MSBA 6420 - PROJECT REPORT

BY:

Michael Thompson, Ashwin Sharma, Sameeksha Aithal, Fang Huang, Arnav Sivaram, Mohammed Saad Ahmad

Table of Contents

Project Definition & Introduction

Background & Context

The Google G-Store is a one-stop shop for all Google branded apparel and swag ranging from hats and shirts to notebooks and pens. The store is a place for highly loyal customers, employees, and universities to purchase gear. The G-Store serves as a point for Google to evangelize their brand and build loyal customers across all their offerings. G-Store is an online store where every website customer visit is tracked and monitored by a multitude of metrics from behavior, demographics, and contextual points. Understanding how customers visit their website and purchase goods is important for G-Store operations ranging from understanding how to increase sales, better manage inventory, and increased brand loyalty.

Problem Statement

The goal of this project is to predict *total transaction revenue* of customers from May 1st 2018 to October 15th 2018 in the Kaggle challenge. More broadly the goal is to predict any amount of future transaction values by customers who visit the G-Store. Predicting the future serves management teams to increase the customer experience, product availability, and chance of customer purchases. In addition, it can help marketing teams to understand what behaviors of customers are highly correlated with transactions to better conduct target marketing for G-Store products.

Technical Specifications & Data Sources

Software & Tools

Most analysis was performed in python. R was used in data exploration for determining irrelevant features and feature structure. Python was used in expanding our data set into a tabular format for friendly analysis model creation, and evaluation. As we utilized neural networks we opted to run our models using Google Colab which allows for GPU acceleration to reduce the time necessary for training a model on a large dataset as presented with this challenge.

R Version 3.5.3	Python Version 3.7
• Data Explorer • Dplyr • Tidyr	• Pandas • Numpy • Keras • Sklearn & more (Scipy, Matplotlib, Datetime, JSON, etc.)

Exhibit 1: Software Versions

Data Sources

The data is large with over 2.1 million records and 13 complex columns which when expanded equate to over 150+ fields. The data also includes a range of features types including behavioral, device, and session/contextual information about the customer upon their online visit at Google G-Store. A list of some of the most impactful features of each type can be seen in Exhibit 2.

1. **train_v2:** Contains an updated dataset found in Kaggle with customer visit/transaction data from August 1st 2016 to April 30th 2018.
2. **test_v2:** Contains an updated dataset found in Kaggle with customer visit/transaction independent features from May 1st 2018 to October 15th 2018.

Behavioral Attributes	Device Attributes	Session/Context Attributes
• Advertisement Clicks • # of Page Views • Date & Time • Frequency of Visits • # of Pages Clicked • & more	• Browser • Operating System • Device Model • Device Language • Browser Version • & more	• Country, State & City Location • Visitor Start Time • Visit Number • Latitude/Longitude • Visitor ID & more

Exhibit 2: Features

Overview of Approach

We started with understanding the features and their distributions. After understanding the nature of the features and the required prediction, we chose two modelling techniques Light GBM and Neural Network to model the prediction. In order to further increase the accuracy, we applied up-sampling techniques and re-ran the model. Finally, we compared across both the modelling techniques and reported the performance.

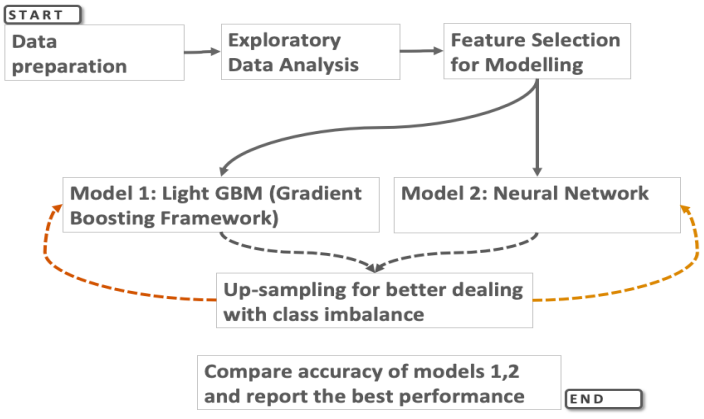


Exhibit 3: Approach Map

Data Preparation

Data Expansion

The first problem presented within the data deals with the data' semi-structured format. As this data was an automated process of collecting visitor characteristics by each visit it is common for data capturing systems to be optimized for speed of writing due to the large volume and speed of data. Therefore, many records include inconsistent data and are tied up in complicated data objects (JSON strings). As seen in Exhibit 4, the original 13 features are displayed in their original format with examples.

