Starbucks Capstone Project

Stephen Blystone

Udacity ML Engineer Nanodegree | December 2019

Table of Contents

Introduction 3
Problem Domain 3
Problem Statement 3
Datasets and Inputs 3
Data Cleaning and Feature Engineering 6
Exploratory Data Analysis 7
Preparing the Data 13
Evaluation Metrics 15
Models Used: 17
Hyperparameter Tuning 21
Conclusion 24
Future Improvements 24
References 24

This full report can also be found as a Medium article here.

Introduction

This is my capstone project for the Udacity Machine Learning Engineer Nanodegree. The full source code can be found on my GitHub. Udacity partnered with Starbucks to provide a real-world business problem and simulated data mimicking their customer behavior.

Problem Domain

As a father of twin toddlers, I am a frequent Starbucks customer. When they wake up in the middle of the night, I need caffeine to function the next day. Starbucks has a Rewards program that allows me to earn points for purchases. There is also a phone app for their Rewards program where they send me exclusive personalized offers based on my spending habits.

This project is focused on tailoring those personalized offers to the customers who are most likely to use them. The Machine Learning terminology for this is "propensity modeling".

"Propensity models are often used to identify the customers most likely to respond to an offer". -Gary Childs (https://www.campaignlive.co.uk/article/propensity-model/165289)

Problem Statement

We want to determine which kind of offer, if any, to send to each customer based on their purchases and interaction with the previously sent offers. Some customers do not want to receive offers and might be turned off by them, so we want to avoid sending offers to those customers.

Datasets and Inputs

For this project I will be using the dataset provided by Starbucks. The data consists of 3 files containing simulated data that mimics customer behavior on the Starbucks Rewards mobile app. It was captured over a 30-day period.

1. **Portfolio.json** contains information about the offers. There are a total of 10 offers. This file contains the following columns:
   - The ***reward*** given in dollars for completing an offer.

- The ***channels*** (email, mobile, social, web) the offer was sent on.
- The ***difficulty***, i.e. the minimum dollar amount the customer must spend to complete an offer.
- The ***duration*** the offer lasts in days.
- The ***offer type*** (BOGO, discount, or informational).
- The unique ***id*** for the specific offer.

	reward	channels	difficulty	duration	offer_type	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 - Complete Portfolio Data

2. **Profile.json** contains demographic information about the customers. There are a total of 17,000 customer profiles listed. The columns are:

- The ***gender*** of the customer. This can be M (Male), F (Female), O (Other), or None if they did not provide a gender.
- The ***age*** of the customer. If no age is provided the birth year defaults to 1900. Since this data is from 2018, the default age is 118.
- The ***id*** of the specific customer.
- The date they ***became a member on*** in format YYYYMMDD.
- The ***income*** of the customer, or NaN if none provided.

	gender	age	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 2 - First 5 rows of customer information in Profile

3. **Transcript.json** contains information about a customer's purchases and their interaction with the offers. There are 306,534 events recorded. The columns are:

- The ***person***'s unique customer id.
- The type of ***event***. Either a *"transaction"* if the customer made a purchase, *"offer received"* if they received an offer, *"offer viewed"* if they viewed the offer, or *"offer completed"* if they successfully completed an offer.
- The ***value*** is a dictionary with values depending on the type of event. It contains the 'amount' spent if *"transaction"* event, the 'offer id' if *"offer received"* or *"offer viewed"*, or the 'offer id' and 'reward' if *"offer completed".*
- The ***time*** is the time in hours since the start of this test. This value ranges between 0 and 714 (i.e. ~30 days)

	person	event	value	time
44528	855a636c057e4a32aa50b7d39f24d769	transaction	{'amount': 33.81}	114
55541	2cfd7c204f744227b147546e36c961b0	offer received	{'offer id': 'fafdcd668e3743c1bb461111dcafc2a4'}	168
66092	cef9b883c0c947e88659c9eb361b3250	transaction	{'amount': 36.95}	168
91878	3798869304074ff1b1bb1cbf1b352676	transaction	{'amount': 3.38}	234
132694	ac3e5e1cf92a4cbe84636087fa45204a	offer completed	{'offer_id': '2298d6c36e964ae4a3e7e9706d1fb8c2...	348

Figure 3 - Random sampling of 5 events in Transcript

I confirmed that the dataset is balanced by looking at the value counts for all Transcript ***events***:

```
transcript.event.value_counts()

transaction        138953
offer received      76277
offer viewed        57725
offer completed     33579
```

Figure 4 - Transcript event value counts to confirm dataset is balanced

We are primarily concerned with whether a customer completes an offer they have received. To determine the percentage completed we can use the following equation:

$$\frac{76{,}277 - 33{,}579}{76{,}277} = \frac{42{,}698}{76{,}277} = 55.97\%$$

Figure 5 - Calculation to determine if dataset is balanced

This means that 55.79% of the customers completed their offers, while 44.03% received offers but did not complete them. These percentages are close enough to consider this a balanced dataset.

