Google Analytics Revenue Prediction

Dave Palazzo and Samantha Werdel
https://github.com/davepalazzo/Google_Analytics_Revenue_Prediction

Summary

In this report, we attempt to predict the log total sum of transactions per users from the online Google Merchandise Store. Given a dataset that contains information about each visit to the online store from August 1, 2016 to August 12, 2016, we use extreme gradient boosting (XGB) on these explanatory variables to fit our model. As a result, we found XGB to be a robust regression model in predicting our response variable. The residual errors were measured by root-mean-square error (RMSE), and the final model had a RMSE of 2.07. The most significant features of the XGB model in predicting the log sum of transactions were pageviews and time on site. These features are reasonable because the number of pages viewed on a website and the time spent would be a good indicator to determine if a transaction was made. In order to verify the model produced reasonable features a chi-squared test of independence was performed where we found evidence to reject the null hypothesis that these features were independent from total transactions.

For an in-depth look at the code, how we obtained our dataset and the analysis, refer to our GitHub repository, Google Analytics Revenue Prediction, linked above.

Introduction

With the recent rise in electronic commerce, along with our growing computational resources for storing vast amounts of data, companies are incentivized to collect data from traffic to their websites. This information can be used for many applications to increase revenue, from tracking what times of the day are most popular to what path landed the customer on the specific website. In our report, we focus on the Google Merchandise Store, GStore, an online store that sells Google "swag". The dataset used came from a past Kaggle competition, "Google Analytics Customer Revenue Prediction." In this competition the goal was to predict how much the Gstore would make given their customer time-series dataset. The competition gave a large train and test dataset where each row signified one visit to the store. The data fields in each column provided the specifics of each visit to the store; a unique identifier for each visitor, the date, what browser the visitor was using, all the pages visited, the geography of the user. A full list of the data fields and their explanations can be found on the Kaggle competition's website, https://www.kaggle.com/c/ga-customer-revenue-prediction/data.

Based on the size of the Kaggle dataset, for the purpose of this report, we will focus our analysis on a subset from the Kaggle dataset. We sorted this dataset by date and sliced the first 27,000 rows to construct our dataset, giving us a date range from August 1, 2016 to August 12, 2016. In contrast, the date range from the Kaggle competition was August 1, 2016 to October 15, 2018. Before any exploratory analysis was done, some data fields contained sub-categories, which were expanded out into separate variables. Our dataset consisted of 27,000 observations and 56 variables. The dataset was split into train and validate data, approximately 80-20. The train contained the days from August 1, 2016 to August 9, 2016, and the validate data included the remaining days.

Exploratory Data Analysis

Since we expanded the data fields that had sub-categories into separate variables, this resulted in a high proportion of missing data. We'll address the missing data that is prominent throughout the dataset. Figure 1 shows the percent missing by column in our dataset, and only the variables that contained missing values. There are 17 columns which all values were missing, and some others that contain sparse values throughout. We can safely drop the columns full of only NA values.

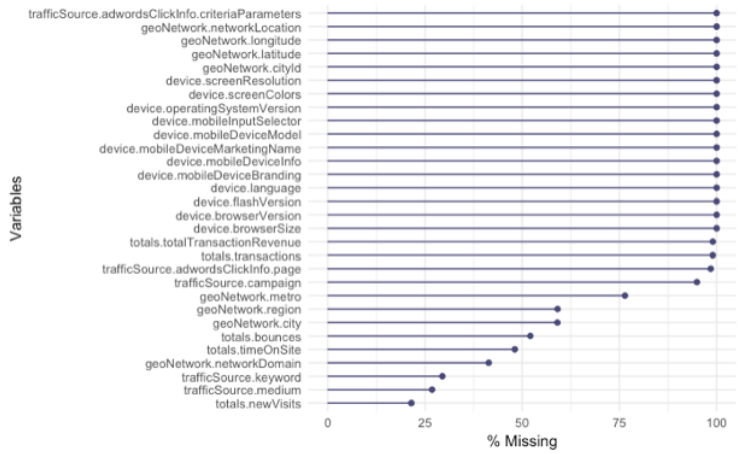

Figure 1. Percentage of Missing Data

Once we accounted for and removed variables with 100% missing, we were left with 32 explanatory variables. From these variables, we are predicting log of revenue transactions. In the Kaggle dataset, revenue transaction is multiplied by 10^6, for the purposes of data exploration in the graph below, we will transform this back to the original dollar value.

