CS 584 Project Report
Movie Recommender Systems

Sohaib Jawad
sjawad@hawk.iit.edu

Victor Samsonov
vsamsonov@hawk.iit.edu

December 3, 2022

github

Abstract

This project report explores the usage of Recommendation System in a movie-oriented environment using the Movie Lens dataset. We cover types of recommendation systems, along with their applications and conclude with our results and thoughts.

Introduction

Summary

Generally speaking, recommender systems are algorithms that make appropriate recommendations to users for things like movies to watch, books to read, products to buy, or other things depending on the industry. During the previous few decades, recommender systems have become more and more prevalent in our lives as a result of the growth of websites like YouTube, Amazon, Netflix, and many more. Currently, recommender systems cannot be avoided in our daily online activities, whether they be for e-commerce or online advertising.

In many businesses, recommendation systems are essential since they may be a large source of revenue when they work effectively or a method to differentiate yourself dramatically from the competition. We might cite Netflix's 2006 "Netflix Reward" competition as evidence of the significance of recommender systems. The challenge's objective was to develop a recommender system that outperformed Netflix's own algorithm for a chance to win a 1 million dollars cash prize.

Previous Work

Since the beginning of the Internet, recommendation systems have existed. These systems and other related technologies have received very little attention from academia and industry, among other sectors. Researchers specializing in information retrieval and cognition science laid the foundations of recommender systems. It was first applied in the Duke University-developed Usenet communication system in the 1970s. They let a group of users to share text information with one another that was categorized into newsgroups and subgroups to make searching easier, as opposed to being developed to target user preferences. It is quite apparent, that since the 1970s, the advances made in recommender systems have been significant to say the least, due to better theoretical understanding of the subject at hand and also due to the greater availability of computation power, which brought the rise of Big Data.

Methods and Results

We used the following methods to solve our problem"

- Collaborative filtering
- KNN
- DLRM (continuous and discrete output)

Problem Description

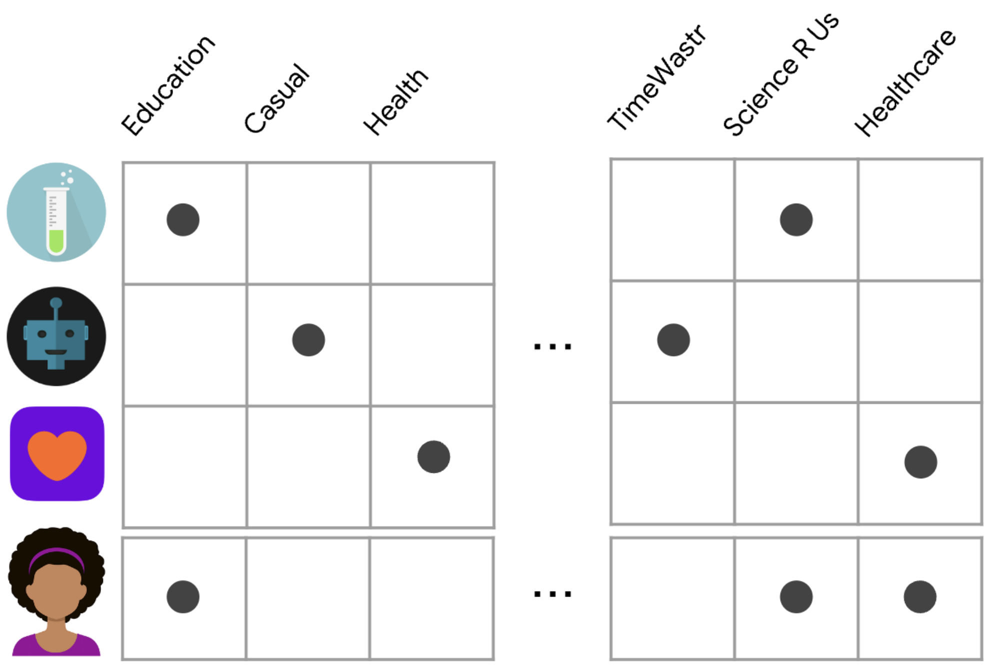
We seek to implement a movie-oriented Recommendtion System such that it can make adequate suggestions and predictions based on the interests of a user, and we will primarily be focusing on a collaborative filtering approach. Before proceding we will provide a background of the types of recommendation systems

Content-based filtering

Based on the user's past behavior or explicit feedback, such as when creating a new profile, content-based filtering uses item features to suggest additional goods that are similar to what the user likes. The model ought to suggest goods that are appropriate for this user. You must choose a similarity metric first in order to achieve this (for example, dot product or cosine similarity). The system must then be configured to grade every candidate item using this similarity metric. The model did not use any data about other users, therefore the recommendations are personal to this user.