Data Cleaning and Feature Engineering

1. **Portfolio Dataset**

- One-Hot Encode the ***channels*** column.
- One-Hot Encode the ***offer_type*** column.
- Rename ***id*** to ***id_offer***.
- Reorder columns so ***id_offer*** is first, for display purposes only.

	id_offer	reward	difficulty	duration	email	mobile	social	web	offer_informational	offer_bogo	offer_discount
0	ae264e3637204a6fb9bb56bc8210ddfd	10	10	7	1	1	1	0	0	1	0
1	4d5c57ea9a6940dd891ad53e9dbe8da0	10	10	5	1	1	1	1	0	1	0
2	3f207df678b143eea3cee63160fa8bed	0	0	4	1	1	0	1	1	0	0
3	9b98b8c7a33c4b65b9aebfe6a799e6d9	5	5	7	1	1	0	1	0	1	0
4	0b1e1539f2cc45b7b9fa7c272da2e1d7	5	20	10	1	0	0	1	0	0	1
5	2298d6c36e964ae4a3e7e9706d1fb8c2	3	7	7	1	1	1	1	0	0	1
6	fafdcd668e3743c1bb461111dcafc2a4	2	10	10	1	1	1	1	0	0	1
7	5a8bc65990b245e5a138643cd4eb9837	0	0	3	1	1	1	0	1	0	0
8	f19421c1d4aa40978ebb69ca19b0e20d	5	5	5	1	1	1	1	0	1	0
9	2906b810c7d4411798c6938adc9daaa5	2	10	7	1	1	0	1	0	0	1

Figure 6 - Complete Portfolio after cleaning

2. **Profile Dataset**

- Remove customers with missing data (***gender*** is None, ***age*** is 118, and ***income*** is NaN). I verified that whenever a customer left one of these optional fields blank, they left all three of them blank.
- Calculate number of days the customer has been a member.
- Store the year the customer became a member on (temporarily for graphing purposes).

- One-Hot Encode the customer’s ***age*** into buckets by decade (10 to 19, 20 to 29, 30 to 39, etc.)
- One-Hot Encode the customer’s ***gender***.
- Rename ***id*** to ***id_customer***.
- Reorder columns so ***id_customer*** is first, for display purposes only.

	id_customer	income	membership_total_days	membership_year	age_[10, 20)	age_[20, 30)	age_[30, 40)	age_[40, 50)	age_[50, 60)	age_[60, 70)	age_[70, 80)	age_[80, 90)	age_[90, 100)	age_[100, 110)	gender_F	gender_M	gender_O
1	0610b486422d4921ae7d2bf64640c50b	112000.0	891	2017	0	0	0	0	1	0	0	0	0	0	1	0	0
3	78afa995795e4d85b5d9ceeca43f5fef	100000.0	958	2017	0	0	0	0	0	0	1	0	0	0	1	0	0
5	e2127556f4f64592b11af22de27a7932	70000.0	606	2018	0	0	0	0	0	1	0	0	0	0	0	1	0
8	389bc3fa690240e798340f5a15918d5c	53000.0	682	2018	0	0	0	0	0	1	0	0	0	0	0	1	0
12	2eeac8d8feae4a8cad5a6af0499a211d	51000.0	772	2017	0	0	0	0	1	0	0	0	0	0	0	1	0

Figure 7 - First 5 rows of cleaned Profile

3. **Transcript Dataset**

- Rename ***person*** to ***id_customer.***
- One-Hot Encode ***events.***
- Get *‘offer id’* from ***value*** column dictionary and place in new column ***id_offer***.
- Get *‘amount’* from ***value*** column dictionary and place in new column ***trans_amt***.