Feature Format	Features in This Format	Example
String Type	channelGrouping, date, fullVisitorId, socialEngagementType, visitId, visitNumber, & visitStartTime	Ex. "Search" or "1"
JSON Object	device, geoNetwork, totals, trafficType	Ex. {"browser": "firefox, …}
List of JSON Like Objects	customDimensions, hits	Ex. [{'hitNumber': '1'...}, {'hitNumber':'2'...}, …]

Exhibit 4: Original Features

The primary challenge with the data in the provided state was in transforming the list of JSON 'like' objects. With the JSON fields it was fairly easy to expand the fields using the python JSON package, but because the 'hits' and 'customDimensions' fields were not proper JSON objects we had to write a custom function called "list_json_converter". With the 'customDimensions' being easier to transform all we had to do was to replace apostrophes with quotes and grab the one object from the list to make a proper JSON object. We then used the JSON package to expand.

However, the hits field was a lot more difficult to expand. First, the number of hits (webpage visits) are in a list which is unequal for each customer visit. This is logical as not all customers visit the same number of webpages when visiting the Google G-Store. We wanted to keep this field as we believed information about how many webpages (including product pages) the customer visited could be predictive of whether and how much they would pay in total. Second, the number of features in each JSON object was inconsistent. In order to fully convert this JSON object would take significant effort. We instead decided to scrape information on the first and last webpage they visited, hoping they would capture as much relevant information as we needed. This allowed us to maintain the frequency, average amount of time spent on pages, and time that they were visiting.

Feature Engineering

Before choosing the features set most useful for our final predictions, we had to obtain the fields of interest. These challenges are important to understand if you are to replicate this analysis on future or bigger data.We

first expanded the composite features into unique features and flattened all of the JSON fields. Much of important information was tied up in complex fields called JSON objects. Flattening of the complex JSON fields such as Hits and Total resulted in a spike in the number of columns. These columns had to be assessed to determine the most important features. We utilized an R programming package called Data Explorer to help identify features with missing values, to understand how normally data is distributed, and see what features may be correlated with another. This package builds exploratory reports that guides easier data analysis on each column to help us understand the features better. The result of the data exploration can be seen in Exhibit 5.

```
library(DataExplorer)
tr <- read_csv("C:/Users/sharm702/Downloads/Ashwin/train_v2.csv", col_types = ctypes)
test_data_report <- create_report(tr)
```