In the case of movie recommendation systems, we can think of providing recommendations based on the movie genre that the user likes or perhaps provide recommendations based on his favorite actor and much more.

Looking at the diagram below, consider an example feature space where each row denotes a mobile application and columns denote the type of category it belongs to. We can clearly see that App 1 and App 4 seem to share more than one category that are the same. So if User likes App 1, it is highly probable that he will like App 4 as well and those the recommender system can recommend it to that user.



Finding the similarity between the users' choice can be done using techniques like Euclidean Distance, Dot Product, and Cosine similarity. Content-based filtering works fine, but there are more advanced techniques that are being used in the industry.

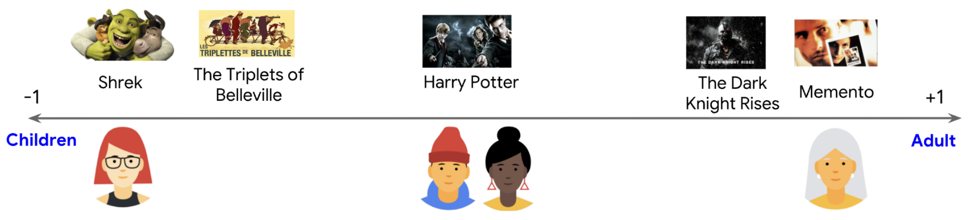
Collaborative Filtering

Collaborative filtering makes recommendations by simultaneously comparing similarities between people and items, addressing some of the drawbacks of content-based filtering. Serendipitous suggestions are now possible; in other words, collaborative filtering algorithms can suggest an item to user A based on the preferences of a user B who shares those interests. Furthermore, instead of relying on manually designing features, the embeddings can be learned automatically.

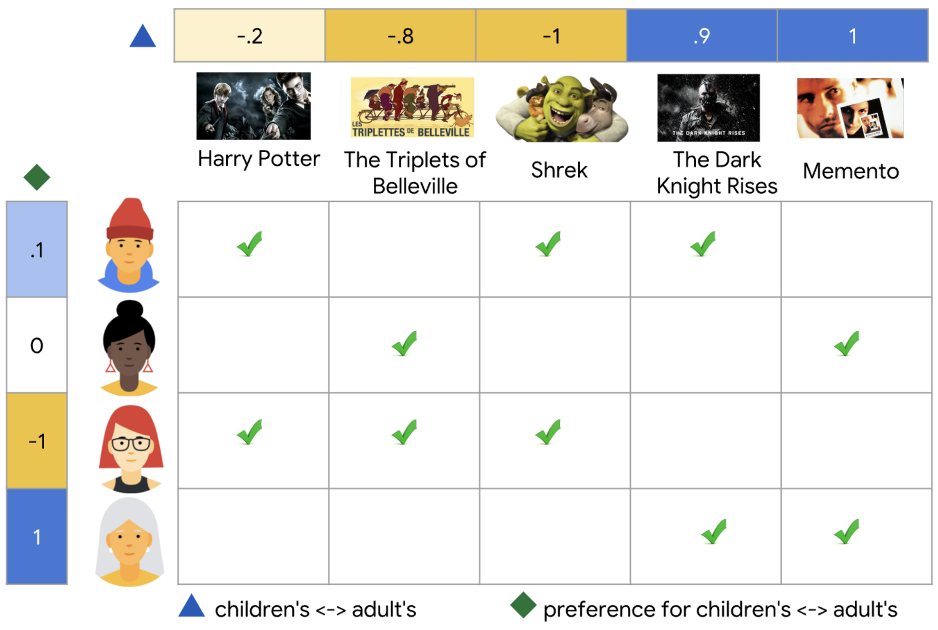
Collaborative filtering however tries to predict whether a user might like a certain product by looking at the preferences of similar users. Here we can think of the standard matrix factorization (MF) approach for movie recommendations where the ratings matrix gets factored into one embedding matrix for the users and one for the movies.

