Predicting Social Media likes based on profile parameters

Arka Mukherjee
Université de Montréal
arka.mukherjee.1@umontreal.ca

Pavithra Parthasarathy
Université de Montréal
pavithra.parthasarathy@umontreal.ca

Emile Dimas
Université de Montréal
emile.dimas@umontreal.ca

Padideh Nouri
Université de Montréal
padideh.nouri@umontreal.ca

KEYWORDS

Machine Learning • Data Science • Stacking Generalization

1. Introduction:

Social media usage has been on the rise since its inception, and we know it produces an enormous amount of data. Moreover, the outbreak of the ongoing COVID-19 pandemic has been a catalyst for the increased usage of social media. This field is very lucrative, and a lot of effort is put into social media analysis. Creating better models can help us understand user habits and make the experience more enjoyable. The goal of this project was to use social media profile features to predict the number of likes of a specific profile by optimizing a regression problem and minimizing the Root Mean Squared Logarithmic Error (RMSLE).

After cleaning, imputing and transforming/normalizing the data (section 2), exploring, creating and selecting useful features (section 3), we tried different models to predict the number of likes. All our best performing models were different types of stack generalizations. The best model we used was a Stack of Adaboost, XGBRegressor, ElasticNet, RandomForest, Bagged SVR, and a final estimator as a Bagged SVR. This model gave us a Root Mean Squared Logarithmic Error of 1.70234 on the public leaderboard and 1.60097 on the private one. Section 4 describes the methodology adopted and the models used during this project. Section 5 illustrates our results. Finally, we conclude and explain the next steps.

Stacked Generalization (called in short, stacking) is an ensemble machine learning algorithm that combines base models' prediction using a meta-algorithm[1]. These base models are trained individually with in-sample data. The trained models then use the out-of-sample data to create the input of the meta-algorithm. K-fold is commonly used with stacking. At each iteration, the base models are trained on data from k-1 folds. The meta-algorithm then uses the last fold to learn the best combination of each of the base-models.

2. Data cleaning:

Data cleaning is a crucial step in any data science project, and according to **Dasu and Johnson**, 80% of data analysis time is spent on cleaning and preparing the data. The first step that we did in data cleaning was to check the features that had missing values. Broadly, there are two ways to deal with missing values – Imputation and Deletion. We tried to come up with different statistical imputations as we wanted to preserve more data. Mean imputation made the results swing due to outliers. Median imputation was a more stable choice in our case. Some of the features had empty spaces instead of missing values. We replaced them with unknown values initially. For the profile colors, we replaced missing values with 'FFFFFF'. Later we changed them in the exploration phase. "Location Public Visibility" had '??' as one of its values. This was replaced in encoding with a numerical feature. There were different cases of alphabets used like 'Enabled' and 'enabled'. But they essentially mean the same. So, in such cases, we replaced all of them with lower cases. We converted the timestamp to DateTime format. Finally, after extracting useful information from the features, we dropped the columns (very low correlation) that had no impact on predicting the 'Num of Likes'.

3. Exploratory Data Analysis:

Images:

Image feature extraction was, without a doubt, one of the most challenging steps in this challenge. We first tried the lazy approach where we used pre-trained VGG-16 to extract feature maps at different depth levels since it is known that the different layers of the CNN extract different levels of granularity, with the last layers being more specific to the images used in training. These features did not bring any significant additional information to the primary model. In fact, the best correlation we were able to obtain between the CNN output and the target variable was only 0.0002. We, therefore, decided to try the manual approach using pre-processing techniques (especially those learned in class) and then create features with the outputted data. For example, we decided to threshold the low-value pixels since they are

acting as outliers. In fact, all images contained a lot of pixels with the value 1, and that was not informative since it corresponded to the background. Moreover, this large amount of non-informative pixels was covering useful information (when analyzing the histogram, for example). This processing step gave rise to more informative histograms[2]. We also tried other transformations, like Log, Gamma, HoG (Histogram of Gradient) transforms and normalization.