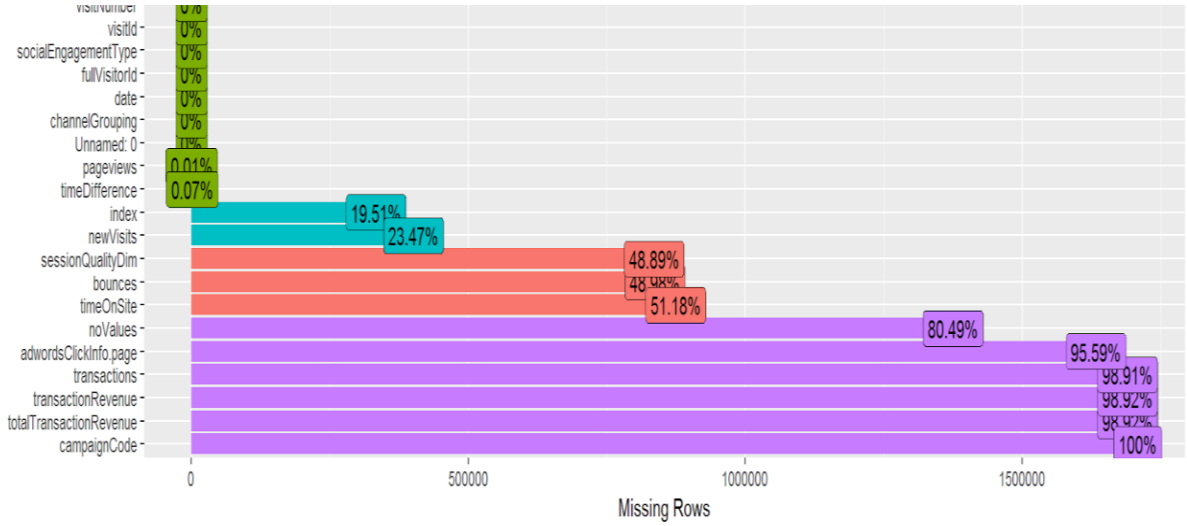


Exhibit 5: Data Analysis

We leveraged the results of data explorer and performed feature elimination. We removed most of the features which have no impact on prediction (Ex. visitor id). Some numeric features including our predictor variable (totalTransactionAmt) consisted of a lot of missing values. However, neither the features nor the records corresponding to missing values could be eliminated from our dataset as this would shrink the dataset by a great amount. Therefore, we decided to replace missing values with 0 for these features. As approximately 80% of the features were categorical (non-numeric), we maintained fields with low levels of missing values by placing missing values as "Other" where appropriate. These categorical columns were then One-Hot-Encoded and then aggregated using sum to get the records from transaction level to customer level. In total, we took 2.1 million visits to 1.6 million unique customers and their average behavior.

Predictive Modelling

Estimating Future Transaction Revenue

General Approach

Given the data we have is highly unbalanced with over 99% of records in our data being customer who did not transact we had to adjust our modeling approach from standard single model approaches. We decided to also handle the imbalance of the data by splitting our prediction into two tasks. First, we developed a model to classify the probability of a customer purchasing an item or not from the G-Store. Second, we used those customer visits which were predicted to estimate the total transaction revenue via a regression model.

Model 1: LightGBM (A Tree Based Algorithm)

Our first model we chose is a natural fit for handling imbalanced data. We decided to use a LightGBM because it is based on a tree architecture which benefits from natural feature importance selection,. The model is also relatively fast to train and has the ability to account for imbalance in the data.

In both of the two stages, we used randomized search for hyperparameter tuning. The optimal hyper-parameters for the classification model were 50 leaves and a maximum depth of 8 layers. As seen on the right, the R2 score (explainability) starts to diverge around 50 leaves indicating a lesser rate of information gain. The model resulted with an AUC score of 0.9895. For the regression model, the optimal hyper-parameter settings were 30 leaves with a maximum depth of 8, which gave us an RMSE of 0.3114.

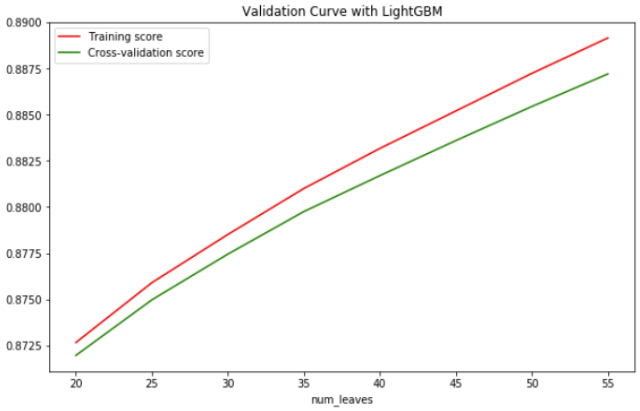


Exhibit 6: Validation curve for LGBM

```
pr_lgb_sum = 0
for i in range(10):
  print("Iteration number ", i)
  param_clf = {"objective" : "binary",
               "max_bin" : 256,
               "learning_rate" : 0.01,
               "num_leaves" : 50,
               "bagging_fraction" : 0.9,
               "feature_fraction" : 0.8,
               "min_data" : 1,
               "bagging_freq" : 1,
               "metric" : "binary_logloss",
               "bagging_seed": 13+i,
               "feature_fraction_seed": 42+i}
  param_reg = {"objective" : "regression",
               "max_bin" : 256,
               "learning_rate" : 0.01,
               "num_leaves" : 30,
               "bagging_fraction" : 0.9,
               "feature_fraction" : 0.8,
               "min_data" : 1,
               "bagging_freq" : 1,
               "metric" : "rmse",
               "bagging_seed": 13+i,
               "feature_fraction_seed": 42+i}
  model_clf = lgb.train(param_clf, lgtrain_clf)
  pr_clf = model_clf.predict(X_test, num_iteration=model_clf.best_iteration)
  model_reg = lgb.train(param_reg, lgtrain_reg)
  pr_reg = model_reg.predict(X_test, num_iteration=model_reg.best_iteration)
  pr_lgb_sum = pr_lgb_sum + pr_reg*pr_clf

pr_final = pr_lgb_sum/10
```

Exhibit 7: Hyper-Parameter tuning in LGBM

Feature Importance

The primary benefit of using an LGBM (Random Forest) model is interpreting what features have the most predictive value. LGBM consists of a number of decision trees. Every node in the decision tree is conditional on a single feature, and it is designed to split the dataset into two parts so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity.

