2021

Machine Learning Capstone Project Final Report

Optimizing the Starbucks App's Promotion Offering Strategy

Elaine Liu

I. Project Overview

One of the challenges businesses face nowadays is the maintenance and refinement their ability to retain and convert customers with evolving preferences and tastes, particularly in industries where new products are constantly introduced to the market.

This project aims to explore and implement a machine learning approach to help the Starbucks App craft personalized promotion offerings, as well as drive the highest return for each offering.

This project utilizes a hybrid model combining decision trees with logistic regression, a framework proposed by Facebook in the paper 'Practical Lessons from Predicting Clicks on Ads at Facebook'.[1] The decision tree model in this project uses the Light Gradient Boosting Machine (LightGBM) framework. The general idea for this hybrid model is that logistic regression alone is an efficient and easy-to-implement classification algorithm, but it only performs well on linearly-separable data and is not suitable for non-linear data. Decision tree models, on the other hand, are able to 'bin' each feature to a leaf node in a tree, thus transforming a continuous feature into a discrete feature. These tree features can then be effectively learned by logistic regression. Therefore, by combining decision trees with logistic regression, the hybrid model is able to achieve both training efficiency and modeling performance.

In addition, this project employs extensive feature engineering that captures customers' behavioral information by combining various offer and customer attribute datasets and leveraging historical transaction data.

By comparing against two benchmark models (one with a logistic regression model only, without tree features, and another with a boosted decision tree model only, as a classification model), this project shows that the hybrid model significantly outperforms both benchmark models in terms of Normalized Cross Entropy, the evaluation metric adopted in this project.

By referring to the feature importance generated by the boosted decision tree model, this project also shows that the most valuable features are those capturing historical transaction information about the customers or the product offerings.

II. Problem Statement

Starbucks sends out promotion offers to users through the Starbucks App once every few days. A promotion can be an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free). Some users may not receive any offers during certain weeks. Not all users receive the same offer. Every offer has a validity period before it expires.

[1] He, Xinran, Stuart Bowers, Joaquin Quiñonero Candela, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, et al. "Practical Lessons from Predicting Clicks on Ads at Facebook." Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining - ADKDD'14, 2014. https://doi.org/10.1145/2648584.2648589.

There are varying customer profiles and behaviors. Some customers might only be receptive to certain types of promotions but not others. Some customers might make a purchase regardless of whether they received an offer.

Because there are costs associated with each promotion, the goal of this project is to devise an optimized strategy using machine learning techniques that drives the highest return on each offer sent by identifying the offers with which customers are most likely to engage.

III. Solution Statement

Methodology

Target Value Definition

Since this project does not provide a target, to define a target variable, I first need to define what is considered a 'positive' action. In our context, a positive action is defined as when a customer views an offer after receiving it; redemption of the offer is not required.

The reason for defining viewing an offer as the positive action rather than redemption is that the viewing rate represents customers' engagement level. Even though the action of viewing an offer does not directly translate to dollar amounts for the business, the viewing rate is of the most important metrics for the business to track.

Hybrid Model Architecture

The hybrid model proposed by the Facebook paper and implemented in this project is a concatenation of boosted decision trees and of a probabilistic sparse linear classifier.

The boosted trees are used to transform features from continuous non-linear inputs to discrete categorical values – 'treating each individual tree as a categorical feature that takes as value the index of the leaf an instance ends up falling in.' [2] The transformed tree features are then treated as categorical inputs to the logistic regression model. The hybrid model architecture can be explained by the **Figure 1**[3] below:

[2] Ibid., 3
[3] Ibid., 2

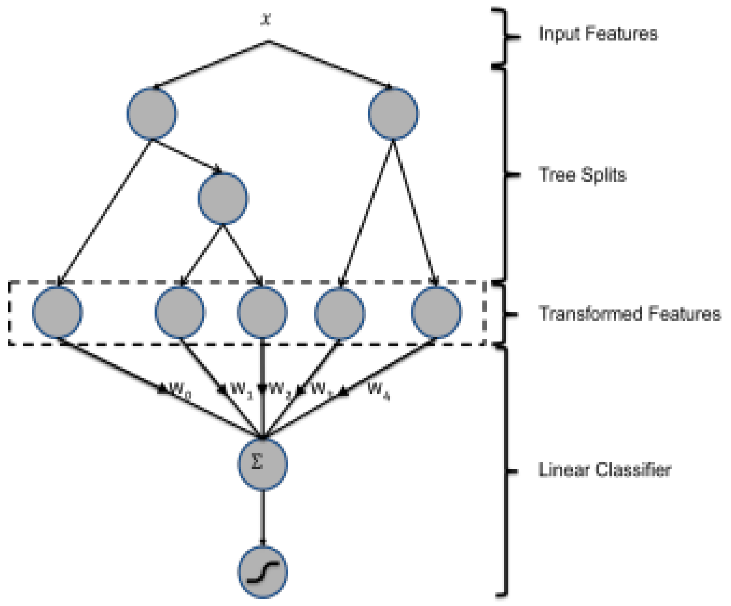

Figure 1: Hybrid model structure

Evaluation Metrics

This project uses Normalized Cross Entropy (NE) as the evaluation metric, same as in the aforementioned Facebook paper. NE is equivalent to the average log loss per impression divided by what the average log loss per impression would be if a model predicted the background click through rate (CTR) for every impression[4]. The formula is as shown below (note that the formula is slightly different from the one in the Facebook paper because the target label in this project is ∈ {0, 1}, whereas the target label in the Facebook is ∈ {-1, 1}):

$$NE = \frac{-\frac{1}{N}\sum_{i=1}^{n}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i))}{-(p * \log(p) + (1 - p) * \log(1 - p))}$$

N: size of the dataset
p_i: estimated probability of viewing an offer p_i where $i = 1, 2, \ldots N$
p: the average empirical probability of viewing an offer

Note that although there is a cost associated with every offer sent, the business would not want to miss any customers who are likely to act on the offer after receiving it, even if it means there would be a higher chance of sending offers to customers who would not be influenced by the offer. In other words, we would need to prioritize minimizing false negatives. Thus, metrics invariant to classification-threshold like AUC would not be optimal for our use case.[5]

[4] Ibid., 2
[5] Classification: ROC Curve and AUC; Machine Learning Crash Course." Google. Google. Accessed March 2, 2021. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

IV. Algorithm and Techniques

Logistic Regression (LR)

LR is a classic binary classification method. There are 3 important concepts to understand in the implementation of the algorithm.

Logistic Function

Using a linear regression model, we can represent output probability as:

$$p(X) = \beta_0 + \beta_1 X$$

However, the obvious downside of using the linear regression model to predict output probability is that we cannot guarantee that it will generate a probability between 0 and 1. LR solves this issue by applying the logistic function to a linear combination of input features to squeeze the output probability between 0 and 1.

$$p(X) = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

The logistic function will always produce an S-shaped curve as shown in **Figure 2**.

