PMLDL Assignment 2 Report

Introduction

This is a report, describing my path of achieving results in this assignment. The task was about creating a recommender system for suggesting films for a given user based on MovieLens 100K dataset. I have tried many approaches and ideas, but firstly I would like to tell about data analysis.

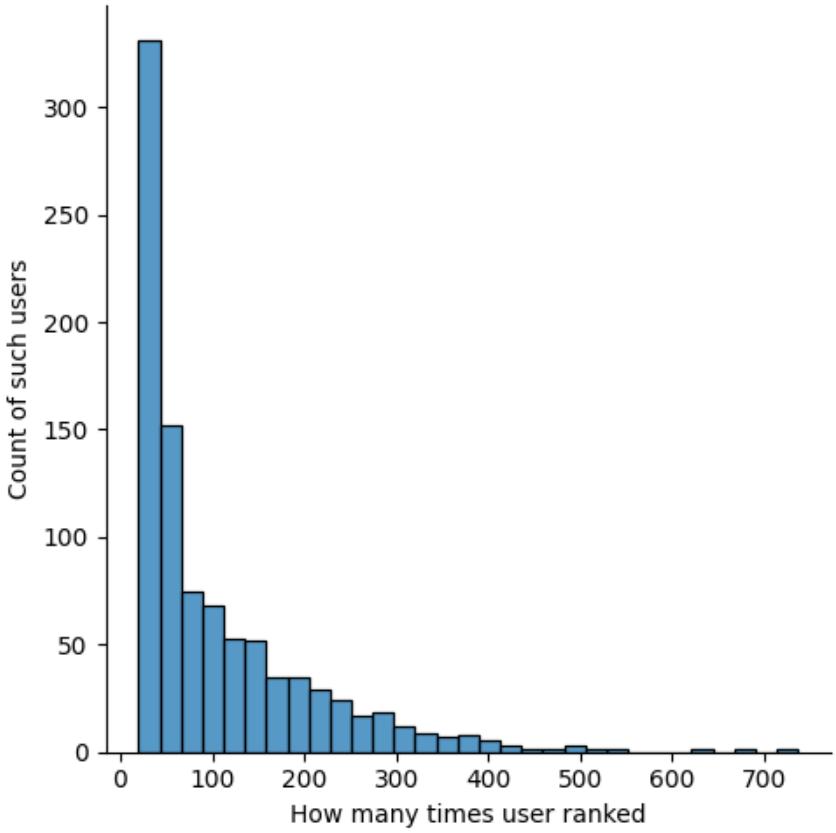
Data analysis

In this assignment I used MovieLens 100K dataset consisting user ratings to movies.

General information about the dataset:
- It consists of 100,000 ratings from 943 users on 1682 movies
- Ratings are ranged from 1 to 5
- Each user has rated at least 20 movies
- There is a separate database about users demographic information (age, gender, occupation and zip-code)
- There is a separate database with films information (title, release_date, IMDB URL, category)

