A Statistical Analysis of Github Users' Activities for Predicting #Target Commits

David Yang
July 15, 2017

1. Introduction

The **primary goal** of this project was to develop a machine learning approach for predicting future number of commits - **target_commits** - given a GitHub account's activity metrics summarized for the previous year.

Two data sets were given for this project, together comprising a year's worth of activity metrics for 55,000 different Github users:

Training Data ("train_data.tsv") had 50,000 rows and 21 columns including 'target_commits' column. This set of data was used for all the data analyses including exploratory data analysis (EDA) and model building.

Testing Data ("test_data.tsv") had 5,000 rows and 20 columns, all having the same format as the Training Data set, except for the target column. This data set was hold-out for make final predictions.

The outcome variable, **target_commits,** was defined as "*the number of commits made by the user during the month immediately following the last month of the year for which data was pulled.*" For example, if data was pulled for the year 2015-04-01 to 2016-03-31, then it would give the number of commits for the month of April, 2016.

2. Methods

A combination of classifier with regression model was mainly conceived in order to handle the observation of a mixture distribution with zero-inflation for the outcome variable. By splitting the **Training Data** into two subsets, I conducted a simple version of cross-validation for model building and model selection:

- **Train_Subset** ('X_Train' contain data on all predictors and 'y_Train' containing the data on the target variable) with 70% randomly selected instances were used for fitting ZIG (zero-inflated Gaussian) model and the RFR (random-forest regressor).

- **Validation_Subset** ('X_Validate' containing data on predictors, and 'y_Validate' contain data on the outcome variable) with rest 30% instances were used for evaluating the performance of the fitted ZIG Models for model selection purpose.

Alternatively, I considered a 'black-box' ensemble approach using Random Forest Regressor (RFR) to predict tart_commits based on the original and derived predictor features. Such a method employed regression trees to handle nonlinearity relationships, mixed variable types (continuous vs categorical), missing values, and outliers.

2.1 ZIG Model

As seen in Figure 2, there was strong evidence that the outcome variable had a mixture distribution of Bernoulli and Gaussian. The former part determines whether target_commits has value zero. If not, then the Gaussian part determines the distribution of non-zero value. Because the values are never negative, the choice of Gaussian is in asymptotic sense. Other choice such as Negative Binomial may be a better choice theoretically.

Denoting *Yi* as the value of target_commits, and *Xi* as feature vector for user *i* (*i=1,2, …,n*) in the training data, then we have

$$Y_i = X_i\beta + \epsilon \qquad if Z_i = 1$$
$$Y_i = 0 \qquad if Z_i = 0$$

where $\epsilon \sim N(0, \sigma^2)$ and *Zi* is latent variable with probability determined by *Xi that can be estimated from a classifier such as a logistic regression:*

$$Prob(Z_i = 1) = 1/(1 + exp(-X_i\gamma))$$

Then, I would fit a Zero-Inflated Gaussian (ZIG) modeling procedure in 3 steps:

- **ZIG Step 1**: Fit a Ridge regressor (RR) with to the instances in Train Subset with positive outcomes (target_commits>0). X_Train_Active and y_Train_Active contain data on features and target_commits, respectively.
- **ZIG Step 2**: Fit a Random Forest Classifier (RFC) using instances from Train Subset (marked as X_Train and y_Train). RFC aims to predict whether an instance would have target_commits of zero or not (i.e., target_positive = 1 if target_commit>0; target_positive = 0 otherwise).
- **ZIG Step 3**: Finally, I use the instances in Validation Sets (marked as X_Validate and y_Validate) to estimate RMSE (root mean squared error) of the above ZIG model defined as the joint use of RFC and RR.

If the performance of ZIG modeling strategy ends up with a small RMSE, then I apply it to the Testing Data for predicting target_commits for final submission.

2.1 Random Forest Regressor (RFR)

RFR is an ensemble method that employs many regression trees, each one is fit to a set of Bootstrapped data set (resample the original data set with replacement). By averaging the predictions of trees, RFR can significantly reduce the RMSE mainly by reducing variances between the trees.

As seen in Figure 1, a regression tree can predict the continuous outcome variable by dividing the feature space into regions, and then giving each region a constant number as estimate. It is a nonparametric method for regression problems and is capable of handling missing value, outliers, mixed categorical and continuous features.