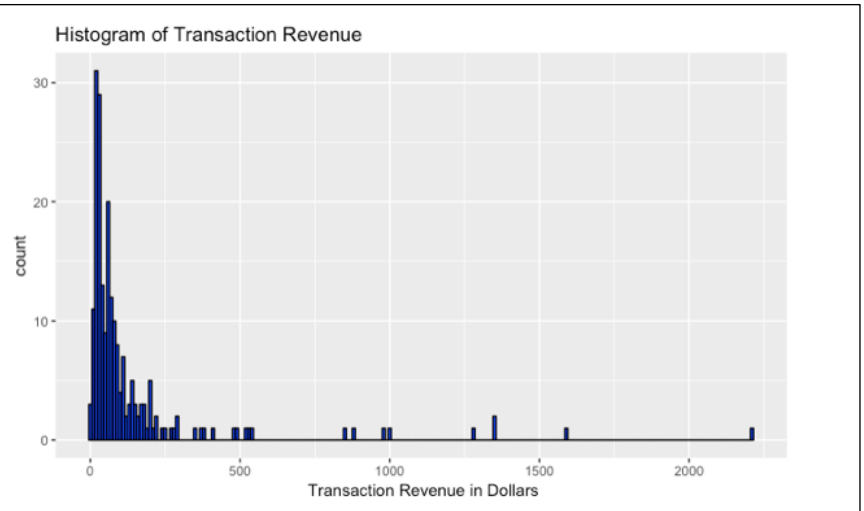

Figure 2. Count of each Transaction (in Dollars)
Summary of Statistics from Graph

```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
1.20   27.18   60.45  139.46  115.85 2211.38
```

The graph in Figure 2 accompanied by the summary statistics, shows all transformed y values. We can see the most frequent transactions are for values below $100. However, we can see a long tail that stretches to a maximum value of $2,000. Additionally, we can note the effect of these larger values on the overall mean by comparing the median value of $60.45 to the mean of $139.46. These numbers make sense given the majority of the items for sale at the Google store are low dollar value items.

The objective of the Kaggle competition prediction is to predict the log of transaction revenue. In Figure 3, we have a density plot of the log transformation on transaction revenue.

