Google Analytics Customer Revenue Prediction

Zhejing Hu, Chi Ma, Yuchong Wang

Department of Statistics,

Columbia University

zh2290@columbia.edu, cm3700@columbia.edu, yc3081@columbia.edu

Abstract — Customer revenue prediction is an act of trying to determine the future revenues for merchandising companies. For boosting the business, marketing teams are often challenged to make appropriate investments in promotional strategies. This paper proposes machine learning models to predict revenue per customer by using the Google Merchandise Store customer dataset. The proposed algorithm includes a linear regression model with elastic-net, regression tree, gradient boost, catboost, XGboost, ensemble, and LightGBM. The proposed models were applied and evaluated using Google analytics datasets and Root Mean Square Error (RMSE). The obtained results showed that LightGBM with free parameters tuning has better prediction accuracy.

Keywords: Google Analytics, Revenue, Regression, Tree, LightGBM.

I.Introduction

Customer revenue prediction has been a focus for years since it can influence every aspect of a merchandising company, especially in production and project development. Predicting customer revenue is not a simple task, mainly as a consequence of different customer background and expectation. The 80/20 rule has proven true for many businesses – only a small percentage of customers produce most of the revenue. As such, it is vital to explore the attributes of past sales and customers. Technical analyses are popular today, especially use machine learning models.

Linear regression (LR) is a commonly used technique due to the high interpretability of input variables. In most cases, linear regression suffers from outliers and overfitting. Proper data cleaning and adding a regularization penalty term to linear regression could eliminate these problems, however, another major flaw of linear regression is that it can only deal with quantitative values.

Also, decision tree (DT) has been developed as an alternative that avoids such limitation. Their practical successes can be attributed to its flexibility in feature selection, the ability to assign specific values to the decision (avoid outlier influences) and the comprehensive nature. But it also suffers from huge computational costs and the nature of growing unwieldy. There are also other models such as cat boost, gradient boost or XGBoost which can solve our problem of predicting customer revenue. In this report, we will focus on a relatively new method called LightGBM.

The perceived advantage of decision tree motivated some researchers to consider a new method, LightGBM, in the context of dealing with a huge dataset and speeding up processes. LightGBM is a powerful machine learning algorithm that been used frequently in recent time. It is a gradient boosting framework that uses the tree-based algorithm. Different from traditional decision tree algorithm, LightGBM grows tree vertically. In other words, LightGBM grows tree leaf-wise while other algorithm grows level-wise. Other reasons why LightGBM gained its popularity these days includes the low memory it requires in running, the support of GPU learning and the focus on result accuracy.

The aim of this report is to find a machine learning model that can predict customer revenue per person accurately with the minimum cost of running time. The performance of the LightGBM is based on the selection of free parameters such as *max_bin, num_leaves*, or *learning_rate*. Multiple tests will be performed to find the best combination of free parameters.

The report is organized as follows: Section 2 represents the previous works related to the problem we are going to solve. System design and dataset usage will be mentioned in Section 3. Algorithm breaks down and experiment results are written in Section 4 and 5. Section 6 represents the conclusion of our project.

II. Backgrounds & Related Work

Google Analytics Customer Revenue Prediction is a featured prediction competition, it predicts how much a GStore customer will spend. It is fairly popular with a $45000 price. There are different methods that were used in this competition. The approaches that have been used include generalized linear models, XGboost, LightGBM, and even Long Short Term Memory networks.

In our project, we will utilize decision tree based algorithm. We also made several modifications on feature selection and parameter tuning base on our understanding of the problem. As the report stated, we mainly focus on the gradient boosting decision tree (GBDT) and used some popular implementation of GBDT such as XGboost in the project. [1]Although many engineering optimizations have been adopted in these implementations, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. A major reason is that for each feature, they need to scan all the data instances to estimate the information gain of all possible split points, which is very time-consuming. Additionally, we used LightGBM, proved later is the best model in terms of efficiency and accuracy, to fit the model. LightGBM integrates with Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). [2]With GOSS, the model exclude a significant proportion of data instances with small gradients, and only use the rest to estimate the information gain. since the data instances with larger gradients play a more important role in the computation of information gain, GOSS can obtain a quite accurate estimation of the information gain with much smaller data size. With EFB, the model also bundles mutually exclusive features (i.e., they rarely take nonzero values simultaneously), to reduce the number of features.