RFR is a conveniently out-of-box solution, fairly robust and usually has high prediction accuracy. Nonetheless, it is a black-box solution, hard for interpretation and requires more computing power, compared with standard regression models.

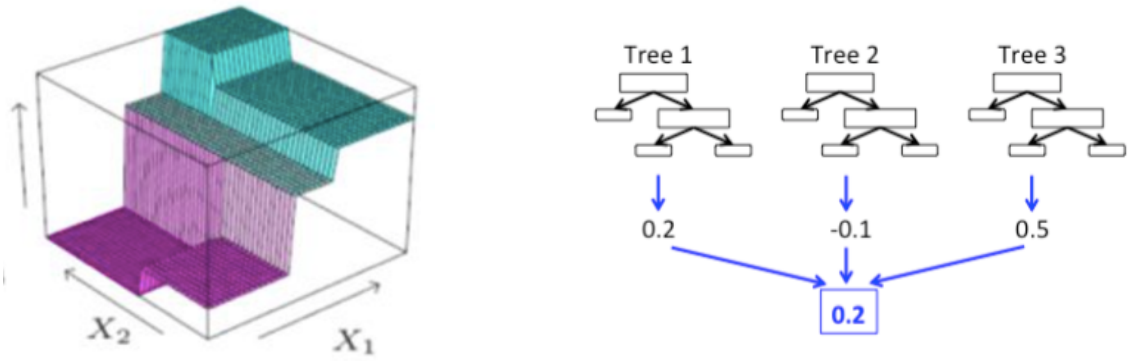

Figure 1. *Random Forest Regressor with Classification & Regression Trees*

3. Results

- **ZIG Distribution for Target Variable**

All statistical predictive modeling starts with the inspection of the target variable's distribution using the training data. As seen in Figure 2, the histogram suggests some violations from simple normality assumption of the target_commits. First, there are lots of instances with target_commits = 0, a phenomenon called "zero-inflation". Second, there are also some target_commits, after log(1+x) transformation, have small positive values such as 0.5 and 0.75. Third, the rest values follow a Gaussian distribution centered around 1.4. As mentioned in the Method section, I adopted two ways to handle this mixed distribution: the ZIG model and RFR approach.

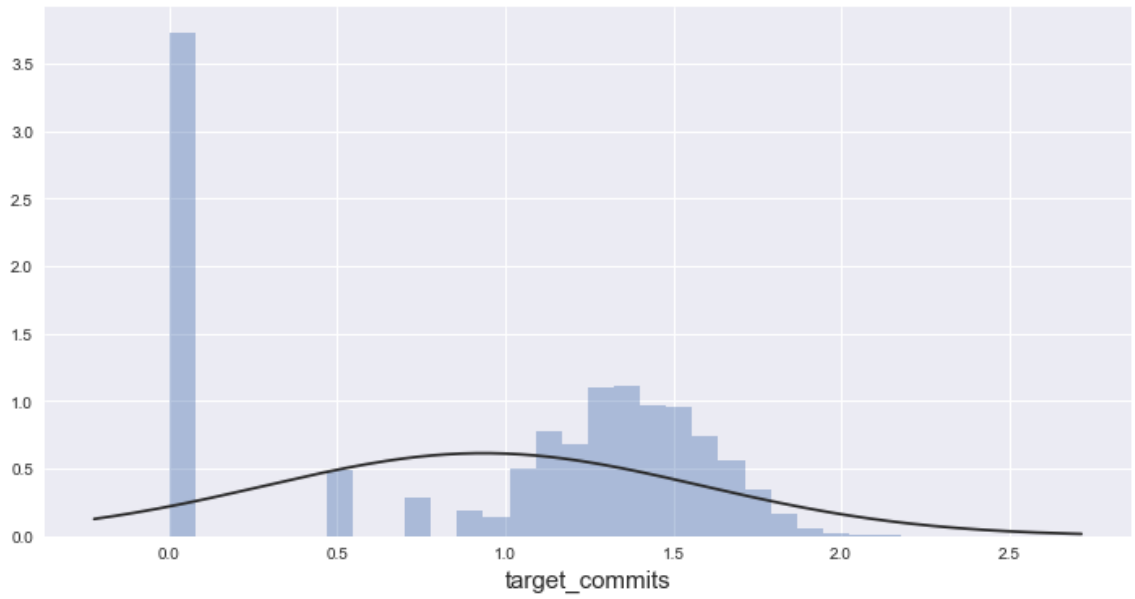

Figure 2. *Histogram of target_commits in the Training Data after log(1+x) Transform*