User ratings amount distribution

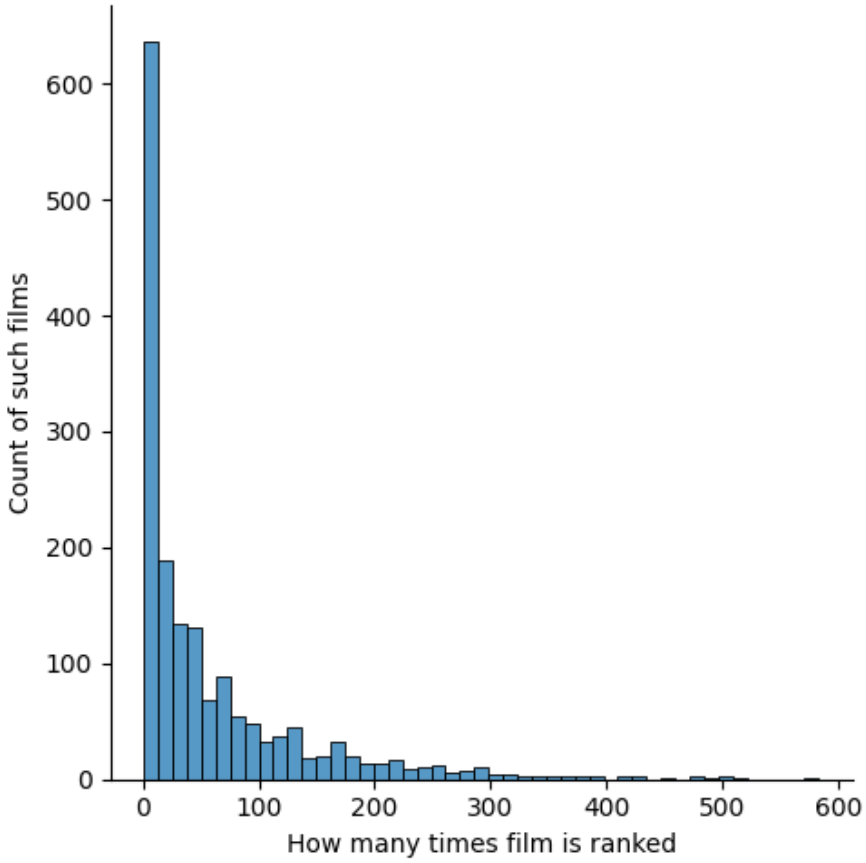
Although it is guaranteed that each user has voted at least 20 times, this is the first thing to check that came to my mind. We cannot make quality suggestions, if users have not got enough votes.



We see, that the most significant part of users has ranked 0-100 times. Surprisingly, many people have voted from 200 to 300 times. Of course, it is good to see. Conclusion — it is not bad. In my opinion, it is enough to build a good model.

Film ratings amount distribution

With users ratings amount distribution above, I believe that these two are the most important parameters to take into account. Talking about film ratings, it is essential for a film to have several ratings in order for average rating to be objective. We cannot trust a film rating if only one user has rated it.



Most of films are rated from 1 to 50 times. I have also checked for amount of films, which have the lowest amounts of ratings:

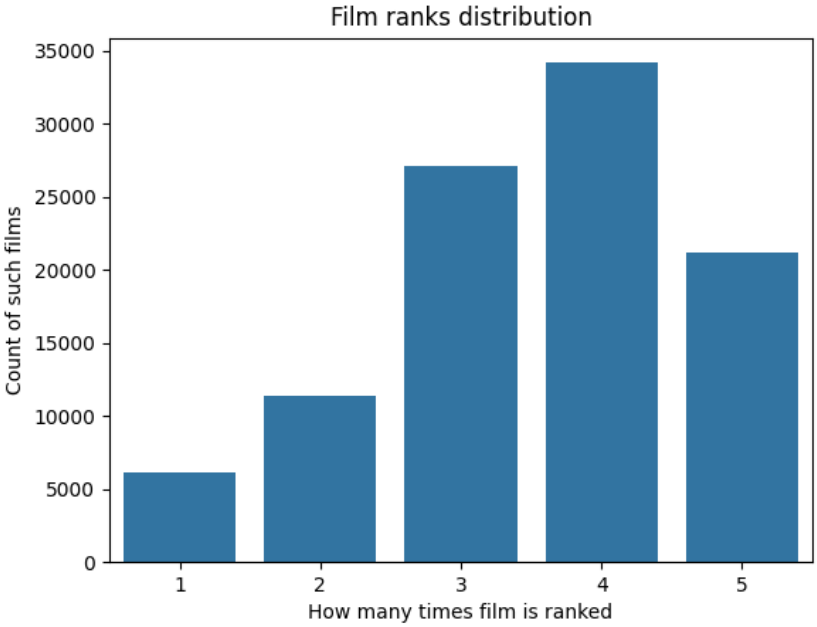
1 rating — 141 films (8% of dataset)
2 ratings — 68 films (4% of dataset)
3 ratings — 60 films (3.5% of dataset)
4 ratings — 64 films (3.8% of dataset)
5 ratings — 51 films (3% of dataset)

Let's assume that a film has low amount of ratings, if it has less than 6 ratings. According to this, there are ~22% of films with low ratings. It is a pretty big amount of films.