III.System Overview

The project is executed in the following sequential steps.

Step 0: Data Collection

The dataset is originally from Google Merchandise store but being preprocessed by Kaggle. The dataset is in CSV format, also contains JSON blobs of varying depth. It can be found on Kaggle: Google Analytics Customer Revenue Prediction competition. *transcationRevenue* is the target variable that we want to predict. Since the value of *transcationRevenue* is preprocessed, we are predicting the natural log of the sum of all transactions **per user**. The targeted time period for the test dataset is from **December 1st, 2018 to January 31st, 2019,** and the target is:

$$y_{user} = \sum_{i=1}^{n} transaction_{user_i}$$

$$target_{user} = \ln(y_{user} + 1)$$

Data fields:

- ***fullVisitorId*** - A unique identifier for each user of the Google Merchandise Store.
- ***channelGrouping*** - The channel via which the user came to the Store.
- ***date*** - The date on which the user visited the Store.
- ***device*** - The specifications for the device used to access the Store.
- ***geoNetwork*** - This section contains information about the geography of the user.
- ***socialEngagementType*** - Engagement type, either "Socially Engaged" or "Not Socially Engaged".
- ***totals*** - This section contains aggregate values across the session.
- ***trafficSource*** - This section contains information about the Traffic Source from which the session originated.
- ***visitId*** - An identifier for this session. This is part of the value usually stored as the _utmb cookie. This is only unique to the user. For a completely unique ID, you should use a combination of fullVisitorId and visitId.
- ***visitNumber*** - The session number for this user. If this is the first session, then this is set to 1.
- ***visitStartTime*** - The timestamp (expressed as POSIX time).
- ***hits*** - This row and nested fields are populated for any and all types of hits. Provides a record of all page visits.
- ***customDimensions*** - This section contains any user-level or session-level custom dimensions that are set for a session. This is a

repeated field and has an entry for each dimension that is set.

- ***totals*** - This set of columns mostly includes high-level aggregate data.

The side of the training dataset is about 1.7 million rows (21 gigabytes) and the test dataset is around 0.4 million rows (7 gigabytes).

Step 1: Data preprocessing

1. Convert all the JSON fields to a flattened CSV format which generates more features.
2. Remove features with only one unique value.
3. Remove features with more than 80% missing values.
4. Impute missing data with median, mean or certain values.
5. Turn text features to lowercase and remove all the punctuations.

Step 2: Exploratory Data Analysis and Feature engineering

Exploratory Data Analysis (EDA) is performed to examine domain knowledge on important features. For instance, Is there a relationship between *channelGrouping* and *transcationRevenue* during 2016 to 2018. Various plots are made to explore the relationship between different features and targeted variables by using python library package and Tableau.

Following with feature engineering, a total of 5 changes are made before model training.

1. Generate some new features like a weekday, month, transaction status and etc.
2. Create buckets for some of the categorical features with too many categories based on domain knowledge.
3. Convert other categorical features into dummy variables.
4. Turn text features like geo_networkDomain into TF-IDF scores.
5. Create aggregated features(sum, min, max, mean, median) per user.

Step 3: Model selection and parameter tuning

In this step, a total of 7 models have selected includes:

1. Linear regression with elastic-net
2. Regression tree
3. Gradient boosting tree
4. XGboost
5. LightGBM
6. CatBoost
7. Ensemble

Model 1 and 2 are used as baseline models for regression and decision tree methods. Since we are already decided to depend on decision tree framework, model 3-6 are trained to compare results. The Ensemble method is a combination of LightGBM, CatBoost, and XGboost, since results favor LightGBM, the combination of Ensemble is 70% LightGBM, 25% Catboost and 5% XGboost. Last but not least, the LightGBM is performed twice, the latter is with parameter tuning.