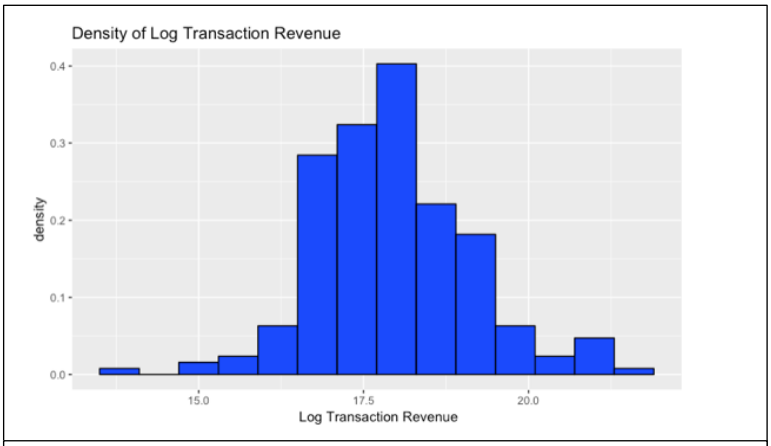

Figure 3. Histogram of Log Transaction Revenue

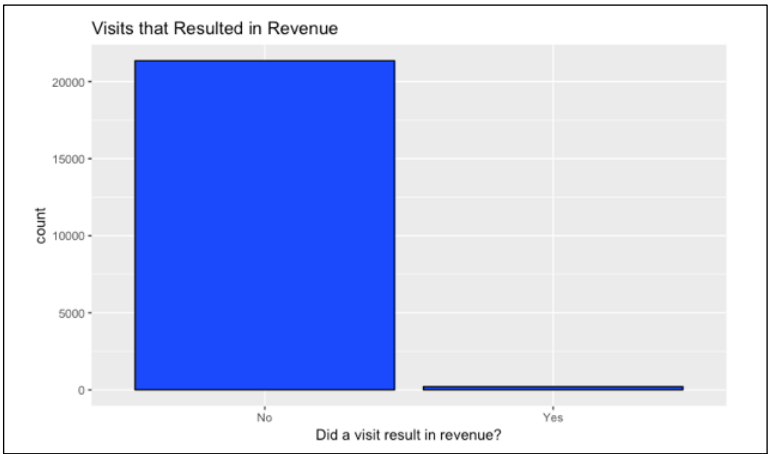

Figure 4. Visits that Resulted in Revenue

This transformation has normalized the distribution. We will use this transformed variable for our analysis. It is also worth looking at the level of balance in the dataset. As our data consists of visits to the Google Store website, we would expect a fairly low number of visits to result in a purchase.

The bar plot in Figure 4, shows that as expected only about 1% of visits to the site result in revenue for the store. This graphic shows the challenge that our model will face predicting an event of such low occurrence.

Method: Gradient Boosting

The model we used to predict the natural log sum of all transactions per user from the explanatory variables in our analysis is gradient boosting, more specifically extreme gradient boosting. Boosting are machine learning algorithms which take weak classifiers and fit them to become strong predictors. Gradient boosting is used to implement supervised learning; which attempts to learn a function that maps input values to output values based on input/output pairs and labeled training examples. In the equation below, F represents the final model generated with gradient boosting.

$$\hat{y} = F(x)$$

To achieve that final model, gradient boosting uses a sequential ensemble technique. Each subsequent model is constructed off the previous model. The first model is a simple weak classifier. The successive models build from the previous models and attempts to correct its residual error. This residual error is calculated by a loss function. In the equation below, m is a representation of a stage in the gradient boosting progression and h represents the residual error given by the user-specified loss function.

$$F_{m+1}(x) = F_m(x) + h_m(x)$$

In gradient boosting the loss function is subjective as long as it minimizes the gradient of error between the predicted y and actual y value. Our analysis uses the root-mean-square error. RMSE squares the errors before taking the average and then takes the overall square root. This penalizes larger errors more by giving them a greater weight.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

The function below shows the process at one step, using the minimized error to estimate the next model. The sum of the residual errors from the loss function is added to the previous model. This boosting continues until the residual errors have not improved, giving the final prediction model.

$$F_{m+1}(x) = F_m(x) + argmin_h \sum_{i=1}^{n} L\big(y_i, F_m(x_i) + h(x_i)\big)$$

In our report, we used an optimized gradient boosting model, extreme gradient boosting (XGB), to predict the log sum of all transactions per user. XGB is a robust regression that has hyperparameters to create a fast and efficient model, without overfitting. The hyperparameters most effective to help with overfitting are nrounds and early_stopping_rounds. The nrounds parameter is the maximum number of models to generate when fitting the predicted y. The early_stopping_rounds number represents the number at which to stop if the model has not improved, which avoids the overfitting of the model.

Model Selection and Parameter Tuning

The gradient boosting algorithm that we will explore is a non-parametric method, and as such makes no assumptions of the distribution of the data. In order to prepare the data for the model however, we will convert all categorical variables into integers. While gradient boosting does not infer parameters based on the data, there are a number of parameters which need to be manually set to run the algorithm. In this next section, we will explore some of these parameters and their effect on overfitting a model.