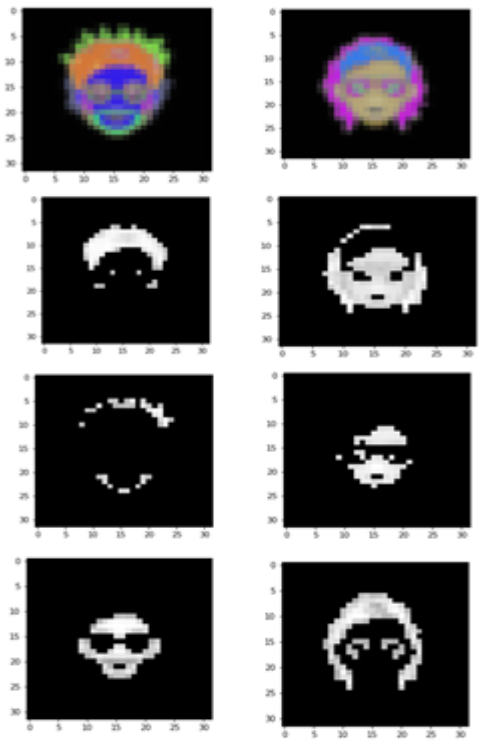


Figure 1: Exploring different filters on the image data

We attempted different methodologies to create additional features: brightness of the pictures, sentiment analysis using OpenCV and Haar Cascades[3], along with pre-trained CNNs on the FER+ dataset. We also tried to use the average color in an image, and the RGB components, the peak value in the histogram and many more. The highest correlation between the features created and the number of likes was only 0.03 for the feature "number of pixels with value > 20". We tried to train the model with this feature added to our baseline features. Unfortunately, the model was performing worse. We, therefore, decided to discard the images for the model.

Text:

Let us now explain our approaches for the CSV data. With basic imputation and conversion to numerical features, we see that we have significantly less correlation (Pearson's correlation) with 'Num of Likes'. So, we checked the distributions of each of the features and transformed them to get a higher correlation. "Num of Followers", "Num of People Following," "Num of Status Updates", "Num of Direct Messages", and "Avg Daily Profile Clicks" had skewed distributions. A log transformation was applied to all of them so that we obtain a near-normal distribution. This way, the prediction is on the linear features rather than exponential features, and we have a better loss. For "Location Public Visibility", all categories have peaked distribution, and a label encoding performed better than one-hot encoding.

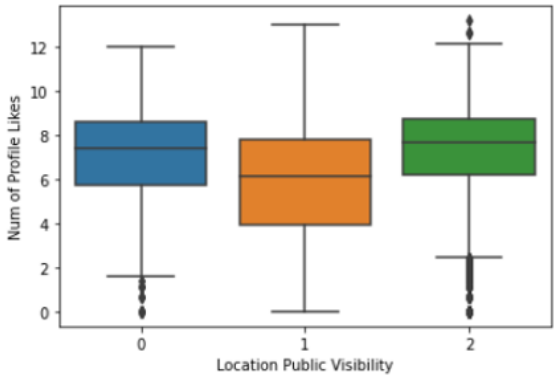


Figure 2: Location Pub. Visibility v/s Likes

A Quantile Transformation was applied to remove outliers in "Avg Daily Profile Visit Duration". A one-hot encoding was applied to "Profile Categories" and "Languages". The ones that had lesser correlation were dropped. Other features that had just 2 categories were also encoded to binary categories. For the colours, we converted the hex to RGB integer values, but this had a lesser correlation. A simple frequency encoding gave a lower value of the loss. In the case of "UTC Offset", we first tried to impute the missing values using the Location (where it was available) and PyTZ library in Python by mapping the Location to the closest PyTZ timezone. However, this gave us very little correlation. Instead, merely imputing with the mean and applying a Standard Scalar showed a higher correlation with the target.

We had observed that 56% of "Personal URL" data was missing. But we came up with a new feature based on the presence or absence of URL. Those users with a URL, had a higher number of likes. This new feature had a maximum correlation with the target.