	id_customer	time	event_offer_viewed	event_transaction	event_offer_received	event_offer_completed	id_offer	trans_amt
306527	24f56b5e1849462093931b164eb803b5	714	0	0	0	1	fafdcd668e3743c1bb461111dcafc2a4	NaN
306529	b3a1272bc9904337b331bf348c3e8c17	714	0	1	0	0	NaN	1.59
306530	68213b08d99a4ae1b0dcb72aebd9aa35	714	0	1	0	0	NaN	9.53
306531	a00058cf10334a308c68e7631c529907	714	0	1	0	0	NaN	3.61
306532	76ddbd6576844afe811f1a3c0fbb5bec	714	0	1	0	0	NaN	3.53

Figure 8 - Last 5 rows of cleaned Transcript

Exploratory Data Analysis

Below are the things I observed when exploring the data:

1. **Portfolio** Exploration

- Every offer was sent via the email ***channel***, in addition to other channels.
- Informational ***offer types*** are never “completed” since they have no ***difficulty***.

2. **Profile** Exploration

- The three optional fields for users when creating a profile are ***gender***, ***age***, and ***income***. I was concerned that customers may have provided some information but left other fields default (ex: they enter ***gender*** and ***age*** but leave ***income*** default). I verified that the whenever a field was left default, they were all left default. There are 2175 customers that left these optional fields default.

- The youngest customer ***age*** is 18. The oldest actual customer ***age*** (not the default 118) is 101 years old.
- The earliest ***membership*** is July 29, 2013 and the most recent ***membership*** is July 26, 2018.
- This graph shows the number of memberships per year. Note that the data only contains information up to July 26, 2018, so the 2018 data is a little over half complete.

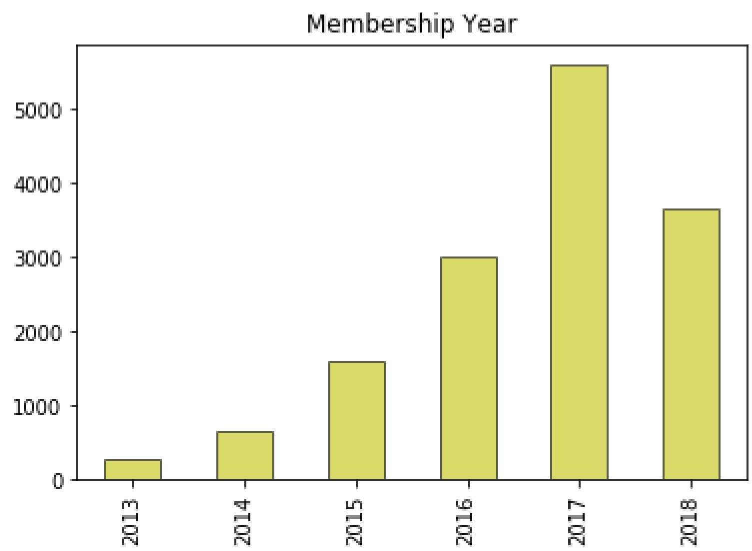

Figure 9 - Profile Memberships Per Year (2018 is partial year)

- This pie chart shows the distribution of ***gender*** (Male, Female, Other):