- **Compare Training and Testing Feature Spaces**

When applying a predictive mode built from the Training Data set to make predictions using features in the Testing Data, we need to ensure the joint distribution of (*X, Y*) do not change much between the two scenarios. Because we do not have data on *Y* in Testing Data, we only can compare them upon *X*. When X contains large number of features, I would resort to PCA (principal component analysis) to reduce the dimensionality. Figure 3 depicts that the top two principal components (PCs) for instances in the Training and Test sets. It indicates that the two feature spaces are fairly overlapped, although the testing cases look more concentrated than the training cases. Thus, it should be fine that outliers in the training data be detected and excluded from the regression analyses.

- **Exploratory Data Analysis (EDA)**

Correlational analyses showed that all the usage metrics were positively correlated with each other and with the outcome variable. This make a lot of sense for metrics like *total_commits* (total number of commits in the previous year), *total_pushes* (total number of pushes down in the previous year), and *total_issue_comments* (total comments made for the issues in the previous year). The scatter matrix for *total_commits*, *total_pushes*, and *target_commits* is shown here for illustration; see Figure 4. The definition of days related metrics, e.g., *days_since_pr* (#days since the last pull request or NULL if no request was made in the last year), does not have understandable meaning as the data reflect. I would imagine that longer *days_since_pr* would associate with smaller number of target_commits, but data suggested the other way.

Figure 3. *Scatter Plot of Top 2 Components from PCA in the Training and Testing Data*

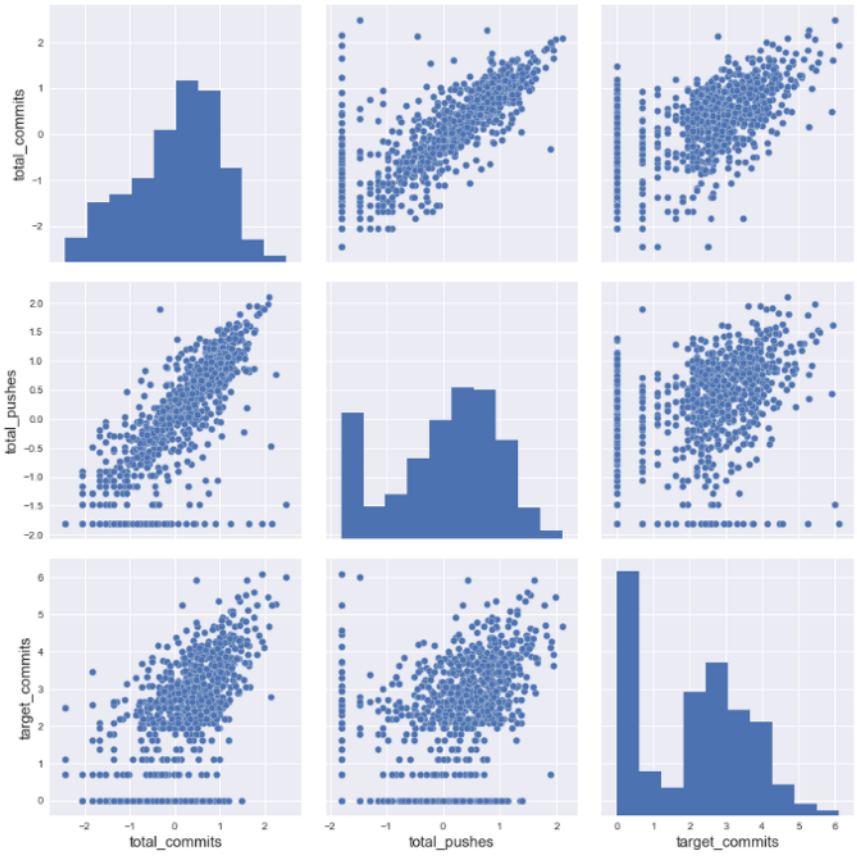

Figure 4. *Scatter Matrix of Top Correlated Variables (randomly sampled cases depicted)*

- **Feature Engineering**

Univariate distributions of the numerical features suggest: (1) logarithmic transformation, log(1+x), should be applied to all quantitative features (except six "days" related variables) and the outcome variable. For each numerical feature that contained a significant portion of zero values, I created new binary variables to indicate a user being active or inactive (no activity at all).