To accomplish this, we will run a number of experiments on our train and validation datasets. We will iterate over optional values for a given parameter, keeping all other parameters constant, to assess the individual impact of different values. We'll assess the effect of eta, max_depth, and gamma on the RMSE of the training and validation dataset.

The first parameter that we'll assess is eta, or the learning parameter. This parameter is sometimes referred to as the shrinkage parameter, it shrinks the level of contributions from each tree. In other words, it controls how much information from the latest iteration is included in the updated model. Float values between 0 and 1 are valid inputs. We can see from the plots in Figure 5, that as the parameter increases towards one, the training set error decreases at a faster rate. However, when we look at the performance on the validation set, we can see that the higher values of 0.5 and 1 perform poorly. This is a clear sign that using values of 1 and 0.5 severely overfit the training data.

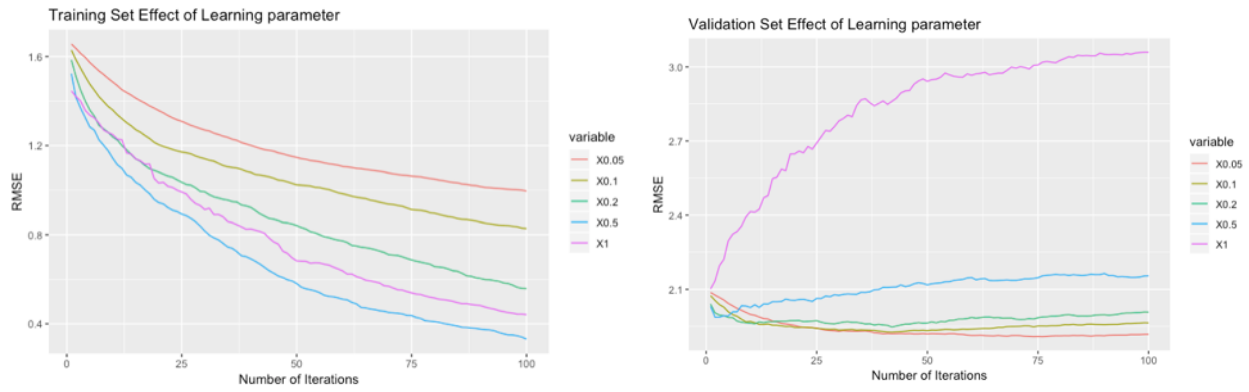

Figure 5. Parameter tuning on the Learning Parameter (ETA)

We've run the same experiment below, this time on the max_depth parameter. This parameter controls the maximum depth of the trees. Given the training dataset, we can clearly see in Figure 6, that the depth of the tree has a significant impact on the rate of decrease in the RMSE. Greater depth, results in better RMSE scores. However, when we compare this to how the model performs on unseen data, the plot changes. Depths of 2 and 10 perform poorly, compared to the depths of 4 and 6. This shows evidence of an overfit model at a depth of 10 and an underfit model at a depth of 2.