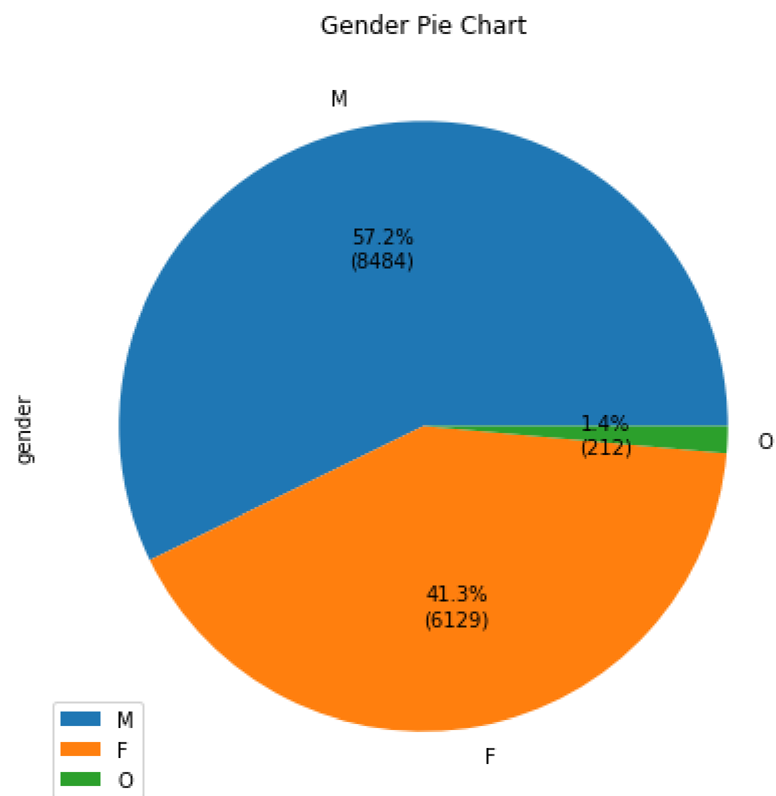

Figure 10 - Profile Gender Percentages and Counts

- This graph shows the ***age*** distribution with mean 54 and standard deviation 17:

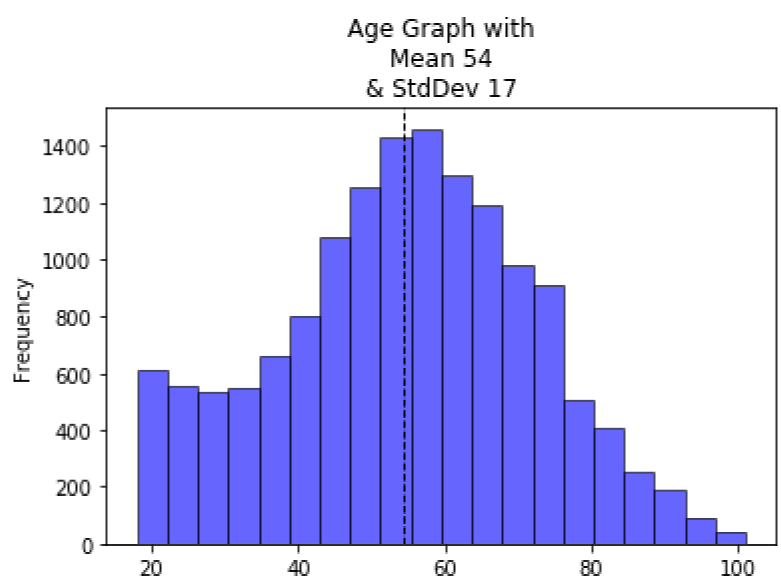

Figure 11 - Profile Age Distribution

- This graph shows the ***income*** distribution with mean $65,404.99 and standard deviation $21,598.30:

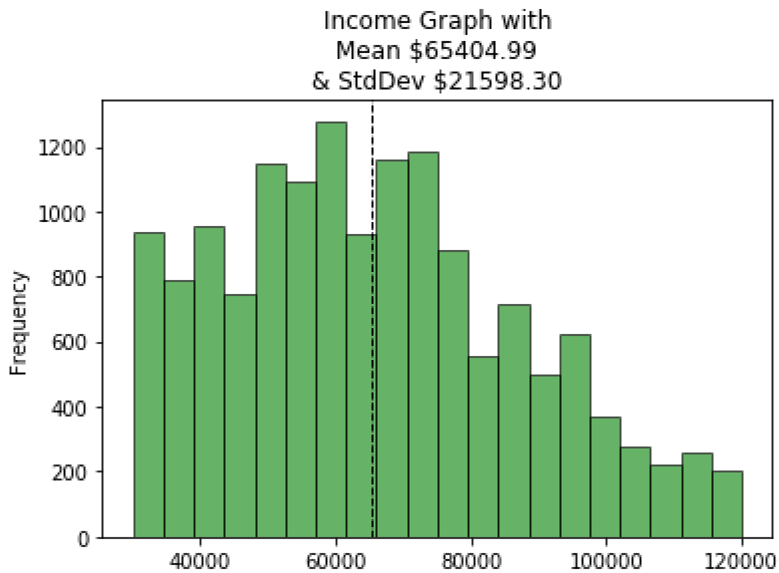

Figure 12 - Profile Income Distribution

3. **Transcript** Exploration

- This pie chart shows the distribution of ***events***:

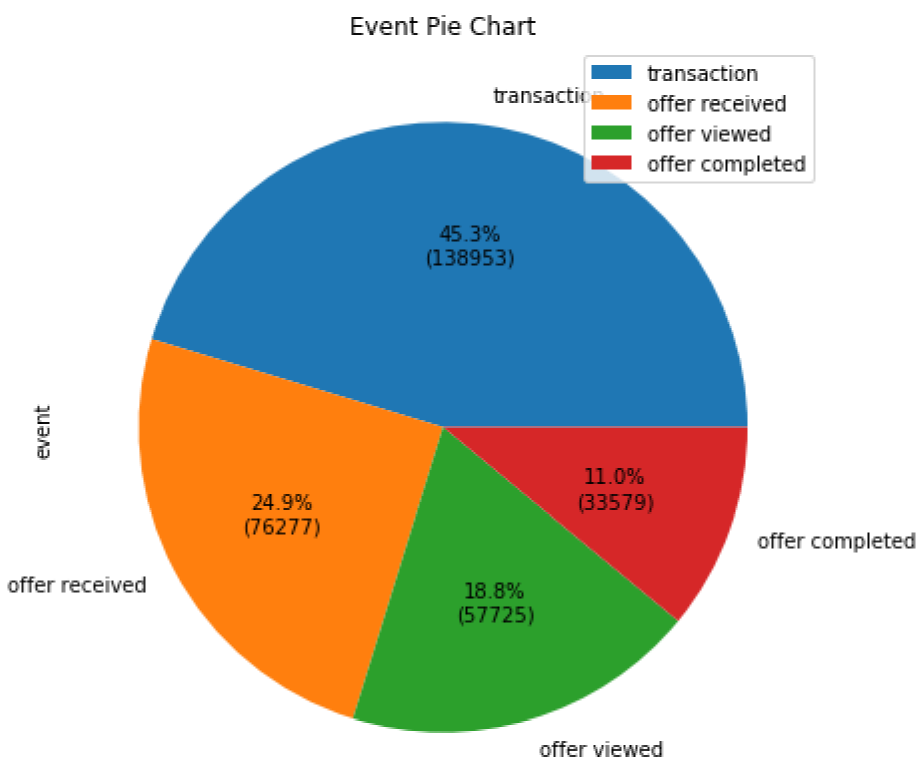

Figure 13 - Transcript Event Percentages and Counts

- Every customer received at least 1 offer, and some customers received up to 7 offers.
- Some customers received the same offer multiple times.
- No customers received multiple offers at the same timestamp.
- There were many instances where a customer would receive another offer before the previous offer expired, resulting in multiple offers being active at the same time.

4. **Combined** Exploration

These are things observed after combining the above 3 datasets:

- Below is a graph of the % of Offer Events per Gender. "Received" ***events*** were fairly even across all genders. The majority of all genders "viewed" the ***event***. Roughly half of Other and Female "completed" the ***event***, while less than half of Males "completed".

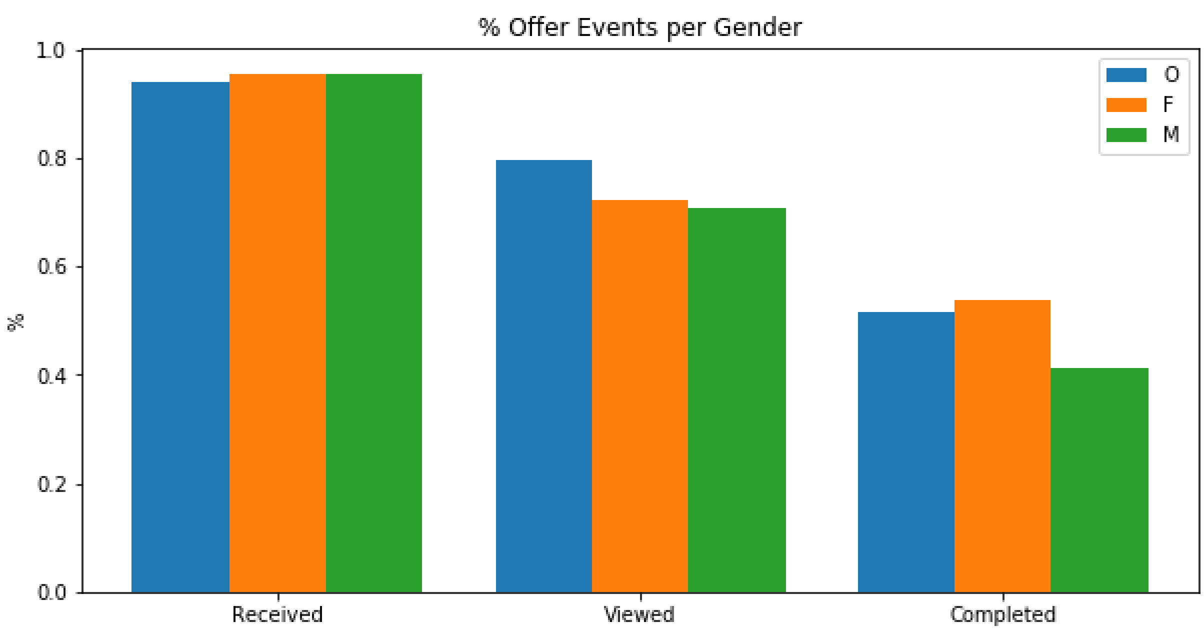

Figure 14 - Percent Offer Events per Gender

- Below is a graph of the % of Offer Events per Age Group. The interesting thing is that the younger age groups did not complete as many offers as the older age groups did. The number completed is also roughly linearly increasing percentage-wise from age buckets 18 through 59.

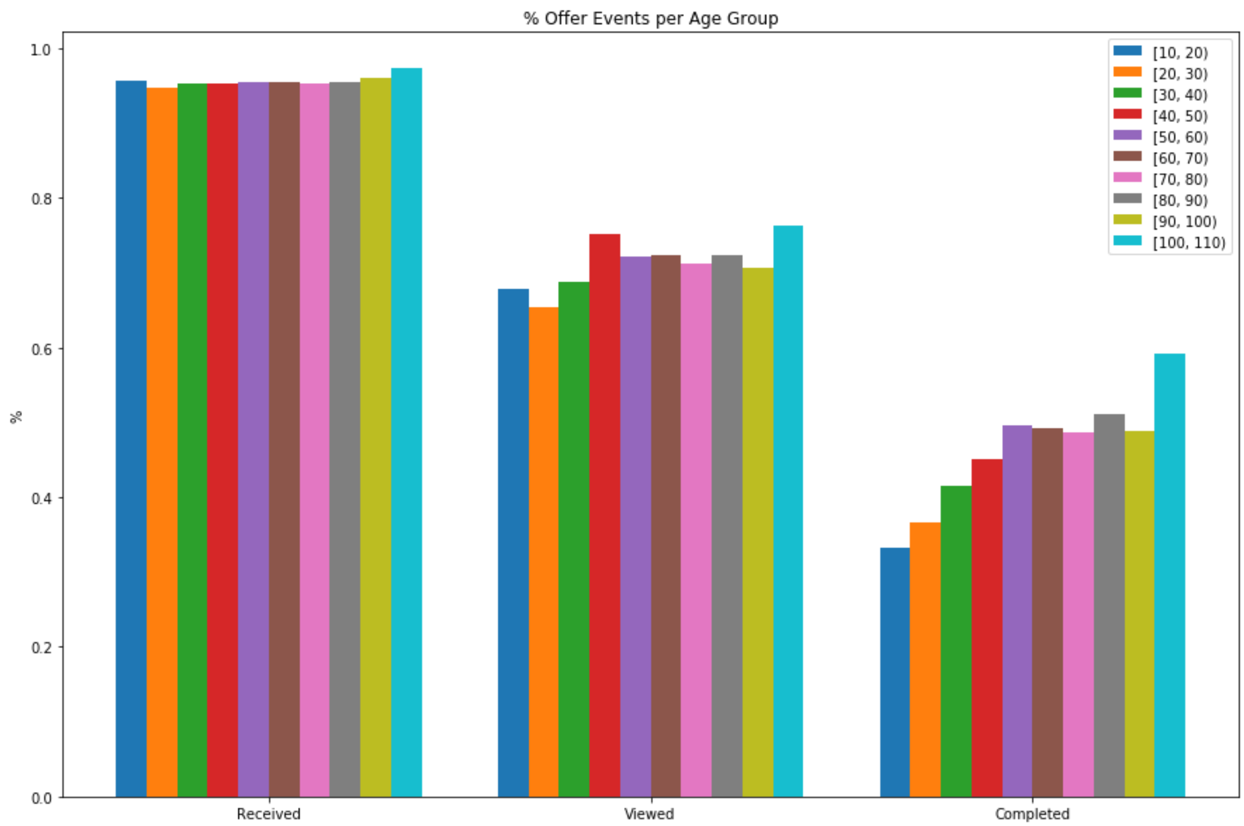

Figure 15 - Percent Offer Events per Age Group

- Below is a graph of the % of money each gender spent towards an offer versus how much they spent while no offer was active. This shows a nearly double increase in spending when offers are active.

