Udacity Capstone Project Report

Starbucks Capstone Project

Presented by: Syed Muhammad Haider Jafri

I. Definition

A. Project Overview

Starbucks is a coffee company based out of Seattle, WA, with a global presence in over 75 countries and operates over 27,000 stores worldwide.

Since Starbucks does not rely on the franchise model for expansion, it needs to innovate at very high rates to ensure, maintain and increase their profit margins in the cut-throat business of coffee. Starbucks wants to be your “third” place - not just your coffee supplier. (The concept of third place is that there is your home, office, and the “third” place where you could hang out with your family and/or friends. “Third place” used to be the shopping malls in the US, but as they are in decline, Starbucks wants to fill the vacuum and become your “third place”).

Starbucks, therefore, must upsell to existing and loyal customers, and create attractive offerings to non-frequent customers at the same time to increase profit margins.

B. Problem Statement

I started with a very simple problem. **Will the customer respond to an offer?**

This is the basic question from where I started my project. There are a lot of techniques I learned and used throughout the course. All code cells are explained in detail in the Jupyter notebook. Here, I will go over the major steps involved in the process.

The basic idea is to create a dataset which is a combination of all 3 datasets combined, joined together on a single parameter, which in our case was the “offer_id” column. This dataset is called out as “final_production_data” in my model.

In this project, we have to craft product offerings to our customers, both loyal and otherwise, to increase their spending in the Starbucks ecosystem. No customer receives more than 1 offer at any given time.

We will assume the following offerings that we were able to craft;

A. BOGO (buy one, get one free).
B. Discount code.
C. Simple advertisement.
D. No offer for the week.

All the code is available in the Jupyter notebook with detailed explanations. Please check my GitHub repo at https://github.com/mohammadjafri1992/StarbucksMachineLearningProject

<u>Solution Strategy</u>

My basic solution strategy includes using classical machine learning models like Logistic Regression and Random Forest Classification.

I will perform some basic data exploration to look at the features and will try to perform feature engineering on the dataset to reduce the number of features. The lower the number of features, the lower will be the utilization of system resources.

After preparing the final production dataset, I will strip down the label column from the main data and save it as a separate pandas series. Then I will drop the label column from the main dataset which will create my feature dataset. After the creation of these datasets, I will pass the datasets through the models imported from the built-in library of "sklearn".

Applying basic machine learning models first is quick and is very light on the system resources. Therefore, if we have a nominal amount of data, we should always use these models first and avoid the deep learning models because those models will end up overfitting the dataset.

C. Metrics

I solved the problem in such a way that the problem became a classification problem. As mentioned above, my question was "will the customer respond to an offer?", which is a binary classification problem i.e. **yes** or **no**. The label column in the dataset is "offersuccessful" column which is part of the

For classification problems, we have 4 metrics that we can measure.

1. Accuracy,
2. Recall,
3. Precision,
4. F1- score.

An intuitive explanation of these parameters can be extracted from a table called the confusion matrix. It is shown below.

<u>Confusion matrix</u>

GROUND TRUTH	**PREDICTED VALUES**	
	POSITIVE	**NEGATIVE**
POSITIVE	True Positive (TP)	False Negative (FN) a.k.a. Type-II Error
NEGATIVE	False Positive (FP) a.k.a. Type-I Error	True Negative (TN)

All metrics can be calculated from the confusion matrix above.

A. Accuracy = (TP + TN) / (TP + FP + TN + FN)
B. Recall = TP / (TP + FP)
C. Precision = TP / (TP + FN)
D. F1-score = 2 * ((precision * recall) / (precision + recall))

Since F1-score is the harmonic mean of precision and recall, we can eliminate those individual values and just focus on Accuracy and F1-Scores of the problems. F1 score is a harmonic mean and not just simple arithmetic mean because the harmonic mean punishes the model more when it is not very accurate, relative to simple arithmetic mean.

Side note:
If it were to be a regression problem then the accuracy metrics we could have chosen is RMSE (Root Mean Square Error)

II. Analysis

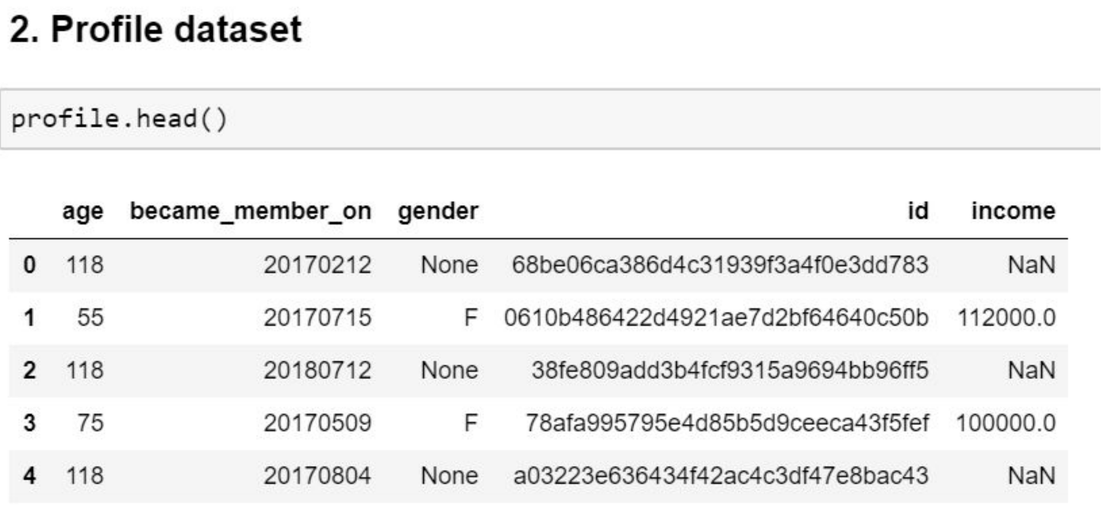
A. Data Exploration

For this project, we are given 3 datasets in JSON for this project, provided by Starbucks.

