`

Machine Learning Challenge Report

The provided Python code aims to predict the self-reported well-being score measured while people were playing a video game designed to lower stress and improve mental health.

Data Processing

We initially used backward fill for imputing missing values for the categorical variables ‘CurrentGameMode’, ‘CurrentTask’, and ‘LastTaskCompleted’ as our data represented a time series. However, we then opted to fill the missing values with mode imputation because it performed better than backward fill. The missing values of the numerical features ‘LevelProgressionAmount' and ‘CurrentSessionLength’ were imputed with the median instead of the mean, as both features exhibited non-normal distributions during our exploratory data analysis (EDA), and the median is, therefore, the more robust choice. Regarding outlier removal, our group experimented with winsorization and IQR filtering, where we found out that the former did not effectively remove outliers in our data set, which is why we selected IQR filtering. Moreover, the negative values in ‘CurrentSessionLength’ were dropped as well.

Feature Encoding and Feature Engineering

For feature encoding, we transformed all of our categorical and text-like variables with one-hot encoding. Still, we removed them in the end since they did not show any high feature importance during our feature importance checks. Despite experimenting with frequency encoding ‘CurrentTask’ and ‘LastTaskCompleted’, no improvements could be seen. We then converted ‘TimeUtc’to a datetime format and feature-engineered a handful of new features such as: 'Second', 'Minute', 'Hour', 'Day', 'DayOfWeek', 'IsWeekend', 'Month', 'Quarter', 'ElapsedTime'. We further added cyclic features like ‘Hour_sin’, ‘Hour_cos’, ‘DayOfWeek_sin’, and ‘DayOfWeek_cos’. In addition, we tested a few time series methods such as rolling window, lag features, and exponential moving averages (EMA) to enhance model performance, but without significant success. Through ‘UserID’, we created five new features, which reduced our overall mean absolute error (MAE) notably. The first feature, ‘UserID_encoded’ employs frequency encoding on ‘UserID’ across both training and test sets, where missing values in the test set were filled with zeros. The second feature ‘UserID_targetencoded’ target encoded the ‘UserID’ based on the ‘ResponseValue’ of the training set, and replaced the missing values in the test set with a global mean of the ‘ResponseValue’. Through hyperparameter tuning, we identified an optimal smoothing parameter of 0.7 for ‘UserID_targetencoded’. The final features are aggregated summary statistics such as mean, min values, and max values of the ‘ResponseValue’ (named ‘UserID_mean’, ‘UserID_min’, and ‘UserID_max’ respectively). Those features were then grouped by the ’UserID’ for both training and test sets, and missing test values were imputed with the corresponding means from the training data.

Learning Algorithms and Hyperparameter Tuning

After feature engineering, we tried modeling with different models such as Lasso Regression, XGBoost, DecisionTreeRegressor, and RandomForestRegressor. At the same time, we considered using time series models like ARIMA, SARIMA, and LSTM but opted for a simpler model due to implementation challenges. The best-performing model was a RandomForestRegressor model with manually tuned hyperparameters (n_estimators=300, max_depth=20, min_samples_split=10, min_samples_leaf=3). Our final RandomForestRegressor model achieved a validation MAE value of 82.729 and an MAE score of 108.3801 on the Codalab competition site. (Group 32, Codalab name: Bladee)

Model Improvement

To further improve our MAE score, we could have implemented a time series model as mentioned above or a deep learning algorithm with embedded methods, which would be more adept at handling textual variables like ‘LastTaskCompleted’ and ‘CurrentTask’. Additionally, applying KNN imputation on the numerical features might have improved our overall MAE score as well. Despite our efforts in applying GridSearchCV and RandomizedSearchCV regarding hyperparameter tuning, it took significant amounts of time to run the models with them, which is why we opted not to implement them.

`

Hsuan Jung Chu **u542596**	**Dominik Duy Duy Nguyen** **u674667**	**Theun Schmalz** **u843860**	**Paul Van Eijkeren** **u202130**
Adjusting existing features, Creating new feature: UserID_encoded & UserID_targetencoded, Testing XGBoost, LSTM Writing the report	Adjusting existing features, Creating new features: aggregate statistics for 'UserID', 'TimeUtc', Testing RandomForestRegressor, Writing the report	Adjusting existing features, Testing features for: LevelProgressionAmount & CurrentSessionLength, Testing DecisionTreeRegressor Writing the report	Hyperparameter tuning, Testing Lasso Regression, ARIMA, SARIMA, Writing the report

References

Banerjee, P. (2019). Random Forest Classification with Python and Scikit-Learn. https://gist.github.com/88545fa33780928694388779af23bf58.git

Kuhn, M., & Johnson, K. (2019). Feature Engineering and Selection: A Practical Approach for Predictive Models (1st ed.). Chapman and Hall/CRC. https://doi-org.tilburguniversity.idm.oclc.org/10.1201/9781315108230

Scikit-learn: Machine Learning in Python, Pedregosa et al., 2011

Galli, S. (2022). Python feature engineering cookbook: over 70 recipes for creating, engineering, and transforming features to build machine learning models (2nd edition). Packt Publishing.

Breiman. L., Adele. C., Random Forests, https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#missing1