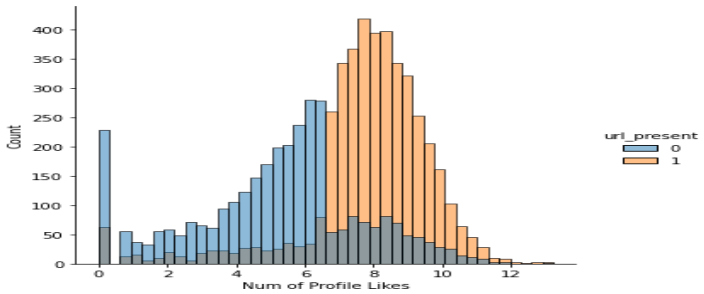


Figure 3: URL vs Likes Analysis Plot

Another feature that we came up with was the "Number of Months", since the account creation and year in which the profile was created. We dropped the latter because of a lesser correlation. One of the most important choices made was to Log-transform the 'Num of Likes' and trained the model on this data because of its skewed distribution. The predicted values were then transformed back to the exponential values. Since there were some outliers in the target, we imputed some of the outliers using a random forest model. But that gave us a lousy accuracy as the data generated probably did not belong to the same distribution as those newly imputed values. Kindly refer to the appendix for the visualization of some of these distributions.

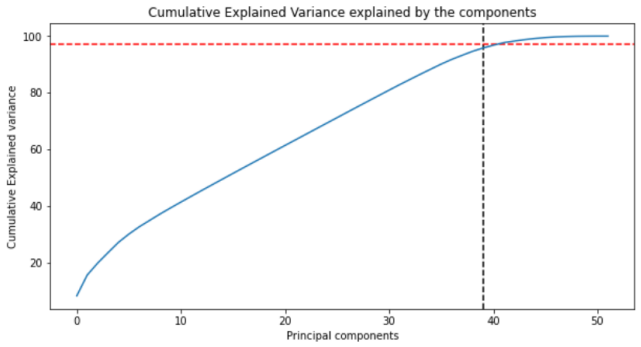


Figure 4: PCs vs Cumulative Explained Variance

We had applied PCA (Principal Component Analysis) to capture the variance in all the features but examining the cumulative explained variance. The graph shows that we need 40 components to get the 95% variance, which is way more than the number of features. Hence, we had to proceed to feature selection. Kindly refer to the appendix for the correlation visualization.

4. Methodology

Imputation and Deletion: Imputation generally refers to filling missing values with other values based on statistical parameters or values of other columns, and deletion refers to removing missing rows from the training data altogether. We tried different imputations, as mentioned in the previous sections.

Frequency and OneHot Encoding: Frequency encoding is an encoding technique that encodes categorical feature values to their frequencies. OneHot encoding is a combination of values with a single high bit and all the other bits as low.

Regularization: This is a Machine Learning technique that is used for penalizing certain weights of models. For our ElasticNet model, we could observe that an L2 Regularization fared better than an L1 Regularization. This technique was also used in our XGBoost, MLP and Ridge Regressor.

Feature Scaling: Finally, we applied standard scaling for all the features, including the categorical ones. This strategy reduced the RMSLE by a lot.

Cross-Validation: This is a broad group of techniques used to validate how a prediction model would fare on unseen data. We employed two different kinds of cross-validation – Holdout CV and K-Fold CV.

Holdout Cross-Validation: In this technique, we roughly split our dataset into two sets – a training dataset with x% of the data, and a test set with y% of the data, and generally, x >> y.

K-Fold Cross-Validation: We divide our data into K different blocks/folds, and one group is used for testing the model, and the rest are used for training. This is usually more robust compared to Holdout CV.

The best performing models for this challenge are reported in **Figure** . XGBoost **(Tianqi Chen, 2016)** and LightGBM **(Guolin Ke, 2017)** which have similar optimization algorithms based on Gradient boosting show relatively better performance compared to other algorithms as a solo model. To improve the performance and reduce the generalization error we combine the capacities of some of our best performing models. We used stacking method to combine the models reported in **Table 1** (Appendix).