1. Profile.json

   Rewards program users (17000 users x 5 fields)

   I. gender: (cateforical) M,F,O or null
   II. age: (numeric) missong value encoded as 118
   III. id: (string/hash)
   IV. became_member_on:(date) format YYYYMMDD
   V. income: (numeric)



2. Profile dataset

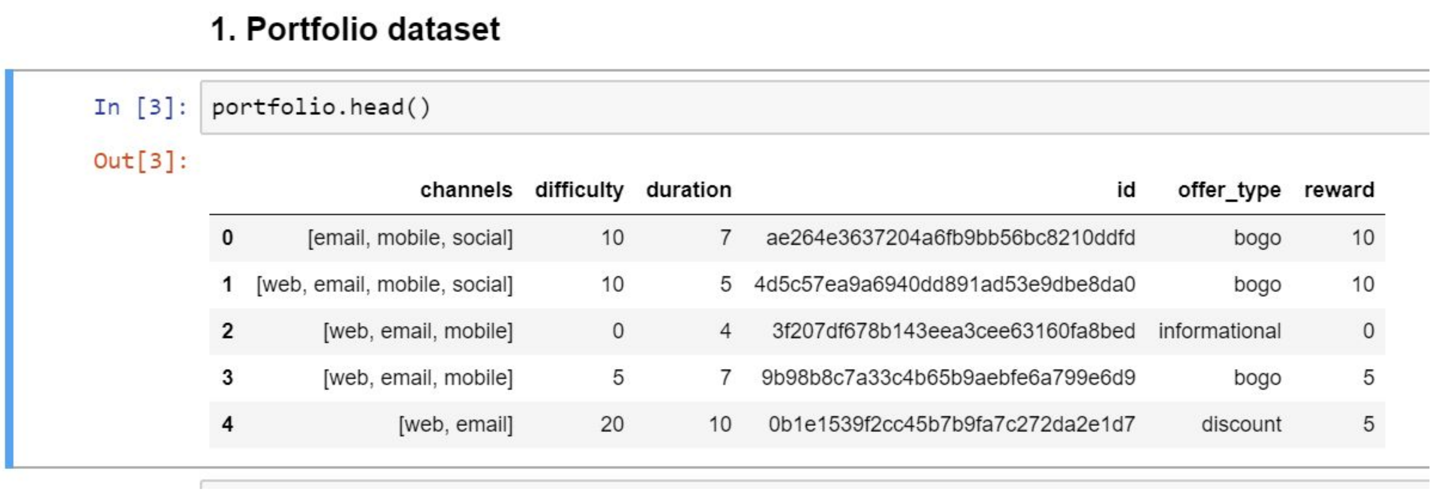
```
profile.head()
```

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

2. Portfolio.json

   Offers sent during the 30-day test period (10 offers x 6 fields)

   a) Rewards: (numeric) money awarded for the amount spent
   b) Channels: (list) web, email, mobile, social
   c) Difficulty: (numeric) money required to be spent to receive rewards
   d) Duration: (numeric) time for the offer to be open, in days
   e) Offer_type: (string) BOGO, discount, informational
   f) id: (string/hash)



1. Portfolio dataset

```
In [3]: portfolio.head()
```

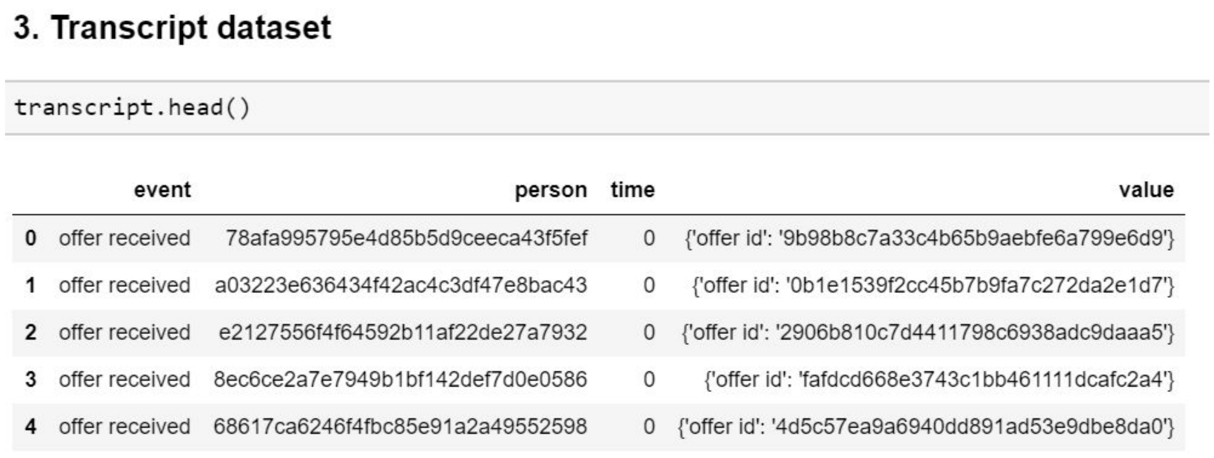
Out[3]:

	channels	difficulty	duration	id	offer_type	reward
0	[email, mobile, social]	10	7	ae264e3637204a6fb9bb56bc8210ddfd	bogo	10
1	[web, email, mobile, social]	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	10
2	[web, email, mobile]	0	4	3f207df678b143eea3cee63160fa8bed	informational	0
3	[web, email, mobile]	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	5
4	[web, email]	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	5