A disadvantage of classic MF is that we cannot use any side features e.g. movie genre, release date, etc., the MF itself has to learn them from the existing interactions. Also, MF suffers from the so-called "cold start problem", meaning a new movie that hasn't been rated by anyone yet, cannot be recommended. Content-based filtering solves these two issues, however, it lacks the predictive power of looking at similar users' preferences.

To better understand the Collaborative Filtering method, consider the following example. Imagine we define a 1-D space. Say people to the left are the ones who prefer watching movies for children and towards the right means people who like watching adult/serious movies. Similarly for movies, towards the left are children movies and towards the right are serious movies. The product of user embeddings and movie embeddings should be higher (close to 1) for movies that we expect the user to like. Notice how we can group the users based on this information even looking at the diagram below.



The same diagram can be represented in a tabular format as follows:



◆	▲	-.2	-.8	-1	.9	1
		Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
.1		✓		✓	✓	
0			✓			✓
-1		✓	✓	✓		
1					✓	✓

▲ children's <-> adult's ◆ preference for children's <-> adult's

We can also add more features to this problem too. Although we handpicked the features for this example, in practice collaborative filtering figures these things out themselves while training.

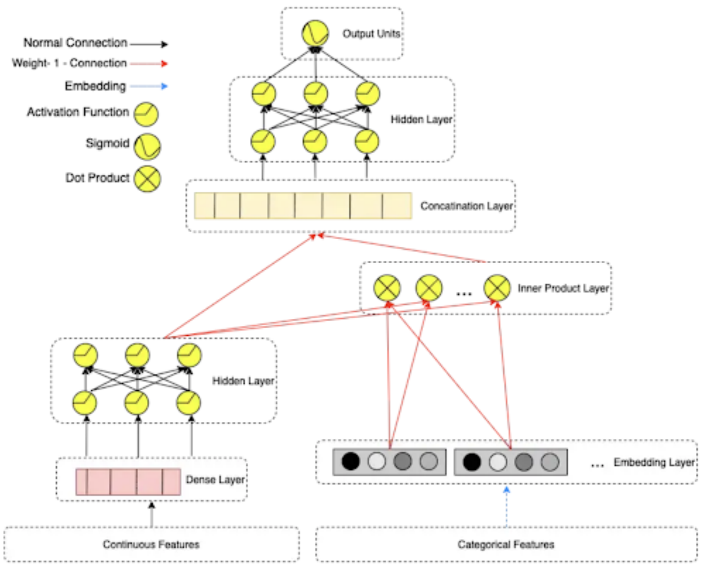
KNN

One of the main algorithms we've used to perform recommendations was the K-Nearest Neighbors ML algorithm. KNN is a supervised learning algorithm that is used to solve regression and classification problems easily.

It finds distances by using a predefined metric and k is a hyperparameter (number of examples closest to an instance), then either performs voting or averaging based on the task.

DLRM

We used this model to train a SoTA deep-learning based recommender system. DLRM was proposed by facebook in 2019 and is considered as the best recommender system. The structure according to its research paper is as follows:



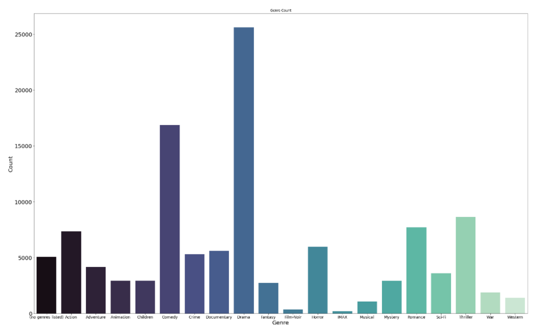
The DLRM architecture is composed of the following components:

- Embeddings: these layers are very powerful and allow to map categories to into an abstract space, and are able to capture context
- Matrix Factorization: since we want to represent the i-th product by a vector $w_i \in R^d fori = 1, ..., n$ and j-th $forj = 1, ..., m$ to find all the ratings where n and m are simply the number of products and users.
- Factorization Machine: FMs incorporate second-order interactions into a linear model with categorical data by defining a new model. FMs are distinct from SVMs that posses a polynomial kernel due to the fact that they are capable of producing a factorization matrix into latent factors.
- Multilayer perceptrons: these are the key components of any Deep Learning architecture. Multilayer perceptrons introduce non-linerity which allows us to uncover interesting properties in the data