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important is the feature.

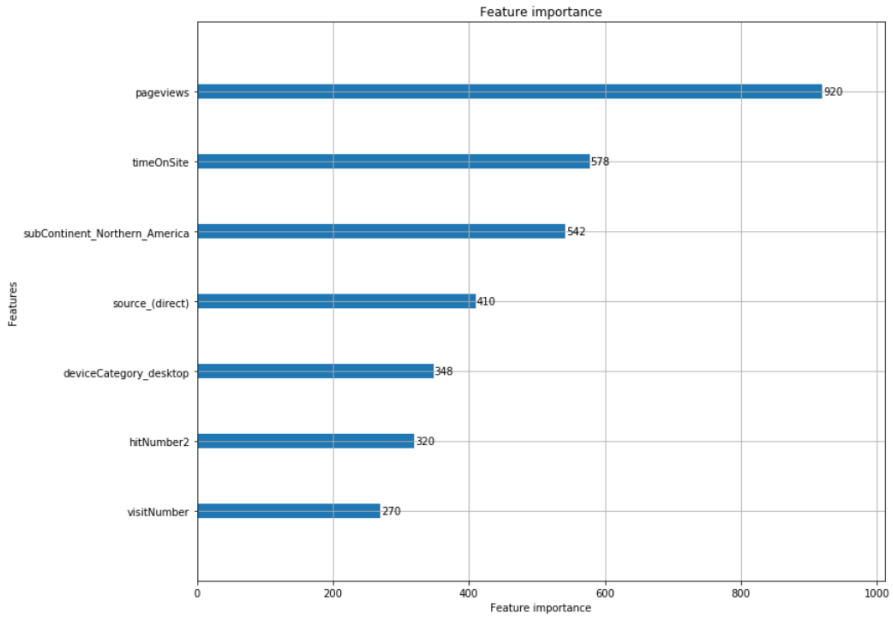


Exhibit 8: Feature Importance with the LGBM Model

The metrics which are indicators of how much time a customer has spent on the website and how active his session is seem to be very strong predictors of a likelihood to purchase. This is intuitive, since a serious buyer may want to assess the product options, read about it, and spend time in consciously deliberating his decision.

Interestingly, it seems like customers who own a Mac are more likely to purchase. In terms of a session original, customers who organically reach a website are more likely to make a purchase, compared to those who come through search or social media advertising.

Model 2: Neural Network

```
# Defining the architecture
NN_Regressor = Sequential()

NN_Regressor.add(Dense(500, kernel_initializer='normal',\
                       input_dim = X_reg_train.shape[1], activation='relu'))

# Adding the Intermediate Hidden Layers :
NN_Regressor.add(Dense(500, kernel_initializer='normal',activation='relu'))
NN_Regressor.add(Dense(500, kernel_initializer='normal',activation='relu'))

# Adding the Output Layer :
NN_Regressor.add(Dense(1, kernel_initializer='normal',activation='linear'))

# Compile the network :
NN_Regressor.compile(optimizer='adam',loss='mse', \
                     metrics=['mean_squared_error', 'mean_absolute_error', \
                              'mean_absolute_percentage_error', 'cosine_proximity'])
NN_Regressor.summary()
```

The next model we chose was a deep neural network because of its natural ability to conduct both feature selection and its renowned ability to learn even the most difficult of prediction problems. A neural network works much like a brain having each node/cell activate upon a stimulus. Just like the LGBM our neural network was made in two parts: classification and regression model. Structurally each model was similar, both having two hidden layers with 500 neurons per hidden layer. ReLu was used as activation functions through the hidden layers for both models. ReLu was an

obvious choice considering it is widely viewed as an industry standard for deep learning practices. However, the output activation function for the class probability model was sigmoid (binary) for our classification model trained on optimizing by binary cross-entropy. Our regression neural network output via a linear function with being optimized by mean squared error MSE. Lastly, we chose ADAM as our optimizer as it is computationally efficient, and can handle sparse gradients on noisy problems such as our data.

Our regression neural network was trained on all customers that have made a transaction in the past. Our final revenue prediction of customers was only done for customers who had a predicted purchase probability greater than 0.5 from our classification model. We trained each model on 50 epochs and we saw the model performed well on both the training and validation set as seen in Exhibit 10

. We tried expanding our two layer neural network to include more hidden layers, but predictive accuracy dropped from our final 0.9587 to 1.38. Hence, we decided to revert to our original model.