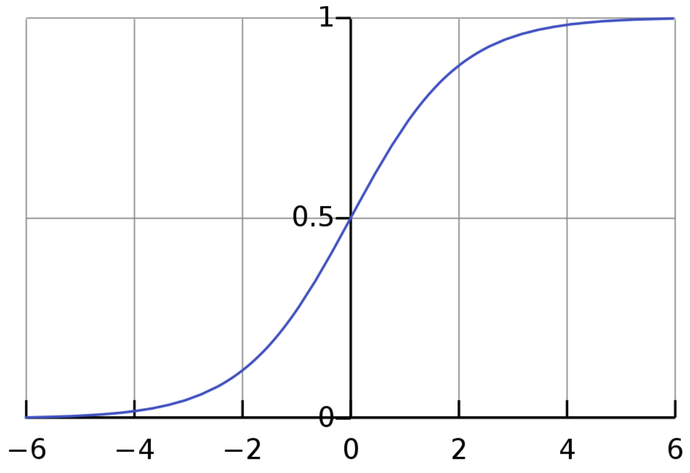

Figure 2: Logistic function

Maximum Likelihood

To fit the LR model, the maximum likelihood estimation (MLE) method is used. MLE is a frequentist probability framework that aims to estimate the value of the unknown parameter so that the model generates results as closely as possible to the observed data[6]. For a binary classification problem where each classification label should be interpreted as a Bernoulli random variable, where $y_i \in (0,1)$ and $0 < p < 1$, the likelihood function can be represented as:

[6] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York :Springer, 2013.

$$L(\beta_0, \beta) = \prod_{i=1}^{n} f(y_i; p) = p^{y_1}(1-p)^{y_1} \times p^{y_2}(1-p)^{y_2} \times \cdots \times p^{y}(1-p)^{y_n}$$

We need to find the p that maximizes the likelihood $L(\beta_0, \beta)$, which requires us to differentiate the likelihood function with respect to p. To make the differentiation easier, we apply a logarithmic function on the likelihood function to turn it to log-likelihood:

$$\log L(p) = \left(\sum y_i\right) \log(p) + (n - \sum y_i) \log(1-p)$$

Since the logarithm is an increasing function, maximizing log-likelihood is the same as maximizing likelihood.

Because it is common practice to minimize a cost function, we can invert maximizing the log-likelihood to minimizing the negative log-likelihood (cost). For m observations, we can calculate the cost as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y_i \times \log p_i + (1 - y_i) \times \log(1 - p_i)]$$

Gradient Descent[7]

Our goal is to minimize the cost $J(\theta)$. Unlike linear regression, there is no closed-form formula to obtain parameters that minimize the negative log-likelihood, so we would need to resort to the gradient descent algorithm. The gradient descent algorithm is an iterative optimization algorithm where with each iteration, we take small steps (we can tune the step size, known as the learning rate) in the direction of the gradient (partial derivative of the log-likelihood with each parameter) to update each parameter:

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

If we keep updating the parameters, they are expected to converge to their best values, and by then we would reach our local minimum. For LR, the result will be a global minimum because it is guaranteed to be convex for all inputs with only one minimum.

Gradient Boosted Decision Tree (GBDT)

Decision Trees

Decision trees can be used to predict the class or value of the target variable by learning simple decision rules inferred from the features of the training data.

[7] Tokuç, A. Aylin. "Gradient Descent Equation in Logistic Regression." Baeldung on Computer Science, January 30, 2021. https://www.baeldung.com/cs/gradient-descent-logistic-regression

A decision rule is a simple IF-THEN statement consisting of a condition and a prediction. For example, a decision rule for our project can be 'IF a customer is a female AND had 3 purchases in the last month, THEN this customer is going to view an offer.'

A decision tree is a tree-like graph with nodes representing an attribute (feature value) and asking a question. Edges represent answers to the questions, and leaves represent the actual output. By traversing the tree from the root to a leaf node, a prediction can be made. **Figure 3** illustrates an example learned decision tree for our project.

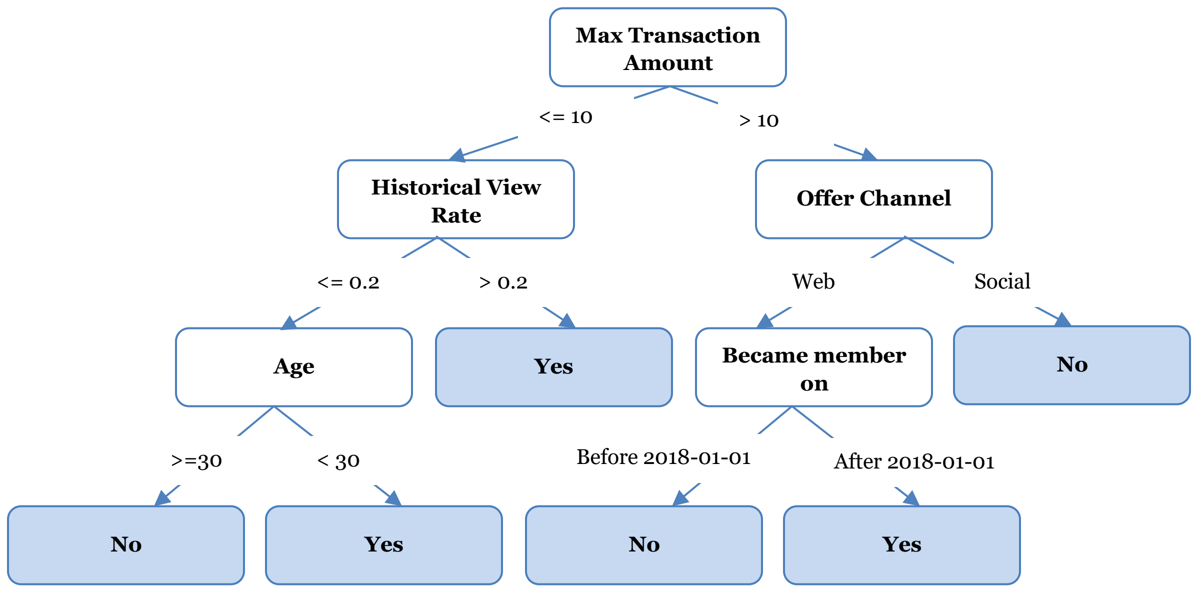

Figure 3: Example learned decision tree

The decision tree splits the nodes on all available variables into sub-nodes. This process is performed multiple times during the training process until only homogenous nodes are left. There are several ways to evaluate homogeneity, such as variance when the target variable is continuous and entropy when the target variable is categorical. The lower the variance or the entropy, the more homogenous the nodes are.

Gradient Boosting

Gradient boosting is a machine learning technique applicable for both regression and classification problems. The core of gradient boosting is the boosting algorithm, which combines weak learners sequentially to achieve a strong learner. A weak learner is defined as one that performs at least slightly better than a random guess.

In a gradient boosted decision tree, the weak learner is the decision tree and each tree only focuses on learning the examples that previous trees misclassified or found it difficult to classify. The final model aggregates the results from each tree to generate a strong learner. This process can be best explained in **Figure 4**.

