Movie Recommender System

Personal Details

Name: Danil Meshcherekov
Email: d.meshcherekov@innopolis.university
Group number: B21-DS-02

Introduction

Recommender systems are ML-based information filtering algorithms that suggest content to users based on their past behavior.

There are two types of recommender systems:

- Collaborative filtering algorithm whose main idea is “similar users like similar items”. This method compares users by their features (age, location, occupation, etc.) and suggests items liked by the most similar users.
- Content-based filtering whose idea is “similar items will be liked by the same user”. This method compares items by how similar their content is and suggests new items with the most similar content

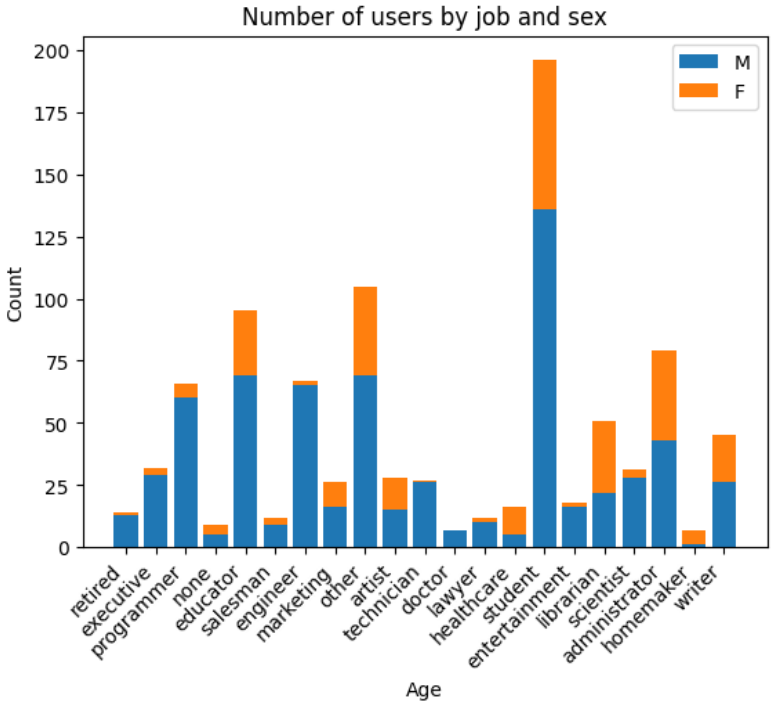
These days, recommender systems are used everywhere: video algorithms on tiktok or youtube, products suggestions on amazon, the ‘made for you’ playlist on spotify, and many others where it’s crucial to retain users’ attention.

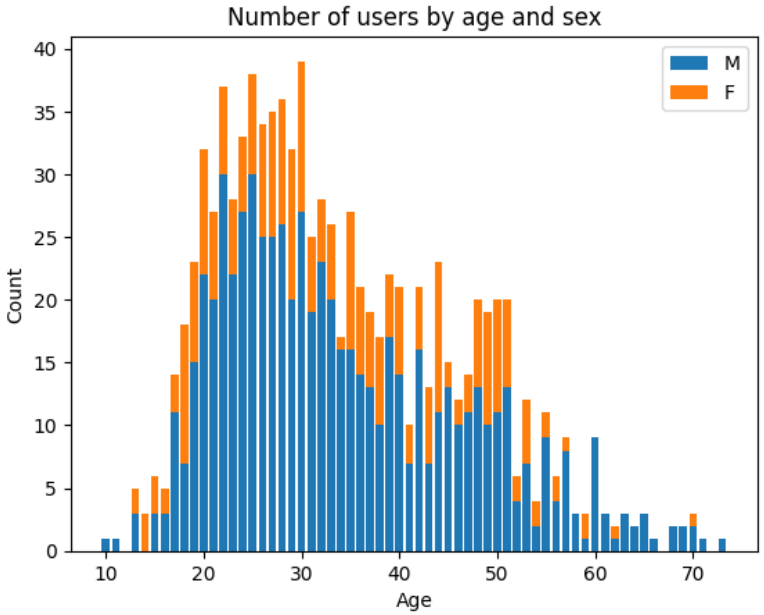
In this assignment I attempt to implement a recommender system. As the dataset for training and evaluating the model I’ll be using a MovieLens dataset 100K that consists of simple demographic information of 943 users, list of genres for 1682 movies, and of 100,000 ratings from 1 to 5 from these users on these movies.There are in total 943 users that made 100,000 ratings on 1682 movies.