Below there are other metrics and visualizations, which are not critical, but very important too.

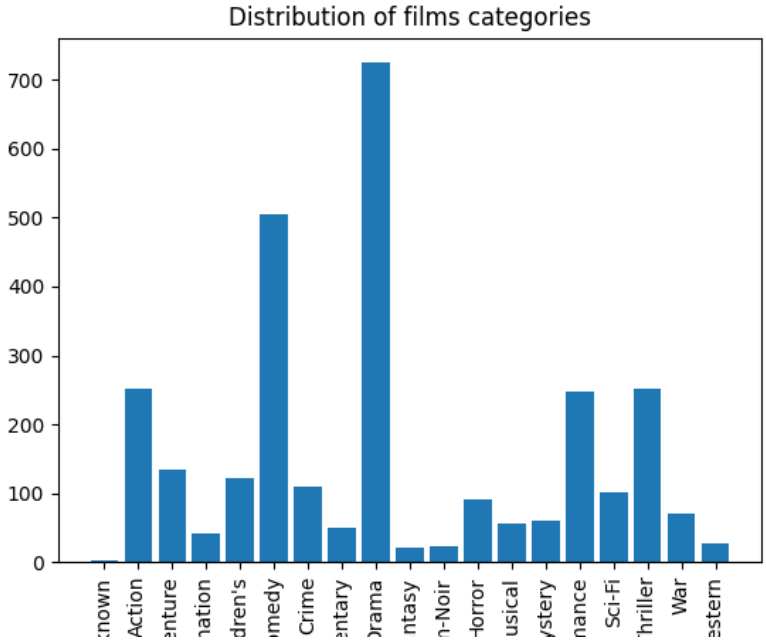
Distribution of film ranks

Looks good, but minor bias to better ratings.