3. Transcript.json

   Event log (306648 events x 4 fields)

   a) Person: (string/hash)
   b) Event: (string) offer received, offer viewed, transaction, offer completed
   c) Value: (dictionary) different values depending on event type
      (1) Offer id: (string/hash) not associated with andy “transaction”
      (2) Amount: (numeric) money spent in “transaction”
      (3) Id: (string/hash)



3. Transcript dataset

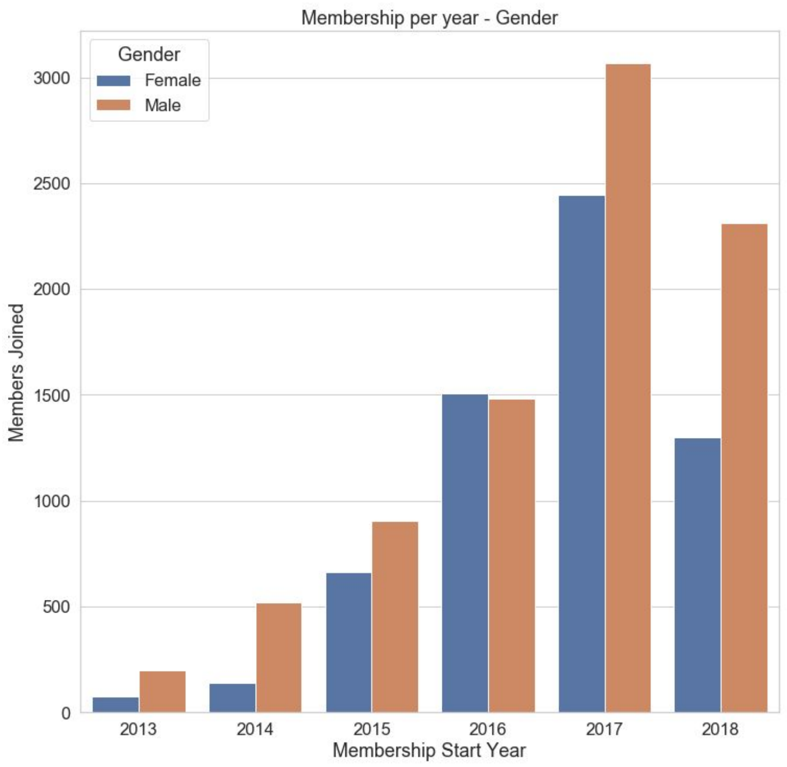
```
transcript.head()
```

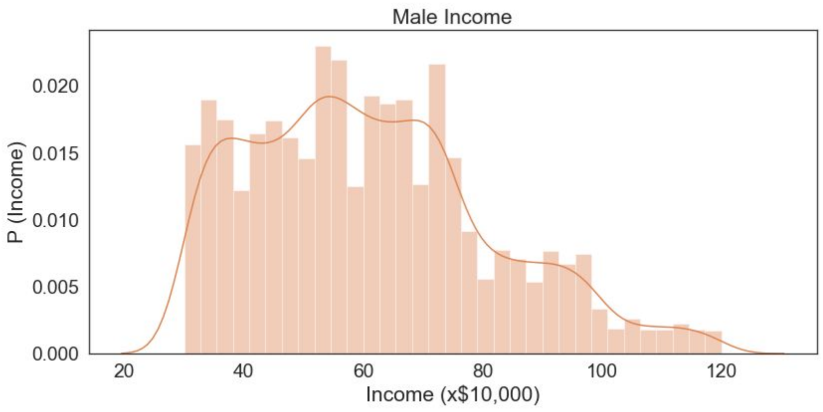
	event	person	time	value
0	offer received	78afa995795e4d85b5d9ceeca43f5fef	0	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}
1	offer received	a03223e636434f42ac4c3df47e8bac43	0	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}
2	offer received	e2127556f4f64592b11af22de27a7932	0	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}
3	offer received	8ec6ce2a7e7949b1bf142def7d0e0586	0	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}
4	offer received	68617ca6246f4fbc85e91a2a49552598	0	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}

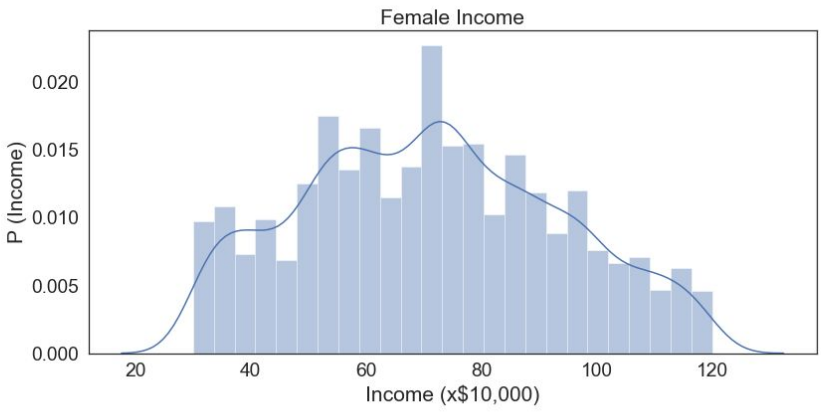
B. Exploratory Visualization

Performing basic data exploration on the provided datasets gives us the following.

