Recommender Systems

Niels Van den Broeck

May 16, 2025

1 Introduction

For this assignment, I made a Jupyter Notebook in which the workflow of the project is explained. Additionally, this report holds the reasoning, explanations and design choices of the data preprocessing part, the selected models, and methods of evaluation.

2 Data preprocessing

To easily access and work with the 2 given datasets, I merged movies.csv and ratings_train.csv into one dataframe, joining on the MovieId attribute. I checked whether there are any missing values present in the dataset. This is not the case, so there is no need to handle them (see later for releaseYear).

The genres are given in the genres column, which is a string of multiple genres, separated by the vertical line character. These are split and put into a list that is stored in the dataframe, for easier access if needed later on.

Ratings contain timestamps that represent the exact time the movie is rated. However, this timestamp is in Unix epoch format, which is not easily readable by humans. I converted them into a more readable datetime format which is supported by the pandas library.

In the title of the movies, the year in which it is released is added between brackets. These years could give us more insights on the recommendations of movies or the impact on ratings (e.g. ratings that come long after the release of the movie tend to be significantly better than ratings that are near the release date of the movie). Some entries do not include the release year of the movie. To solve this, I took the first rating of that movie and gave that year the release year. This covered all missing values, and thus the data is now fully usable to experiment with.

3 Task 1: Building Recommender Models and Evaluation

There are multiple ways to train a model for recommending movies based on previous ratings. Furthermore, the train/test split and evaluation method could have a huge impact on selecting the best model. In this section, we will go over all possibilities and find the best suited model to later predict and fill in the recommendations in ratings_test.csv

3.1 Train/Test split

Choosing the right train/test split is crucial when training a model. 80% training data and 20% testing is known to be a great separation and is therefore used in this project. The common way to split data is by **randomly sampling** from the original dataset into the train and test dataset. This is smart and optimal in many cases, but we want to be able to recommend movies to new users, for whom we have no data yet. With random sampling, 100% of the users in the test dataset are also present in the training dataset, meaning that it is not known how the user would react to new users. This is solved when ratings are not split randomly, but depending on the time it is published. In the **time-based splitting** method, some users will still be present in both datasets (e.g. when they rated movies over long periods of time), but most users have not rated anything yet (before the moment we split on) and thus will only appear in the test dataset. When performing an 80/20 split, around 25% of the test users are present in both datasets. We would like this percentage to be higher, but it is a great starting point for testing. The last method that I tried is **user wise** splitting, were each user's ratings are split in 80% for the training dataset and 20% for the test dataset. This once again gives a 100% common dataset rate, but it could be interesting to see how well the model performs when it has a lot of references for each user.

3.2 Models

As indicated in the assignment, 2 models are tested for recommending movies to users based on their previous ratings. The first algorithm is **KNNBasic**, which works by finding either similar users (*user-based*) or similar movies (*item-based*) and uses those similarities to predict how a user might rate an unseen movie. We experimented with different **similarity measures** and settings for the KNN model:

- {`"name": "cosine", "user_based": True`}: Finds similar users using **cosine similarity**.
- {`"name": "cosine", "user_based": False`}: Finds similar items using **cosine similarity**.
- {`"name": "pearson", "user_based": True`}: Finds similar users using **Pearson correlation**.
- {`"name": "msd", "user_based": True`}: Finds similar users using **Mean Squared Difference (MSD)**.

Secondly, we tested the **SVD** algorithm, a matrix factorization technique commonly used in recommendation systems. SVD works by decomposing the user-item rating matrix into lower-dimensional latent factors that capture the hidden relationships between users and items.

To evaluate the effect of different hyperparameter settings, we experimented with the following configurations:

- {`"n_factors": 50, "n_epochs": 20, "lr_all": 0.005, "reg_all": 0.02`}: Uses 50 latent factors, trains for 20 epochs, with a learning rate of 0.005 and regularization of 0.02.
- {`"n_factors": 100, "n_epochs": 30, "lr_all": 0.007, "reg_all": 0.05`}: Uses 100 latent factors, 30 training epochs, and slightly higher learning rate and regularization.
- {`"n_factors": 30, "n_epochs": 15, "lr_all": 0.01, "reg_all": 0.01`}: A smaller model with faster training but potentially less accuracy.

Results are shown in 3.4.

3.3 Evaluation

It is important that the predictions, given by the model, are actually good recommendations. Recommending a user a movie that he has seen, but rated very bad, is not a good recommendation. Therefore, we will count a recommendation as a hit when the user has seen it in the test dataset and gave it a rating equal to or above 3.5/5. Hits are then used to calculate precision@K and recall@K, with k in [1,3,10]. F1@K is calculated as well, but it is not used for analysis or decision-making, as it can provide a misleading representation of the model's performance.