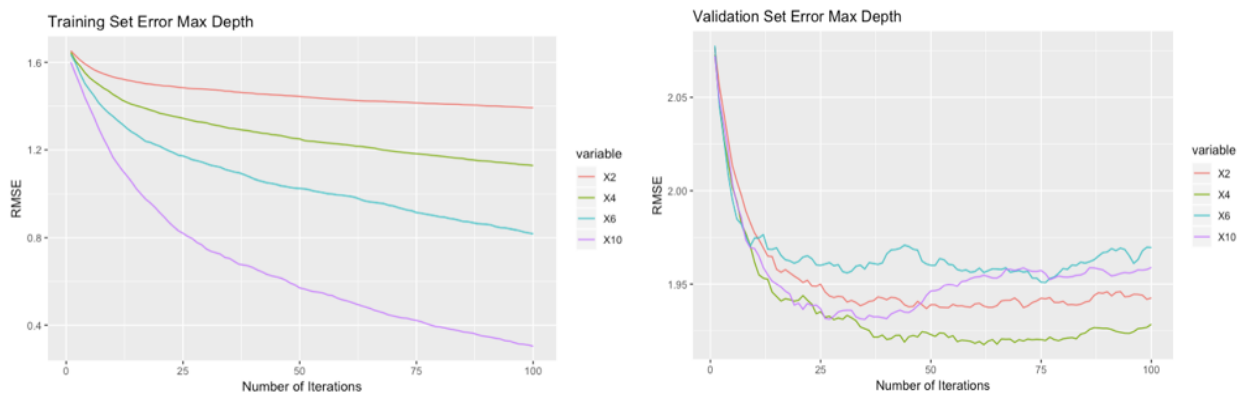

Figure 6. Parameter tuning on Max_depth Parameter

The last parameter that we will assess is the gamma value. From the plot in Figure 7 below, we see much less variation in the training dataset with the exception of the extreme value of 100. Values 0.1 through 10 fit similarly on the training data. Looking at the performance on the validation set, again, we do not see as much variance in the performance as we have seen with eta and max_depth. Gamma values of 100 and 1 appear to perform the best.

.

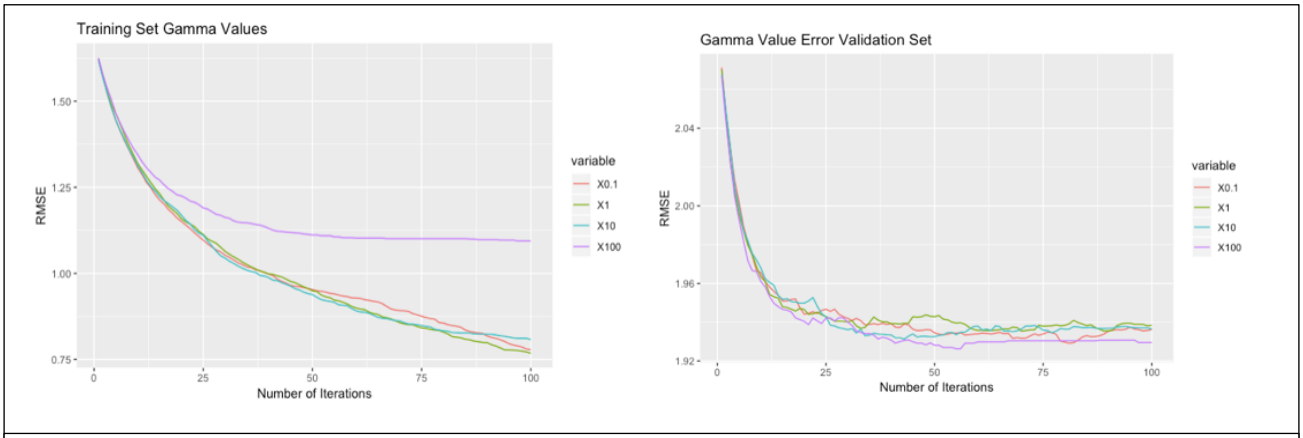

Figure 7. Parameter tuning on Gamma Parameter

In order to tune all of our hyperparameters we will use the library mlr to validate the performance of the parameters on unseen data. Our best result on our validation set was achieved with the following parameters:

> objective=reg:linear, nrounds=5, eta=0.372, booster=gbtree, max_depth=3, min_child_weight=4.4, subsample=0.801, colsample_bytree=0.958

At this point we are ready to run the model on the full training dataset with the parameters above.

Model Evaluation

After training our model on the training dataset and testing the performance on the validate dataset, we found our model produced a RMSE of 2.07. Based on this result, we would have expected to finish around 600th place in the Kaggle competition. Given that our dataset did not include the full visits data, we might expect to improve this result by training our model on the full dataset. Moreover, our RMSE on our training dataset is a 1.58, this is over 30% lower than what was achieved on the validation set. It is likely that we have overfit our data using the k-fold cross validation for hyperparameter tuning. A better approach may have been to use a train, validate, test split in the model selection phase.