For each "days_since_…" metric, if the #days is larger than 330, then I created an indicator variable to mark the account's low level of activity. The "days_since_" metrics were scaled into [0, 1] interval and a power transformation was further applied to make the distribution less skewed.

Finally, I created quadratic terms for each of the numerical variables and added interaction terms between each binary variable and its root features. For example, *total_commits* was first transformed using log(1+x), then dichotomized into *bin_total_commits* (0 if no commit; 1 otherwise), and finally *total_commits_2* (quadratic term) and *bin_total_commits_int* (interaction term between *bin_total_commits* and *total_commits*) were derived.

As we know "there is no free lunch in this world", there is also no free lunch in the world of data science, either. A successful data analysis task is often based on large amount of effort on data cleaning, exploration, and feature engineering. I spent quite a lot of time in this take-home challenge project, mainly to illustrate this emhasis.

For example, I even thought the two text features, '*issue_titles*' and '*pr_titles*', might contain some predictive information. I first created two numerical features to measure the relative length of titles (count the numbers of words), and found some predictive power from them (although small), hence they were included for all the analyses. Then, TF-TDF vectorization was applied, and the derived vectors were added to the linear regression models (with L1 and L2 regularizations) and RFR, but found no value. Thus, these "bag-of-words" type of features were not used in ZIG and RFR modeling.

- **Other Data Pre-Processing and Cleaning Steps**

Missingness all happened on "days_since…" variables. The two metrics with around 50% missingness ("*days_since_inline_pr_comment*", "*days_since_commit_comment*") were excluded from analysis. The rest 4 incomplete features were imputed with median values observed on them, individually.

Normalization was applied to all quantitative features and their derived variables so that for the numerical features included into the regression modeling all had mean value around 0 and standard devision of 1.0.

- **ZIG Modeling Results**

ZIG Ridge Regressor (RR): Using instances in the training data with *target_commits* > 0, adopting a nested cross-validation step, I tried to fit linear regressors with L1 and/or L2 regularization (i.e., Lasso, Ridge, and Elastic-Net regressors). The best regressor was a Ridge Regressor (L2 regularization with tuning parameter alpha=30). It has RMSE as small as 0.82 for *target_commits* positive user cases in the Training Data. Figure 5 depicts the L2 regularized regression coefficients of the chosen ZIG Ridge Regressor. It indicates that the terms related to total_commits have dominant impact to the level of target_commits (if it is larger than zero). Residual analyses shown by plots in figure 6 confirmed the acceptance of this ZIG model component. There were some difficulty for fitting a regressor that could satisfactorily predict the observed target_commits with transformed level close to zero.