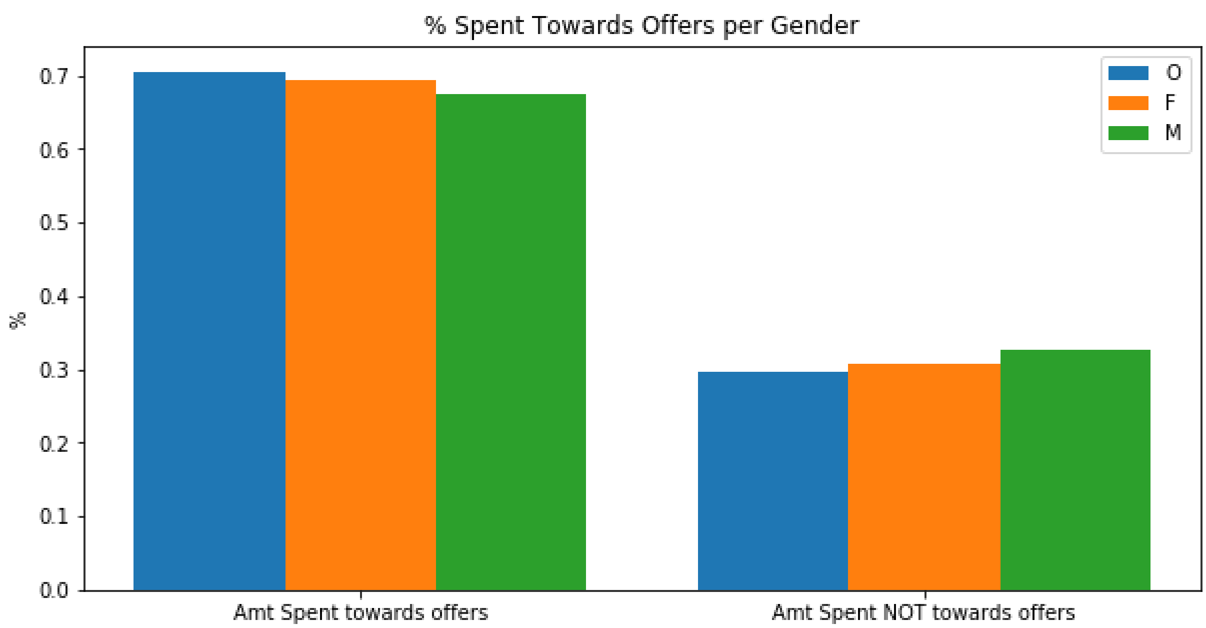

Figure 16 - Amount Spent towards Offers vs. Not towards Offers for Gender

- Below is a graph of the % of money each age group spent towards an offer versus how much they spent while no offer was active. Similar to above, there is a roughly double increase in spending when offers are active.