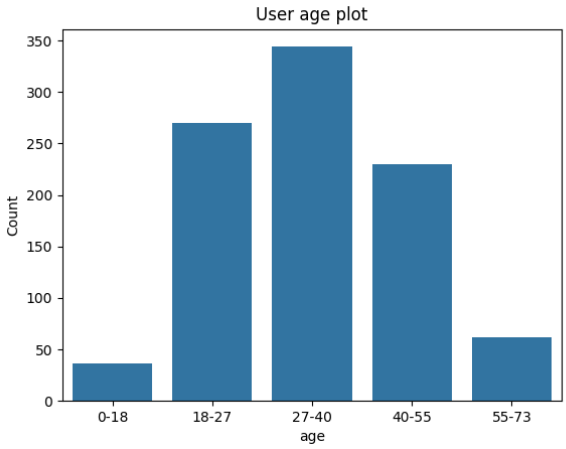


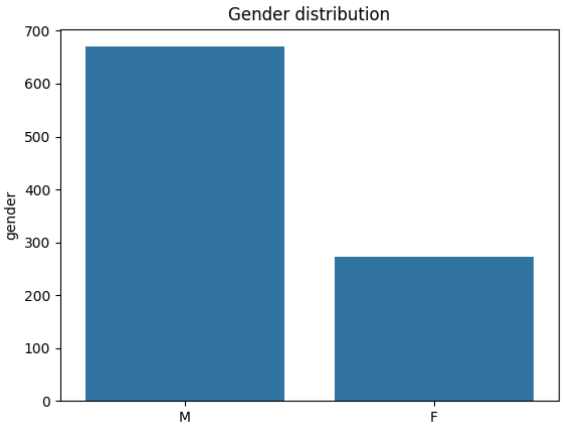
Distribution of film categories

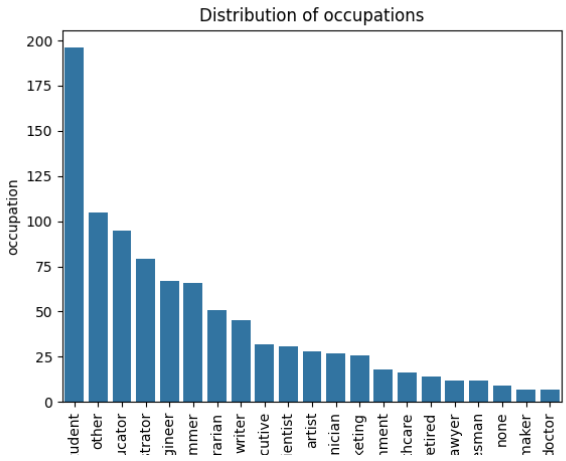
Not bad.



User demographic data distributions







Model Implementation V1

Firstly I decided to try a very straightforward approach: join films and users to ratings and then just train a Linear Regression or a Neural Network predictor.

The resulting data consisted of 104 columns, which includes rating, release date, category (one-hot-encoded), state (converted from zip-code, one-hot-encoded), gender (one-hot-encoded) and age-ranges (one-hot-encoded).

My final model consists of 3 layers with hidden sizes (256, 512) and ReLU activation function.

Further I will not talk about linear model, because its results were even worse.

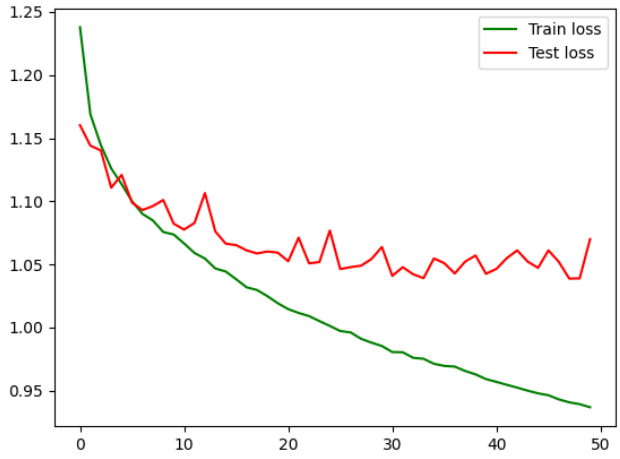
Advantages and disadvantages

Advantages:
1) It is very and very simple
2) Trains and converges really fast
3) No need to have historical data for users and films

Disadvantages:
1) Low quality of predictions

Training process

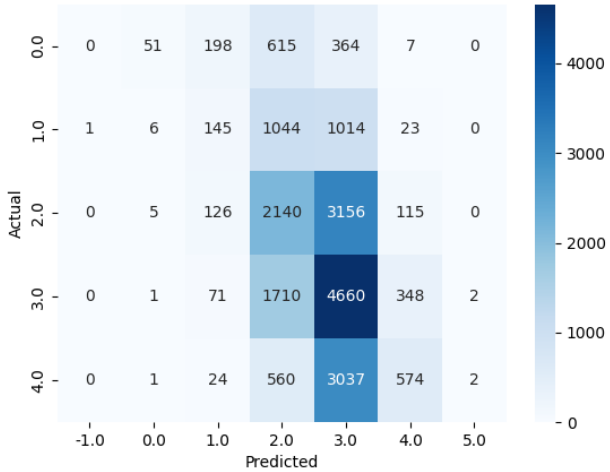
The model converges at around 50th epochs, the whole process takes around 2.5 minutes. Below is the plot of difference between train loss and validation loss.