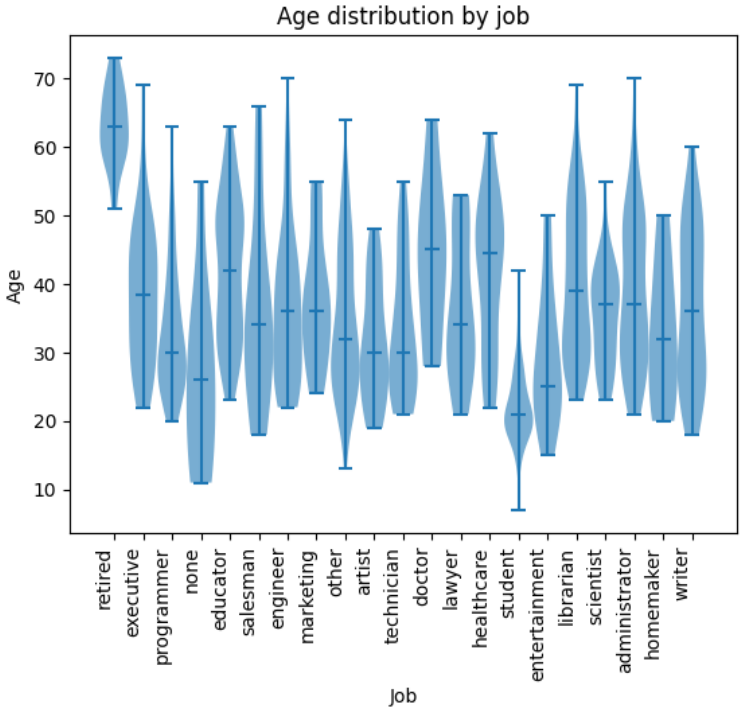
Data Analysis

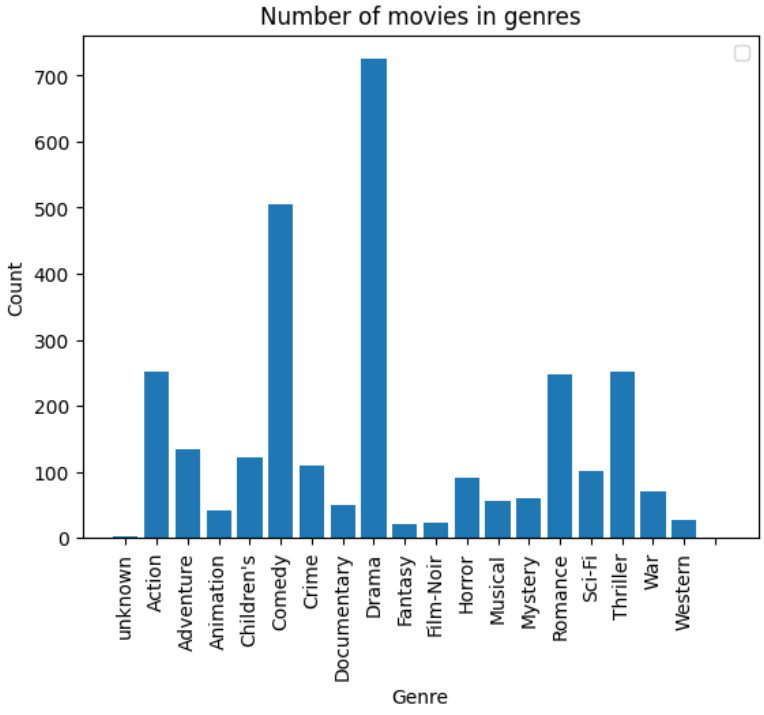
There are in total 943 users that have made 100,000 ratings on 1682 movies.

