PMLDL Assignment 2, Report

Ilia Milioshin, i.mileshin@innopolis.university, B20-RO-01

December, 2023

1 Introduction

Firstly let's reformulate the recommendation task. We can consider the given data as an undirected graph with nodes that represent users and movies. In a such graph edge is a user's review of some movie. By this definition of an edge, it is obvious that connections are present only between users and movies. Thus, the main task is to predict the edge for some specific user and movie.

To simplify a task let's predict only connections that represent a good review (at least 3 out of 5). For handling such a task I would use a GNN. The main idea behind it is to train embeddings in such a way that connected nodes would have highly similar embeddings (in the sense of some similarity function, usually a dot product).

2 Data analysis

Let's consider some properties of the future graph. Firstly let's consider a rating distribution.

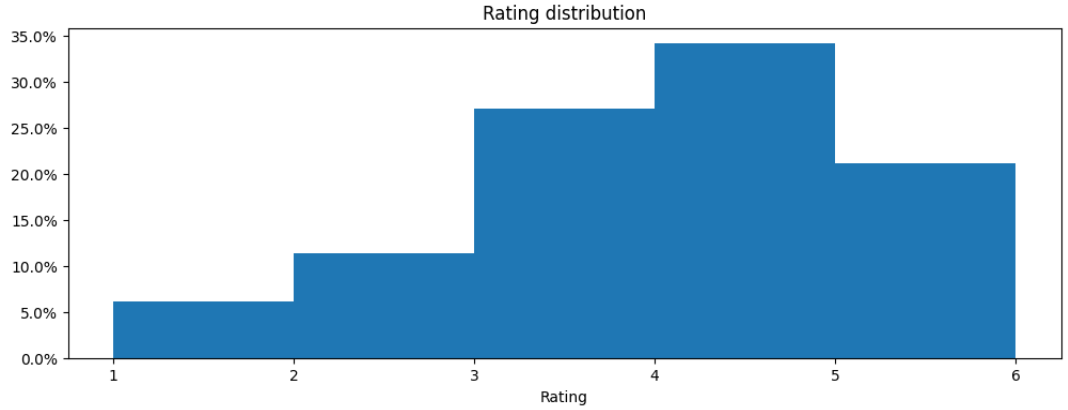

Figure 1: Rating distribution

As we can see in Fig. 1, almost 85% of all reviews are not negative. Therefore after removing "negative" edges obtained graph would be not so sparse.

Now let's see a review dates distribution:

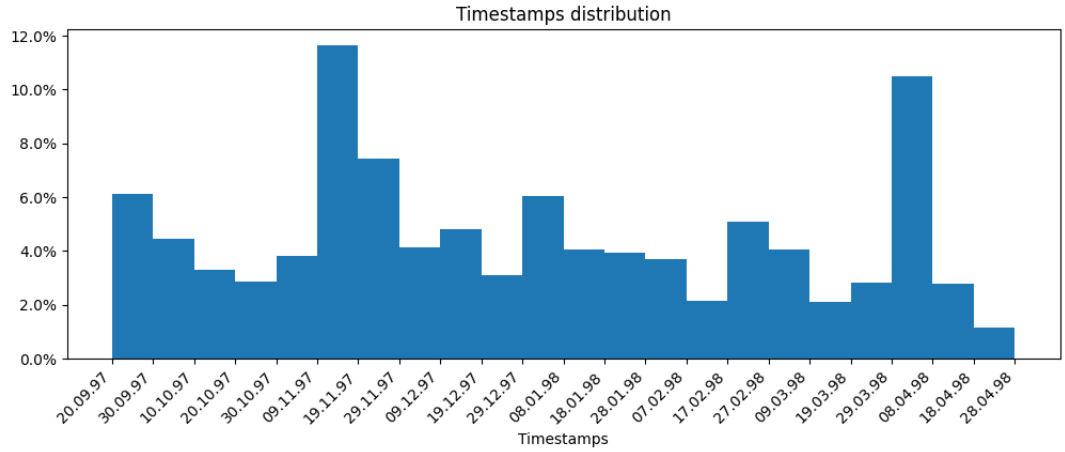

Figure 2: Dates distribution

Amount of reviews per user distribution:

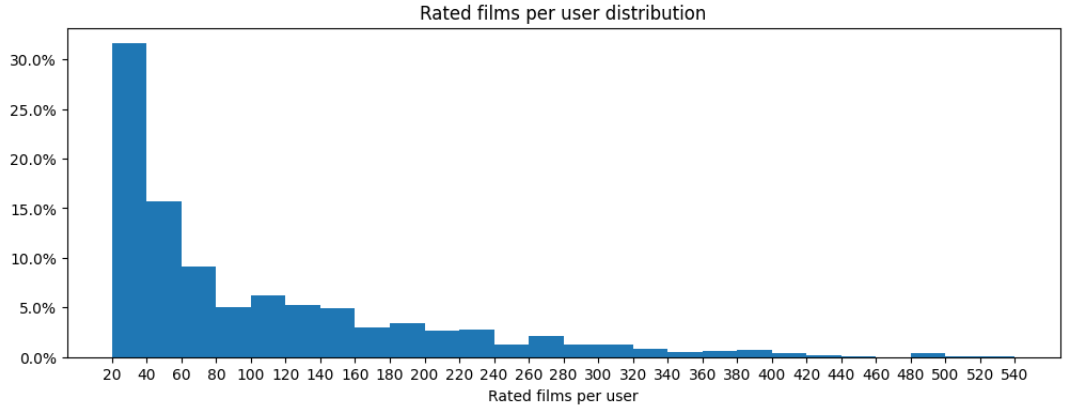

Figure 3: Amount of reviews per user distribution

From Fig. 3 we see that every node would have at least 20 connected neighbors.

Other information about data:

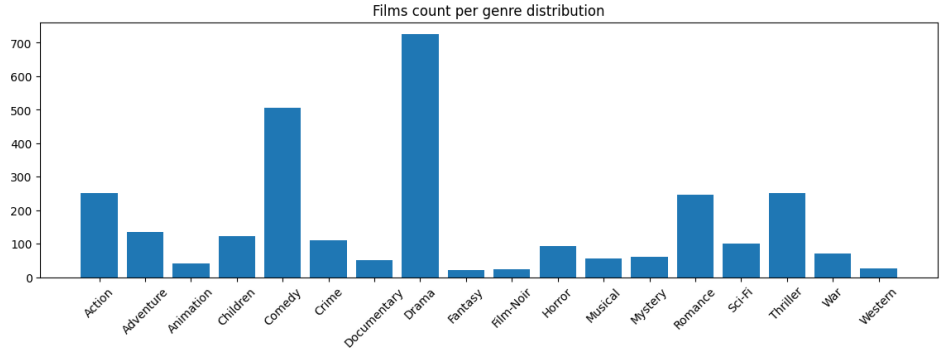

Figure 4: Films count per genre distribution

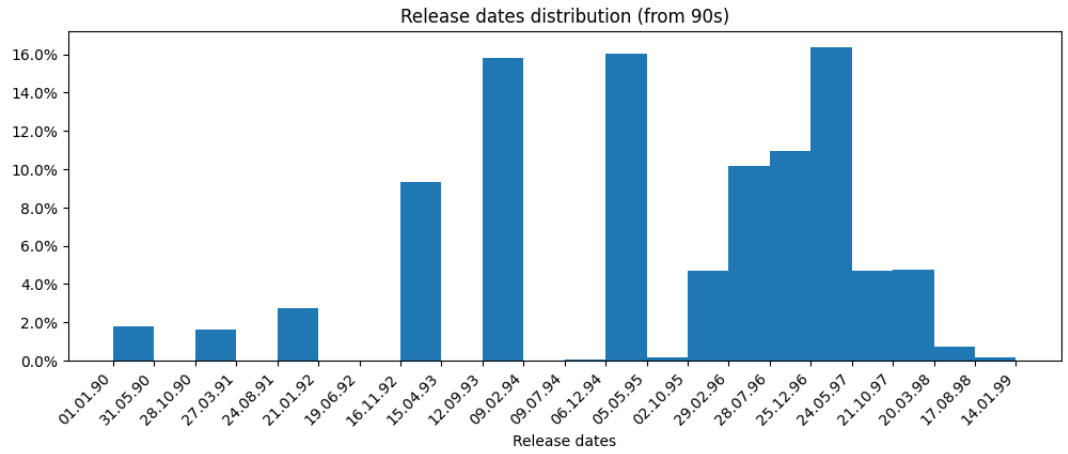

Figure 5: Release dates distribution

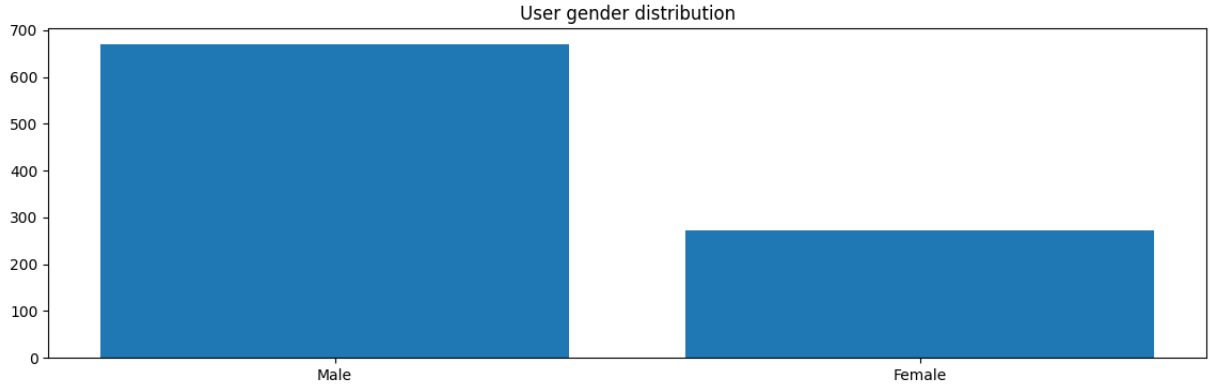

