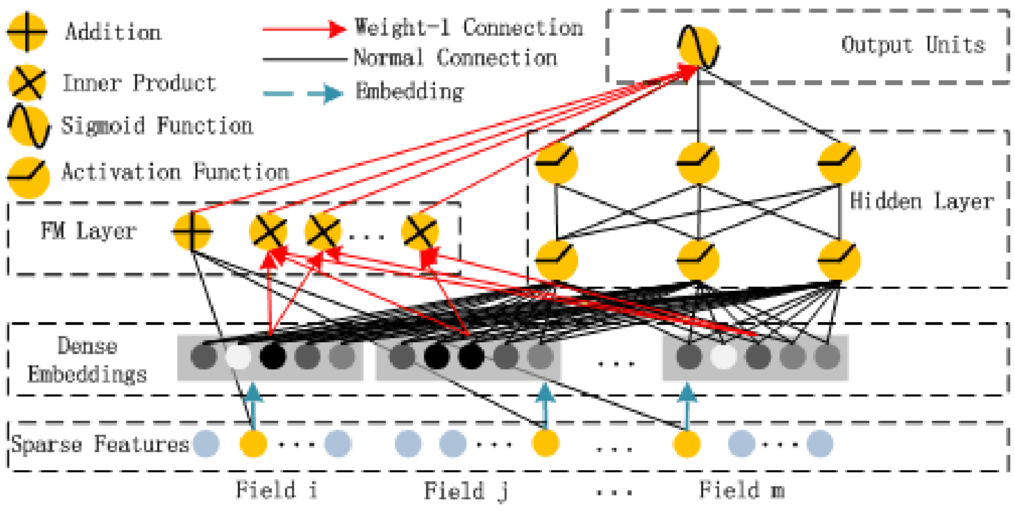

Figure 1: "Wide & deep architecture of DeepFM" from the original paper.

Additionally, these dense features are passed to a shallow multi-layer perceptron, which allows for additional tuning of the output logit.

DeepFM can be viewed as a sum of a linear contextual bandit, via the linear projection from sparse features to output logit, a factorization machine, which is the set of pair-wise inner products between dense features, and a multi-layer perceptron, stemming from the dense features. Therefore, it is highly versatile and adaptive to the specific problem at hand. Additionally, each of these components can be nullified by using zero weights, and thus DeepFM can collapse to any of these individual components if demonstrated to be the optimal modeling approach.

4 Methods

4.1 Data

We wrote Python scripts using BeautifulSoup to scrape the user pages for roughly 6500 top LetterBoxd users to extract their movie-watching data. This included timestamps, a 5-star rating, and boolean values for whether a user liked, rewatched, or reviewed the movie. We scraped the text of these user reviews as well.

We downloaded an initial Kaggle dataset of movie data up to 2022 and supplemented it with data we gathered from the TMDB API for movies released from 2022-2024. These dataset fields are listed in the Appendix.

We featurized the movie data relevant to our recommendation algorithms. For categorical variables like `genres`, `production_countries`, and `spoken_languages`, we used one-hot encoding. We converted time-based values like `release_date` and `year_released` to ordinal values. Additionally, we encoded `overview`, a natural language description of the movie, using the pre-trained model all-MiniLM-L6-v2 from the SentenceTransformers package. We chose this model because of its high performance on sentence embedding tasks and fast embedding speed.

4.2 Learning and Recommendations

DeepFM Learning Objective. Here we apply the DeepFM framework. For a given user/movie pair, we assign a reward of $r = 1$ when the user gave the movie a rating ≥ 7, and a reward of $r = 0$ otherwise. We then train our model to regress the reward value through mean-squared-error loss. To prevent overestimating model performance, we create a baseline predictor which simply estimates $r = 1$ when a movie's average rating (across all users) is ≥ 7, and $r = 0$ otherwise. Thus, if our model can outperform the baseline, it has learned a method of recommending movies to users in a more selective/personalized fashion than global metrics for movie quality.

Language Model Learning Objective. We also attempted to use movie descriptions directly as inputs into an autoregressive language model: GPT-2. The motivation was that the pre-trained representations from GPT-2 would be powerful, and by using an end-to-end approach with self-attention, we could avoid the step of training “user vectors" and instead directly infer user preferences from their interaction history.