EDA

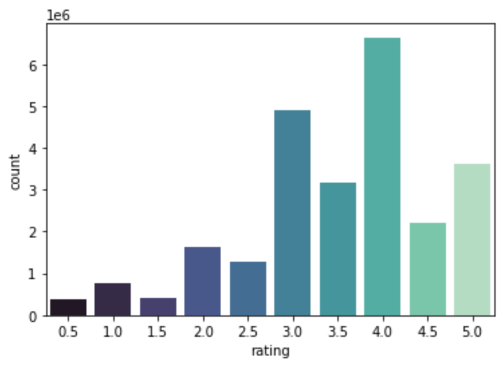
We will work on using the MovieLens 1M dataset which contains 1 million movie ratings and tagging activities since the year 1995 and is structured in the following main files:

- tag.csv
- movie.csv
- rating.csv
- link.csv



Genre counts (Drama and Comedy are the most frequent genres)

```
The mean rating is: 3.533854451353085
The mode rating is: 4.0
```



Ratings counts

Results

We implemented 2 DLRM models:

- continuous DLRM (output is continuous, regression task)
- discrete DLRM (the output is discrete, classification task)

training background:

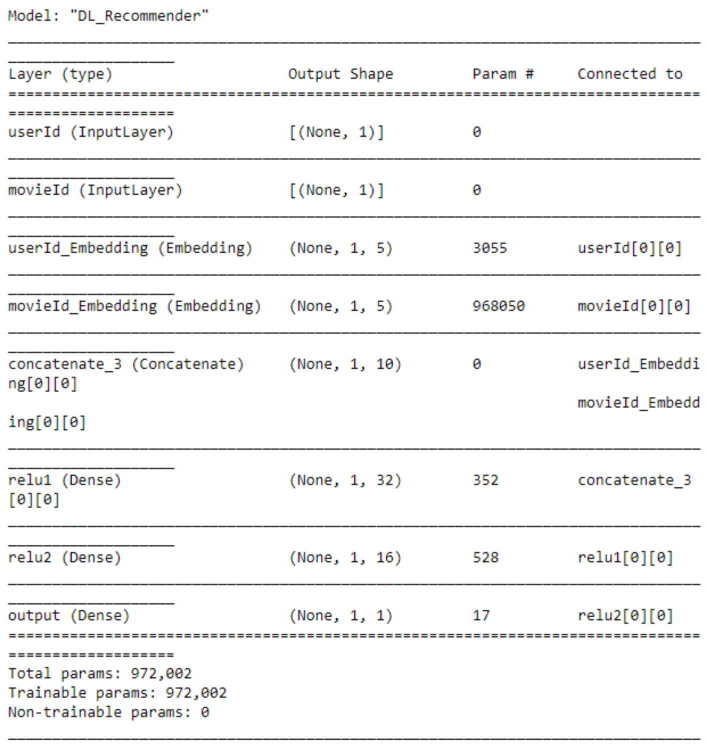
In order to ensure our model performs well, it is important to establish good data splits when it comes to our training, validation and testing.

Our team first defined the size of the test set to be 20% (samples were randomly selected). When it comes to defining a validation set, we simply reserve 10% of the training set for the purpose of validation. Based on if we are using a continuous DLRM or discrete DLRM we preprocessed the data differently, which were by either centering or label encoding them.

Training

We first train using a subset of our training set and validation, which allows us to determine when our models start to overfit:

Model Architecture:



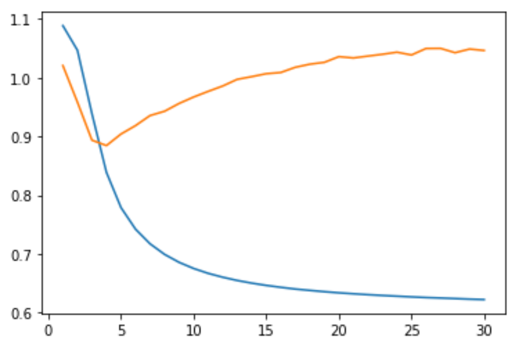
```
Model: "DL_Recommender"
__________________________________________________________________________________________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
userId (InputLayer)            [(None, 1)]          0
__________________________________________________________________________________________________
movieId (InputLayer)           [(None, 1)]          0
__________________________________________________________________________________________________
userId_Embedding (Embedding)   (None, 1, 5)         3055        userId[0][0]
__________________________________________________________________________________________________
movieId_Embedding (Embedding)  (None, 1, 5)         968050      movieId[0][0]
__________________________________________________________________________________________________
concatenate_3 (Concatenate)    (None, 1, 10)        0           userId_Embeddi
ng[0][0]
                                                                movieId_Embedd
ing[0][0]
__________________________________________________________________________________________________
relu1 (Dense)                  (None, 1, 32)        352         concatenate_3
[0][0]
__________________________________________________________________________________________________
relu2 (Dense)                  (None, 1, 16)        528         relu1[0][0]
__________________________________________________________________________________________________
output (Dense)                 (None, 1, 1)         17          relu2[0][0]
==================================================================================================
Total params: 972,002
Trainable params: 972,002
Non-trainable params: 0
__________________________________________________________________________________________________
```