Figure 6: Gender distribution

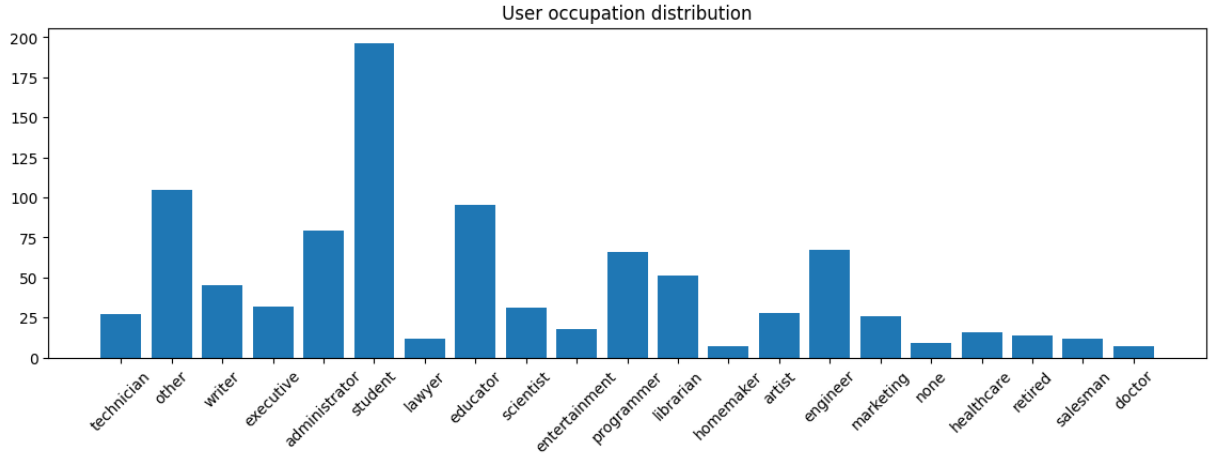

Figure 7: User occupation distribution

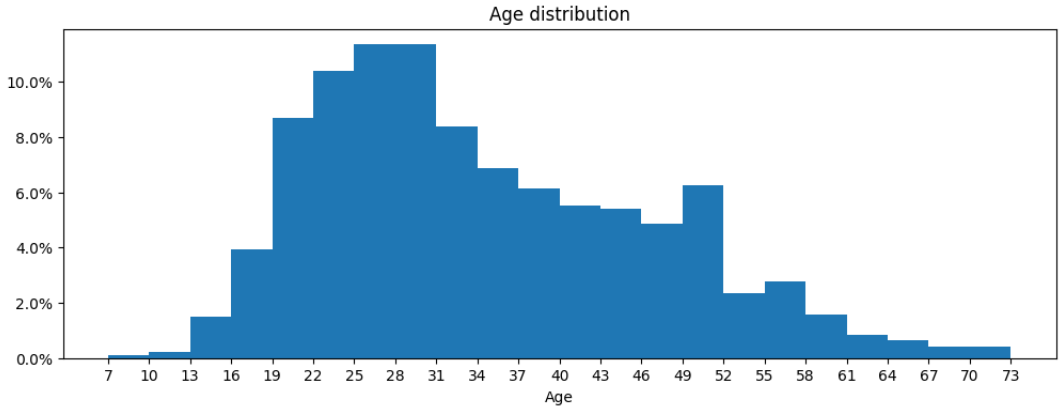

Figure 8: Age distribution

