Submission Information

Author: Timothy Gorman

Project: Starbucks Capstone Challenge

Definition

This section will provide background on the project, a description of the problem statement and the proposed solution. ## Project Overview As part of the Udacity Machine Learning Engineer Course, Starbucks has provided a data science experiment for us to attempt. This experiment is about understanding what the best offer is for each customer demographic that can be found in the Starbucks app at an invidualized, personalized level. The way that the challenge is presented leaves the door open for different approaches to this challenge. For example, I could build a machine learning model that predicts how much someone will spend based on demographics and offer type, I could build a model that predicts whether or not someone will respond to an offer, or I could decide not to build a machine learning model at all and instead define something like a rules engine.

Generally speaking, this project falls under marketing analytics which is the field of optimizing marketing campaigns for increased return on marketing investment. This interests me because marketing analytics is part of my everyday job at Huntington National Bank (HNB). In my role at HNB, I support model building and model deployment for marketing campaigns. Tackling this project will provide me with relevant experience to the problems I'm faced with every day at work.

The data is provided in the AWS Starbucks Capstone Challenge work space. The data is contained in three files: * portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.) * profile.json - demographic data for each customer * transcript.json - records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

- portfolio.json
  - id (string) - offer id
  - offer_type (string) - type of offer ie BOGO, discount, informational
  - difficulty (int) - minimum required spend to complete an offer
  - reward (int) - reward given for completing an offer
  - duration (int) - time for offer to be open, in days
  - channels (list of strings)
- profile.json
  - age (int) - age of the customer
  - became_member_on (int) - date when customer created an app account

  - gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
  - id (str) - customer id
  - income (float) - customer's income
- transcript.json
  - event (str) - record description (ie transaction, offer received, offer viewed, etc.)
  - person (str) - customer id
  - time (int) - time in hours since start of test. The data begins at time t=0
  - value - (dict of strings) - either an offer id or transaction amount depending on the record

To use this data I downloaded it from the provided workspace and uploaded it into my AWS Account for this section of th class.

Brief Description of the Problem

As described in the previous section, there are multiple ways to analyze the Starbucks dataset. For this project, I chose to build a model that predicts whether individuals will accept offers presented through the Starbucks app.

Solution Statement

My solution was developed using Sagemaker Studio in the AWS account associated with this course. After appropriately analyzing and cleaning the data in a notebook, I split the data into a training sets, validation sets, testing sets. I saved those data sets in S3 to and used them in Sagemaker Training jobs. I used a Sagemaker implementation of a package called LightGBM to predict whether or not individuals will accept the offers presented through the Starbucks app. LightGBM is a gradient boosting framework that uses tree based learning algorithms. Tree based algorithms are well-suited for tabulaur data like in this Starbucks dataset and LightGBM in particular is advantageous because it is designed to be fast, have low memory usage, allow for parallel, distributed, and GPU learning, and handle large-scale data. I trained and tested the model using sagemaker processing. I then compared this model output to to a logistic regression model and analyzed the results based on the metrics described below

Metrics

I measured success based on the AUC score (area under the ROC curve). This is an appropriate metric for classification problems that will give me a sense of the false positive and the true positive rate (extractable from the ROC Curve). Beyond that, AUC is a desirable metric for the following reasons: (1) AUC is scale-invariant (doesn't depend on absolute value of scores) and (2) AUC is classification-threshold-invariant (measures quality of prediction independent of

selected classification threshold). I also considered other metrics such as the F1 Score and accurracy.

Data Exploration

My data exploration is all performed in the notebook "01_Exploratory_Data_Analysis". To run this notebook and the notebooks described later, use the following settings in Sagemaker Studio.

- Instance: ml.t3.medium
- Image: Data Science
- Kernel: Python3

The important aspects of the data exploration are described below.