As observed, out model has the following components:

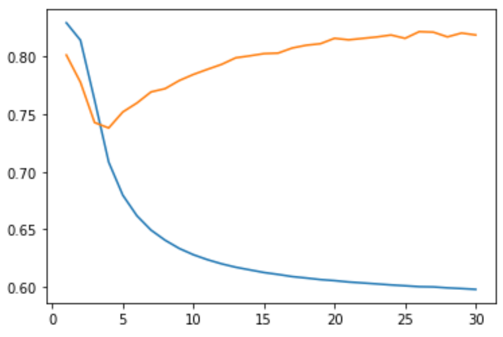
- 2 Inputs for the user and movie information
- 2 embedding layers corresponding to the user and movie.
- 2 hidden fully-connected layers with 32 and 16 units (relu activation).
- an output layer with no activation or softmax based on the task.

Other information about the model:

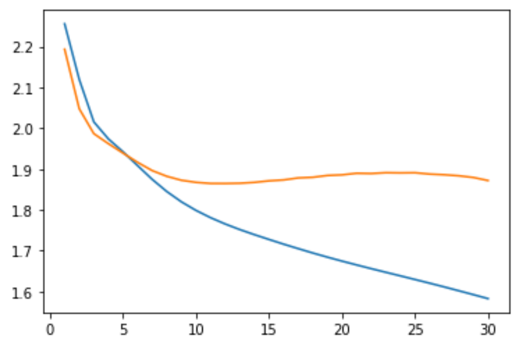
- Uses Adam optimizer
- Has a learning rate of 0.0001
- uses an embedding size of 5



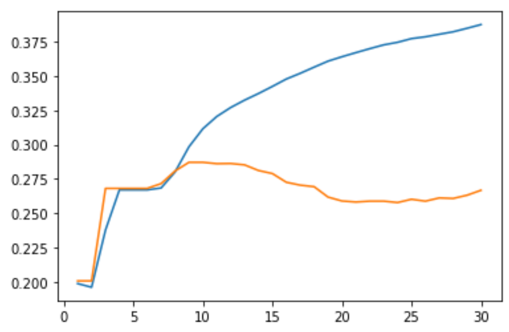
Loss continuous DLRM (training loss: 0.62 and val_loss: 1.0463)



MAE continuous DLRM (training MAE: 0.5978 and val_MAE: 0.8189)



Loss discrete DLRM (training loss: 1.5826 and val_loss: 1.8719)



accuracy discrete DLRM (training: 0.3873 and val: 0.2668)

Evaluation:

Based on the graphs, we can determine that the models start overfitting somewhere between 15 and 20 epochs, therefore we will only train for 15 epochs both of our final models.

For the continuous DLRM, we achieve a MAE of 0.8124.

For the discrete DLRM, we achieve an accuracy of 28.5% and a MAE of 0.811

Conclusions and Future work

Based on our research and work on Recommedation Systems, we can safely say these systems are widely used across the industry and one the most used ML/DL models in production. These systems have evolved greatlt over the years and will continue to do so. Big companies like Netflix, Google and Amazon are investing high capital on these systems to stay ahead of their competitors in this game.

Having tried different approaches for Recommeder systems, we came to a conclusion that Deep Learning based Collaborative filtering models are the SoTA models currently in the industry as they produce substantially good results as compared to other non deep-learning based models.

It is safe to say that big companies continue to invest heavily in this domain and will do so in the future too as their business model revolves around these systems.

References

- Deep Learning Recommendation Model - Paper 2019
- Google Developers - Content Based Filtering
- Google Developers - Collaborative Based Filtering
- Modern Recommender Systems