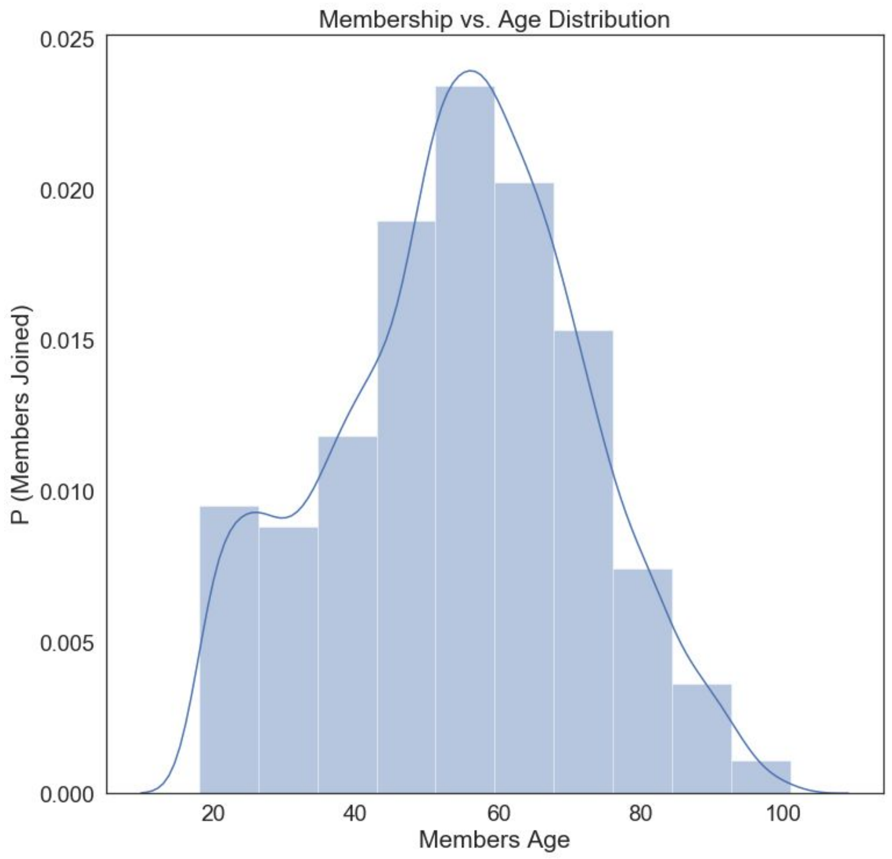
And,


Membership per year - Gender
Gender
Female
Male
Members Joined
Membership Start Year
0
500
1000
1500
2000
2500
3000
2013
2014
2015
2016
2017
2018


Male Income
P (Income)
Income (x$10,000)
0.000
0.005
0.010
0.015
0.020
20
40
60
80
100
120


Female Income
P (Income)
Income (x$10,000)
0.000
0.005
0.010
0.015
0.020
20
40
60
80
100
120

And,



C. Algorithms and Techniques

After working with a lot of machine learning and deep learning models, I have figured out that the best place to start a machine learning project is with a simple classical ML model. The 2 models I used (other than the base/benchmark model) are;

1. Logistic Regression Model
2. Random Forest Classifier

In my experience, the best place to start for a classification problem is a Linear Regression model, which I used first. I applied the model on the prepared dataset and then performed the “prediction” task on the test dataset to get the final model prediction values. I then compared those values against the benchmark model I set up. Logistic Regression model performed much better than the benchmark.

Then I applied the Random Forest Classifier model to the prepared dataset and then performed the “prediction” task on the test dataset to get the final model prediction values. I then compared those values against the benchmark model I set up. Random Forest Classifier model performed even better than both, the benchmark model and the Logistic Regression model.

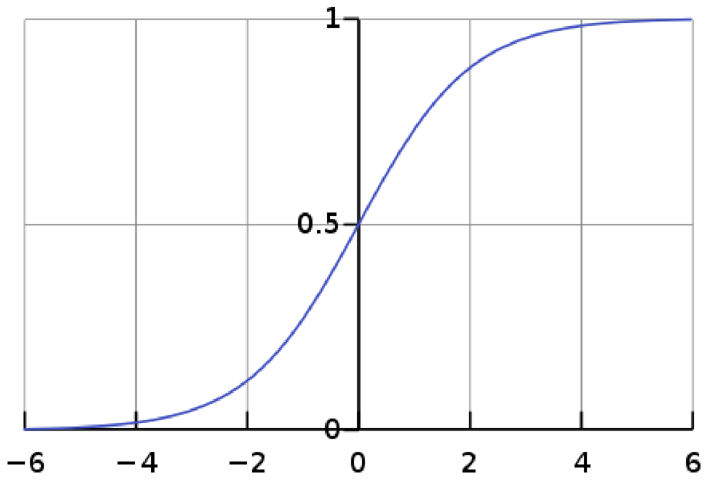
LOGISTIC REGRESSION

Logistic Regression is a binary classification algorithm. The function outputs “yes” or “no”, “true” or “false”, 0 or 1, etc. It sounds confusing because there is “regression” in the name, but it is nevertheless a “classification” algorithm. In its simplest form, logistic regression function (sigma(t)), which is basically an implementation of a sigmoid function (for binary classification) can be expressed as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

where ;
Sigma = Logistic function
e = exponential function
t = real input (t belongs to a set of Real numbers, R)

The logistic regression function is centered around 0, as shown below.



The goal of the Logistic Regression function is to classify the value to be either 0 or 1, which we want in our case because of how we formulated our problem for the project.

How does it work?

The way it works is that after the function has solved one instance, it decided whether the final value is <=0.5. If it is <= 0.5, the function predicts 0, and 1 otherwise.

It is important to note that the Logistic Regression function only models the **statistical classification**, which is **not** the same as a classifier model or predictor. To use it as a classification model, we must perform the final operation of binning the predicted probability of an instance as either 0 or 1 by using “floor” or “ceiling” functions in Python if we are modeling it manually. Since we are using the built-in function of sklearn, this method has already been implemented for us.