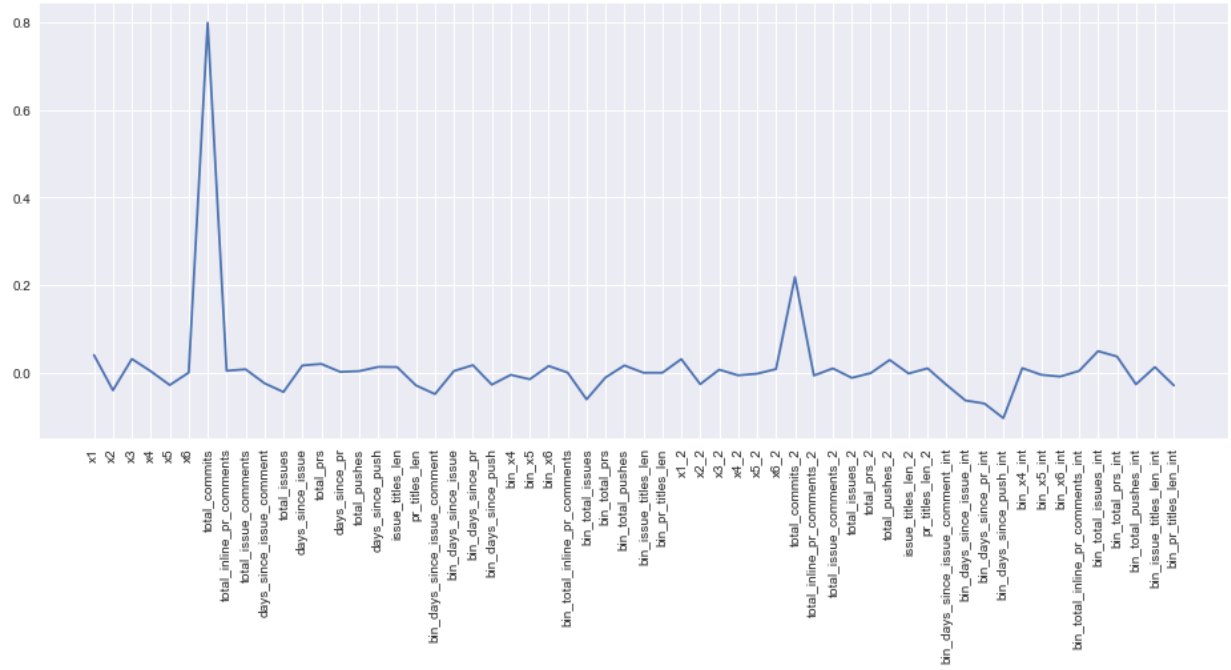

Figure 5. *ZIG Ridge Regressor Coefficients*

ZIG Random Forest Classifier (RFC): By dividing all the users in the Training Data into two parts: those with positive *target_commits* (*target_positive* = 1) and those with zero *target_commits* (*target_positive* = 0); *'target_positive'* plays the counterpart of Zi as defined in 2.1 for the ZIG model. 29% of users had *target_positive* =0, not close to a unbalanced classifier problem. A standard logistic regression model (as defined in section 2.1 for Zi) was first fit but found the accuracy was only around 65%. Thus, I resorted to a RFC. With out much tuning effort, I fitted a RFC model that provided 83% prediction accuracy. Variable importance plot in Figure 7 suggests that there were quite some informative activity metrics (e.g., total_commits, total_pushes, and

total_issue_comments) from which one could use to predict whether a user would be active or not in the dichotomous sense in the month following the year.