As discussed in the Project Overview Section, there are some similar columns between the various data sets but they are not exactly the same, so I first standardized names across datasets so that customer IDs are "customer_id" and offer IDs are "offer_id". With those column name updates, I will describe the three different datasets below.

Portfolio

The portfolio dataset contains all 10 different offers presented in the Starbucks app and can be seen below.

	reward	channels	difficulty	duration	offer_type	offer_id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5	7	bogo	9b98b8c7a33c4b65b9aebfe6a799e6d9
4	5	[web, email]	20	10	discount	0b1e1539f2cc45b7b9fa7c272da2e1d7
5	3	[web, email, mobile, social]	7	7	discount	2298d6c36e964ae4a3e7e9706d1fb8c2
6	2	[web, email, mobile, social]	10	10	discount	fafdcd668e3743c1bb461111dcafc2a4
7	0	[email, mobile, social]	0	3	informational	5a8bc65990b245e5a138643cd4eb9837
8	5	[web, email, mobile, social]	5	5	bogo	f19421c1d4aa40978ebb69ca19b0e20d
9	2	[web, email, mobile]	10	7	discount	2906b810c7d4411798c6938adc9daaa5

Figure 1: Raw Portfolio Data

The offers have differing numbers and types of channels that they can be presented through with differing difficulties, rewards, and durations. The bogos come with the highest rewards and the discount comes with the highest difficulty. For this data to be properly used in a model, I will need to extract the individual channels out of the "channels" column and one-hot encode them.

Profile

The profile dataset contains the relevant features of customer profiles as can be seen in the image below. “age”, “customer_id”, and “became_a_member_on” all have non-null values but “gender” and “income” both have null values that will need to be imputed before modeling.

```
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 5 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            14825 non-null  object
 1   age               17000 non-null  int64
 2   customer_id       17000 non-null  object
 3   became_member_on  17000 non-null  int64
 4   income            14825 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 664.2+ KB
```

Figure 2: Profile Info

A snapshot of the profile dataset looks like the following image.

	gender	age	customer_id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN

Figure 3: Raw Profile Data

By inspecting this sample of data we can see that the offer ID is hashed like the customer ID is from the portfolio dataset, and we can see that gender is presented as a categorical type. One column that stands out is “became_a_member_on”, which can be reformed into something like customer tenure, which I think will have a strong impact on whether or not an offer will be accepted.

We can learn more about our numeric columns by ploting histograms of them, which ar shown below.

“age” appears to have a fat tail on the distribution towards the lower end, with faster drop off on the higher end accompanied by a spike at the age of 118. We