Parameters vs Hyperparameters

There is always some confusion about the parameters and the hyperparameters of any model.

In short, the model parameters are the values that the model learns on its own after the training has completed. Model parameters are learned “after” we have set the “hyperparameters”.

These include;

- Weights,
- Biases,
- Coefficients,
- Model Split points, etc.

Model hyperparameters, on the other hand, include the parameters or variables which help in the “learning process” of any model. These are the minor tweaks that we perform during the model training phase to improve the model performance.

Hyperparameters include;

- Parameter distribution,
- No. of parallel jobs,
- No. of iterations,
- Model verbosity,
- Scoring parameters (e.g. fbeta_score with beta in our case < 1 because we wanted to give more weight to “precision”. If beta > 1, the fbeta_score gives more weight to “recall”),
- Type of solver to use,
- Type of optimizer to use, etc.

These variables help in learning the model its parameters which can be used to make future predictions.

Model Extension

Since the Logistic Regression model predicts the probability of an instance, it can be extended and modified to make predictions for multiple classes as well, rather than just 2 classes.

RANDOM FOREST CLASSIFIER

Random forest classifier is an implementation of a popular statistical method called Random Forest.

Random forest is an advanced machine learning method that belongs to a category of ensemble algorithms. This method can be used for both classification and regression models.

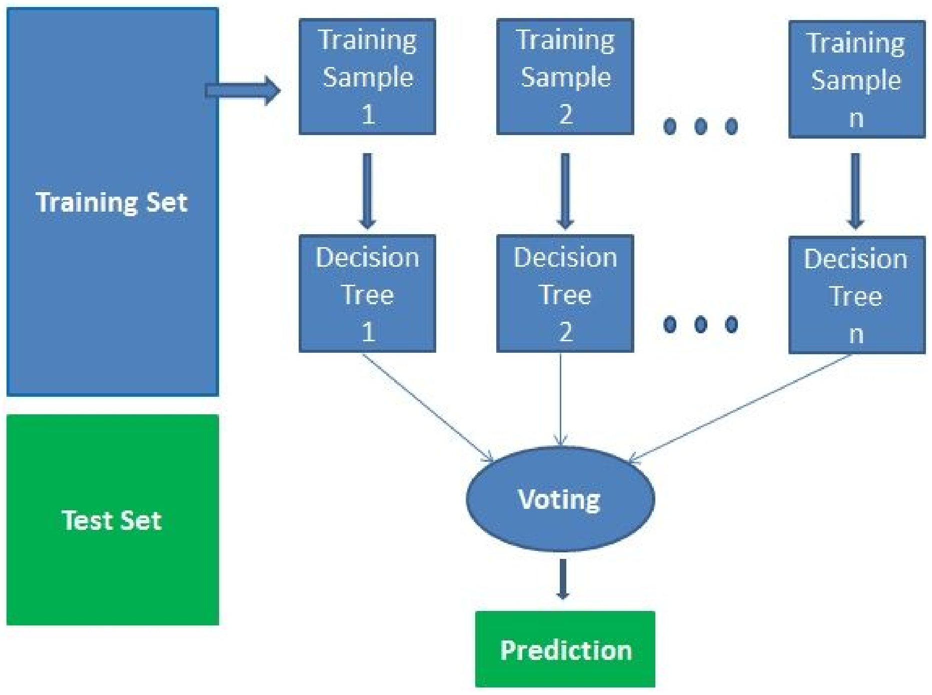
An ensemble model is built out of several learning models, stacked on top of one another or running in parallel, to improve the accuracy of the final model. In our case, the model is built out of several decision trees.

In Random forest classification, any instance of the dataset is classified into a certain category based on the mode of the resultant decision trees. In other words, a classification is performed when there is a consensus among the decision trees that a certain instance should belong to a certain class, based on the data that was learned by the model. Random forest models are used because of their built-in nature to autocorrect for model overfitting.

How does it work?

The model works on the modified bagging model, which is also called the bootstrap aggregating model. This modification step is where the magic of random forest happens. In the training phase, the model randomly selects a subset of the features it trains and applies to all the trees it generated. This sampling is done with replacement. When a tree is built, all of the data is passed through the tree and the representative features are compared with the representative features of other trees. Then another tree is built and the same steps are performed.

The size of the sample parameters selected is usually around 33% of the total parameters.



Unbiased model

Random forest is inherently unbiased. This is because the features used to build a tree are different, and are randomly selected from the dataset. This feature of the model makes it unbiased and it rarely overfits the dataset, if ever at all.

In the random forest model, we do not apply the dropout or other regularizations like L1 and L2.

Handling of missing data

Random forests handle the missing data easily, especially the implementation of random forests in sklearn. Usually, in the training phase, the missing values are replaced with the median of the feature values. This slows down the training a little bit, but this happens automatically for the random forest. This method applied only to numerical features.