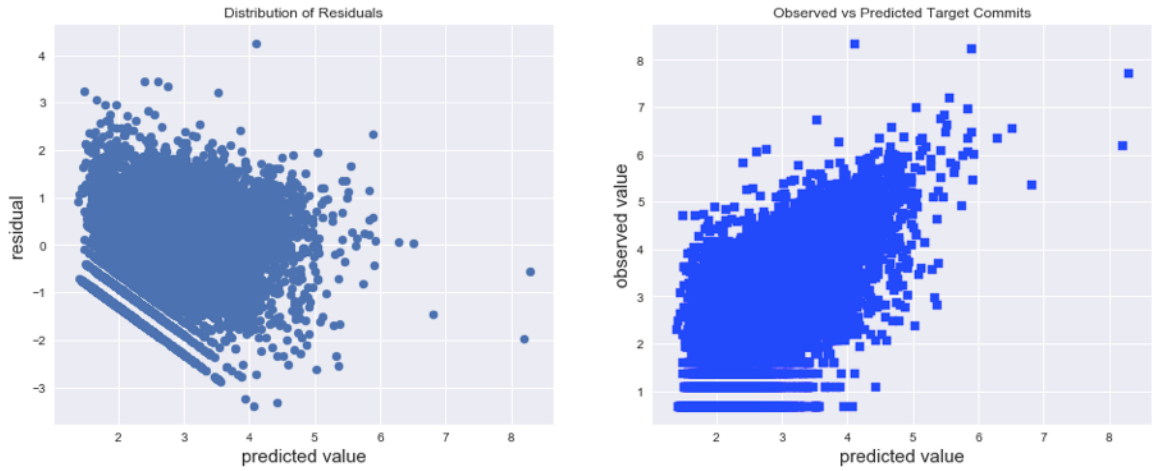

Figure 6. *Residual and Predicted Values from the ZIG Ridge Regressor*

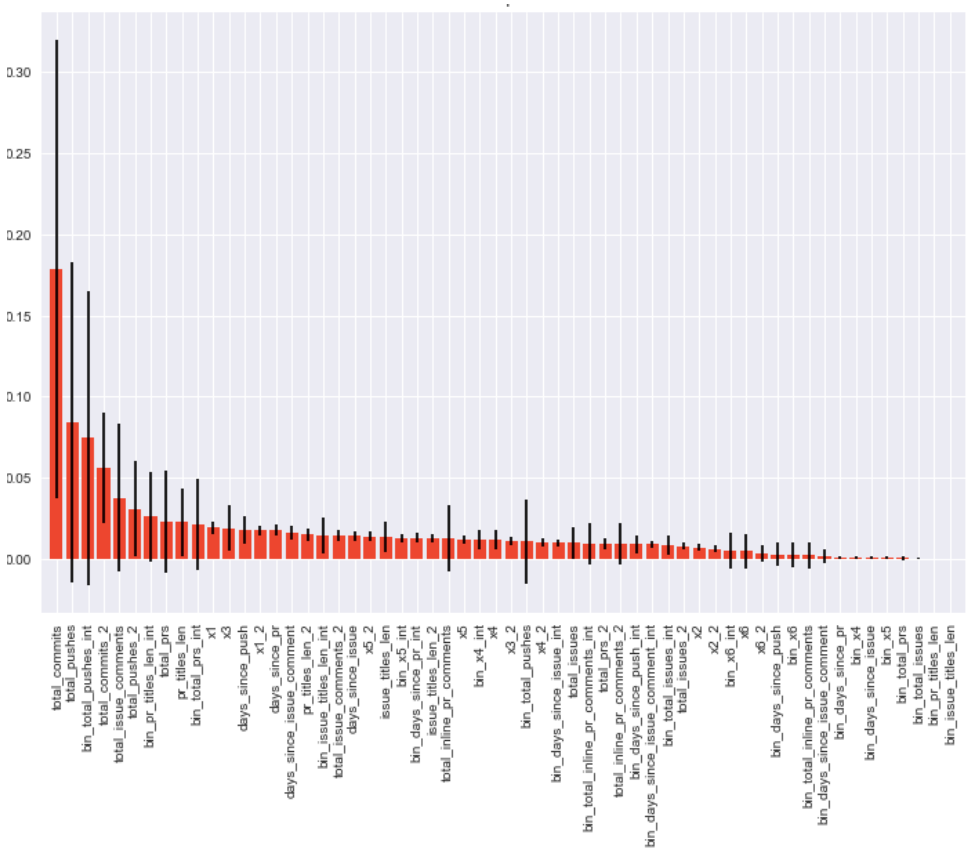

Figure 7. *Variable Importance from the ZIG Random Forest Classifier*

Productivity of ZIG (RR+RFC): When the ZIG model applied to the Validation set, I first used RFC to classify instances to *target_positive*=1 vs *target_positive*=0 groups. Then, in the positive group, I applied RR to predict the absolute value of *target_commits* in the algorithmic scale. In other words, I used zero to represent those users who have a high chance to be inactive in the next month given on previous-year's activity metrics. Such a model ended up RMSE estimate at 1.16.

- **RFR Modeling Results**

My experience as a data scientist told me that random forest approach would always be a great candidate in practical applications. Hence, I finally fitted a RFR model to the data (after all the above feature engineering had been conducted). This approach ended up with an RMSE estimation of 1.07, much smaller than 1.16 of the above ZIG model. Therefore, for the purpose of obtaining a potential better result in the final submission, I applied the RFR to predict *target_commits* for the 5000 test cases. Figure 8 depicts the relative importance of the features in predicting *target_commits*, which clearly suggests that *total_commits* and its quadratic term were dominantly strong predictors. Compare to Figure 7, you can see that activity metrics play different roles when the task is different (classification versus regression).