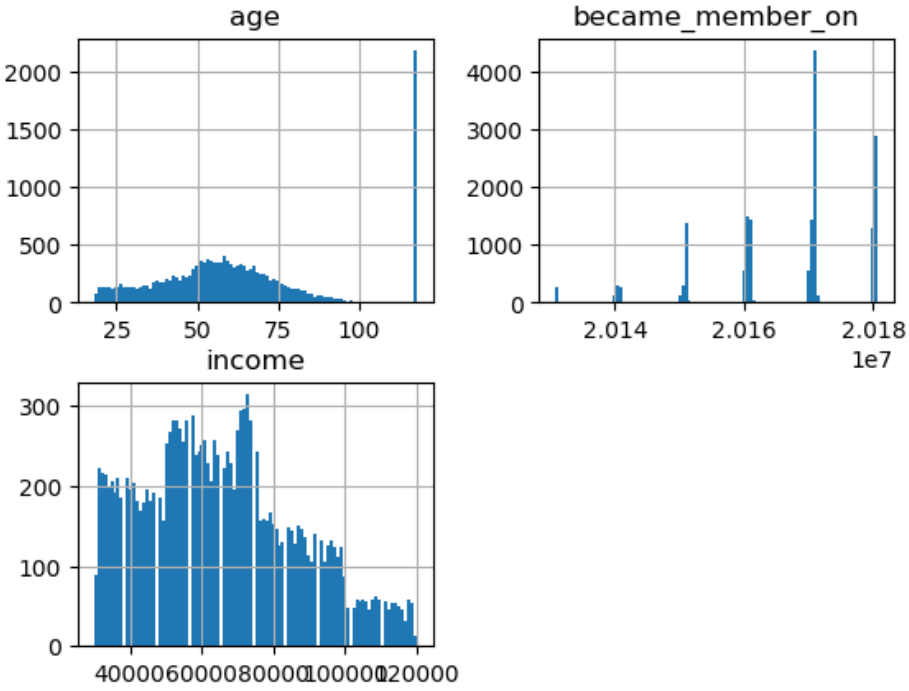

Figure 4: Profile Dataset Histograms

know that there are not that many profiles with an age equal to 118 so we will be removing that from our dataset that gets fed into the model.

"became_a_member_on" is grouped by year, which is due to how the date is formatted in the column. Because of this, some information is lost but we can still see that there is a peak for the year 2017 and then a drop off as we go further back in time.

"income" appears to have for distinct groupings that we can later use to segment the data into different income populations.

Transcript

Transcript is the largest data set provided with 306534 records provided. All records are populated with no null values as seen below along with a sample of data from the transcript dataset.

```
RangeIndex: 306534 entries, 0 to 306533
Data columns (total 4 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   customer_id  306534 non-null  object
 1   event        306534 non-null  object
 2   value        306534 non-null  object
 3   time         306534 non-null  int64
dtypes: int64(1), object(3)
memory usage: 9.4+ MB
```

Figure 5: Transcript Basic Info

	customer_id	event	value	time
306529	b3a1272bc9904337b331bf348c3e8c17	transaction	{'amount': 1.5899999999999999}	714
306530	68213b08d99a4ae1b0dcb72aebd9aa35	transaction	{'amount': 9.53}	714
306531	a00058cf10334a308c68e7631c529907	transaction	{'amount': 3.61}	714
306532	76ddbd6576844afe811f1a3c0fbb5bec	transaction	{'amount': 3.5300000000000002}	714
306533	c02b10e8752c4d8e9b73f918558531f7	transaction	{'amount': 4.05}	714

Figure 6: Transcript Snapshot

The "event" column contains the following kind of events: transaction, offer recieved, offer viewed, and offer completed. This is the main column we will use to identify a successful offer along with the "time" column from this dataset

and the "duration" column from the portfolio dataset. In order for an offer to be considered successful it would need to to have an offer recieved, viewed, and then completed within the duration (from the portfolio dataset) of the offer. If the offer is viewed and not completed, then the offer is not succesful, and if the offer is completed and not viewed then the offer is not considered successful.

Continuing with analysis of the Transcript Dataset, if we take the difference between the max and the min times from this dataset (time is in hours), we find that the offer trial lasted about 30 days. This gives us a sense of the timescale over which our model predictions will be helpful. If this were a real model, I would think that the refresh timescale of the model (retraining) would maybe be on the order of months or a year becuase our training time window is only one month.

Algorithms and Techniques

As mentioned in the Solution Statement, LightGBM is the model that I chose to use in this project. In particular, I have chosen to use the AWS Sagemaker Jumpstart implementation of the LightGBM Algorithm. This implementation allows me to use a pre-trained LightGBM model that easily integrates with sagmeaker training, hyperparameter tuning, and inference without the need to write a series of custom scripts. I also chose to use AWS Sagemaker Jumpstart implementation of the SKLEARN Logistic Regression algorithm. This has all of the benefits that the LighhtGBM implementation has. Using these implementations gives me the full power of Sagemaker with the least amount of setup – perfect for modeling a new set of data. Algorithms and techniques used in the project are thoroughly discussed and properly justified based on the characteristics of the problem.

My expectation for performance is that a hyper-parameter-tuned LightGBM model will be able to achieve a higher ROC AUC score when compared to a simple Logistic Regression Model.