Step 4: Compare and output results

The results of the eight models are displayed in the last section. Decision tree models perform better than the regression model. It is expected since lots of the variables are not qualitative.

IV. Algorithm

In this work, we have applied seven different models, the linear regression model with the elastic net, regression tree, gradient boost model, XGBoost, Light GBM, and catboost model. We ran the first three simple model on Spark as our baseline models. These models can be categorized as the regression-based model and tree-based model. Since our goal is to predict the customer revenue based on multiple features, so the first model we want to use is the basic regression model.

A.Linear Regression Algorithm with Regularization

Linear regression is an approach to modeling the relationship between a scalar dependent variable y and one or more explanatory variables (or independent variables) denoted X. For more than one explanatory variable (independent variable), the process is called multiple linear regression which is the case in our project.

$$Y = (y_1, \dots y_n)^T, n \times 1$$
$$X = (x_1, \dots x_p)^T, n \times p$$

Linear regression is to find a linear combination β of predictors x = (X1 , … Xp) to describe the relationship between y and (X1 , … Xp), use yhat = xT* β to predict y. However, using Linear Regression is hard to interpret when there are more predictors, it will need a smaller subset to exhibit the strongest effects.

We used linear regression with elastic net since regularization is a commonly used technique in machine learning to avoid over-fitting of models by introducing additional penalty term to the original loss function. Nearly all learning algorithm serves to minimize a loss function. The most commonly used regularization is L1 and L2, which stands for LASSO and Ridge respectively.

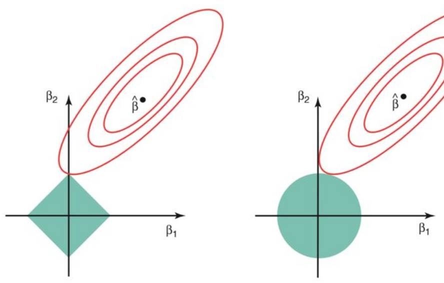
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

Lasso (Least Absolute Shrinkage Select Operator) adds absolute coefficient as the penalty term.

Unlike Lasso, Ridge adds squared coefficient as the penalty term.

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2=\mathrm{RSS}+\lambda\sum_{j=1}^{p}\beta_j^2$$

Geographically, the constraint for Lasso and Ridge are shown in the plot below. Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, therefore the ridge regression coefficient could not be 0. However, the lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero.



The key difference between them is that Lasso could be used for feature selection by shrinking some parameters to zero while ridge shrinks the parameter toward 0 at a faster rate than Lasso. Ridge is most often used to deal with multicollinearity.

We chose Elastic Net as our regularization method, which is a combination of L1 loss and L2 loss, and the formula defines below:

$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y-X\beta\|^2+\lambda_2\|\beta\|^2+\lambda_1\|\beta\|_1)$$

The combination of L1 and L2 solves the limitations of both methods, and provide users the flexibility to adjust the weight. Therefore, elastics net is preferred in many situations especially when features are complicated. And the tools we used for this method is Pyspark especial spark.ml API for regression and elastic net.

B. Light GBM

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification, and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

It is designed to be distributed and efficient with the following advantages[3]:

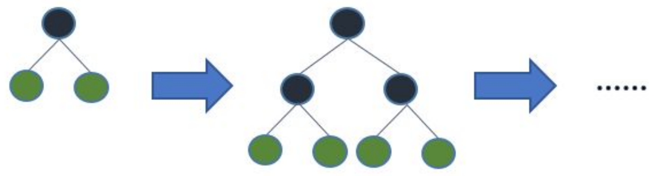
B1.Optimization in Speed and Memory Usage:

LightGBM uses histogram-based algorithms[4, 5, 6], which bucket continuous feature (attribute) values into discrete bins. This speeds up training and reduces memory usage. The histogram-based algorithms can Reduce the cost of calculating the gain for each split, reduce memory usage and reduce communication cost for parallel learning, which allows a lower training time.