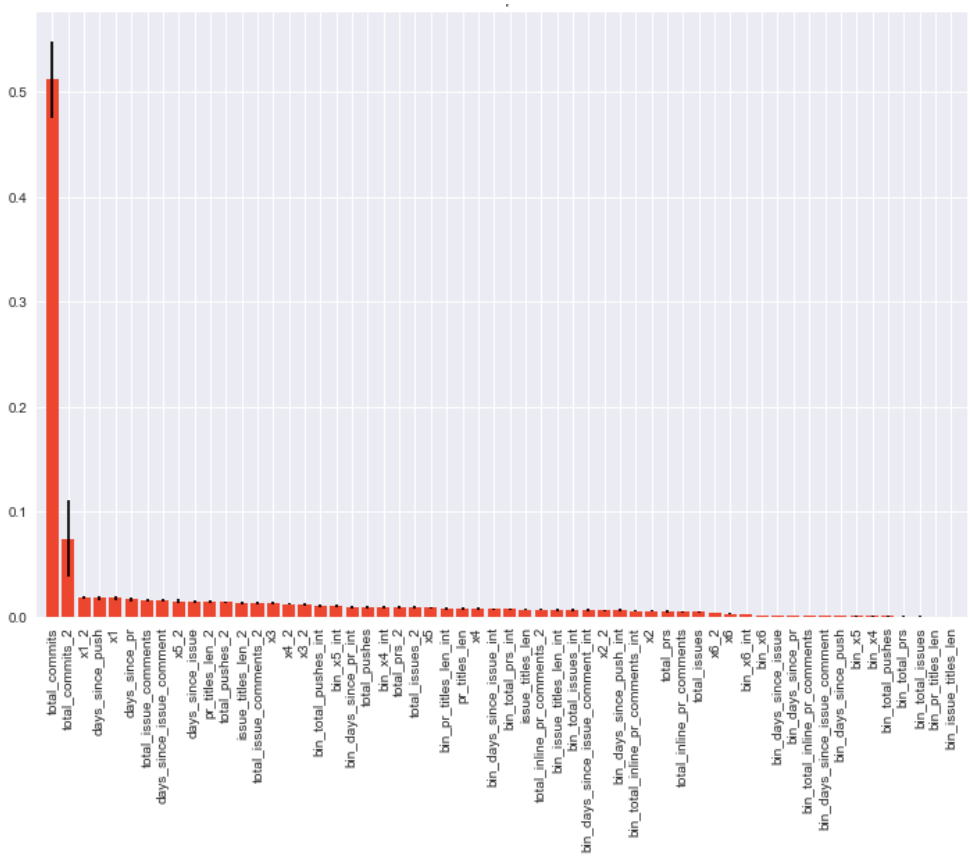

Figure 8. *Variable Importance from the fitted Random Forest Regressor*

4. Discussion

In this report, I summarized two primary methods used for tacking the assigned task. The ZIG model demonstrates how I would approach the problem and the analytical steps that I would follow as a statistical. Although it may not provide the best performance, it definitely provides a good foundation.

Given the limited time, I only illustrated how feature engineering should be done. It was not done completely. If I had more time, I would consider creating more cross-metric interaction terms and even investigate whether nonlinear kernel regressors should be applied. Regressors such as SVM (support vector machine) and MLP (multi-layer perceptron) would be ideal choices to try for this problem. But I believe the key for success lies more in feature engineering, rather than finding a sophisticate form of modeling.

The data sets (*train_data.tsv* and *test_data.tsv*) have some inconsistencies from the column definitions provided by the *data_overview.pdf* file. I spent quite a long time to figure out problems such as which column is "*is_org_member*". By definition, it is binary (0/1), but there was no column that satisfied this assumption. Similarly, I found that the "user_age_days" had a lot of "huge" values that are impossible in reality. It took me quite a while to realize that 'huid' actually is the line record. Anyway, without a colleague around to clarify things like these, I chose to use "X1-X6" to label some variables requiring further investigation.

Using a year's summary to predict the future activity in the following month is a "decomposition task". When all the 12 months activities were aggregated into summary metrics, we lost important information in the time domain such as seasonality. If monthly broken data had been given, we would have stronger signals to make predictions.

If the goal is solely on making predictions as accurate as possible, then I will definitely try to spend more time finding an ensemble method. Gradient Boosting Trees and XGBoost (extreme gradient boosting) would be my next trial.