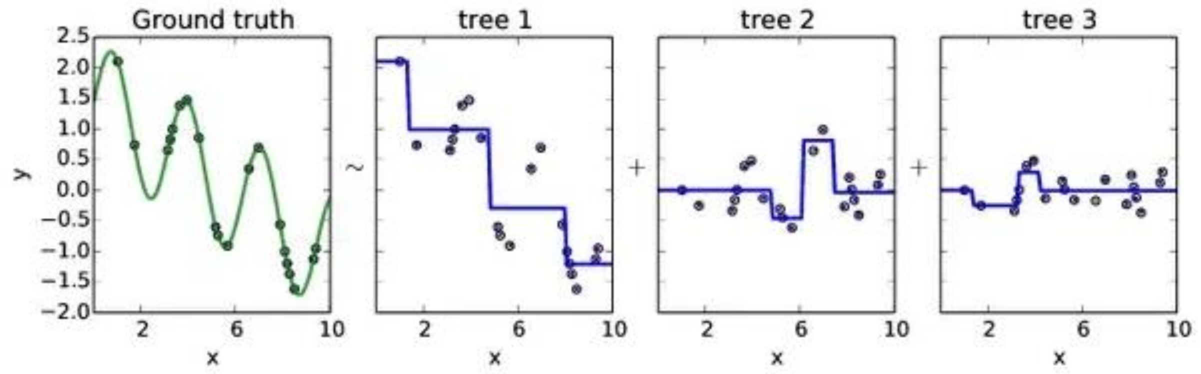

Figure 4: Boosting algorithm

A simplified algorithm can be explained as follows[8]:

For a number of pre-determined boosting rounds M and a differentiable loss function L:

Step 1: Initialize a fit of the data, F_0, a constant value that minimizes the loss function L:

$$F_0 = arg_\gamma min \sum_{i=1}^{n} L(y_i , \gamma)$$

Step 2: With the initial guess of F_0, we can calculate the gradient (or pseudo residuals), of L, our loss function with respect to F_0

Step 3: Fit a decision tree $h_1(x)$ to the residuals:

$$r_{i1} = \frac{\partial L(y_i, F_0(x_i))}{\partial F_0(x_i)}$$

Step 4: Apply gradient descent to minimize the loss for each leaf with a step size γ_1 and a learning rate $\lambda_1$to shrink the step size, thus yielding a new boosted fit of the data:

$$F_1(x) = F_0(x) + \lambda_1 \gamma_1 h_1(x)$$

V. Data Overview

Portfolio

Portfolio data contains offer IDs and metadata about each of the 10 offers. The details of the portfolio data are shown in **XI. Appendix – Portfolio**.

[8] Wyk, Andrich van. “An Overview of LightGBM.” avanwyk. avanwyk, June 21, 2020. https://www.avanwyk.com/an-overview-of-lightgbm/.

Profile

Profile data contains demographic data for each of the 17,000 customers. The details of the profile data are shown in **XI. Appendix – Profile**.

Transcript

Transcript data contains records for transactions, offers received, offers viewed, and offers completed. The details of the transcript data are shown in **XI. Appendix – Transcript**.

VI. Implementation

Create Target Label

With a fair amount of data wrangling, each transaction was assigned a label of 1 or 0, representing whether a customer viewed an offer before transacting or not, after receiving the offer.

Among ~76,000 transactions where the customer received an offer, ~68,000 (90%) transactions show that the customer viewed the offer (label = 1) and ~17,000 (10%) transactions show that the customers did not view the offer (label = 0). Therefore, the offer view rate is about 90%.

Feature Engineering

Aside from the offer and customer features provided in the Portfolio and Profile datasets, I created additional features derived from the Transcript data that serve to uncover underlying characteristics of offers and customers that are not represented in the Portfolio and Profile datasets.

Prevent Data Leakage

One important callout is that all these features were calculated based on training (historical) data only to avoid data leakage - the model would only use past information to make predictions on future data. I divided the Transcript into training and testing - for each customer, based on the time she/he received an offer, if the time was smaller than the fourth quartile of the time period, the transactions would be included in training; otherwise, testing. These features, calculated only using training data, later are included in the testing data, so the training and testing datasets have the same feature space.

The fourth quartile is chosen to make sure that the training dataset has roughly triple the size of the testing dataset.

The following offer features were created based on training the Transcript data (details are shown in **XI. Appendix – Portfolio features based on training Transcript**):

- Average probability of each offer to be viewed (i.e. 'view_rate_portfolio')

- Average/minimum/maximum time it takes for an offer to be viewed (i.e. ‘avg_duration_view_portfolio’, ‘min_duration_view_portfolio’, ‘max_duration_view_portfolio’)

The following customer features were created based on training the Transcript data (details are shown in **XI. Appendix – Profile features based on training Transcript**):

- Average probability of each customer to view an offer (i.e. ‘view_rate_profile’)
- Average/minimum/maximum time it takes for a customer to view an offer after receiving it (i.e. ‘avg_duration_view’, ‘min_duration_view’, ‘max_duration_view’)
- Average/minimum/maximum amount each customer spends on a transaction (i.e. ‘avg_amount’, ‘min_amount’, ‘max_amount’)
- Average number of times each customer transacts (i.e. ‘avg_trx_cnt’)

Data Preprocessing

Log-transformation skewed features

I used log-transformation to transform the skewed features to approximately conform to normality. Although feature normalization is not likely to have an impact on the performance of GBDT, the benchmark model - LR - is among the gradient descent based algorithms and would require the data to be scaled so the gradient descent can move smoothly toward the local minimum.

Apply One-Hot-Encoding

One-hot-encoding is applied on ‘offer type’ and ‘gender’ features. Although LightGBM offers good accuracy with integer-encoded categorical features and performs better than one-hot encoding, I chose one-hot-encoding to satisfy the requirement of the LR model.

Remove highly correlated features

The issue with multi-collinearity among features does not affect the performance of LightGBM because the algorithm is based on Boosted Trees. Additionally, the regularization implemented by default in sklearn LR also addresses the issue of multi-collinearity. Still, highly-correlated features were removed to reduce the feature space.

Figure 5 below shows the features left for modeling and their pairwise correlation with other features.

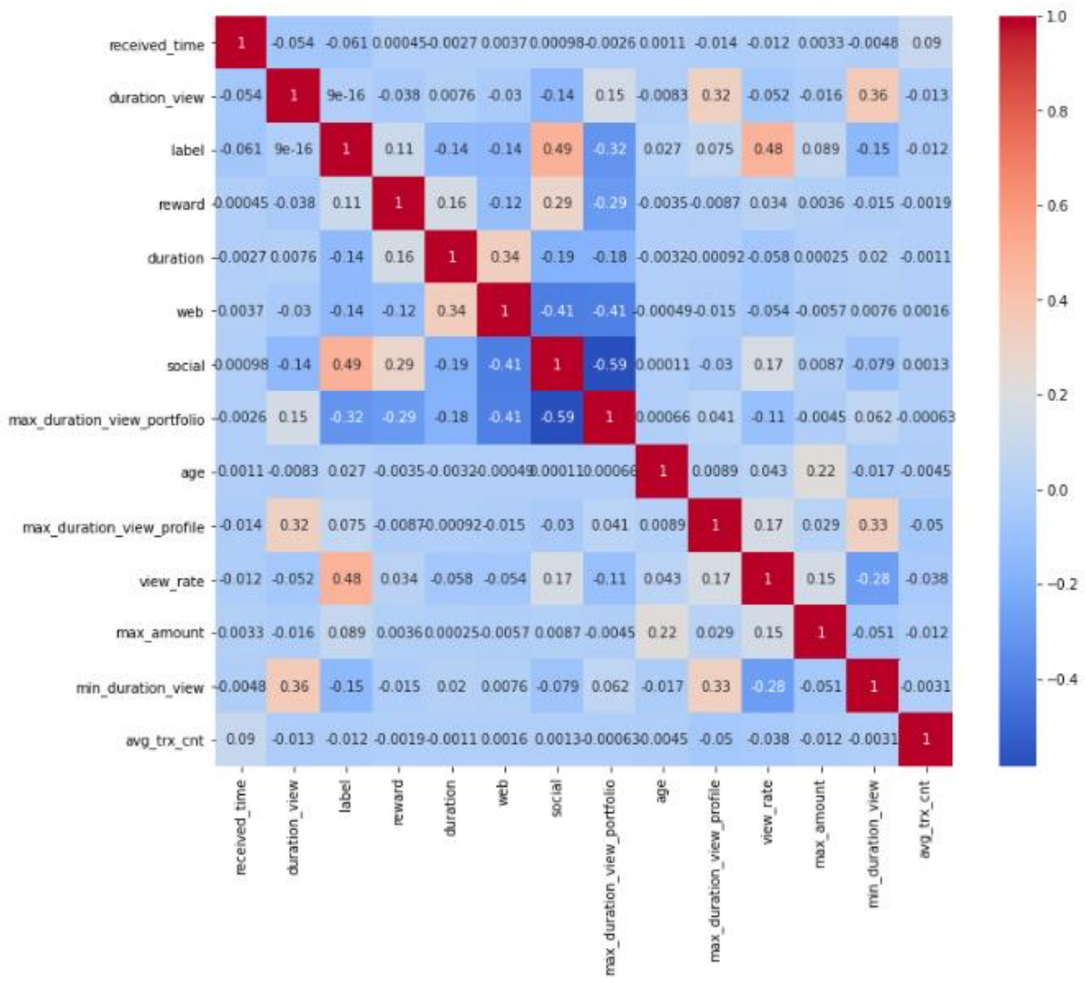
Figure 5: Correlation heat map for features left for modeling