Methodology

In this section, I will describe how the data was preprocessed, how the features were engineered, and how the models were trained.

Data Preprocessing and Feature Engineering

A significant portion of the time for this project was spent on preprocessing the datasets for the LightGBM and Logistic Regression models. I will first describe how I preprocessed each dataset and then how the were joined, labeled, and finalized for the model. All of these preprocessing and feature engineering steps can be found in the "02_Feature_Generation.ipynb".

Portfolio

To prepare the portfolio dataset for modeling, I decided to one-hot encode both the "offer_type" column and the "channels" column using using the Sci-Kit Learn MultilabelBinarizer method and the Pandas get_dummies method, respectively. A snapshot of the final portfolio dataset is below.

	reward	difficulty	duration	offer_id	email	mobile	social	web	bogo	discount	informational
0	10	10	7	ae264e3637204a6fb9bb56bc8210ddfd	1	1	1	0	1	0	0
1	10	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	1	1	1	1	1	0	0
2	0	0	4	3f207df678b143eea3cee63160fa8bed	1	1	0	1	0	0	1
3	5	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	1	0	1	1	0	0
4	5	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	1	0	0	1	0	1	0

Figure 7: Final Portfolio Dataset

Profile

To prepare the profile dataset for modeling I needed quite a lot of preprocessing. First I filled nulls in the "gender" column with a "no_G" label and then one-hot encoded the "gender" into "M", "F" "O" (other), and "no_G" columns. Next I binned and one-hot encoded the "age" column into quartiles that were calculated excluding the spike in counts at 118. This means that I bucketed all customers with age = 118 into the oldest age group, "age4", which I think is an acceptable choice for this clearly incorrect age. In order from youngest to oldest, the 4 age buckets were "age1", "age2", "age3", and "age4". After that I binned and one-hot-encoded the "income" column into 4 buckets from low to high income: inc1, inc2, inc3, inc4, and 1 other bucket, no_inc, for missing income. These groups were based on the 4 groups seen in profile_hist.png. Lastly I decided to reformat the "became_member_on" column into a "years_as_member" column which assumed the current year was 2018. I assumed this beacuse there were no customers that became members in years after 2018. Instead of bucketing "years_as_member" and one-hot encoding it, I decided to keep it as a single column and retain ordinality, which I thought may be an important aspect of this feature for the model. A snapshot of the final profile dataframe can be seen below.

	customer_id	F	M	O	no_G	age1	age2	age3	age4	inc1	inc2	inc3	inc4	inc_miss	years_as_member
0	68be06ca386d4c31939f3a4f0e3dd783	0	0	0	1	0	0	0	1	0	0	0	0	1	1
1	0610b486422d4921ae7d2bf64640c50b	1	0	0	0	0	1	0	0	0	0	0	1	0	1
2	38fe809add3b4fcf9315a9694bb96ff5	0	0	0	1	0	0	0	1	0	0	0	0	1	0
3	78afa995795e4d85b5d9ceeca43f5fef	1	0	0	0	0	0	0	1	0	0	0	1	0	1
4	a03223e636434f42ac4c3df47e8bac43	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Figure 8: Final Profile Dataset

Transcript

To preprocess the transcript dataset , I had to unpack the value column into separate columns for “amount”, “offer_id”, and “reward” and then populate those columns based on the “event” column. For example if the “event” equaled “offer received” then I needed to populate the “offer_id” column with the “offer_id” from the “value” column. Or if the “event” equaled “offer completed” then I needed to populate “offer_id” and “reward” from the “value” column. An example of the transaction dataset where I filtered only to “offer completed” events is seen below.