B2.Optimization in Accuracy:

Leaf-wise (Best-first) Tree Growth

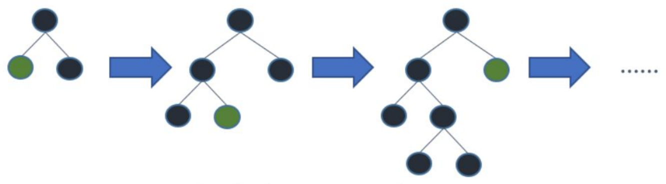
Most decision tree learning algorithms grow trees by level (depth)-wise, like the following image:



Level-wise tree growth

LightGBM grows trees leaf-wise (best-first)[7]. It will choose the leaf with max delta loss to grow. Holding #leaf fixed, leaf-wise algorithms tend to achieve lower loss than level-wise algorithms.

Leaf-wise may cause over-fitting when data size is small, so LightGBM includes the max_depth parameter to limit tree depth. However, trees still grow leaf-wise even when max_depth is specified.



Leaf-wise tree growth

In addition, LightGBM supports a variety of features including metrics, sub-sample, bagging, etc, which allows further parameter tuning.

V. Software Package Description

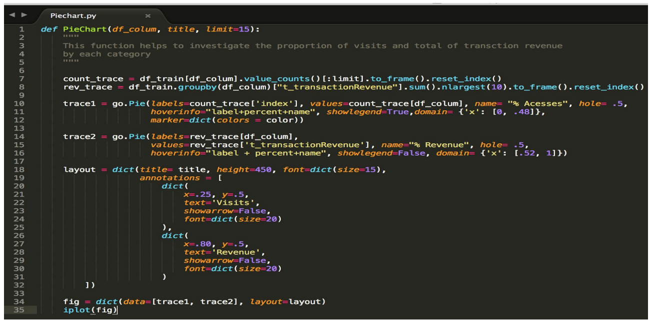
1) yaml package

We convert some non-standard json format to python objects with yaml.load()

```
>>> yaml.load("""
... none: [~, null]
... bool: [true, false, on, off]
... int: 42
... float: 3.14159
... list: [LITE, RES_ACID, SUS_DEXT]
... dict: {hp: 13, sp: 5}
... """)

{'none': [None, None], 'int': 42, 'float': 3.1415899999999999,
'list': ['LITE', 'RES_ACID', 'SUS_DEXT'], 'dict': {'hp': 13, 'sp': 5},
'bool': [True, False, True, False]}
```

