Introduction

The objective of the hackathon was to develop a machine learning approach to predict the engagement between a user and a video. The training data already had an engagement score feature, which was a floating-point number between 0 and 5. This considerably simplified matters, as in recommender systems, calculating the engagement score is often more challenging than predicting them. The challenge, therefore, was to predict this score based on certain user related and video related features.

The list of features in the dataset is given below:

Variable	**Description**
row_id	Unique identifier of the row
user_id	Unique identifier of the user
category_id	Category of the video
video_id	Unique identifier of the video
age	Age of the user
gender	Gender of the user (Male and Female)
profession	Profession of the user (Student, Working Professional, Other)
followers	No. of users following a particular category
views	Total views of the videos present in the particular category
engagement_score	Engagement score of the video for a user

Initial Ideas

Two main approaches were considered:

- **Regression Models**: Since the engagement score was a continuous variable, one could use a regression model. There were several reasons that I did not use this method:
    1. The lack of features on which to build the model. The features “user_id”, “category_id” and “video_id” were discrete features that would need to be encoded. Since each of these features had many unique values, using a simple One-Hot Encoding would not work due to the increase in the number of dimensions. I also considered using other categorical encoders, like the CatBoost encoder, however, I’ve found that most categorical encoders only do well if the target value is a categorical variable, not a continuous variable like the case here. This would leave us with only 5 features on which to make the model, which never felt enough. The 5 features also had very little scope for feature engineering, apart from some ideas on combining gender and profession and age and profession.
    2. The fact that traditional regression models have never done well as recommender systems. The now famous Netflix challenge clearly showed the advantage collaborative filtering methods had over simple regressor models.

- **Collaborative Filtering**: This was the option I eventually went with. There was enough evidence to see that Collaborative Filtering was in almost all cases better than regression models. There are many different ways of implementing collaborative filtering, of which I finally decided to use the Matrix Factorization method (SVD).

Final Model

The first few runs of collaborative filtering were not very successful, with a low r2 score on the test set. However, running a Grid Search over the 3 main hyperparameters – the number of factors, the number of epochs and the learning rate, soon gave the optimal SVD. The final hyperparameters were:

- Number of Factors: 100
- Number of Epochs: 500,
- Learning Rate: 0.05

The r2 score on the test set at this point was 0.506, which took me to the top of the leaderboard.

The next step was to try to improve the model further. I decided to model the errors of the SVD model, so that the predictions of the SVD could be further adjusted by the error estimates. After trying a few different models, I selected the Linear Regression model to predict the errors.

To generate the error, the SVD model first had to run on a subset of the training set, so that the model could predict the error on the validation set. On this validation set, the Linear Regression was trained. The SVD was run on 95% of the training set, therefore, the regression was done on only 5% of the entire training set. The steps in the process were:

1. Get the engagement score predictions using the SVD model for the validation set.
2. Calculate the error.
3. Train the model on this validation set, using the features – “age”, “followers”, “views”, “gender”, “profession“ and “initial_estimate”. The target variable was the error.
4. Finally, run both models on the actual test set, first the SVD, then the Linear Regression.
5. The final prediction is the difference between the initial estimates and the weighted error estimates. The error estimates were given a weight of 5%, since that was the proportion of data on which the Linear Regression model was trained.
6. There could be scenarios of the final prediction going above 5 or below 0. In such cases, adjust the prediction to either 5 or 0.

The final r2 score was 0.532, an increase of 2.9 points.

Ideas for Improvement

There are many ways I feel the model can be further improved. Some of them are:

1. **Choosing the Correct Regression Model to Predict the Error**: It was quite unexpected that a weak learner like Linear Regression did better than stronger models like Random Forest and XGBoost. I feel that the main reason for this is that dataset used to train these regressors were quite small, only 5% of the entire training set. While the linear regression model worked well with such a small dataset, the more complicated models did not.
2. **Setting the Correct Subset for the SVD**: After trying a few different values, the SVD subset was set at 95% while the error subset set at only 5%. The reason for setting such a high percentage was that the SVD was the more powerful algorithm and I wanted that to be as accurate as possible. However, this severely compromised the error predictor. Finding the perfect balance here could improve the model performance.
3. **Selecting the Correct Weights for the Final Prediction**: The final prediction was the difference between the initial estimate and the weighted error estimate. Further analysis is needed to get the most optimum weights. Ideally, the weights should not be needed at all.
4. **Feature Engineering**: The error estimator had no feature engineering at all, in fact, I removed the feature “category_id” as well. Adding new features could potentially help in improving the error estimates, however, the benefits would be low, as it accounts for only 5% of the final prediction.