Even with these shortcomings, we can still gain valuable insights into our dataset. From the model we can generate an importance matrix, which is a data table with scores for each attribute in the following metrics; gain, cover, and frequency. The values for these metrics are all relative values (i.e. sum across all features is equal to 1). The gain metric is the relative contribution to each tree in the model. Based off of

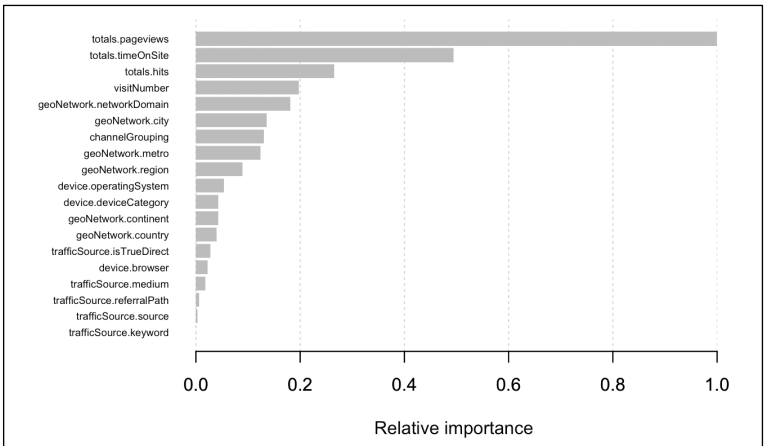

Figure 8. Importance of Features from the Model

this metric alone we see that page views, hits and total time on site achieve the best scores. The full results of this metric are plotted in Figure 8.

We can see for the alternative metrics, cover and frequency, the top three variables remain the same across each metric. Cover is the relative number of observations related to a feature, whereas frequency is the percentage representing the number of times a particular feature occurs in the trees of the model.

The results seem to make intuitive sense, as you would expect an increase in the number of page views, hits and total time on site to increase the probability of a transaction taking place. Nonetheless, we can verify that these are statistically significant attributes by performing a chi-squared test of independence with the attribute and our response variable. We performed a chi-square test on the top three attributes in the importance matrix and one rated of low importance. Our results confirmed that our model identified highly correlated attributes with transaction revenue. The three important features produced highly significant p-values, where the low importance feature, traffic source keyword, produced a p-value equal to 1.

Conclusion

Extreme gradient boosting proved to be an effective method for modeling transaction revenue from visits to the Google Merchandise online store. While the model produced a satisfactory RMSE score, additional care should be taken in hyperparameter tuning, which can easily lead to overfitting these types of models.

Works Cited

1. A Gentle Introduction to XGBoost for Applied Machine Learning. (2016, September 21). Retrieved from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

2. Chi Square test for feature selection. (2018, July 05). Retrieved from http://www.learn4master.com/machine-learning/chi-square-test-for-feature-selection

3. Google Analytics Customer Revenue Prediction. (n.d.). Retrieved from https://www.kaggle.com/c/ga-customer-revenue-prediction/overview

4. Gradient boosting. (2019, May 30). Retrieved from https://en.wikipedia.org/wiki/Gradient_boosting

5. Hastie, T., Friedman, J., & Tisbshirani, R. (2017). *The Elements of statistical learning: Data mining, inference, and prediction*. New York: Springer.

6. Root-mean-square deviation. (2019, April 19). Retrieved from https://en.wikipedia.org/wiki/Root-mean-square_deviation

7. Srivastava, T. (2017, May 04). How to use XGBoost algorithm in R in easy steps. Retrieved from https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/

8. Tseng, G., & Tseng, G. (2018, April 13). Gradient Boosting and XGBoost. Retrieved from https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5