Most of the users are males from 20 to 40 years old. Many of them are either students, educators, programmers, or administrators. You can see more details in Notebook 1 with initial data exploration.









The most common movie genres are Drama and Comedy, and Fantasy ranks the last one.

TODO add more info

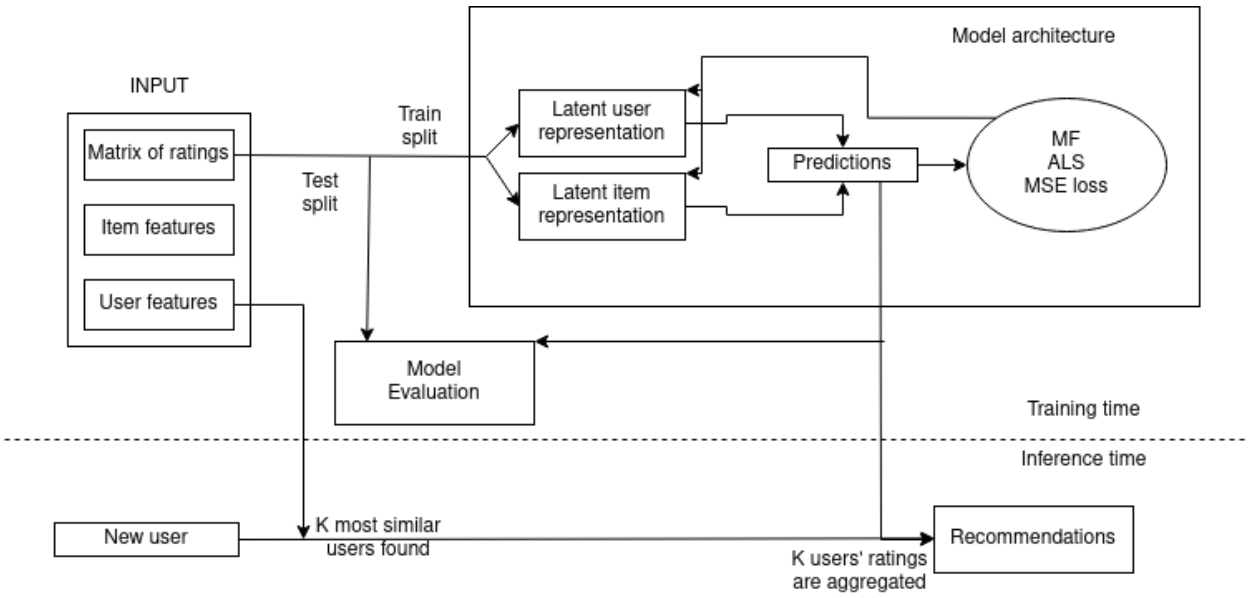
Model Implementation

My recommender system uses two collaborative filtering methods of two different stages of the algorithms.

At the model initialization time I use the Matrix Factorization method with Alternating Least Squares. The approach of this method is to decompose the rating matrix of size (n_users, n_items) into two matrices that store latent vector representations for each user and item: (n_users, lat_dim) and (n_items, lat_dim). To learn the correct latent representations and make correct predictions we optimize a regularized MSE loss over the known ratings.

At the inference time, when we need to recommend some items to a new user, I perform a k-NN algorithms: k users with the most similar features to that of the new use are chosen, their (predicted) feedbacks are averaged, and the items with the highest predicted feedback are displayed for the new user.

This is an overview of my model architecture and the training and prediction process:



Model Advantages and Disadvantages

Model advantages:

- Training is very efficient: ALS method can potentially be parallelized
- Minimization of the regularized MSE loss is a convex optimization problem that has a solution in a closed form. Training with such a loss function is even more efficient.
- The solution uses user information to make more accurate predictions

- The model can recommend items for new users that have not yet interacted with any items;
- The model doesn't require information about items

Model disadvantages:
- It doesn't utilize item information, which can make it less accurate.
- The model is a black box: the latent representations of items and users are difficult to interpret.

Training Process

In the original dataset there are seven train/test splits. However, I only use one of them to train and evaluate the model.

For each iteration, I recorded the error, accuracy, and precision (see the next section for evaluation metrics). Here are the results for training my model for 1000 iterations. The results are reproducible and can be found in Notebook 2.

Evaluation

For evaluation I used 4 metrics:
- Root mean squared error. This metric is commonly used in evaluating the performance of recommender systems because it penalizes big mistakes (for example, if the model predicted a good rating and the user feedback was bad or vica versa), and penalizes small mistakes less;

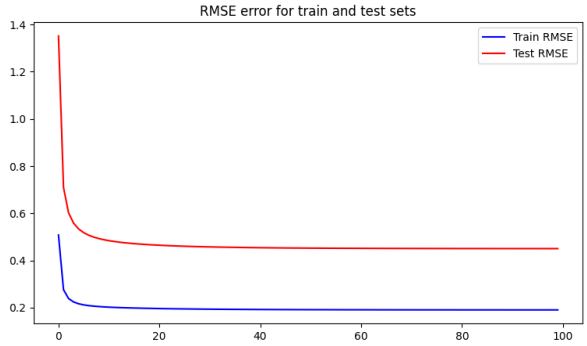
- Accuracy. Because there are multiple values of feedback, accuracy would not work correctly with the default rating system. Therefore, I firstly chose that ratings of 4 and 5 are good feedback, and other ratings are bad feedback. The accuracy is scored based on how many movies were correctly classified as having a good or bad feedback, rather than based on a predicted feedback in range from 1 to 5;

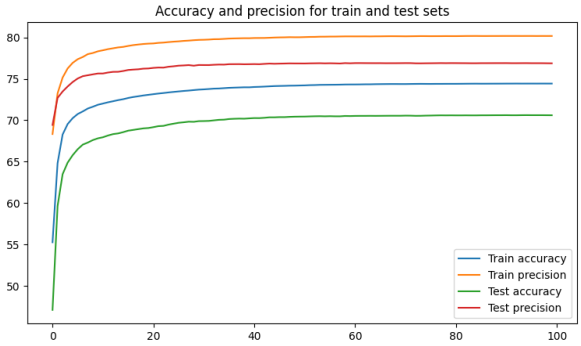
- Precision. I chose this particular metric because just like any classification problem, recommender systems may be imbalanced: some users hate everything they watch and other users love anything they watch. In particular, there are two situations: a good movie is classified as bad and not recommended (which is not bad), and a bad movie is classified as good and is recommended. The latter type, false positive, is not preferable, because this user may stop using our platform. Therefore, I included it as one of the metrics.

- Diversity. This metric simply shows how diverse recommended results are on average. The main motivation for this is that a low diversity may not increase motivation in our users to stay on our platform because they will find what they need and leave it. On the

other hand, if recommendations are too random (diverse), users may often receive items that are not relevant for them, and we will lose the audience again.

Results





After training a model for 100 epochs, the evaluations on the test set are as follows:

```
RMSE:      0.11255
Accuracy:  70.605%
Precision: 76.87676%
Diversity: 2.21706%  Bad; recommendations are too similar
```

Also, according to the graphs, around 10 epochs for training would be enough because at that time the model has already reached its local minimum.

Precision, which is the most important metric here, stands at 77%, which means that users will like around 4 out of 5 recommended items.

All the results are reproducible. You can them and the code to generate them in Notebook 2.