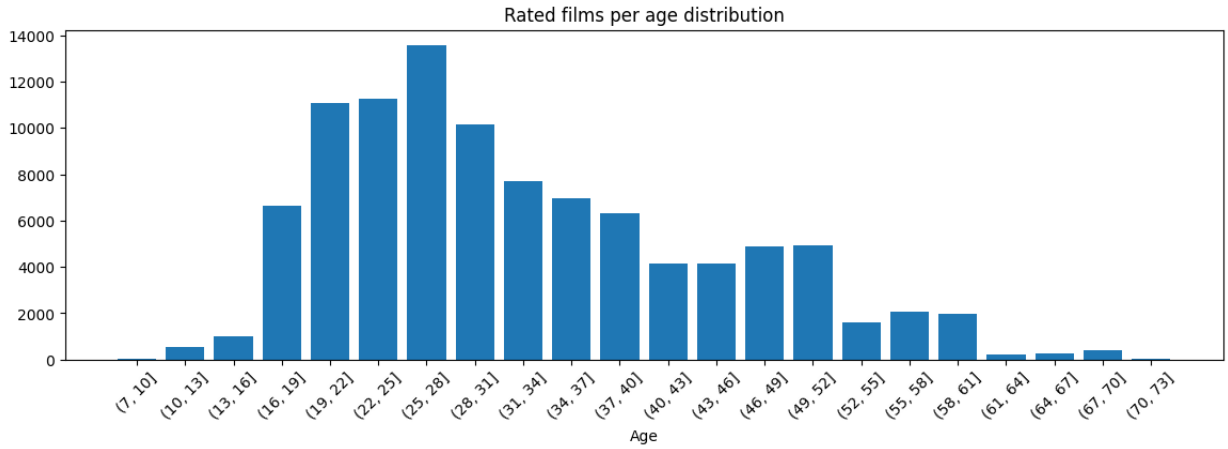

Figure 9: Rated films per age distribution

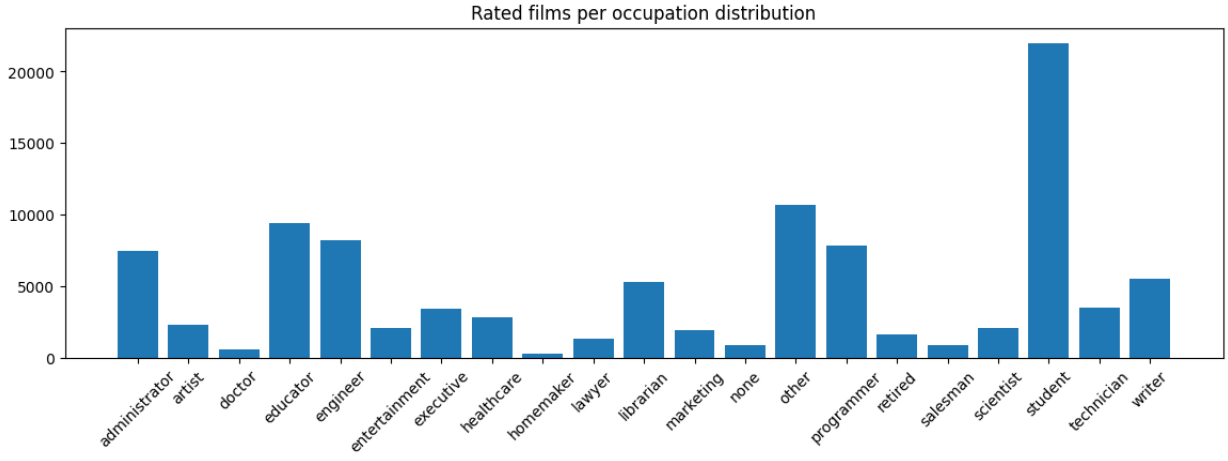

Figure 10: Reated files per occupation distribution

Let's see the mean rating distributions:

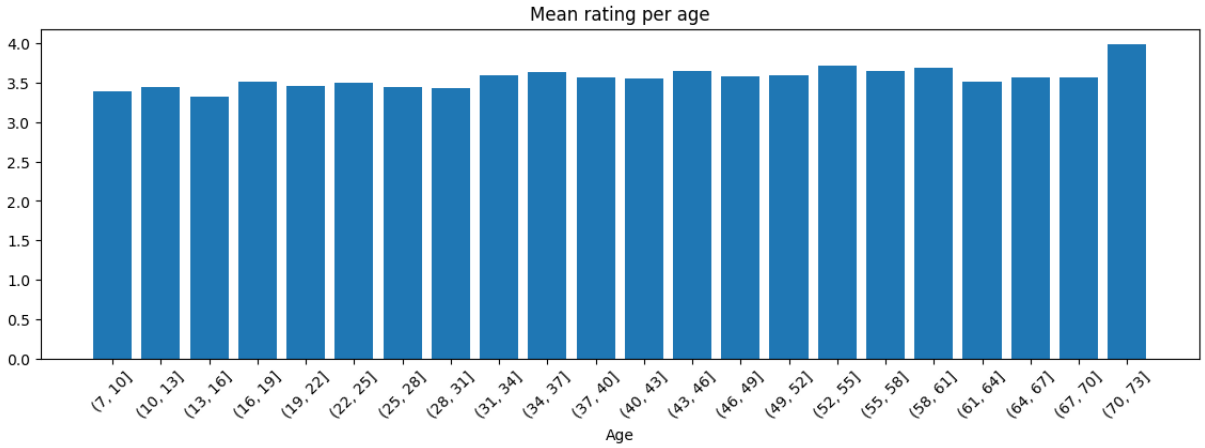

Figure 11: Mean rating per age distribution

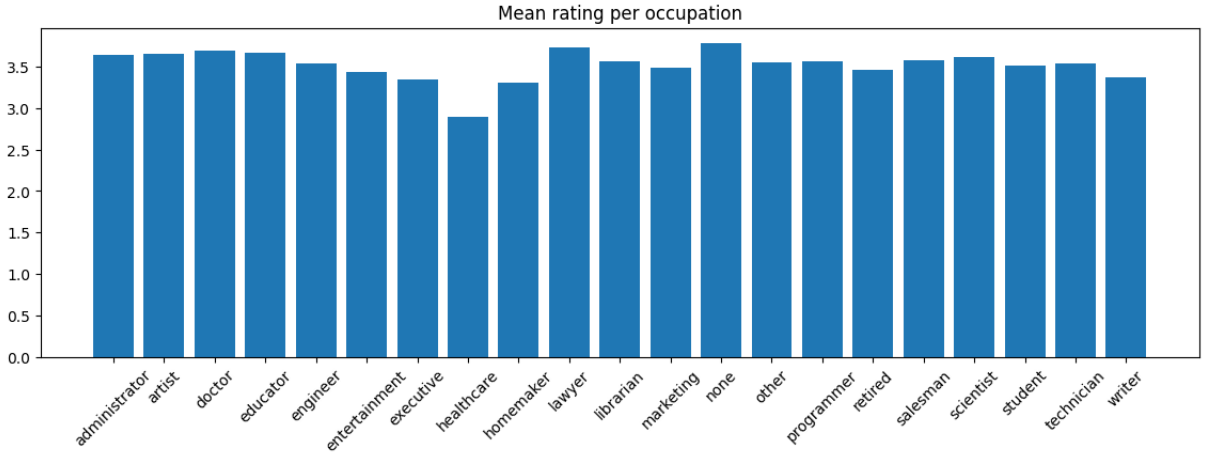

Figure 12: Mean rating per occupation distribution

Here is the list of the most favorite films per occupation:

occupation	movie title	rating
administrator	Wizard of Oz, The (1939)	4.632
artist	Star Wars (1977)	4.667
educator	Young Frankenstein (1974)	4.647
engineer	Young Frankenstein (1974)	4.600
executive	Twelve Monkeys (1995)	4.471
librarian	When Harry Met Sally... (1989)	4.684
marketing	English Patient, The (1996)	3.765
other	Young Frankenstein (1974)	4.730
programmer	Young Frankenstein (1974)	4.765
scientist	Star Wars (1977)	4.222
student	Young Guns II (1990)	4.632
technician	Toy Story (1995)	4.562
writer	Willy Wonka and the Chocolate Factory (1971)	4.875

3 Model Implementation

As mentioned above to solve the task I used a GNN (more accurately Light GNN). I took the idea from **Lab 11** of **PMLDL** course. Let's consider this solution as a baseline.

To improve it, I decided to add feature dependency. Let U be users features vectors, and I be movies features. Therefore, $U \in \mathrm{R}^{n\_users \times user\_features}$, and $I \in \mathrm{R}^{n\_movies \times movie\_features}$. Here, **n_users** is the number of unique users, **n_movies** is the number of unique movies, **user_features** is the number of user features, and **movie_features** is the number of movie features.

Hence, we update the initial embeddings in the following manner (W is trainable):

$$E_{\text{users}} = \hat{E}_{\text{users}} + U \cdot W_{\text{users}}$$

$$E_{\text{movies}} = \hat{E}_{\text{movies}} + I \cdot W_{\text{movies}}$$

As user features I used an occupation field, for movie features, I chose a genre. The main idea behind the choice is a hypothesis that people of a specific profession prefer particular genres of movies.

The training sequence remained unchanged.

4 Model Advantages and Disadvantages

The main advantage of the chosen model that I found is its' simplicity and lightness. The training does not require fancy GPUs and powerful CPUs. On the other hand, this model is a kind of smart SVM. Therefore, the trained embeddings can be easily interpreted.

However, the simplicity of the solution does not guarantee its accuracy. Moreover, it is hard to add non-vectorizable features, such as movie text, posters, and e.t.c.

5 Training Process

BPR (Bayesian Personalized Ranking) loss was used. As regularization embeddings squares were utilized. Below you can see the history of training.