3.4 Results

It is clear that the Time based split results in the best recommendations overall. This is unexpected due to the fact that most users (75%) are cold cases which the model have no data of. However, it might be possible that new users tend to watch and rate the same popular movies, resulting in high accuracy. SVD clearly outperforms KNN by a lot, with an astonishing 15,35% precision@10 with the following hyperparameters: "n_factors": 50, "n_epochs": 20, "lr_all": 0.005, "reg_all": 0.02. We will use this model in Task 2.

Split	Model	Precision@1	Recall@1	Precision@3	Recall@3	Precision@10	Recall@10
TimeBased	KNN-cosine-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-cosine-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-cosine-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-cosine-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-cosine-item	0.184211	0.002156	0.093567	0.002570	0.042982	0.003810
	KNN-cosine-item	0.184211	0.002156	0.093567	0.002570	0.042982	0.003810
	KNN-cosine-item	0.184211	0.002156	0.093567	0.002570	0.042982	0.003810
	KNN-cosine-item	0.184211	0.002156	0.093567	0.002570	0.042982	0.003810
	KNN-pearson-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-pearson-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-pearson-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-pearson-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-msd-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-msd-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-msd-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	KNN-msd-user	0.184211	0.002156	0.093567	0.002570	0.041228	0.003360
	SVD-f50-lr0.005	**0.473684**	0.010913	0.213450	0.013525	**0.149123**	0.026862
	SVD-f100-lr0.007	0.008772	0.000122	0.160819	0.010268	0.069298	0.012765
	SVD-f30-lr0.01	0.017544	0.000177	0.192982	0.010782	0.122807	0.020680
Random	KNN-cosine-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-cosine-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-cosine-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-cosine-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-cosine-item	0.000000	0.000000	0.000000	0.000000	0.000170	0.000014
	KNN-cosine-item	0.000000	0.000000	0.000000	0.000000	0.000170	0.000014
	KNN-cosine-item	0.000000	0.000000	0.000000	0.000000	0.000170	0.000014
	KNN-cosine-item	0.000000	0.000000	0.000000	0.000000	0.000170	0.000014
	KNN-pearson-user	0.003407	0.000625	0.001136	0.000625	0.000511	0.000909
	KNN-pearson-user	0.003407	0.000625	0.001136	0.000625	0.000511	0.000909
	KNN-pearson-user	0.003407	0.000625	0.001136	0.000625	0.000511	0.000909
	KNN-pearson-user	0.003407	0.000625	0.001136	0.000625	0.000511	0.000909
	KNN-msd-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-msd-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-msd-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	KNN-msd-user	0.001704	0.000341	0.000568	0.000341	0.000170	0.000341
	SVD-f50-lr0.005	0.080068	0.005671	0.067007	0.010981	0.050937	0.029098
	SVD-f100-lr0.007	0.069847	0.004809	0.059057	0.011756	0.039864	0.025938
	SVD-f30-lr0.01	0.081772	0.005981	0.072686	0.013667	0.051789	0.028638
UserWise	KNN-cosine-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-cosine-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-cosine-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-cosine-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-cosine-item	0.000000	0.000000	0.000572	0.000005	0.000343	0.000018
	KNN-cosine-item	0.000000	0.000000	0.000572	0.000005	0.000343	0.000018
	KNN-cosine-item	0.000000	0.000000	0.000572	0.000005	0.000343	0.000018
	KNN-cosine-item	0.000000	0.000000	0.000572	0.000005	0.000343	0.000018
	KNN-pearson-user	0.001715	0.000017	0.001144	0.000874	0.001029	0.002082
	KNN-pearson-user	0.001715	0.000017	0.001144	0.000874	0.001029	0.002082
	KNN-pearson-user	0.001715	0.000017	0.001144	0.000874	0.001029	0.002082
	KNN-pearson-user	0.001715	0.000017	0.001144	0.000874	0.001029	0.002082
	KNN-msd-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-msd-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-msd-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	KNN-msd-user	0.001715	0.000017	0.000572	0.000017	0.000343	0.000874
	SVD-f50-lr0.005	0.048027	0.004157	0.045740	0.008993	0.036878	0.022875
	SVD-f100-lr0.007	0.027444	0.001416	0.027444	0.005187	0.026587	0.015266
	SVD-f30-lr0.01	0.053173	0.003106	0.051458	0.007594	0.036878	0.019448

4 Task 2: Making Predictions for Specific Users

Now that we know which algorithm performs the best, we can apply this to the ratings_test.csv file. Each user will get 10 recommendations which the model predicts to fit the best to that user. Since we are using SVD, users that are new will get recommendations assigned based on the top rated movies from the training dataset. Since we don't have to test the model anymore, we can use the training data completely to train the model without splitting it into test and train at all. This may seem like experimenting the different splitting techniques was useless, but showing that this algorithm performed better in any splitting technique, gives us a great insight and better confidence of its accuracy.