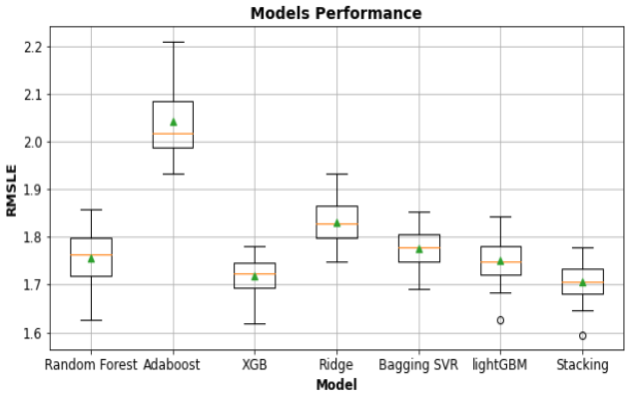


Figure 5: Models and RMSLE scores

As stated by David Wolpert in the original paper on Stacked Generalization. **(Wolpert, 1992)** We do not know a priori which choice of the model would be the best for base or meta-model. Therefore, after extensive Random and Grid Search, we present our top models in **Table 1**. The Libraries used for models include Sklearn, XGBoost[5] and LightGBM[4]. For instance, Ridge regression with linear least squares loss function and L2 regularization can show good results as a meta-algorithm. More details about models and hyperparameter choices are available in the result.

5. Results and Discussion

Linear Regression gave us the highest RMSLE, with a K-Fold Cross Validation score of around 1.9. We could conclude from the results that it was underfitting, as the decision boundary was relatively straightforward given the problem and the data. As we increased the model complexity, our results started to improve, which is why we decided to try models with a higher capacity (a more varied decision boundary). For instance, it can be observed in Figure tweaking the learning rate of XGB can have a significant influence on RMSLE. More information on choices of hyperparameters and the observations using a 10-fold cross-validation method is available in the appendix. The ranking of our models is coherent in the public and private leaderboard. This could be another confirmation that hyperparameters' choices and our method for cross-validation were successful, and we did not overfit the data.

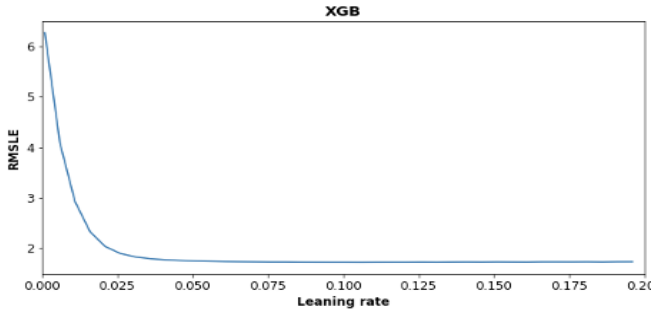


Figure 6: RMSLE variation as a function of the learning rate

6. Conclusion and Future work

We could conclude that given this dataset, we can create a robust regression model by preferring imputation over deletion during data cleaning and dropping columns, which have a very low correlation value. Furthermore, tree-based models/models with a higher capacity performed better individually, and the overall stack of regressors performed the best when each individual model had a performance in the same range. It is also imperative for a data science project to make sure the data has very few outliers and the distribution is closer to a standard bell curve. One also has to make sure that the target variable has a symmetric distribution and not a skewed one. Without appropriate skew fixes and normalizations, we were getting an RMSLE value of around 2.1 during our initial tests. Hence, more work in the image feature extraction section might potentially help us get a better prediction, and further hyperparameter tuning in LightGBM can aid the forecast as well.