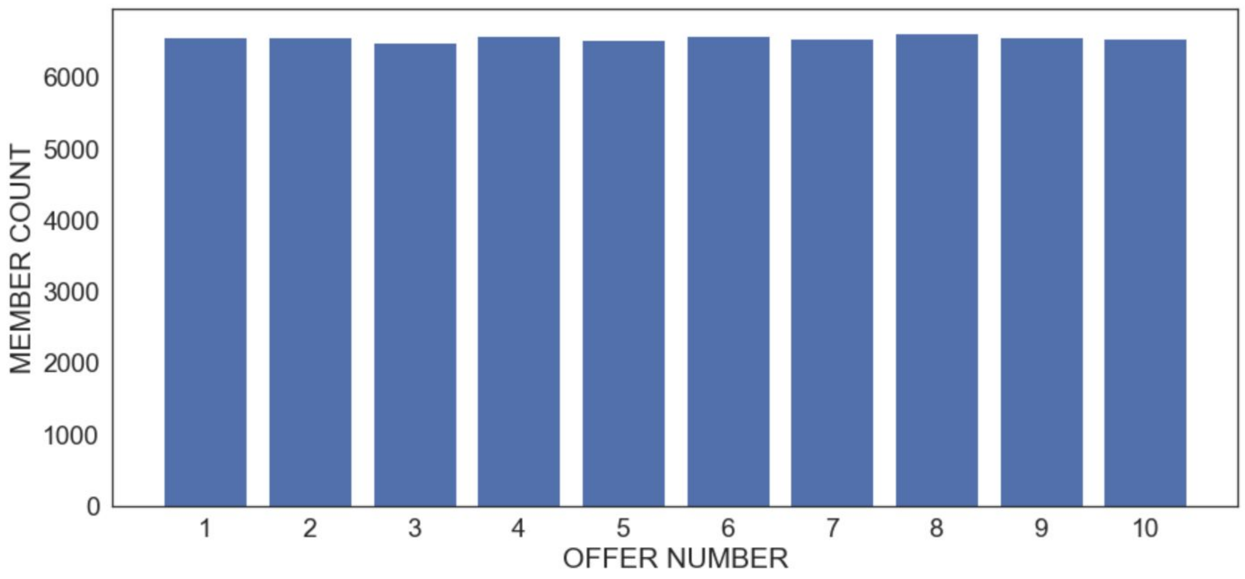
Model Hyperparameters

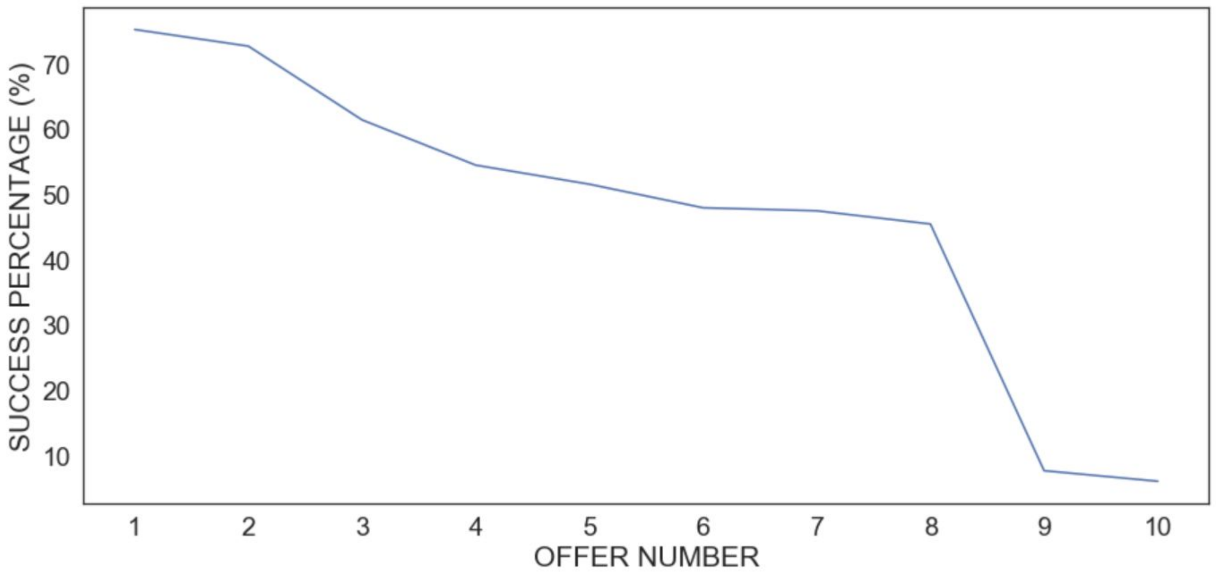
The model hyperparameters used in the Random Forest model creation are as follows;

- Number of estimators
- Maximum depth of a tree
- Minimum number of splits at each node
- Minimum number of leaves at each node
- Scoring parameters (e.g. fbeta_score with beta in our case < 1 because we wanted to give more weight to “precision”. If beta > 1, the fbeta_score gives more weight to “recall”)
- Number of jobs in parallel
- Number of iterations
- Model verbosity

D. Benchmark

Earlier, I had planned to use a clustering algorithm, but during the data exploration process, I decided to make a navie model benchmark which assumes that all the offers go successful. This was then compared against the other two ML models I had selected.





III. Methodology

A. Data Processing

The biggest challenge in the project was the data cleaning part. A lot of data had to be removed from the datasets because of the NULL or NaN values. This stage took the most amount of time.

First of all, I looked at the NULL and NaN values in the datasets and dropped the rows containing any of the values. These rows were useless for the type of analysis I needed to perform. This reduced the number of rows, but we had enough data to perform classical machine learning tasks.

The project was designed in the following way.

In general, we split the training and testing data into 70% - 30%. But depending upon the number of instances we have, we can change this ratio up or down. e.g. if we have 10 million samples, then keeping 3 million samples just for the testing phase makes very little sense. And since we know that the performance of a machine learning model improves with the amount of data we supply, we can get better results by keeping only ~100k examples in testing set and using the rest of the dataset for training.

1. I split the master and cleaned data into 85% training and 15% testing.
2. I selected 2 Machine Learning models,
   a) Logistic Regression
   b) Random Forest Classification
3. I ran Logistic Regression and Random Forest Classification on the data separately and compared them against the naive benchmark model which assumes that all the offers were successful.
4. After performing all the tests, I compared the results of all models and selected the Random Forest Classifier for our production and further tuning.