Evaluation

I have got the following results:
1) Accuracy = 0.3785
2) RMSE loss = 1.0344238

Confusion matrix:



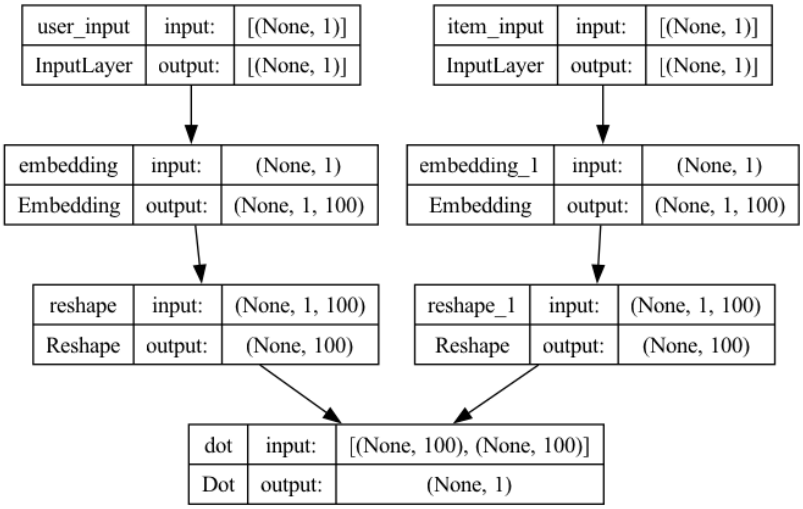
Unfortunately, it is clearly seen, that the model has just trained to predict mostly rating ‘3’.

Results

I believe that the main this approach did not succeed is that model does not take specific user info and movie info into account. So it cannot learn dependencies of specific preferences. In my opinion, this approach is failure.

Model implementation V2

And that is the reason why I decided to try another approach, which is more specific to recommender systems. My second approach is a collaborative filtering movie recommender system using matrix factorization with embeddings. I am going to break down the key components.



1. Input Layers:
   - Two input layers are defined for user and item IDs, respectively.
   - Each input layer takes a single integer as input, representing the user or item ID.
2. Embedding Layers:
   - Two embedding layers are created, one for users and one for items (movies).
   - These layers are used to convert user and item IDs into dense vectors of fixed size (specified by k_factors).
   - Embeddings are trainable parameters, and their values are learned during the model training process.
3. Reshape Layers:
   - Reshape layers are applied to the embedded vectors to ensure compatibility for the subsequent dot product operation.
4. Dot Product Layer:
   - The dot product is computed between the user and item embeddings using the Dot layer.
   - This operation effectively captures the interaction between a user and an item in the latent space defined by the embeddings.

This collaborative filtering approach is based on the idea that user preferences and item characteristics are represented in a lower-dimensional latent space. By learning these representations through embeddings and performing a dot product, the model predicts user ratings for unseen user-item pairs.

Model advantages and disadvantages

Advantages:

- Implicit Feature Learning:
  The model effectively captures latent features or characteristics of users and items implicitly through learned embeddings. This allows the model to uncover complex patterns in user-item interactions.

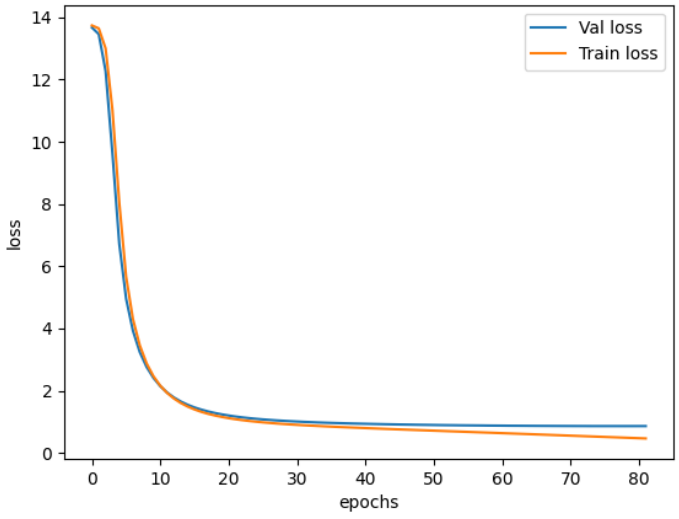
- Scalability:
  Embedding-based collaborative filtering is computationally efficient, especially compared to traditional matrix factorization methods. The use of embeddings significantly reduces the number of parameters compared to a full user-item interaction matrix.
- Interpretability:
  The learned embeddings provide a low-dimensional representation of users and items, which can offer insights into the underlying factors driving user preferences and item characteristics.
- Flexibility:
  The model architecture is flexible and can be easily adapted to handle additional features or incorporate more complex structures if needed.
- Predictive Accuracy:
  Collaborative filtering, when well-tuned, can produce accurate predictions, especially in scenarios where users share similar tastes.