During some of our tests, we observed that we could get a correlation up to 0.022 when computing the log-transformed mean of the histogram of individual images. This suggests that there could be a way to extract features from images that improve the predictions. Furthermore, HoG and darkness/brightness channel analysis showed promise for some images, but there was no overall benefit. We can propose a hypothesis that the images could be useful for predictions if the features are extracted differently, which, can improve the RMSLE score of our model. We had also observed that tree-based models like LightGBM were the best suited for this problem and performing an eloquent Grid-Search technique on such a model will improve the results obtained. However, the computational cost is extremely high, as LightGBM models have many parameters that are inter-dependent.

7. Statement of Contribution

All team members contributed equally and worked together for each task - data analysis, developing methodology, coding and optimizing, result and graph analysis, and writing the report and presentations. We hereby state that all the work presented in this report is that of the authors. All work is unique, and our own and referenced works used has been cited.

8. Bibliography

Guolin Ke, Q. M.-Y. (2017). *Advances in Neural Information Processing Systems 30.*

Tianqi Chen, C. G. (2016). XGBoost: A Scalable Tree Boosting System.

Wolpert, D. H. (1992). *Stacked Generalization, Neural Networks, Wolume 5, Isssue 2* .

Dasu T, Johnson T (2003). Exploratory Data Mining and Data Cleaning. Wiley-IEEE

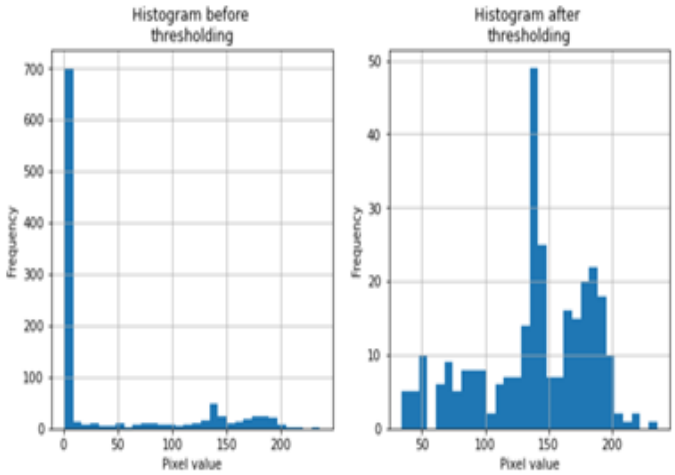
Wickham, Hadley. "Tidy data." *Journal of Statistical Software* 59.10 (2014): 1-23.

9. References:

1. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html
2. https://likegeeks.com/python-image-processing/
3. https://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html
4. https://lightgbm.readthedocs.io/en/latest/
5. https://xgboost.readthedocs.io/en/latest/parameter.html