VII. Modeling

Data

I combined the Transcript, Portfolio, and Profile data, along with additional features created, as the final dataset for the model to train and predict.

Hybrid GBDT & LR Model Implementation

GBDT – Hyperparameter tuning

GBDT in this project uses the LightGBM framework and applies Bayesian Optimization, optimized on the mean of the Normalized Entropy using a custom evaluation metric function, to find the best hyperparameters.

LightGBM uses the leaf-wise tree growth algorithm. The leaf-wise growth may be over-fitting if not used with the appropriate parameters. As such, parameter tuning is a must for LightGBM to obtain the optimal model performance.

LightGBM takes many parameters. It is not realistic to tune every single one. As a result, this project only tunes the following parameters with a range of values.

Hyperparameters To Be Tuned	**Tuning Range**
n_estimators	(100,300)
learning_rate	(0.01, 1.0)
feature_fraction	(0.1, 0.9)
max_depth	(5,9)
num_leaves	(200,300)
min_split_gain	(0.001, 0.1)
min_child_weight	(5,50)

Table 1: Hyparameters tuning

The details of the complete set of parameters used in GBDT can be found in **XI. Appendix – GBDT Complete Set of Hyperparameters.**

GBDT – Tree feature creation

The trained GBDT transforms the original features to categorical features for both training and testing data. Each individual tree is a categorical feature that takes as value the index of the leaf an instance ends up falling in[9] (i.e. tree features). Next, one-hot encoding is implemented to transform original tree features into a binary vector with 0 and 1 to indicate whether an instance ends up in a leaf. The final features to be included as input to LR are a combination of the one-hot encoded features based on tree features and the original features.

The goal of GBDT is not to predict the target value, but to transform features to later be included as inputs to LR.

Figure 6 below is the feature importance for the feature inputs to GBDT. Notably, the top 5 features that weighed more tended to be features generated from the historical Transcript data (i.e. 'max_amount', 'view_rate (profile)', 'max_duration_view_profile', 'duration_view (profile)', 'max_duration_view_portfolio'). This project arrived at the same conclusion as the Facebook paper - 'the most important thing is to have the right features: those capturing historical information about the user or ad dominate other types of features.'[10]

[9] Ibid., 2

[10] He, Xinran, Stuart Bowers, Joaquin Quiñonero Candela, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, et al. "Practical Lessons from Predicting Clicks on Ads at Facebook." Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining - ADKDD'14, 2014. https://doi.org/10.1145/2648584.2648589.

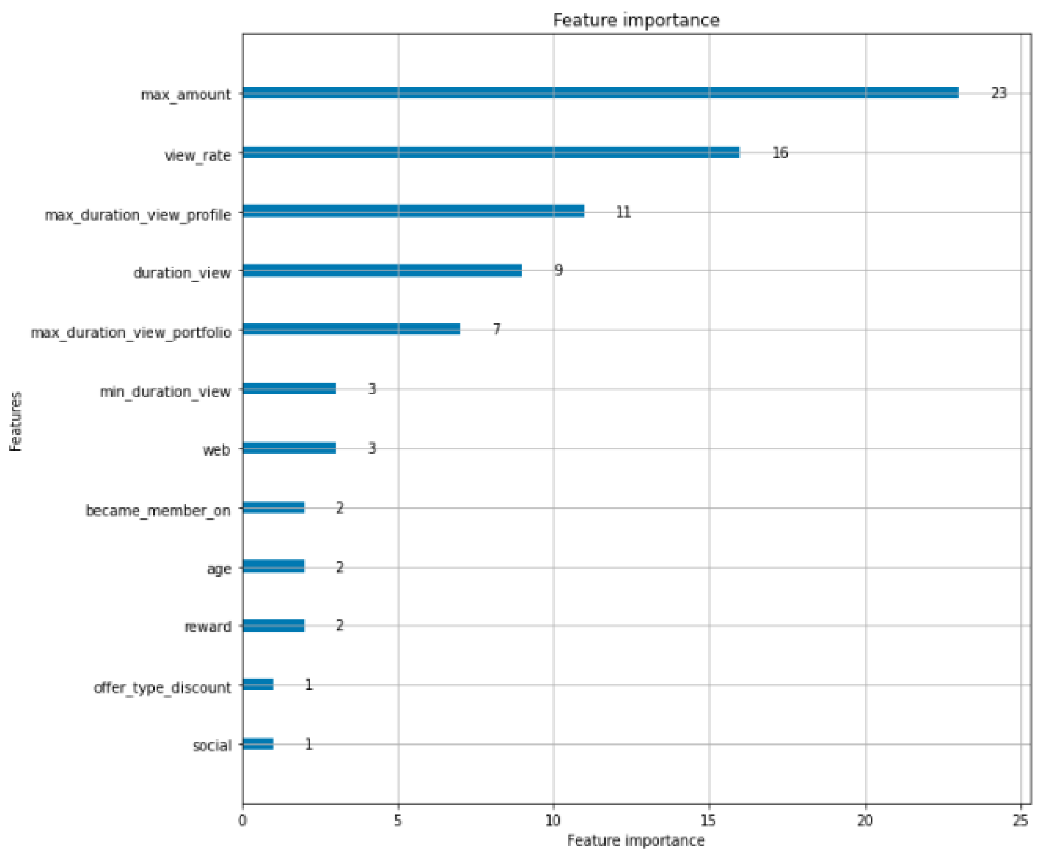

Figure 6: Feature importance generated by GBDT

LR – Result Prediction using Combined Features

This project uses LogisticRegressionCV from the sklearn.linear_model module to implement LR.

The model takes the original features and the one-hot-encoded tree features, with highly-correlated features removed.

The LR model is implemented with 5-fold cross-validations, 'neg_log_loss' as the evaluation metric, and 0 as random_state. Other parameters in this classifier are kept as default. The details of the complete set of parameters used in LR can be found in **XI. Appendix – LR Complete Set of Hyperparameters.**

Note that the evaluation metric here is 'neg_log_loss' instead of the 'Normalized Entropy.' The reason for this is that to calculate the Normalized Entropy, the average log loss would need to be divided by the average log loss for background view rate, and depending on the specific fold in the 5-fold cross validation, the divisor could be zero and we would encounter a 'division by zero' error. Therefore, 'log_loss', the numerator for the Normalized Entropy, was chosen as the evaluation metric.