B. Implementation

During the implementation phase, I performed the following tasks.

1. For individual tasks, all the categorical variables were converted into one hot encoded variable, e.g. “offer_type” in the portfolio dataset.
2. In the same dataset, the feature “channels” was categorical, but could not be converted into the one-hot-encoded columns because one person got the offers through multiple channels. I used the MultiLabelBinarizer module from sklearn.preprocessing module. This would convert such categorical variables into numerical variables.
3. These newly created one-hot-encoded dataframes were then appended to the existing portfolio dataset and the original columns were dropped.
4. Similar feature engineering was performed on the other datasets as well.

5. Finally, all three datasets were appended together on “offer_id”.
6. The data was then explored further for numerous relationships.
7. This dataset was then split into training and test datasets with their respective labels.
8. The results were then compared with the benchmark model and the best performing model was selected.

C. Refinement

One major refinement I could have done to improve the whole project is that I could have created a number of functions to perform the repetitive tasks to reduce the size of the notebook.

I also could have combined all the functions in individual files to make the code a little bit cleaner.

I could have filled out the NaN and NULL values by some other numbers, but that would have compromised the integrity of the data. Since these values were not a lot, therefore I kept most of them.

One improvement I could have performed was tuning the hyperparameters even more. Although I tuned the model a lot and improved its performance by ~5%, the performance then flattened after a while. Spending more time with the models might have improved their performance.

IV. Results

A. Model Evaluation and Validation

In order to measure the performance of our models, we should first set our benchmark, against which we will measure the performance of our models.

This benchmark can be either;

a - How humans make predictions, OR
b - How a naive system would make predictions.

Since we do not have any insights on how a human would predict the outcome of any offer, we will create a naive prediction model.

For our naive model, we can import "accuracy_score" and "f1_score" from sklearn and try to run it on the y_train.
This will give us something to work with, and we will benchmark our models on the outcome of this model.

1. **Logistic Regression**

Using a simple Logistic Regression Model on our dataset has performed exceedingly well.

We had the following performance metrics for the **naive model**;

1. Accuracy - 46.8%
2. F1 Score - 63.8%

And with the **Logistic Regression Model,** we have;

1. Accuracy - 72.1%
2. F1 Score - 71.4%

This clearly shows that out of the two models, we should clearly use Logistic Regression.

2. **Random Forest Classifier**

Using Random Forest Model on our dataset has performed the best so far.

We had the following performance metrics for the **naive model**;

1. Accuracy - 46.8%
2. F1 Score - 63.8%

And with the **Logistic Regression Model**, we have;
1. Accuracy - 72.1%
2. F1 Score - 71.4%

And with **Random Forest Model**, we have;
1. Accuracy - 74.2%
2. F1 Score - 73.4%

This clearly shows that out of all the models, Random Forest Classifier works the best.

In my experience, I have seen that Random Forest Classifiers have worked best for classical machine learning problems, especially when we do not have a lot of data. If we have a lot of data available, then we can use Deep Learning Models which "tend" to perform better than classical machine learning models. Also, the definition of a "lot" of data varies with the scope of the problem to be solved.

B. Justification

Depending on the nature of the underlying data, there are some models that perform better on any given dataset. This is not a one-size-fits-all scenario.

Deep learning models only work well if we have a lot of data. If we had run the DL model here, it would have created an overfit model which would be very bad for generalization.

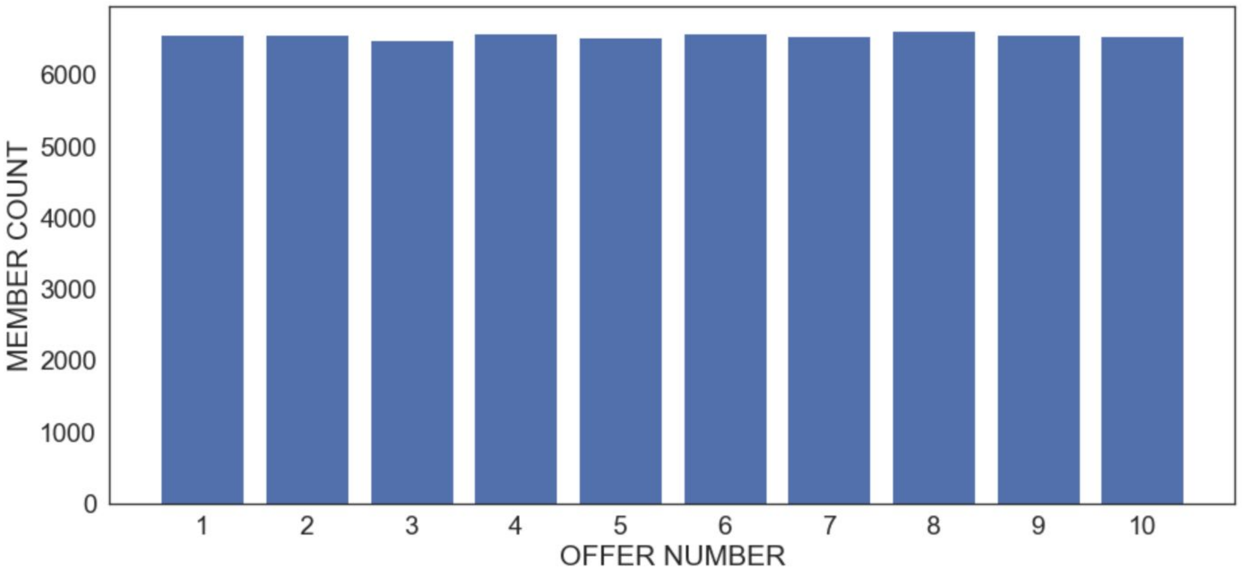
In theory, I could have explored other models as well but in the interest of time, I only used the two models. Other supervised learning classification models might have performed better.