Under this approach, we treat movie recommendations as a sequence modeling objective. Let y_t and x_t represent the user's rating and the information about the movie at time t. We give the model a reward proportional to the rating the user provides, and then train the model to estimate the reward:

$$R_t \sim P(y_t \mid y_{t-1}, x_{t-1}, \dots, y_0, x_0)$$

We let N represent the “context length": the number of ratings the language model can view in order to make its prediction. For brevity, we will from now on denote the sequence of movies and user feedback as τ_N (representing “trajectory"). We define loss via a cross-entropy objective. Because we assume implicit variance in user ratings, model the target distribution, $P(y_{N+1} \mid \tau_N)$, as a softer version of categorical classification. Given y_{N+1}, we train the model with cross-entropy loss against the distribution:

$$P(y \mid \tau_N) = \begin{cases} 0.25 & \text{if } y_{N+1} = y + 1 \\ 0.5 & \text{if } y_{N+1} = y \\ 0.25 & \text{if } y_{N+1} = y - 1 \end{cases}$$

If $y_{N+1} \in \{0, 10\}$, we clip $y = -1$ and $y = 11$ and rescale the remaining probabilities to sum to 1. The purpose of this is to enforce some numerical structure: The model should be penalized less harshly for similar predictions than for distant ones.

5 Experiments

5.1 DeepFM Experiments

First, we created a small development set consisting of 30 unique movies (and a large set of users who have provided ratings for those movies). We found that after training for a small number of epochs (50), our model predicted reward correctly with 81% accuracy, where “correctness" is defined as $\frac{\mathbb{I}\{\text{Round}(\hat{R})=r\}}{N}$ (N being the number of instances in the training set). The baseline estimator, using no user personalization, had an accuracy of 70%.

Having implemented DeepFM for our recommendations, we scaled up our experiment to test various dataset sizes (results shown in Table 1). We noted that our model strongly overfitted to the training dataset.

Table 1: Accuracy of DeepFM before overfitting correction.

Split	**Acc. (No Personalization)**	**100 Epoch Acc. (train)**	**200 Epoch Acc. (train)**	**100 Epoch Acc. (test)**	**200 Epoch Acc. (test)**
30 train, 0 test	0.7106	0.9040	0.9889	-	-
500 train, 100 test	0.769	0.7860	0.8430	0.5695	0.5732
1000, 150 test	0.745	0.6030	0.9040	0.5170	0.5369
9000 train, 1000 test	0.7322	0.8437	0.8878	0.5641	0.5662

5.2 DeepFM Overfitting

We found that the DeepFM model tended to overfit. The training accuracy would climb to over 90% rapidly but the testing accuracy remained between 50% and 60%. This was especially true of the smallest datasets used in development which approached 99% accuracy on the training set. Thus, we sought to address overfitting.

Dropout We implemented dropout and L2 regularization. The dropout was applied to the DeepFM model where it contains the variable p which is the probability for a given element to be zeroed. When $p = 0.5$ then with a probability of 50% the element will be zeroed which ensures that the model does not learn too specifically the training data but instead generalizes.

L2 Regularization: We implemented L2 Regularization in the Adam optimizer. By adding a small weight_decay term to the optimizer on the order of $1e^{-5}$ the optimizer adds a regularization term to the loss function proportional to the squares of the weights of the model. This has the effect of shrinking the parameter estimates and biases the model towards simpler solutions. By punishing complex weights, L2 regularization discourages training set memorization and limits overfitting. However, our model's accuracy only improved marginally, on the order of 1% in our splits.

Model simplification: We removed natural language embeddings from the training data, to reduce the amount of information about each movie that could be used for memorization. We also simplified the model by reducing the number of hidden MLP layers and the number of dense embeddings. Finally, we reduced user vectors to a single dimension, in an attempt to reduce the impact of potential user-level overfitting entirely. However, even with this, we were not able to improve validation accuracy.

We believe a significantly deeper inspection of model architecture is necessary to diagnose why overfitting occurs and what steps could be used to address it in DeepFM. We leave this to future work.

5.3 Language Model Experiments

Given our large dataset of movie rating histories, sorted chronologically and collated by user, we uniformly sample sliding context windows τ_N. For each movie that the user rates in a given context window, we format a context string for each movie that includes the movie's description, the genre, the languages spoken, and the average vote. Then, we add the string `Rating:  X / 10`, where `X` is replaced with the rating value.

We sample context trajectories of length 10, and truncate them to fit in GPT-2's context window (1024). Improved sampling techniques to reduce skewed sampling towards users with disproportionately more ratings than others can be left to future work. We then train the model to predict all movie ratings simultaneously, with a decoder-only attention mask to enforce rating predictions to only be made based on prior ones. We experiment with two loss objectives:

1. **Unweighted**: Apply an autoregressive loss, in which each token from the formatted movie string is trained to predict the next token. This means the model is additionally trained to predict movie details, such as the description and genre.
2. **Weighted**: Here, we only apply the loss function to rating tokens. The motivation is that less of the training objective would be dedicated to memorizing movie details, in case such information bottlenecked the model's ability to predict ratings correctly.

We train each model with $100,000$ rating trajectories, or a total of approximately $100,000,000$ tokens. This takes approximately 4 hours on a single NVIDIA A100 GPU (which we accessed through Rivanna). We use the Adam optimizer and a learning rate of 10^{-4}.