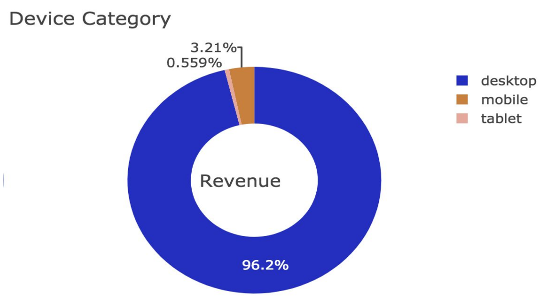
2) Piechart.py

We wrote up the Piechart function to plot pie charts



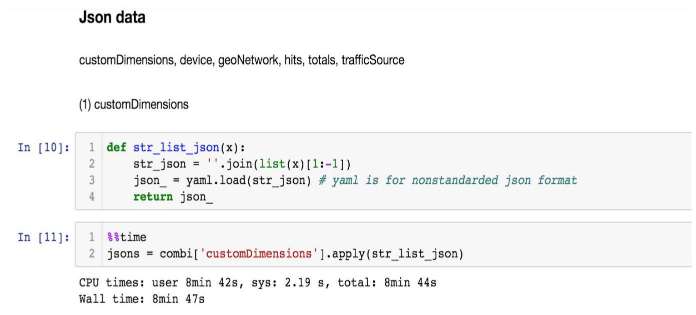
```
def PieChart(df_colum, title, limit=15):
    """
    This function helps to investigate the proportion of visits and total of transction revenue
    by each category
    """

    count_trace = df_train[df_colum].value_counts()[:limit].to_frame().reset_index()
    rev_trace = df_train.groupby(df_colum)["t_transactionRevenue"].sum().nlargest(10).to_frame().reset_index()

    trace1 = go.Pie(labels=count_trace['index'], values=count_trace[df_colum], name= "% Acesses", hole= .5,
                    hoverinfo="label+percent+name", showlegend=True,domain= {'x': [0, .48]},
                    marker=dict(colors = color))

    trace2 = go.Pie(labels=rev_trace[df_colum],
                    values=rev_trace['t_transactionRevenue'], name="% Revenue", hole= .5,
                    hoverinfo="label + percent+name", showlegend=False, domain= {'x': [.52, 1]})

    layout = dict(title= title, height=450, font=dict(size=15),
                  annotations = [
                      dict(
                          x=.25, y=.5,
                          text='Visits',
                          showarrow=False,
                          font=dict(size=20)
                      ),
                      dict(
                          x=.80, y=.5,
                          text='Revenue',
                          showarrow=False,
                          font=dict(size=20)
                      )
        ])

    fig = dict(data=[trace1, trace2], layout=layout)
    iplot(fig)
```

Here’s one of the pie chart output:



3) Data Preprocessing code (Data_Preprocessing.ipynb)

This is a jupyter notebook .ipynb file, user can load the train_v2.csv and test_v2.csv data. Then flattening some of the nonstandard JSON objects into a pandas data frame by using the yaml python package.



Json data

customDimensions, device, geoNetwork, hits, totals, trafficSource

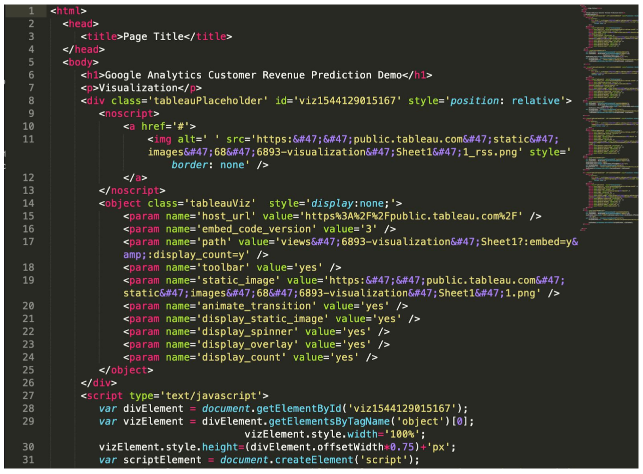
(1) customDimensions

```
In [10]:  def str_list_json(x):
              str_json = ''.join(list(x)[1:-1])
              json_ = yaml.load(str_json) # yaml is for nonstandarded json format
              return json_
```

```
In [11]:  %%time
          jsons = combi['customDimensions'].apply(str_list_json)

          CPU times: user 8min 42s, sys: 2.19 s, total: 8min 44s
          Wall time: 8min 47s
```

4) EDA code (**Exploratory_Data_Analysis.ipynb, visualization.twb, demo.html**)

