Introduction

The challenge was to create a movie recommendation system based on the user's previous ratings and demographic information. My idea was to train a model to predict movie ratings and suggest the highest rated movies to users. To solve this problem, I used various ML methods, like classical ML techniques and Neural Networks. As a result, I got decent values for my main evaluation metric.

Data analysis

As I started analyzing the dataset, I noticed that data is distributed between multiple folders, containing information about users and movies. For my solution I decided to use a content-based approach. So, that's why I merged all the data together, encoded non-numerical parts of it, added normalization, dropped less useful features and finally obtained training and testing datasets to train my models on.

Model Implementation

Initially, I decided to build my solution based on the classical ML approach. I was not sure where to start, so I used the tpot module that provides an evolutionary algorithm to find the best pipeline for my input data. As a result, after several generations, I got a model based on the Random Forest with an accuracy of 0.445. *(See 1.0_data_exploration_and_naive_approach.ipynb in models folder for code)*

Then, I moved on and tried to train a Dense Neural Network on my data, because I wanted to find out if it would be better than the Classical ML model I got earlier. I experimented with different sizes and numbers of dense layers and training parameters, but Neural Network's metrics were always worse or equal to the previous model's results. Also, it was much slower to train. So, I decided to abandon the NN approach and try to improve the solution in another way. *(See 1.1_NN_approach.ipynb in models folder for code)*

My next thought was to add more sparseness to the train data by one-hot encoding some of the features to potentially improve predictions. So, I transformed the data and once again trained the tpot classifier. However, there was no improvement compared to previous results. *(See 1.2_more_sparseness.ipynb in models folder for code)*

Such that, the best of the approaches was the initial one, Random Forest Classifier with fine-tined parameters. *(2.0_final_solution_evaluation_and _recommendation.ipynb for code)*

Model Advantages and Disadvantages

The main advantage of the Random Forest model is the speed of training and inference because it is based on the Classical ML. Also, it works well with both categorical and continuous values and does not require data normalization as it uses a rule-based approach. It also has some disadvantages, if we compare it to oher Classical ML methods.

For example, It requires a lot of computing power as well as resources as it builds many trees to combine their results.

Training Process

The final classifier I used belongs to the scikit-learn library. Given that, training was done automatically and pretty fast, in contrast to the Neural Network training for which I had to build the model and carefully choose main parameters.

Evaluation

As the main evaluation metric for my predictions, I chose NDCG score because of non-binary notions of relevance, in our case - ratings. It represents how well I ranked movies that I want to suggest to the user in comparison to ideal recommendation.

Results

My best model resulted in a 0.928 NDCG score, which is quite good. Also, it achieved 0.865 Mean Absolute error.