Below are the 5-fold cross validation results (adjusted by dividing 'neg_log_loss' with the overall background view rate).

We can see that the cross-validation results are almost the same across different folds for the each of the 10 inverses of regularization strengths (i.e. Cs).

	Cs1	Cs2	Cs3	Cs4	Cs5	Cs6	Cs7	Cs8	Cs9	Cs10
Fold 1	0.414	0.136	0.03	0.006	0.001	0.0	0.0	0.0	0.0	0.0
Fold 2	0.422	0.132	0.03	0.006	0.001	0.0	0.0	0.0	0.0	0.0
Fold 3	0.426	0.14	0.033	0.006	0.001	0.0	0.0	0.0	0.0	0.0
Fold 4	0.421	0.132	0.031	0.006	0.001	0.0	0.0	0.0	0.0	0.0
Fold 5	0.417	0.137	0.03	0.006	0.001	0.0	0.0	0.0	0.0	0.0

Table 2: 5-fold cross-validation results of the hybrid model

Benchmark Model Implementation

Benchmark - GBDT

To serve as the benchmark model, GBDT takes the same model without adjustment from the GBDT model used in creating tree features. The only difference is that the input features only include original features (the same original features used in the hybrid model).

Benchmark – LR

The LR model takes the same parameters as in the hybrid model. The only difference is that the input features only include original features (the same original features used in the hybrid model).

VIII. Model Performance Result

Model Structure	Normalized Entropy (Training)	Normalized Entropy (Validation)	Normalized Entropy (Testing)
GBDT + LR	1.03e-05%	1.06e-05%	**1.54e-05%**
GBDT only	0.31%	0.32%	**1.12%**
LR only	45.74%	45.64%	**83.00%**

Table 3: Model performance result comparison

IX. Conclusion

Based on the model performance result, comparing the two benchmark models, the GBDT-only model achieves a significantly lower Normalized Entropy than the LR-only model. As such, the GBDT-only model is remarkably more performant than LR in this specific task (1.12% vs. 83%).

However, even though the GBDT-only model already achieves small Normalized Entropy, the hybrid (GBDT + LR) model generates extremely small testing Normalized Entropy (1.54e-05%).

This project arrives at the same conclusion as the Facebook paper: that combining the decision tree based model with probabilistic linear classifiers significantly improves the prediction accuracy of a model using just the probabilistic linear classifiers alone.

X. Future Work

Based on the hybrid model's almost perfect Normalized Entropy result, the hybrid model might be too complex for this project. A good model not only should be able to predict results accurately but also should maintain a simple structure. Future work might involve applying the hybrid model to a larger dataset or other tasks to test if this model can be well extended to a broad range of use cases.

XI. Appendix

Portfolio

	Count	Explanation
Offer Type		
BOGO	4	
Discount	4	
Informational	2	
Difficulty		Minimum required spend to complete an offer
0	2	
5	2	
7	1	
10	4	
20	1	
Reward		Reward given for completing an offer
0	2	
2	2	
3	1	
5	3	
10	2	
Duration		Time for offer to be open, in days
3	1	
4	1	
5	2	
7	4	
10	2	
Channels		Whether the offer is sent through web email, mobile, or social channels. Offers can be sent through multiple channels
Email	10	
Mobile	9	
Web	8	
Social	6	

Profile

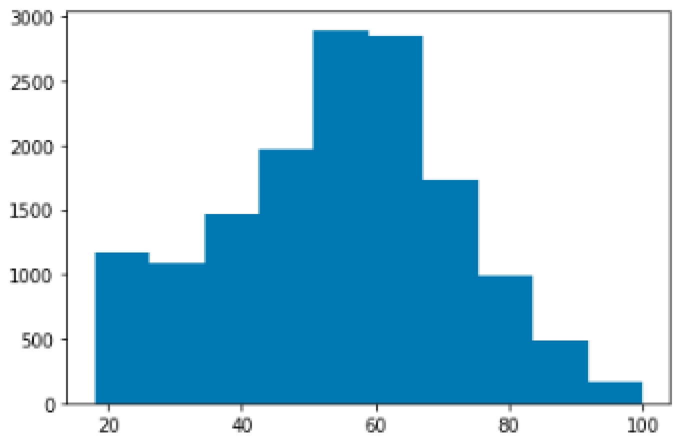

Figure 4: Distribution of customer age (extreme values/outliers removed)