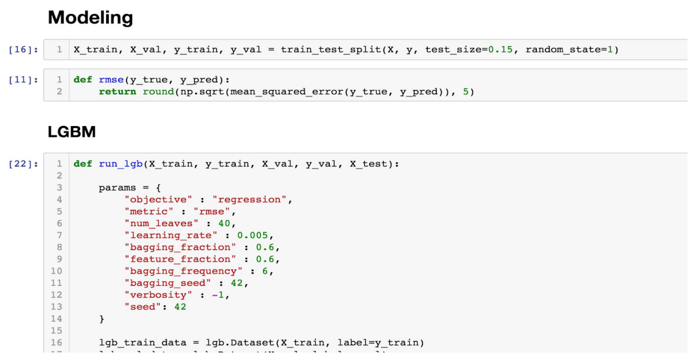
The EDA part includes 1 python notebook file, 1 tableau file and 1 HTML file which is the demo for presentation. The following graph is the HTML code snapshot.



```
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Google Analytics Customer Revenue Prediction Demo</h1>
    <p>Visualization</p>
    <div class='tableauPlaceholder' id='viz1544129015167' style='position: relative'>
      <noscript>
        <a href='#'>
          <img alt=' ' src='https:&#47;&#47;public.tableau.com&#47;static&#47;
          images&#47;68&#47;6893-visualization&#47;Sheet1&#47;1_rss.png' style='
            border: none' />
        </a>
      </noscript>
      <object class='tableauViz'  style='display:none;'>
        <param name='host_url' value='https%3A%2F%2Fpublic.tableau.com%2F' />
        <param name='embed_code_version' value='3' />
        <param name='path' value='views&#47;6893-visualization&#47;Sheet1?:embed=y&
        amp;:display_count=y' />
        <param name='toolbar' value='yes' />
        <param name='static_image' value='https:&#47;&#47;public.tableau.com&#47;
        static&#47;images&#47;68&#47;6893-visualization&#47;Sheet1&#47;1.png' />
        <param name='animate_transition' value='yes' />
        <param name='display_static_image' value='yes' />
        <param name='display_spinner' value='yes' />
        <param name='display_overlay' value='yes' />
        <param name='display_count' value='yes' />
      </object>
    </div>
    <script type='text/javascript'>
      var divElement = document.getElementById('viz1544129015167');
      var vizElement = divElement.getElementsByTagName('object')[0];
                              vizElement.style.width='100%';
      vizElement.style.height=(divElement.offsetWidth*0.75)+'px';
      var scriptElement = document.createElement('script');
```

5) Modeling code (Feature_Analysis_and_Modeling.ipynb)

This jupyter notebook .ipynb file is for final modeling. We perform feature selection and feature engineering first. Then with all the processed features, we feed them into modeling.



Modeling

```
[16]:  X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.15, random_state=1)
```

```
[11]:  def rmse(y_true, y_pred):
           return round(np.sqrt(mean_squared_error(y_true, y_pred)), 5)
```

LGBM

```
[22]:  def run_lgb(X_train, y_train, X_val, y_val, X_test):

           params = {
               "objective" : "regression",
               "metric" : "rmse",
               "num_leaves" : 40,
               "learning_rate" : 0.005,
               "bagging_fraction" : 0.6,
               "feature_fraction" : 0.6,
               "bagging_frequency" : 6,
               "bagging_seed" : 42,
               "verbosity" : -1,
               "seed": 42
           }

           lgb_train_data = lgb.Dataset(X_train, label=y_train)
```

VI. Experiment Results

6.1 Evaluation Metric:

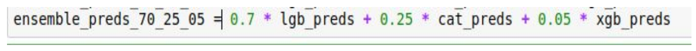
In order to evaluate the performance of our prediction models, we use Root Mean Square Error(RMSE) as our metric since it is also the evaluation metric used in the Kaggle Competition. RMSE is defined as:

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

where y hat is the natural log of the predicted revenue for a customer and y is the natural log of the actual summed revenue value plus one.

6.2 Model Result:

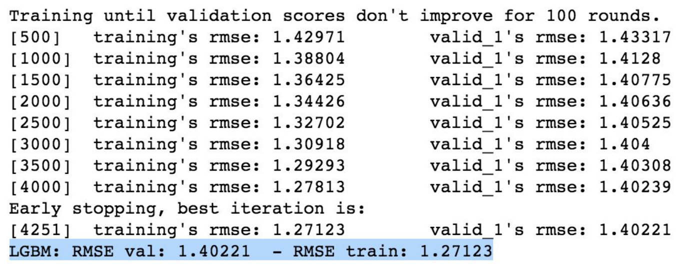
We tried six basic models, which are regression with elastic nets, regression tree, gradient boosting, xgboost, LightGBM and catboost. The result shows that the complicated models like xgboost, LightGBM, and Catboost outperform simple models like regression model, regression tree and gradient boosting. In addition, the LightGBM performs best among all of these six models. We also did an ensemble model based on XGBoost, Light GBM, and Catboost by combining the predicting result of all three models. The equation shows below[4]:



```
ensemble_preds_70_25_05 = 0.7 * lgb_preds + 0.25 * cat_preds + 0.05 * xgb_preds
```

The ensemble model achieved an MSE of 1.483 when we submit the result.