5.4 Language Model Results

We evaluate the model's performance by randomly sampling $n = 1000$ rating trajectories from a test split of the dataset. Then, we compare the model's predicted rating to the user's true rating. We plot these in 5. We find that generally, model performance increases as context length increases. It is difficult to determine whether these benefits are from the language model/LM knowing more about the user given their rating history, or from the general theme of LMs achieving better performance on a task after being shown a few examples. The weighted model has slightly higher recall, and the unweighted model has slightly higher precision, indicating that it is ambiguous whether changing the loss objective had a strong positive impact.

We show scatterplots as a function of context length for each model: The model trained with the unweighted objective is shown in 2, and the model trained with the weighted objective is shown in 3.

We find that the models are able to obtain the following average rewards: As is shown in 5.4, the language models under both training objectives are able to achieve rewards of approximately 0.73. This is similar to the reward on the training set that DeepFM was able to receive, but surprisingly, GPT-2 is able to generalize much more effectively. We hypothesize that this is due to GPT-2's large-scale pretraining.

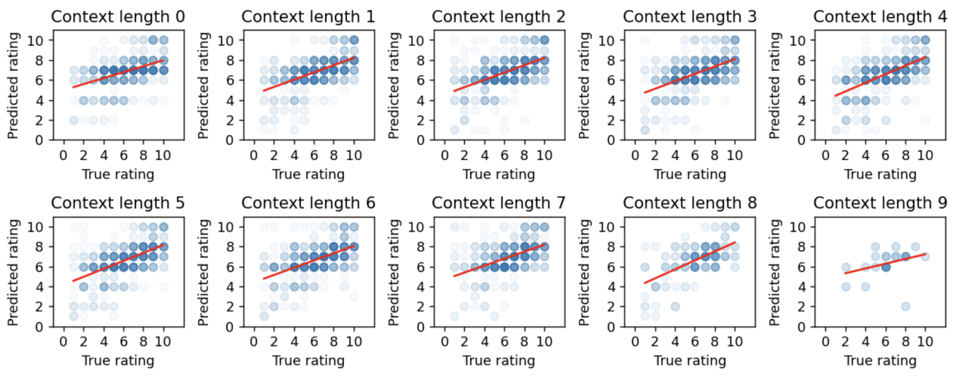

Figure 2: Scatterplot (Unweighted)

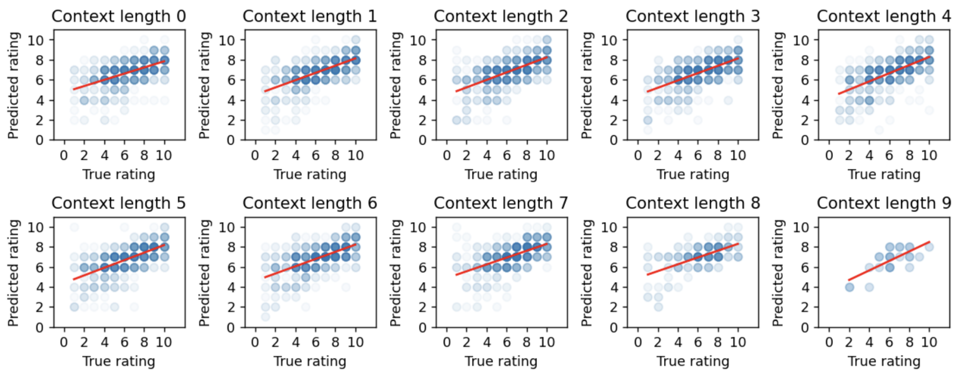

Figure 3: Scatterplot (Weighted)

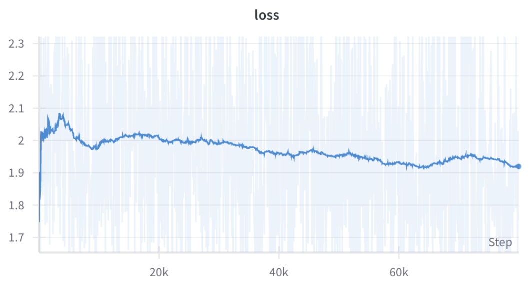

Figure 4: Learning Curve for GPT-2 Model

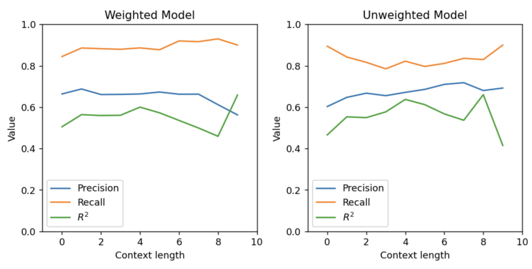

Figure 5: Comparison between Weighted and Unweighted Models

Context Length	Weighted	Unweighted
0	0.7087	0.6533
1	0.7411	0.6907
2	0.7197	0.7099
3	0.7225	0.6946
4	0.7331	0.7248
5	0.7334	0.7240
6	0.7237	0.7381
7	0.7042	0.7353
8	0.6545	0.7068
9	0.6800	0.8000