	customer_id	event	time	offer_id	amount	reward
12658	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	offer completed	0	2906b810c7d4411798c6938adc9daaa5	None	2.0
12672	fe97aa22dd3e48c8b143116a8403dd52	offer completed	0	fafdcd668e3743c1bb461111dcafc2a4	None	2.0
12679	629fc02d56414d91bca360decdfa9288	offer completed	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	None	5.0
12692	676506bad68e4161b9bbaffeb039626b	offer completed	0	ae264e3637204a6fb9bb56bc8210ddfd	None	10.0
12697	8f7dd3b2afe14c078eb4f6e6fe4ba97d	offer completed	0	4d5c57ea9a6940dd891ad53e9dbe8da0	None	10.0

Figure 9: Final Transcript Dataset

Merging and Generating the Target Label

The data was merged by left joining the profile and portfolio datasets to the transcript dataset on “offer_id” and “customer_id” respectively. To generate the target label I sorted the merged dataset by “customer_id”, “offer_id”, and “time”. This resulted in a dataset where all transactions for a given customer ID were in order of offer ID and also in chronological order. This arrangment of the dataset allowed me to search for “sucessful offers”, which I did using conditional statements and looping over the rows of the dataframe. As a reminder, the criteria for a sucessful offer are the following:

1. an offer was recieved,
2. an offer was then viewed,
3. an offer was subsequently completed,
4. and criteria 2 and 3 were satisfied within the duration of the offer in question.

When a successful offer was found, the associated “offer recieved” record was labeled as “offer_successful” equal to 1, and 0s were assigned to unsuccessful offers.

After labeling, the dataset was found to be somewhat imbalanced with 30.5% of records haveing an “offer_successful” label of 1 and 69.5% having and “offer_succesful” label of 0.

The Final Dataset: Train, Validation, Test Split

The columns of the final dataset were filtered down to the following, where "offer_successful" is the target column:

Feature #	Feature
Target	offer_successful
1	reward
2	difficulty
3	duration
4	email
5	mobile
6	social
7	web
8	bogo
9	discount
10	informational
11	F
12	M
13	O
14	no_G
15	age1
16	age2
17	age3
18	age4
19	inc1
20	inc2
21	inc3
22	inc4
23	inc_miss
24	years_as_member

The only records kept were those for "offer received" events, all other records were dropped because they did not add more relevant information from a modeling perspective.

The data was then split into training, validation, and testing datasets using a stratified train-test split method from Sci-Kit Learn with 75% of the data in the training set the remaining 25% split between the testing and validation datasets.

The data was then sent to s3 buckets from where it was loaded into the training jobs described in the next section.

Implementation

Model training was accomplished in the "03_model_training.ipynb" notebook and the training process therein is described below. I largely based the training steps on this AWS example notebook in Github and this "How to Use Sagemaker LightGBM" Documentation for the LightGBM model and the Sagemaker Jumpstart notebook for Sci-Kit Learn Linear Classification.

LightGBM Training

As mentioned before, I decided to use the AWS Sagemaker implementation of the LightGBM classification model. This was convenient because that implementation provided the relevant images, training script, and inference script. In order to see the benefit of hyperparameter tuning I first trained the LightGBM model without hyperparameter tuning using the sagemaker.estimator.Estimator.fit method. I used the defualt hyperparemeters associated with the Sagemaker implementation of the LightGBM model except for setting the "metric" hyperparameter equal to "binary_logloss" and the "num_boost_round" hyperparameter = 500. Training involved training on the training dateset and finally validating on the validation dataset at the end of training job.

After finishing that training job, I then proceeded to tune the LightGBM model with the following hyperparameter ranges using the sagemaker.tuner.HyperparameterTuner method.

```
hyperparameter_ranges = {
    "learning_rate": ContinuousParameter(1e-4, 1, scaling_type="Logarithmic"),
    "num_boost_round": IntegerParameter(50, 1000),
    "early_stopping_rounds": IntegerParameter(2, 30),
    "num_leaves": IntegerParameter(10, 50),
    "feature_fraction": ContinuousParameter(0, 1),
    "bagging_fraction": ContinuousParameter(0, 1),
    "bagging_freq": IntegerParameter(1, 10),
    "max_depth": IntegerParameter(5, 30),
    "min_data_in_leaf": IntegerParameter(5, 50)
}
```

The best estimator from the tuning job was retained for comparsion to the untuned estimator. The untuned and tuned model binaries can be found in the "trained_models/lightgbm" folder under "untuned_model" and "tuned_model", respectively.