Disadvantages:

- Cold Start Problem:
  The model may struggle with the cold start problem, where it's challenging to provide accurate recommendations for new users or items with limited interaction history.
- Data Sparsity:
  Collaborative filtering relies on historical user-item interactions, and the model's performance may suffer in sparse datasets where users have rated only a small fraction of items.
- Lack of Diversity:
  The model may recommend popular items more frequently, leading to a lack of diversity in recommendations. This is a common issue in collaborative filtering systems.
- Limited Contextual Information:
  The approach does not explicitly consider contextual information such as genre, time, or explicit user preferences. This can limit the system's ability to capture diverse user behaviors.
- Overfitting:
  In the case of small datasets, the model may be prone to overfitting, especially when dealing with a large number of latent factors.
- Difficulty Handling Dynamic Preferences:
  The model might struggle to adapt to changes in user preferences over time, as it relies on historical data to learn user-item interactions.

Training process

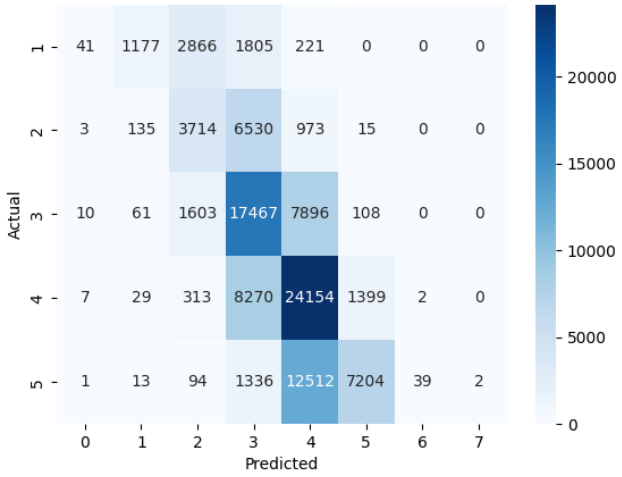
The model converges for about 2 minutes, it takes approximately 82 epochs.
Below is the plot of difference between train loss and validation loss.



Evaluation

I have got the following results:
1) Accuracy = 0.538
2) RMSE loss = 0.9

Confusion matrix:



Lets assume that when we will predict user rankings for all films, then it will have approximately the same distribution of values in confusion matrix. Lets also assume that user loves a films, when he rates it like 4 or 5. In such case we can suggest films with rating prediction of 5.

User will give:
- '5' to 82% of films
- '4' to 16% of films
- '3' to 1% of films
- '2' to 0.1% of films
- '1' to 0.0% of films

This basically means that user will like 98.5% of films that are commended. But what about amount of the recommended films? We can face lack of recommendations problem, if we pick only top prediction films.

Below is the distribution of recommendations amount for users.

(0, 2]	110 = 19%
(2, 6]	106 = 18%
(6, 10]	60 = 10%
(10, 30]	119 = 21%
(30, 100]	118 = 21%
(100, 200]	33 = 6%
(200, 600]	31 = 5%

In my opinion, it is not bad. It is also possible to move the threshold from 4.5 to 4.2, for example, and the results will change. We can play with quality vs quantity.

Conclusion

In my opinion, the second approach is success. I think the overall quality of its predictions is good.

I like the way, how it predicts. I also like that it is possible to play with threshold in order to change quantity of recommendations.

All in all, the assignment was very challenging and interesting for me. Challenging because I had no experience in the recommender systems field at all. And now I am aware of different methods and approaches, which is brilliant!