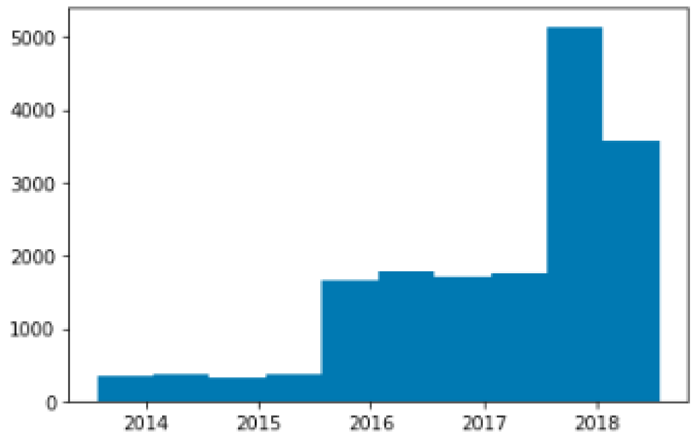

Figure 5: Distribution of when customers became members

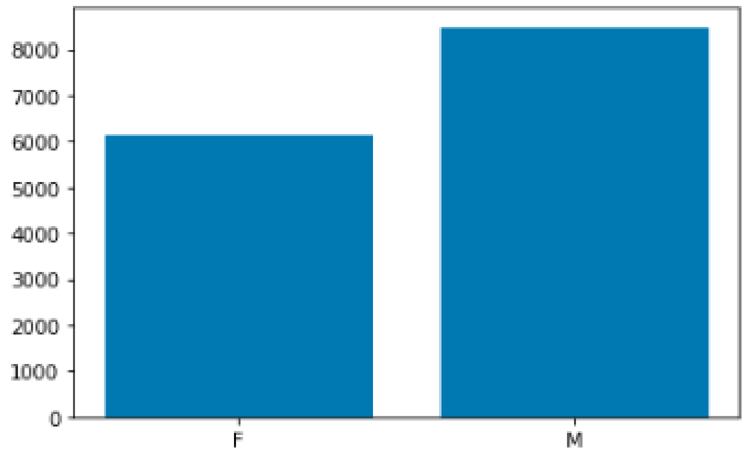

Figure 6: Distribution of customer gender

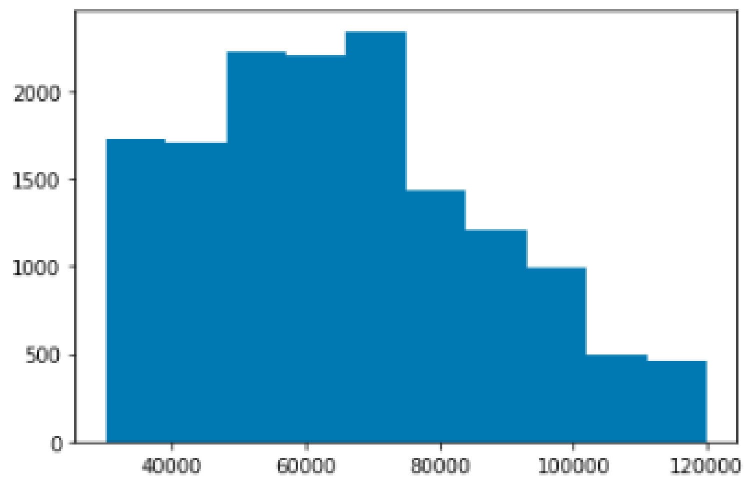

Figure 7: Distribution of customer income

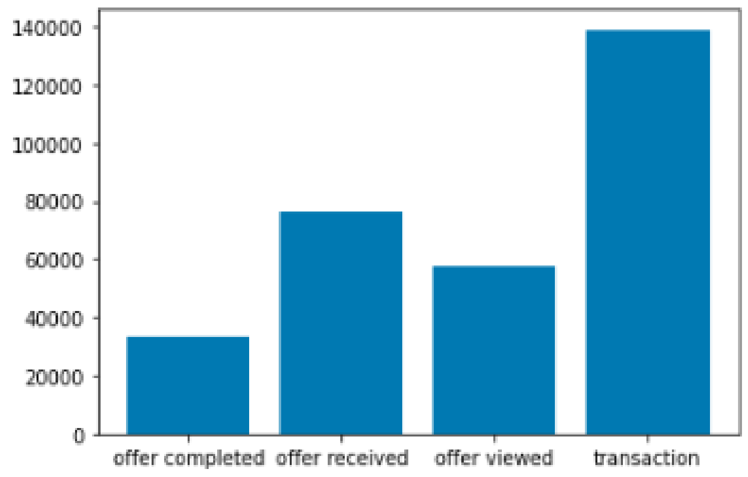

Figure 8: Distribution of events

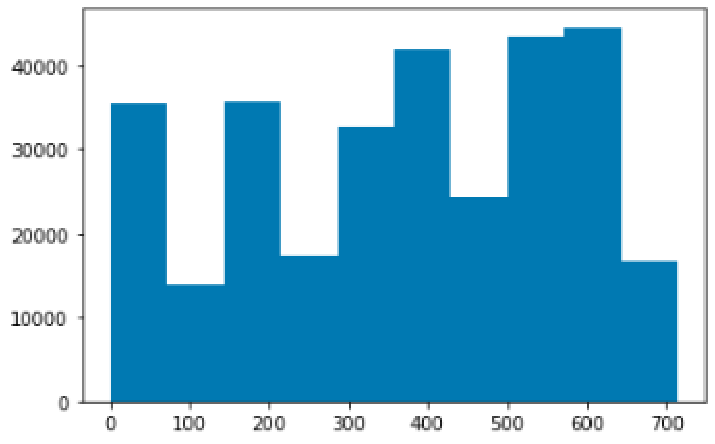

Figure 9: Distribution of time in hours since the start of the test

Transcript

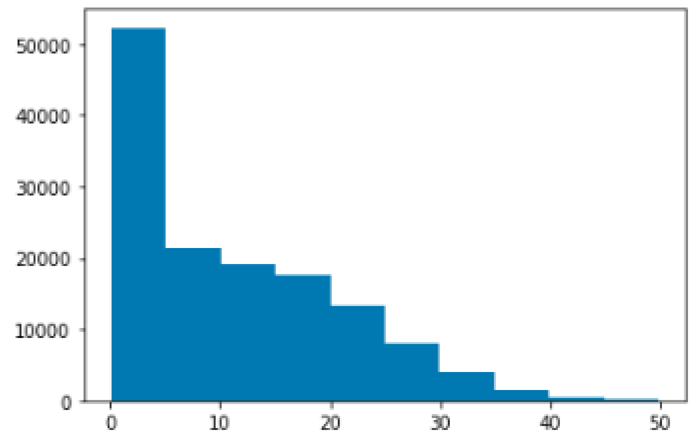

Figure 10: Distribution of transaction amount (extreme values/outliers removed)

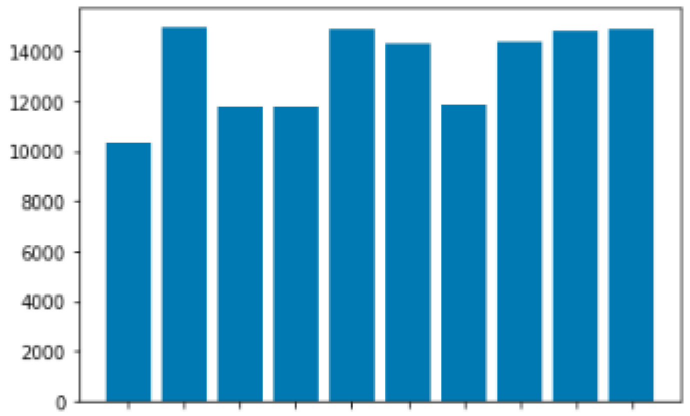

Figure 11: Distribution of offers used in transactions

Portfolio features based on training Transcript

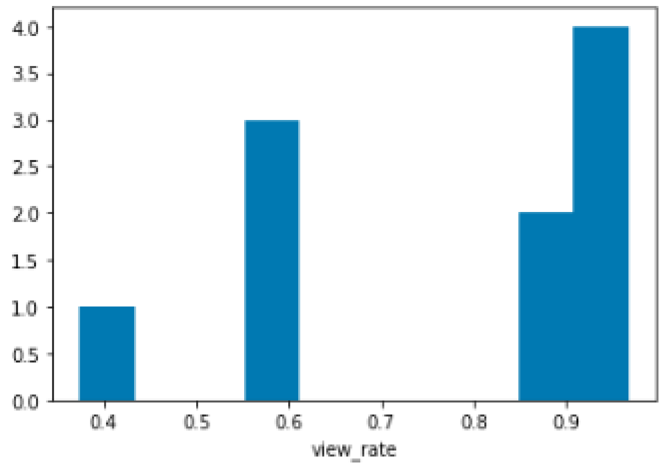

Figure 12: Average view rate of each offer

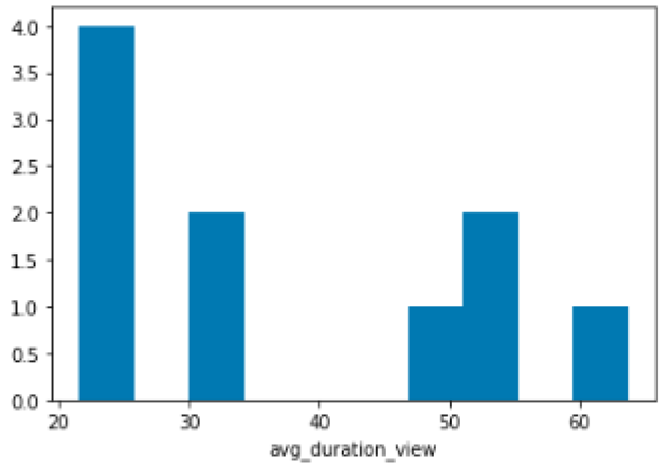

Figure 13: Average duration each offer was viewed

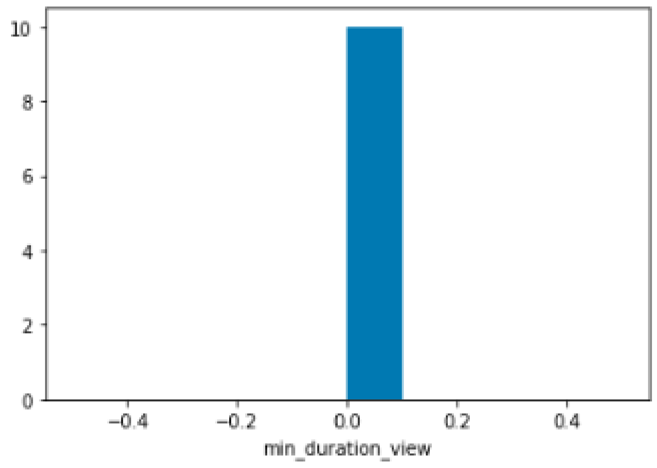

Figure 14: Minimum duration each offer was viewed

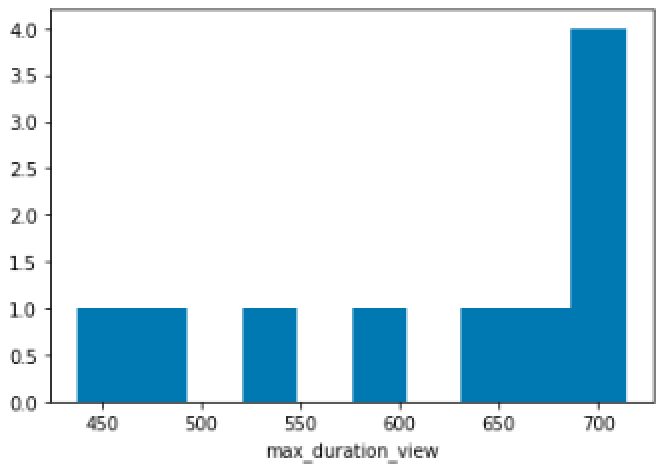

Figure 15: Maximum duration each offer was viewed

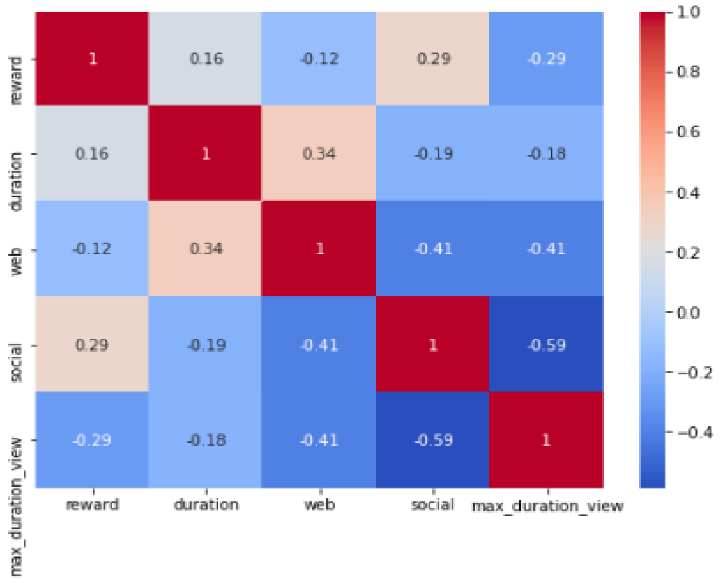

Figure 16: Correlation among all offer features

Profile features based on training Transcript

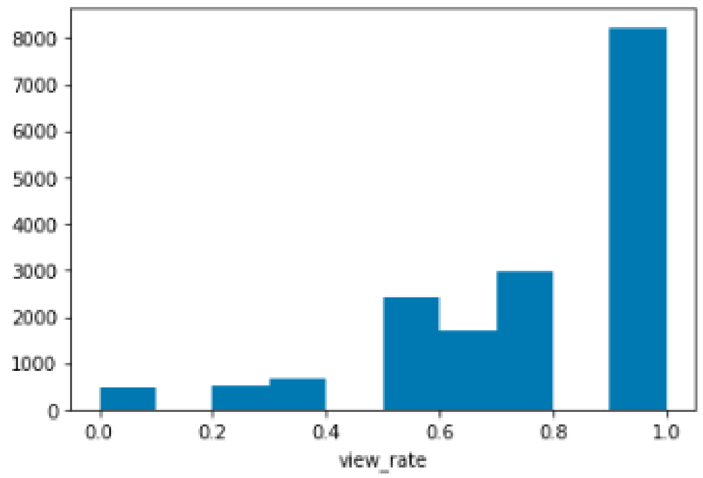

Figure 17: Average view rate of customers

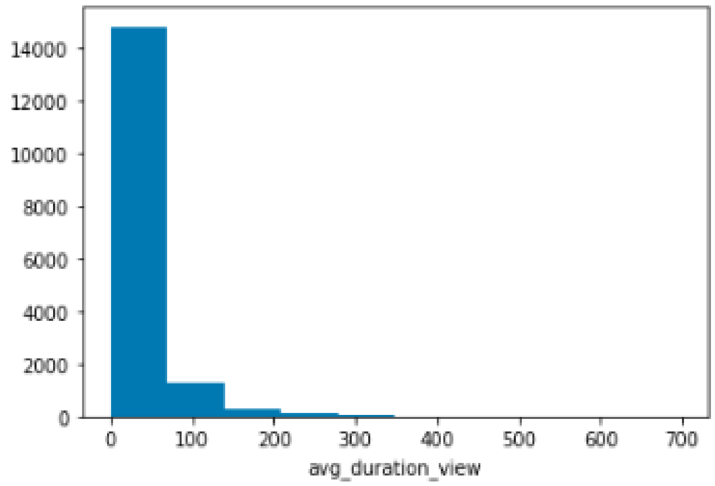

Figure 18: Average duration for which each customer viewed an offer

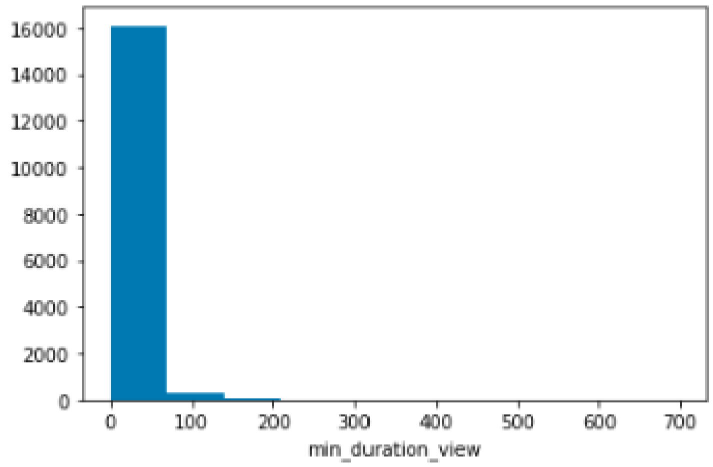

Figure 19: Minimum duration for which each customer viewed an offer

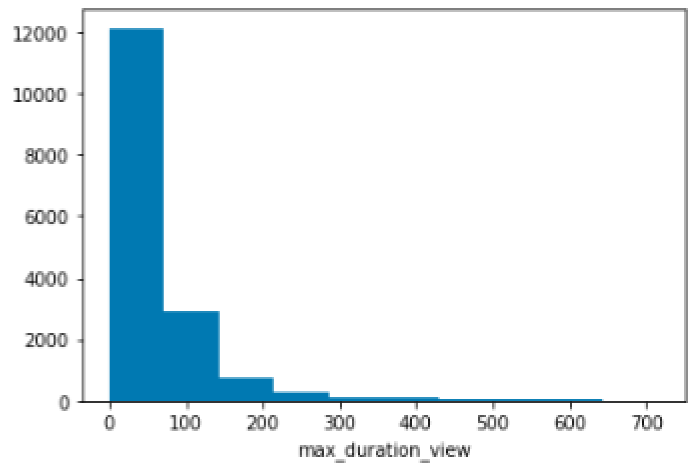

Figure 20: Maximum duration for which each customer viewed an offer

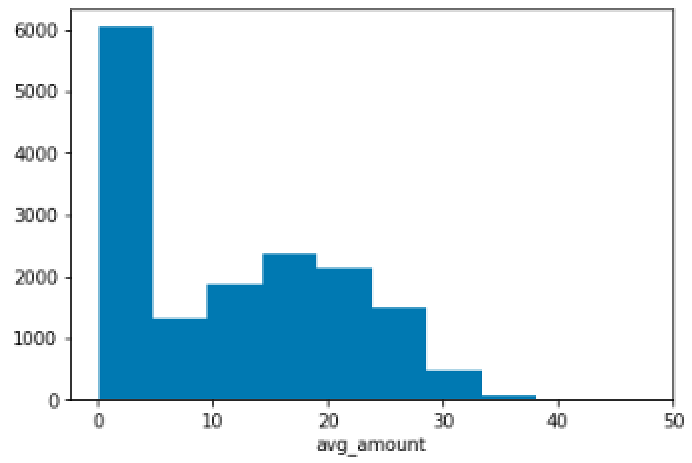

Figure 21: Average amount each customer spent on a transaction

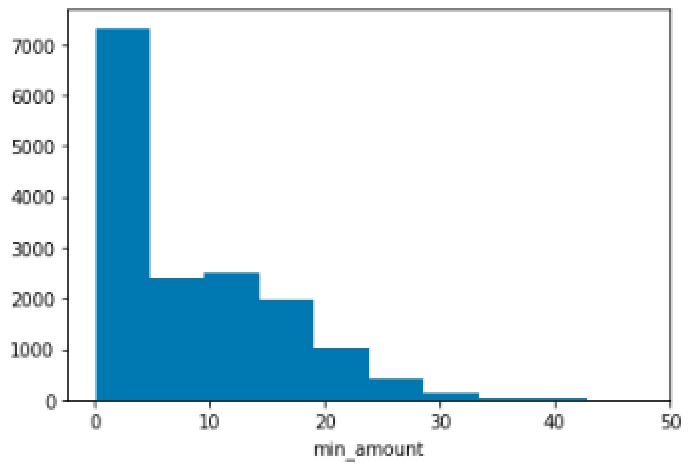

Figure 22: Minimum amount each customer spent on a transaction

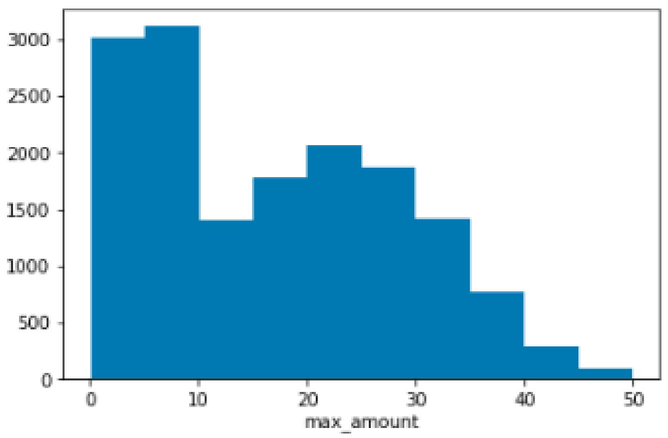

Figure 23: Maximum amount each customer spent on a transaction

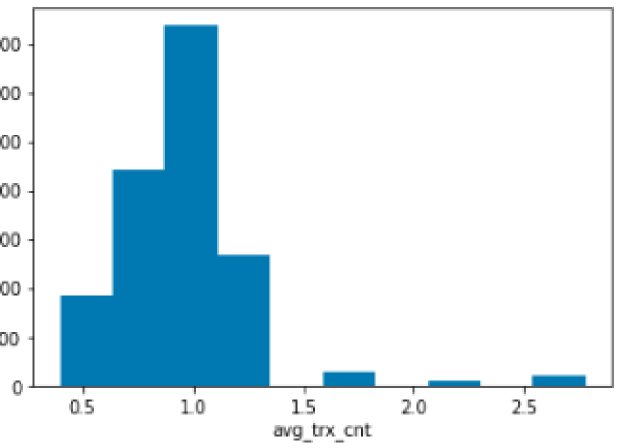

Figure 24: Average number of times a customer made a transaction

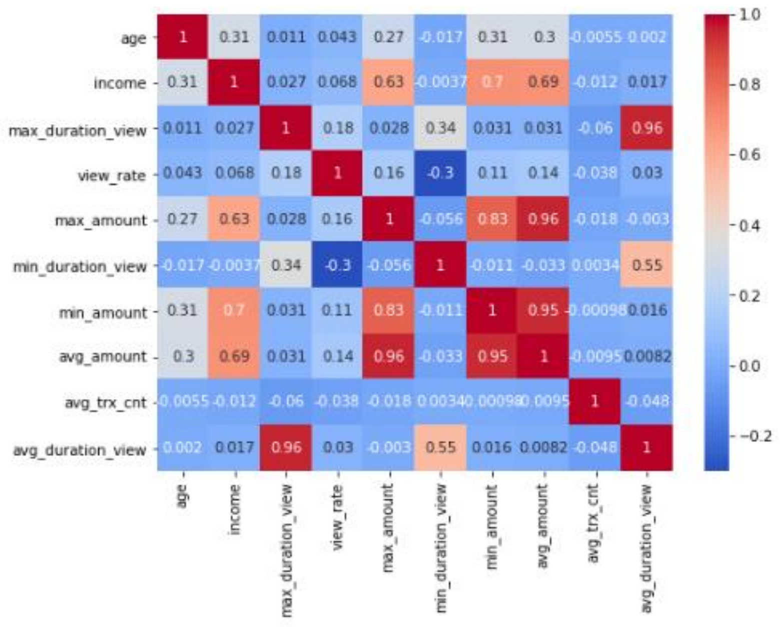

Figure 25: Correlation among all customer features

GBDT Complete Set of Hyperparameters

Hyperparameters	**Tuning Range**	**Final Value**
Tuned using Bayesian Optimization		
feature_fraction	(0.1, 0.9)	0.37
learning_rate	(0.01, 1.0)	0.99
max_depth	(5, 9)	9
min_child_weight	(5, 50)	5.06
min_split_gain	(0.001, 0.1)	0.03
n_estimators	(100, 300)	249
num_leaves	(200, 300)	205
Pre-determined without tuning		
bagging_fraction	N/A	0.2
bagging_freq	N/A	1
boosting	N/A	gbdt
colsample_bytree	N/A	0.8
metric	N/A	Normalized Entropy Defined in customized evaluation function that takes the predicted values, the training data label, and returns the name and result of the evaluation metric, as well as whether the higher the evaluation result is the better (in our case, False)
min_child_samples	N/A	20
objective	N/A	Binary
random_state	N/A	0
reg_alpha	N/A	1
reg_lambda	N/A	1
subsample	N/A	0.8
verbosity	N/A	-1

LR Complete Set of Hyperparameters

Hyperparameters	**Value**	**Default**
class_weight	None	Yes
Cs	10	Yes
cv	5	**No**
dual	False	Yes
fit_intercept	True	Yes
intercept_scaling	1.0	Yes
l1_ratio	None	Yes
max_iter	100	Yes
multi_class	auto	Yes
n_jobs	None	Yes
penalty	L2	Yes
random_state	0	**No**
refit	True	Yes
scoring	neg_log_loss	**No**
solver	Lbfgs	Yes
tol	0.0001	Yes
verbose	0	Yes