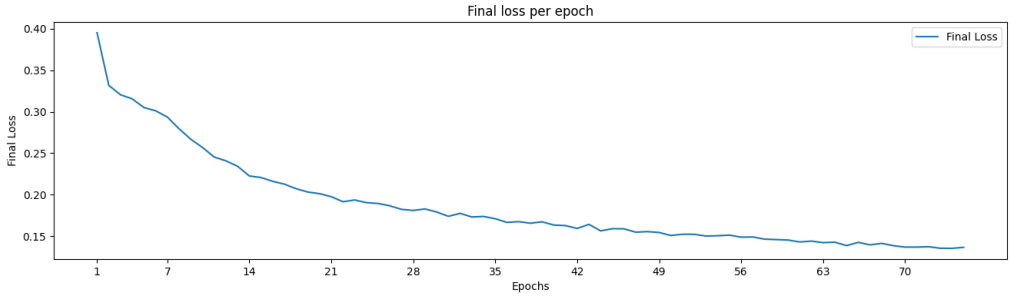

Figure 13: Final loss

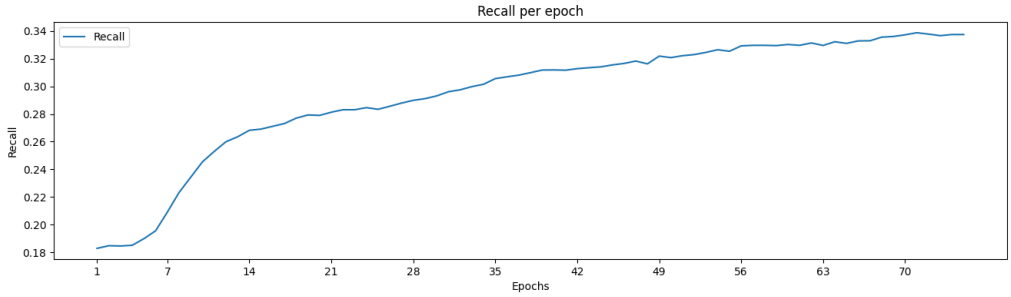

Figure 14: Recall

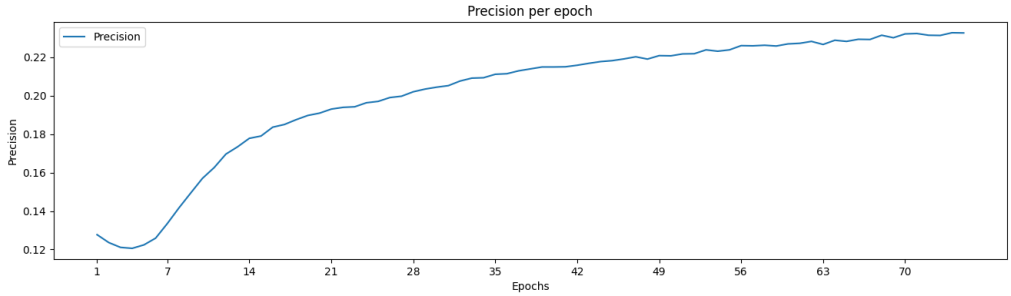

Figure 15: Precision

Here, **Final loss** = BPR loss + 10^{-4} regularization, **Recall** intersection of predicted movies with true ones divided by the amount of truly liked movies, and **Precision** almost the same, but decided by the number of predicted movies (top K most probable).

6 Evaluation

During the training process, I used a custom train-val split, where validation has no films that are not present in the train set. For evaluation, I decided to use sets provided by creators of MovieLens dataset.

7 Results

Without features support:

set number	recall	precision
1	0.2628	0.3807
2	0.2886	0.3103
3	0.3091	0.2561
4	0.3125	0.2456
5	0.3178	0.2348
Mean	0.2982	0.2855

With features support:

set number	recall	precision
1	0.2872	0.4129
2	0.3210	0.3394
3	0.3296	0.2752
4	0.3388	0.2659
5	0.3355	0.2504
Mean	0.3224	0.3088

As we can see adding features improves model predictions.

8 References

GitHub repo
MovieLens