Appendix

1. Image Histogram Analysis:

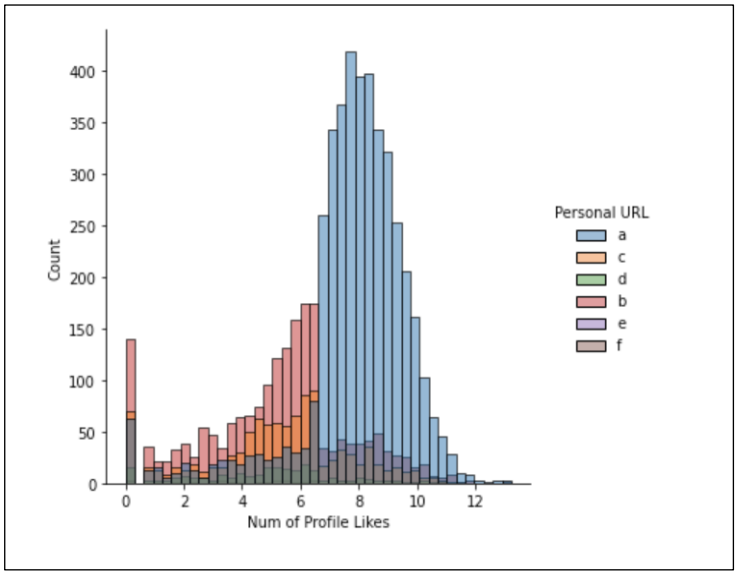


2. Leaderboard results

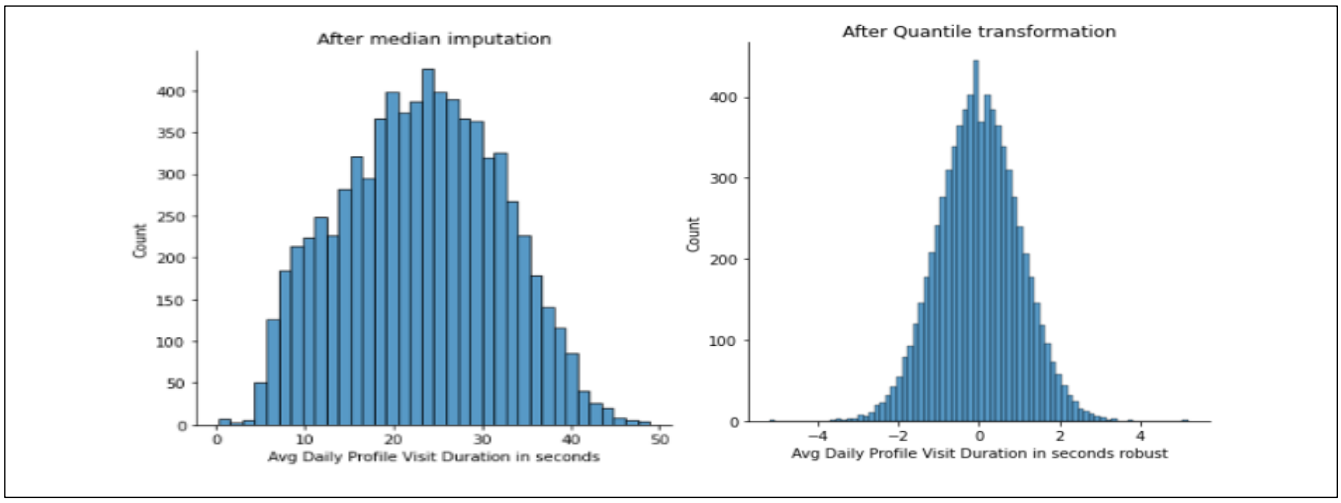
Submission		Private Score	Public Score
Base Models	**Meta Model**		
Adaboost, XGB, Ridge, Random Forest, LightGBM	Ridge	1.59895	1.72006
Bagging SVR, Adaboost, XGB, Ridge, Random Forest, LightGBM	Bagging SVR	1.60512	1.71272
Bagging SVR, Adaboost, XGB, Ridge, Random Forest	Bagging SVR	1.60097	1.70234
Adaboost, XGB, Ridge, Random Forest	Bagging SVR	1.60697	1.70408
Adaboost, XGB, Ridge, Random Forest	SVR	1.64674	1.6702

Table 1: Leaderboard results

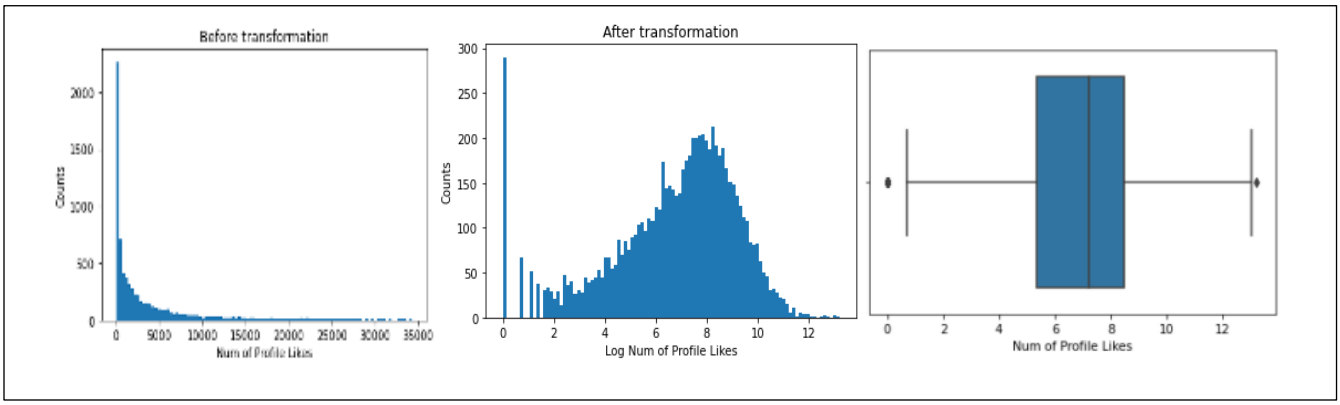
3. Distribution of URL Presence (into multiple classes, based on the URL string length):