6 Conclusions

We evaluate methods for offline movie recommendation using a dataset from LetterBoxd's top users. We find that DeepFM works well on training data but falters due to overfitting when evaluated on the test data. We attempt various standard methods of correcting overfitting, but ultimately are unable to correct entirely for the overfitting issue. We also attempt to formulate the recommendation task as a language modeling token prediction of the movie score given a set of 10 previous movies watched. This approach has several benefits including the use of generalized representations of natural language data for each movie as well as the ability to generalize across users. We also include a file allowing users to test the recommendation in a pseudo-online setting by inputting a list of movies and requesting a recommendation. Future work on this recommendation task could extend our approach to a truly online setting or use additional sources of user data like review text and the user social graph.

We evaluated offline movie recommendation methods using a dataset from LetterBoxd's top users. While DeepFM performed well on the training data, it suffered from overfitting issues on the test set, despite attempting various standard regularization techniques. To address this, we formulated the recommendation task as a language modeling problem, predicting movie scores based on a sequence of previously watched movies. This approach leverages generalized representations of natural language data for each movie and enables generalization across users. Additionally, we provide a pseudo-online setting where users can input a list of watched movies and receive recommendations.

Our language modeling approach offers several advantages: (1) It mitigates overfitting by learning generalized representations from large corpora. (2) It can incorporate additional user data, such as review text and social connections, to enhance recommendations. (3) It enables a seamless transition to a truly online setting by incrementally updating the model with new user interactions. Ultimately, however, our approach faltered due to the limitations of fine-tuning language models for classification tasks.

Future work can explore extending our approach to a fully online recommendation system, incorporating multi-modal data (e.g., movie posters, trailers), and leveraging user-user and item-item similarities in the learned representations. Additionally, future work can address the cold-start and generalization problems using a hybrid recommendation approach in which recommendations are initially based on a learned general representation and, as more interactions are added, recommendations shift towards collaborative filtering. Furthermore, evaluating the recommendation quality through real-world user studies would provide valuable insights into the practical utility of our methods.

References

T. Aditya, K. Rajaraman, and M. Monica Subashini. Comparative Analysis of Clustering Techniques for Movie Recommendation. *MATEC Web of Conferences*, 225:02004, 2018. ISSN 2261-236X. doi: 10.1051/matecconf/201822502004. URL `https://www.matec-conferences.org/10.1051/matecconf/201822502004`.

M. M. Afsar, T. Crump, and B. Far. Reinforcement Learning based Recommender Systems: A Survey. *ACM Computing Surveys*, 55(7):1–38, July 2023. ISSN 0360-0300, 1557-7341. doi: 10.1145/3543846. URL `https://dl.acm.org/doi/10.1145/3543846`.

F. Liu, R. Tang, X. Li, W. Zhang, Y. Ye, H. Chen, H. Guo, and Y. Zhang. Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling, Oct. 2019. URL `http://arxiv.org/abs/1810.12027`. arXiv:1810.12027 [cs].

S. Mehra and T. Choudhury. Sentiment Analysis of User Entered Text. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 457–461, Belgaum, India, Dec. 2018. IEEE. ISBN 978-1-5386-7709-4. doi: 10.1109/CTEMS.2018.8769136. URL `https://ieeexplore.ieee.org/document/8769136/`.

M. Z. Naeem, F. Rustam, A. Mehmood, M. zzud din, I. Ashraf, and G. S. Choi. Classification of movie reviews using term frequency-inverse document frequency and optimized machine learning algorithms. *PeerJ Computer Science*, 8:e914, Mar. 2022. ISSN 2376-5992. doi: 10.7717/peerj-cs.914. URL `https://peerj.com/articles/cs-914`. Publisher: PeerJ Inc.

S. M. Qaisar. Sentiment Analysis of IMDb Movie Reviews Using Long Short-Term Memory. In *2020 2nd International Conference on Computer and Information Sciences (ICCIS)*, pages 1–4, Sakaka, Saudi Arabia, Oct. 2020. IEEE. ISBN 978-1-72815-467-1. doi: 10.1109/ICCIS49240.2020.9257657. URL `https://ieeexplore.ieee.org/document/9257657/`.

Appendix

Movie dataset fields:

`_id`, `genres`, `image_url`, `imdb_id`, `imdb_link`, `movie_id`, `movie_title`, `original_language`, `overview`, `popularity`, `production_countries`, `release_date`, `runtime`, `spoken_languages`, `tmdb_id`, `tmdb_link`, `vote_average`, `vote_count`, `year_released`.