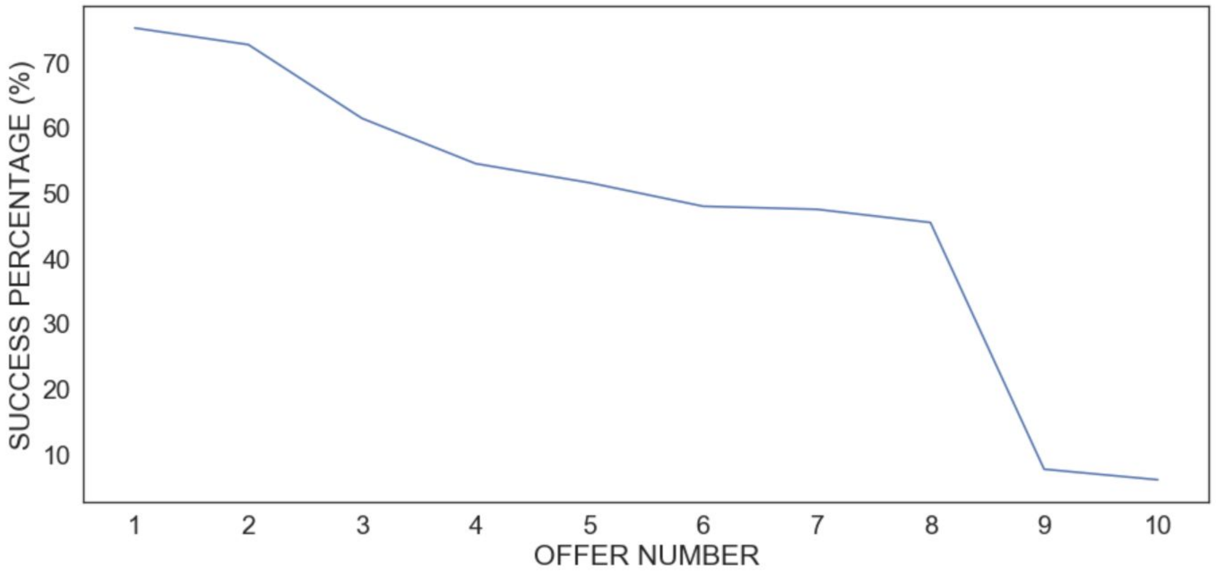
V. Conclusion

A. Free-Form Visualization

The most important quality of the dataset is the thorough nature of the features given to us. In the real world projects, the datasets which are needed for machine learning applications are in various tables and we have to perform feature engineering, data cleaning and EDA on all datasets.

In the datasets, the fact that the least popular types of offers were informational, it was cleverly hidden and I had to explore it by appropriately wrangling the data in the Jupyter notebooks.





From the plots above, we can observe the following things;

1. Almost all offers were provided equally to ~6600 users.

2. The effectiveness or success of the offers gradually reduced, i.e. the first 6 types of offers were the most successful, while the offers 9 and 10 were the least successful offers.

The 2 least successful offers are "informational", which is no surprise since I personally will not consider the informational offers to be "offers" per-se. Customers would consider it to be a memo from the company and are least likely to go buy the product immediately after they learned about the product. Considering the nature of Starbucks customers, we know that they are loyal, are more affluent, and want to be confident of their purchase. With only the informational offers, the confidence part seems to be a bit off, hence the offer completion drops off.

B. Reflection

I selected "accuracy" and "F1-score" to be the benchmark metrics because first of all, it is a classification problem.
Second, the F1-score is a harmonic mean of "precision" and "recall", which eliminates the need to benchmark those metrics individually.

In the model of choice, Random Forest Classifier, if I had more data then I could have trained the model for even longer and fine-tuned the hyperparameters even more. Because machine learning models depend on a lot of data, I might have been able to improve the F1 scores and accuracy.

Finally, this is not the final form of the project/model in any way. I could have spent a LOT more time on making different combinations and refining the model but in the interest of time, I am stopping here.

There is a fine line between doing the project "enough" and getting something out of it, vs. "mastering" the project and getting the best result "after" it was needed. This line is arbitrary and varies for all projects. As a machine learning engineer, we must keep this line in mind all the time.

C. Improvement

One major improvement I could have done to improve the whole project is that I could have created a number of functions to perform repetitive tasks to reduce the size of the notebook.

I also could have combined all the functions in individual files to make the code a little bit cleaner.

I could have filled out the NaN and NULL values by some other numbers, but that would have compromised the integrity of the data. Since these values were not a lot, therefore I kept most of them.

One improvement I could have performed was tuning the hyperparameters even more. Although I tuned the model a lot and improved its performance by ~5%, the performance then flattened after a while. Spending more time with the models might have improved their performance.

In the model of choice, **Random Forest Classifier**, if I had more data then I could have trained the model for even longer and fine-tuned the hyperparameters even more. Because machine learning models depend on a lot of data, I might have been able to improve the F1 scores and accuracy.

Finally, this is not the final form of the project/model in any way. I could have spent a LOT more time on making different combinations and refining the model but in the interest of time, I am stopping here.

There is a fine line between doing the project "enough" and getting something out of it, vs. "mastering" the project and getting the best result "after" it was needed. This line is arbitrary and varies for all projects. As a machine learning engineer, we must keep this line in mind all the time.

For details on the code execution, please take a look at the Jupyter notebook as it explains each cell in detail.