Logistic Regression Training

For the benchmark comparison I used the Sagemaker implementation of the Sci-kit Learn linear classification model. This training was done using the defualt hyperparameters provided by Sagemaker.

Results

In this section I compare and contrast the results from the three training methods described above.

Comparing the Tuned and Untuned LightGBM Models.

The two LightGBM models were compared on the test.csv data set created in the step 02 notebook using the AUC scores which are seen below for each model.

Model	AUC Score
Tuned	0.8389
Untuned	0.8381

The above results are practically identical which means that the hyperparameter tuning as conducted here may have room for further improvement and is saved for future improvements. For now, because these results are so similar, I can choose proceed with the tuned model only for further results and analysis.

The ROC curve for the tuned LightGBM model can be seen below. From this curve you can see a few things. First, the curve looks appropriately convex-upward. A model of random guessing would be a line with slope 1 and an intercept of 0. Becuase the curve is convex-upward and above a "random guess" line, we can conclude that the tuned model did better than randomly guessing. In order to extract the best threshold we can use the Youden's J statistc which gives us a threshold of 0.350969.

Below you can see a figure describing feature importances of the tuned light GBM model (see the table under section "The Final Dataset: Train, Validation, Test Split" for feature names). The top 3 most important features were "years_as_member", "reward", and "difficulty" in that order. I think that this fits my general intuition that tenure of a customer impacts their likeliness to respond to an offer. Additionally, I intuitely believe that "reward" is an important factor because cash rewards are a often-used tactic in marketing campaigns. For example, banks often use premiums to encourage customers to respond to offers just like this Starbuck App. I also understand why "difficulty" (minimum spend) is an important feature because you are asking customers to give up more of their own money in order to complete an offer, something that people are just less inclined to do.

Comparing the Tuned LightGBM and Logistic Regression Models

Below is table of comparing the relevant metrics for the LightGBM and Logistic Regression Models.

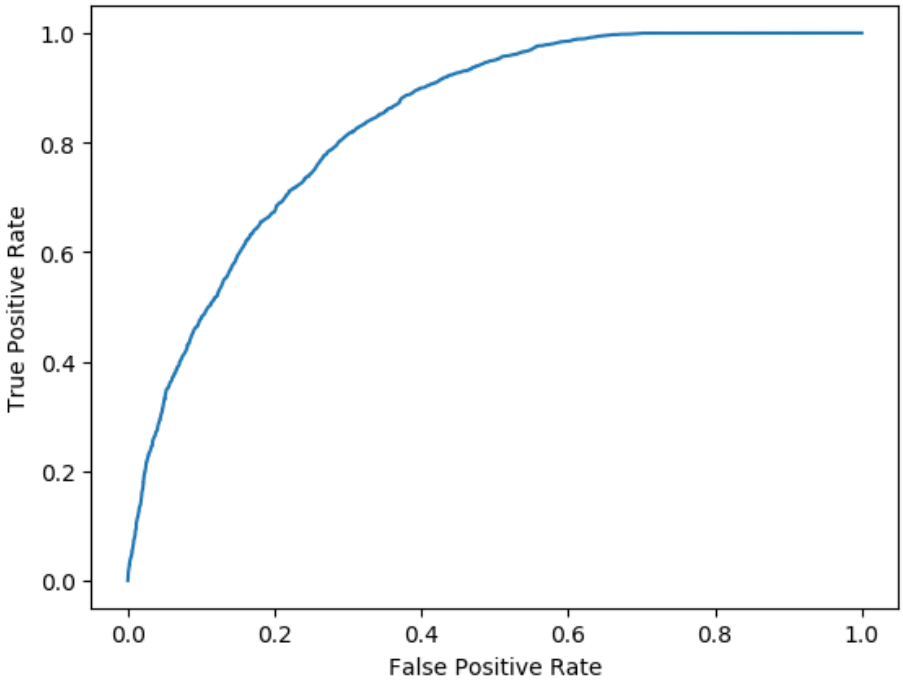

Figure 10: Tuned LightGBM ROC Curve

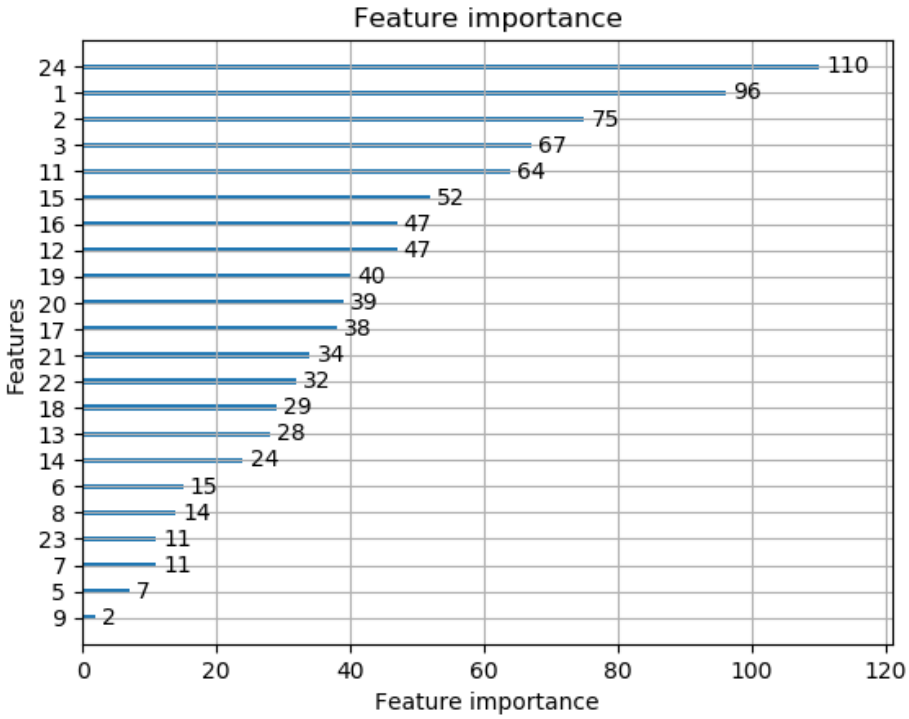

Figure 11: Feature Importance

Model	AUC Score	Accuracy	F1 Score
Tuned LGBM	0.8389	0.7477	0.6433
Logistic Regression	0.8381	0.759	0.5588

The results of these two models are nearly identical, just as we saw above when comparing the tuned and untuned LightGBM models. The differences in accuracy and F1 scores can likely be attributed to the different thresholds used in those calculations between the two models

Endpoint Deployment

Though it was not critical to the purpsoe of this project, I also decided to deploy my tuned LightGBM model to an endpoint. I was able to do this and execute inference on the entire testing dataset, which is a valuable aspect of machine learning engineering operations. The code to do this is shown in the "03_model_training.ipynb" notebook.

Conclusions and Future Improvements

Because all three models had similar AUC scores, I think it's fair to conclude that we've extracted nearly all the predictive power out of the features that I created as they exist now. I don't believe much will be gained by exploring hyperparameter tuning ranges in future iterations because of this. More gains will likely be gotten by creative feature engineering such as adding a feature that is a sum of transactions up to the time that the offer is received. This feature would tell the model something about a customer's willingness to spend money in the Starbucks app. In a real world situation, another simple improvement would be to expand the dataset and collect more records for training.