Since Light GBM performs best on this dataset, so we decide to do parameter tuning on the Light GBM model and we achieved an RMSE of 1.40221 after tuning several parameters including the number of leaves, max depth, number of estimators, etc.



```
Training until validation scores don't improve for 100 rounds.
[500]   training's rmse: 1.42971        valid_1's rmse: 1.43317
[1000]  training's rmse: 1.38804        valid_1's rmse: 1.4128
[1500]  training's rmse: 1.36425        valid_1's rmse: 1.40775
[2000]  training's rmse: 1.34426        valid_1's rmse: 1.40636
[2500]  training's rmse: 1.32702        valid_1's rmse: 1.40525
[3000]  training's rmse: 1.30918        valid_1's rmse: 1.404
[3500]  training's rmse: 1.29293        valid_1's rmse: 1.40308
[4000]  training's rmse: 1.27813        valid_1's rmse: 1.40239
Early stopping, best iteration is:
[4251]  training's rmse: 1.27123        valid_1's rmse: 1.40221
LGBM: RMSE val: 1.40221  - RMSE train: 1.27123
```

We tried five submissions and the light GBM with parameter tuning achieved the highest test RMSE, which is 1.2954 calculated by the Kaggle Leaderboard.

VII.Conclusion

According to the comparison between the model generated results, LightGBM is the winner. We can confirm that using historical Gstore transaction data to predict the sales revenue per customer is practical and realistic. The 80/20 law is also proved by our result.

Some major problems we encountered during the project include:

1) Data leakage during the competition: There is a data leakage problem happened during the competition which made more than 600 of the teams reach the perfect score in less than one week. Therefore, Kaggle released dataset Version 2 after we spent most of the time on Version 1. This is a huge disappointment for us and we have to start all over again since we still want to participate in the competition.

2) Lack of computational power: I would say 25% of this project went into general research and system configuration. Due to the lack of computation power, we established a compute engine on Google Cloud Platform as we described in the presentation. Ultimately we installed a virtual machine instance with Ubuntu16.04 and runs 64 cores. The biggest hardship we faced most is that none of us is familiar with the Linux environment and terminal commands even though we learned a lot during the Big data class.

In the future, there are several improvements to make:

1) The feature engineering and selection are not perfect, we plan to exhaustively try a number of new features as well as perform a detailed feature selection.

2) After parameter tuning, the LightGBM model produced a better result than before, therefore, parameter tuning for other models will also be performed and see if they could result in better accuracy. We will also use cross-validation to find the best components for the ensemble method.

ACKNOWLEDGMENT

THE AUTHORS WOULD LIKE TO THANK PROFESSOR CHING-YUNG LIN FOR HIS DEDICATION IN TEACHING OF BIG DATA ANALYTICS COURSE IN FALL 2018. WE WOULD ALSO THANK TAS FOR THEIR HARDWORKING AND HELPFUL ASSISTANCE.

REFERENCES

[1][2] Ke, G. L., Meng, Q., et al., 2017, "**LightGBM: A Highly Efficient Gradient Boosting Decision Tree**", NIPS

https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree

[3]https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

[4]https://www.kaggle.com/julian3833/2-quick-study-lgbm-xgb-and-catboost-lb-1-66

APPENDIX

Paper: Chi: Section 1,2,3,7; Yuchong: Section 5,6; Zhejing: Section 4,6;

Code: Chi: EDA, Feature Analysis, Presentation Demo; Yuchong: EDA, Feature Analysis, Modeling;

Zhejing: GCP system build, Modeling

We all agree that our entire team contribute to this project evenly.