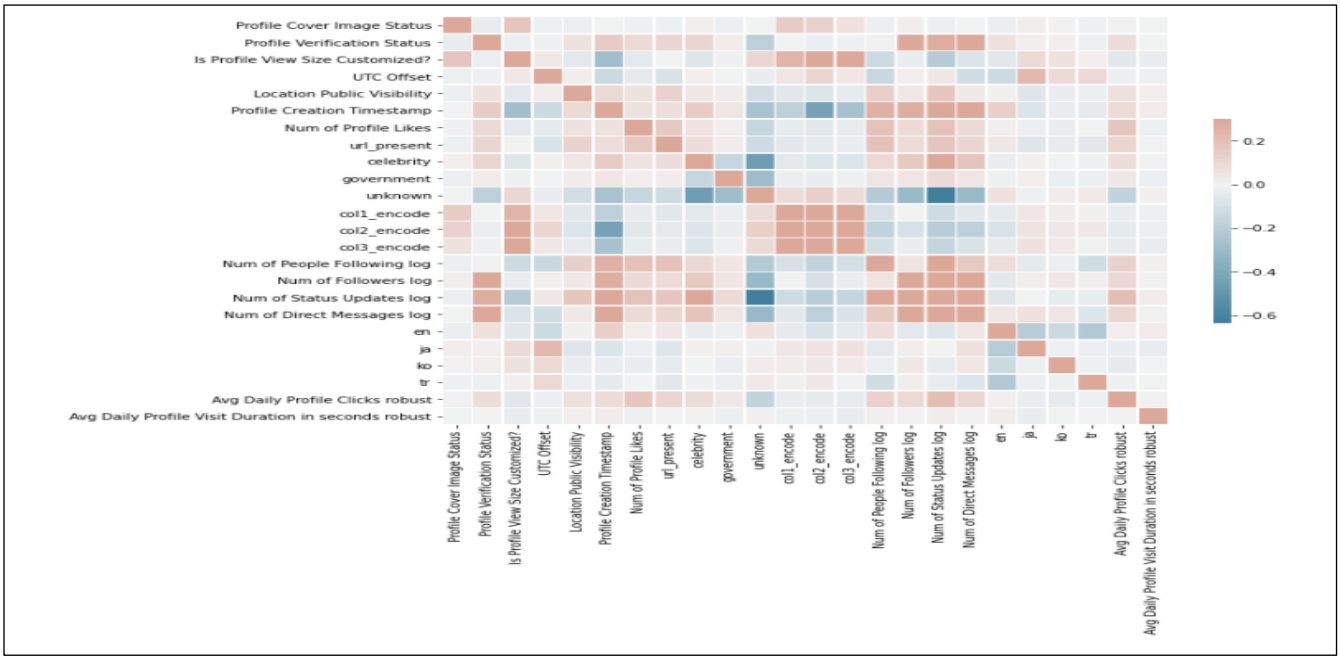
4. Average daily profile visit duration in seconds – 25,75
   Quantile Imputation Strategy:



5. Profile likes transformation and imputation:



6. Correlation between features:



7. Hyperparameter Tuning: