Movinder: A Movie Recommendation System for Groups

Jelena Banjac
jelena.banjac@epfl.ch

Can Yilmaz Altinigne
can.altinigne@epfl.ch

Sofia Kypraiou
sofia.kypraiou@epfl.ch

Panagiotis Sioulas
panagiotis.sioulas@epfl.ch

ABSTRACT

Using the wealth of data available on user preferences, researchers and online streaming companies have extensively researched and deployed movie recommender systems. Most recommendation algorithms process the preferences of each user, and potentially those of other similar users, to predict other movies they might like. However, watching movies is often a social activity shared by multiple actors, such as a group of friends. The social dimension of watching movies contradicts common approaches that strive to satisfy one user at a time. In this project, we analyse different implementations of a movie recommender system that aims to maximize the collective satisfaction of a group of users. Using the combination of Movielens and IMDb datasets, we simulate the group of friends to train our recommender models. The recommender models implemented and discussed include the following types of networks: Neural Collaborative Filtering (NCF), Neural Graph Collaborative Filtering (NGCF), Siamese Neural Network (SNN) and LightFM models. The obtained results are compared. The fastest models are deployed to the website, whereas other models with higher accuracy can be found in the notebooks due to higher computation time. The link to the repository: *https://github.com/Movinder*.

KEYWORDS

recommendation system, graph, neural networks, nearest neighbors

.

1 INTRODUCTION

The concept of recommendations is intertwined with the history of art, tracing back to antiquity in the form of literary criticism. Yet, the recent proliferation of the World Wide Web has brought unprecedented challenges and opportunities to effective recommendations. On the one hand, the abundance of multimedia has increased the need for accurate recommendations to allow users to be time-efficient. On the other hand, users produce a wealth of data on their preferences which can be used to personalize recommendations. As a result, recommender systems have been the subject of extensive body of academic and industrial research.

Movie recommendations are a particularly popular topic in the area of recommender systems. Web-based movie recommender systems are an established type of service, with ventures such as MovieLens dating as early as 1997. The rise of online streaming companies has further fueled the attention to the problem. The stunning one million dollar reward in the Netflix Prize competition in 2009 is a testament to the significance of movie recommendations.

In the background, most algorithms of movie recommender systems consider user preferences in conjunction with movie similarity. The preference data is sparse because users only rate a small subset of the available movies. For this reason, many algorithms use collaborative filtering to include the preferences of other similar users to the recommendation. However, the recommendation still concerns only a single user. By contrast, watching movies is in many cases a social event in which people participate as groups (i.e. friends, couples, associates). Members of the group obviously have different preferences. In this context, the single-user personalized recommendation is unsatisfactory.

In this project, we propose a multi-user recommender system. The per-user preference predictions are then aggregated to produce a prediction for the collective satisfaction of the group.

To validate our approach, we use publicly available datasets on movie ratings and movie characteristics. Specifically, we use the MovieLens dataset[1], which contains 100000 ratings from 943 users on 1682 movies. We merge it with the IMDb dataset[2] which contains information on the cast, the production movies and the related-keywords for each movie. We use the combined features of the two datasets to generate the movie graph that our algorithm requires.

EPFL, January 2020, Lausanne, Switzerland

[1] https://grouplens.org/datasets/movielens/
[2] https://datasets.imdbws.com/

The layout of the report is as follows. In Section 2 we describe our data acquisition process, whereas in Section 3, we explore the collected data. In Section 4, we examine our recommendation algorithm and propose different aggregation methods. Finally, in Section 5, we evaluate our system and we summarize our insights in Section 6.

2 DATA ACQUISITION

The core dataset that we use is the MovieLens dataset. The dataset contains 100000 ratings from 943 users on 1682 movies. Each user in the dataset has rated at least 20 movies. User entries also include demographic information, specifically the age, the gender, the occupation and the zip code. The dataset provider has also annotated each movie with a list of genres. The 19 genres are represented using a bit-encoded vector. Each of the movies also have a composite text that concatenates the name with the year of release. We unpack the information.

The MovieLens dataset contains limited information about the movies, making it difficult to construct an accurate similarity graph. For this reason, we enrich movie entries using information extracted from IMDb. IMDb stores additional information including keywords, cast members and production companies. More precisely, it contains 134170 keywords, 4167491 cast members and 234997 companies. We first convert IMDb to a relational database using imdbpy. Then, we perform data integration by performing a join on the movie name and year of release information. Some MovieLens movies have a distorted name format. For example, articles are places after the rest of the name. Also, international movies have the original title inside parentheses. We use rule-based data cleaning to merge the data and associate MovieLens movie identifiers with IMDb movie identifiers.

3 DATA ANALYSIS

We use the vector representation of the movie entries to build a similarity graph. We compute similarity as the cosine similarity metric:

$$sim(v_1, v_2) = \frac{v_1 \cdot v_2}{||v_1|| \cdot ||v_2||}$$

for feature vectors v_1, v_2. For performance, we normalize the feature vectors and compute similarity as the dot product.

We consider the similarity matrix which is the foundation of the similarity graph. In Figure 1, we plot the histogram of the values in the similarity matrix, excluding zeroes. We observe that the vast majority of similarity values is less than 0.2. Even so, the matrix is not sparse as 61.29% of the values is non-zero. A minority of the values is also high which correspond to very similar movies.

To produce the movie graph, we sparsify the similarity matrix by setting a threshold of 0.1. All the values below that are set to zero. We plot the histogram of the values for the sparsified similarity matrix in Figure 2, excluding zeroes. At first, we considered setting the threshold to 0.2 because it seems to be the rightmost point of the concentration of small similarity values. However, the resulting graph is very sparse and contains 815 nodes without neighbors. By contrast, a value of 0.1 yields only 22 such nodes. The fraction of non-zero values is 6.98 which means that the graph is relatively sparse. Figure 3 shows the degrees of the similarity graph that we build. The majority of nodes has a small degree, but we notice that there exists a significant number of hubs with more than 400 neighbors.

We also perform spectral analysis on the similarity graph using the combinatorian Laplacian matrix. Figure 4 illustrates the eigenvalues of the combinatorial Laplacian. The graph shows significant eigengaps near 1300. Additionally, we perform spectral clustering with 250 clusters. We use cosine distance because of the high number of dimensions. However, since the k-means algorithm provided by sklearn only supports euclidean distance, we normalize the embedded values. The euclidean distance of normalized vectors is linear to cosine distance. Some clusters are reasonable, for example animations (Snow White, Aristocats, Cinderella, Pinocchio and others) and war movies (Apocalypse Now, Full Metal Jacket, Courage under Fire), and are clearly interpretable. Unfortunately, other clusters either consist of a single movie or contain unrelated movies. The latter case usually occurs for very large clusters.

4 RECOMMENDATIONS

In this section, we present different movie recommendation methods that we use.

4.1 Neural Collaborative Filtering

One of the models that we use for collaborative filtering is *Neural Collaborative Filtering* implemented in [2]. The general network combines Matrix Factorization and Multilayer Perceptron to build a deep network structure named as Neural Matrix Factorization. In most recommendation systems, the interaction between users and movies are modeled by applying matrix factorization and then take the inner product on the latent features of users and items. This framework replaces the inner product with a neural architecture that can learn an arbitrary function from data.

The network takes user and item embedding as inputs and concatenates the outputs of matrix factorization branch and Multilayer Perceptron which consists of several feed-forward layers with *ReLU* activation functions. Output activation

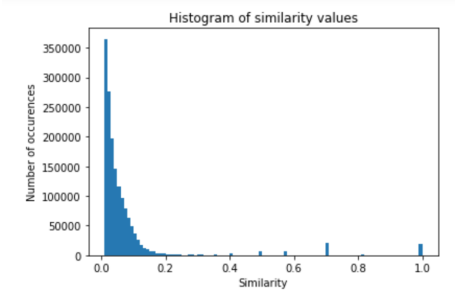

Figure 1: Histogram of values in similarity matrix

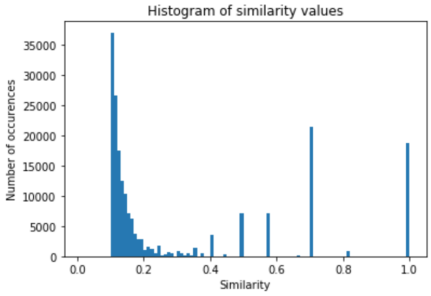

Figure 2: Histogram of values in similarity graph

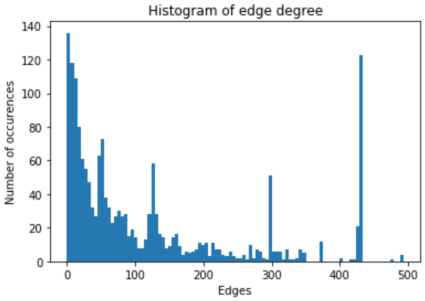

Figure 3: Histogram of node degree in similarity graph

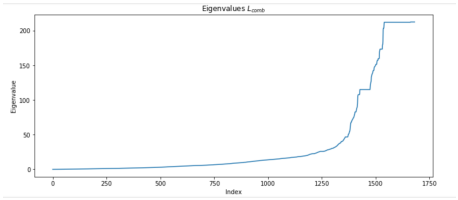

Figure 4: Eigenvalues of similarity matrix

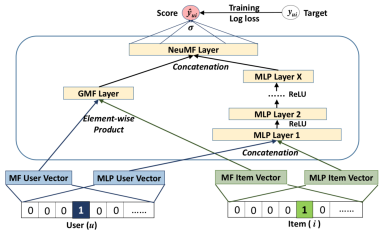

Figure 5: Model overview of Neural Collaborative Filtering network [2].

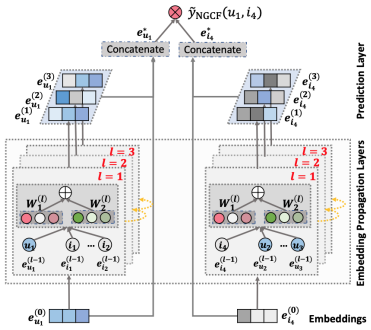

Figure 6: Model overview of Neural Collaborative Filtering network [5].

function is *Linear* activation function, since we consider this problem as a regression problem.

4.2 Neural Graph Collaborative Filtering

User-movie interactions can be presented in a bipartite graph. In *Neural Collaborative Filtering* [2], the interaction between users and movies is not considered. In *Neural Graph Collaborative Filtering*, high-order connectivity is modeled by a graph neural network [5].

For example, the model considers the interaction between the target user and the other users who like the movies that the target user likes. The number of multiple embedding layers can be increased to obtain higher-order connectivity.

4.3 Siamese Neural Network

To create friends similarity graph, we use group of friends and their features as nodes and features similarity as edges. However, the feature similarity is unknown. For that, we decided to use Siamese Neural Network [1]. The inputs to the our network are: friends, positively rated movies (rating

above 3), and negatively rated movies (rating bellow 4). The positively and negatively rated movies use the same weights, thus we have a Siamese architecture that will give us the single embedding for the movies. Last layer of the network is the triplet loss based on the Bayesian Personalized Ranking (BPR) [4]:

$$L_{BPR}(a, p, n) = \sum 1 - \sigma(f(a, p) - f(a, n))$$

where a is an anchor observation, p is the positive sample which should be closer to a than the negative sample n, and σ is the sigmoid function. The function f is the transformation we want to learn, and ϵ is a tuning parameter ($\epsilon > 0$). The learned metric should separate the negative sample n from the positive sample p at least by a positive margin ϵ.

4.4 LightFM

LightFM[3] is Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback, including implementation of Bayesian Personalized Ranking (BPR) and Weighted Approximate-Rank Pairwise (WARP) ranking losses. In its essence, it presents a hybrid matrix factorisation model. We decided to use it since it outperforms collaborative and content-based models as well as sparse interaction data scenarios, as tested in the paper [3] that provides much more implementation details.

5 EXPERIMENTS

We will present several experiments with different models in this section.

5.1 Collaborative Filtering with Deep Learning

We compare *Neural Collaborative Filtering* [2] and *Neural Graph Collaborative Filtering* [5] using user and item embeddings with the size of 64.

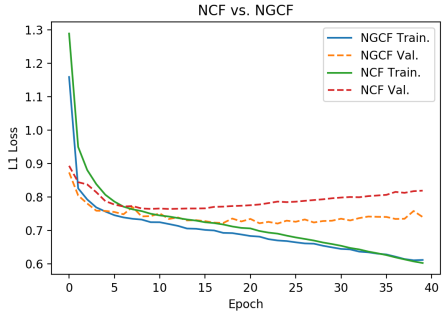

Figure 7: Test set performance of NCF [2] and NGCF [5].

[3]http://lyst.github.io/lightfm/docs/home.html

We did training/validation/test split using 0.6:0.2:0.2 ratio. We perform grid search to choose the best hyperparameters. We use *Adam* optimizer with MAE loss function for both models. Both models have *Linear* output activation function. *NGCF* [5] model achieves **0.72** MAE in test set, whereas *NCF* [2] achieves **0.764** MAE.

5.2 Siamese Neural Network

We start by binarizing the user ratings that are given in range between 1 and 5. The ratings above 3 are the interacting friends-movie pairs, and ratings bellow 4 are non-interacting friends-movie pairs (i.e. creating an implicit feedback data). To construct the triplets a, p, n, we sample from the interacting friends-movies pairs and combine them with randomly sampled non-interaction items for the friends.

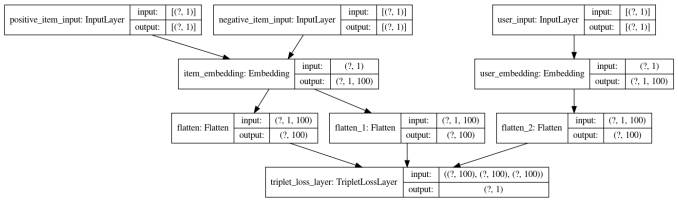

Figure 8: The architecture of implemented Siamese neural network

The model architecture can be seen in Figure **??**. We can see that the dense embeddings of friends and movies are the input to the triplet loss function. By minimizing it, we achieve our goal: learning friends representations and movie representations in embedded space.

The performance evaluation of the implemented Siamese neural network is shown in Figure 9

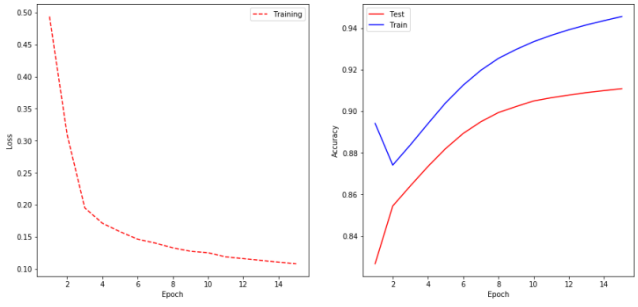

Figure 9: Left plot shows the loss of training the Siamese model. Right plot shows the accuracy on training and test sets.

In the end we have a **94.5%** of accuracy on the train set and **91.1%** of accuracy on the test set.

5.3 LightFM

We train our model using the movie and friends dataset features. We crate the sparse matrices in order to feed our model for training. First, we build ID mappings between the friends and movie ids from our datasets to indices that will be used internally by the model. Then, we crate the interactions matrix that contains interactions between friends and movies (1 if friend rated the movie, 0 otherwise). In addition, we collect friend's features (i.e. their age and gender) and movie's features (i.e. genres, titles, release, genres, and vectorized keywords, cast and company information) and supply it to the model. Building a model included the tuning of following hyper-parameters: number of epochs used, learning rate for the adagrad learning schedule, maximum number of negative samples used during the WARP fitting. We used WARP ranking loss since it was working significantly faster than BPR ranking loss. We also compared the performance of model depending on whether we use additional movie's and friend's features or not. Final used parameters were:

- Number of epochs: 150
- Learning rate: 0.015
- Maximum number of negative samples: 11
- Ranking loss: WARP.

The resulting precision of the model is **83%** on the training set and **68%** on the test set. The resulting accuracy is **87%** on the training set and **88%** on the test set.

5.3.1 Friends graph representation. From the model, we extract the user embeddings which are represented as a 10-dimensional vector for each group of friends. To plot friends similarity graph, need an adjacency matrix and friends embedding coordinates in 2D space. The adjacency matrix was calculated using the cosine similarity. The friends embedding dimension was reduced from 10D to 2D using different techniques of dimensionality reduction, including TSNE which can be seen in Figure 10.

5.3.2 Movie graph representation. Similar to the friends graph represenation, we extract the movie embeddings which are represented as a 10-dimensional vector for each movie. To plot a movie similarity graph, need an adjacency matrix and movie embedding coordinates in 2D space. The adjacency matrix was calculated using the cosine similarity. The movie embedding dimension was reduced from 10D to 2D using different techniques of dimensionality reduction, the TSNE can be seen in Figure 11.

6 CONCLUSIONS

We observe that Graph Neural Network model reaches a better test set error than Neural Network structure on movie recommendation task, since it captures the collaborative signal between user and movies on a bipartite graph structure.

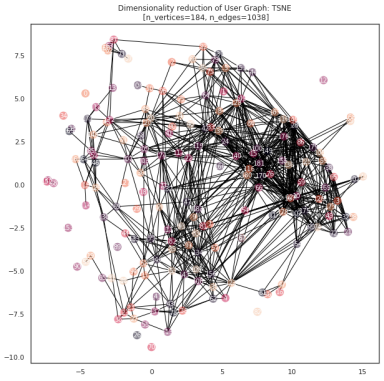

Figure 10: Friends graph embedded in 2D space using the TSNE dimensionality reduction method. In total we have 184 groups of friends each representing the node. The edges are thresholded similarity values (threshold=0.88)

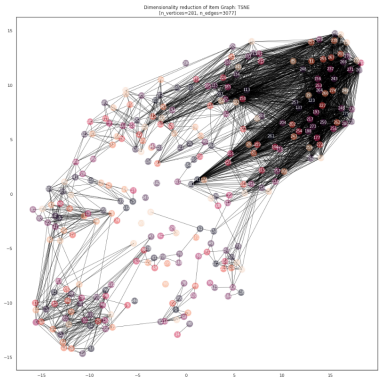

Figure 11: Movie graph embedded in 2D space using the TSNE dimensionality reduction method. In total we have 1251 different movies. To plot it, we selected random 281 movies where each represents the node. The edges are thresholded similarity values (threshold=0.9)

Moreover, we decide to use our two fastest implementations of movie recommendation which are General Matrix factorization and LightFM implementations on a website to recommend movie to group of users.

The product of this project was implemented as a web application on the following link: *https://movinder.herokuapp.com/*. More details on the implementation as well as other visualizations can be found in our github repository: *https://github.com/Movinder*.

REFERENCES

[1] Miguel Campo, JJ Espinoza, Julie Rieger, and Abhinav Taliyan. 2018. Collaborative Metric Learning Recommendation System: Application

to Theatrical Movie Releases. arXiv:cs.IR/1803.00202

[2] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *Proceedings of the 26th International Conference on World Wide Web - WWW '17* (2017). https://doi.org/10.1145/3038912.3052569

[3] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. In *Proceedings of the 2nd Workshop on New Trends on Content-Based Recommender Systems co-located with 9th ACM Conference on Recommender Systems (RecSys 2015), Vienna, Austria, September 16-20, 2015. (CEUR Workshop Proceedings)*, Toine Bogers and Marijn Koolen (Eds.), Vol. 1448. CEUR-WS.org, 14–21. http://ceur-ws.org/Vol-1448/paper4.pdf

[4] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian Personalized Ranking from Implicit Feedback. *CoRR* abs/1205.2618 (2012). arXiv:1205.2618 http://arxiv.org/abs/1205.2618

[5] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR'19* (2019). https://doi.org/10.1145/3331184.3331267