Exhibit 9: Code for Neural Network

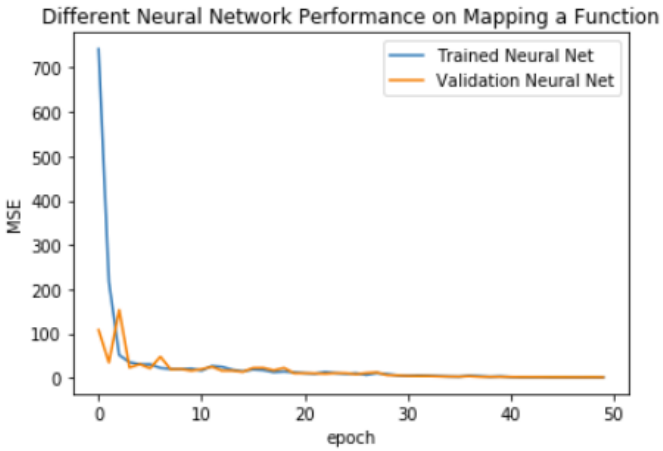


Exhibit 10: MSE Scores on Neural Network

Network Characteristics	Classification Network	Regression Network
Number of hidden layers	2	2
Number of nodes/hidden layers	500	500
Hidden Layer Activation Function	ReLu	ReLu
Output Activation Function	Binary Cross-Entropy	Linear (RMSE)
Optimizer	ADAM	ADAM
Number of Epochs	50	50

Exhibit 11: Summary of neural network characteristics

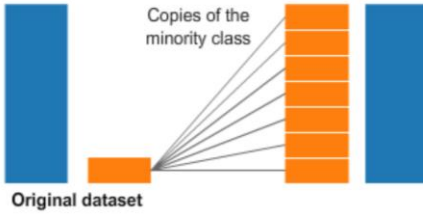
Optimization

Dealing with Class Imbalance

Majority of the transactions in the dataset are visits that didn't convert into a sale. This imbalance in the dataset leads to a bias where the model tends to classify customer visits as no sales, simply because they consist of 99% of the dataset. There are many solutions such as using decision trees and downsampling. We have already implemented one solution, as the LGBM is a variant of a decision tree.

Up-Sampling Process

We chose to do upsampling. Upsampling is a strategy to handle imbalanced classes by repeatedly sampling with replacement from the minority class, to make it in similar proportion as the majority class. To contextualize, we duplicate records of customer visits where a purchase has been made, to have a similar amount of records of data with transactions as do not have transactions. This approach benefits by keeping the data but risks being generally applicable to new predictions.



Model Performance Comparison

Network Characteristics	Classification Network	Regression Network	Network Characteristics
Root Mean Square Error	**17.81**	**0.3114**	**1.1785**
Classifier AUC		**0.9895**	**0.9587**
Final Verdict		**57 times better than random chance**	**15.2 times better than random chance**

Exhibit 12: Model Performance Comparison

Unfortunately, we are unable to compare our RMSE score to others competitors on Kaggle due to a different test dataset used for private submissions. Instead, we decided to compare our predictions to a random chance case. Random chance case is where we predict the minority class (no customer) have transacted. Our random chance RMSE score was 17.81 on our validation data. By itself, this number is meaningless in a business context. However, we can use this as a comparative benchmark to compare the performance of

our LGBM and neural network models. This comparison yields that LGBM model performed 57 times better and the neural network 15.2 times better than random chance.

Another metric to measure performance of Classifier models is the AUC (Area under the ROC Curve), which tells us how good the model is at separating the classes (the higher the better). The LGBM model had an AUC of 0.9895, higher than that of the neural network's AUC of 0.9587. The LGBM model performed overall better than the neural net due to its strong ability to classify customers in the first step.

Conclusion and Closing Comments

In conclusion, our models performed better after upsampling the transaction records. We found that understanding the hour in which the hit occurred, the session number of the customer, the time they spend on the site, and the number of return visits best indicate the customers transaction revenue value.

In our opinion, the LGBM model is a better choice over the neural network model. This is partly because our LGBM model was better at predictions on our validation test, compared to the neural network. And also because it provided a better measure of separability. Additionally, LGBM being a decision tree based provided us with a ranking of features based on their importance in prediction. Finally, retraining and testing a neural network is fairly resource intensive. We suggest Google goes forward with further improving upon our LGBM model which can help the marketing department to target customers more efficiently.