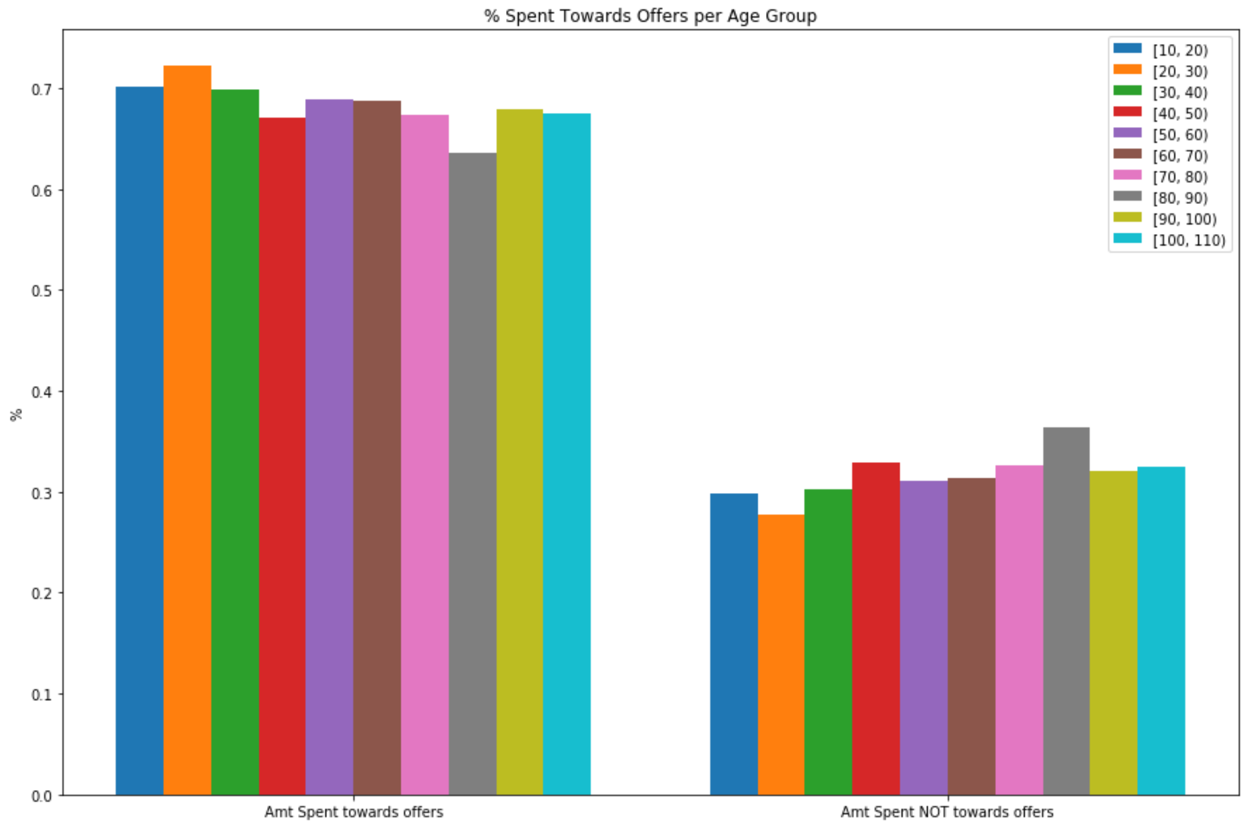

Figure 17 - Amount Spent towards Offers vs. Not towards Offers for Age Group

Preparing the Data

1. **Combining the Data**

Some of the columns in the Transcript data needed additional feature engineering in order to prepare for building the models. Columns like ***time***, ***event_transaction***, ***event_offer_received***, ***event_offer_viewed***, ***event_offer_completed***, and ***trans_amt*** needed aggregated into entries per ***id_customer*** and ***id_offer*.** My models will focus on customer and offer-related data rather than time or individual event-related data.

I created a nested dictionary data structure and iterated over the Transcript data capturing the following for each ***id_customer***:

- For each offer received: the 'num_times_received', the 'num_times_viewed', the 'num_times_completed', and 'total_amt_spent_towards_offer'.

- Also the ‘total_amt_spent_not_in_offer’ to capture all money spent when no offers were active.

This resulted in the following columns added to the data for additional analysis:

- ‘amount_spent_not_in_offer’
- ‘num_times_received’
- ‘num_times_viewed’
- ‘num_times_completed’
- ‘total_amt_spent_towards_offer’
- ‘avg_amt_spent_towards_offer’

After creating these new columns, I could safely remove the ***time***, ***event_transaction***, ***event_offer_received***, ***event_offer_viewed***, ***event_offer_completed***, and ***trans_amt*** columns from Transcript.

I then merged the Transcript, Portfolio, and Profile data into a Combined dataframe.

	id_customer	id_offer	num_times_received	num_times_viewed	num_times_completed	total_amt_spent_towards_offer	avg_amt_spent_towards_offer	reward	difficulty	duration	...	age_[50, 60)	age_[60, 70)	age_[70, 80)	age_[80, 90)	age_[90, 100)	age_[100, 110)	gender_F	gender_M	gender_O	amount_spent_not_in_offer
0	78afa995795e4d85b5d9ceeca43f5fef	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	1	1	37.67	37.67	5.0	5.0	7.0	...	0	0	1	0	0	0	1	0	0	23.93
1	e2127556f4f64592b11af22de27a7932	2906b810c7d4411798c6938adc9daaa5	1	1	0	0.00	0.00	2.0	10.0	7.0	...	0	1	0	0	0	0	0	1	0	39.31
2	389bc3fa690240e798340f5a15918d5c	f19421c1d4aa40978ebb69ca19b0e20d	2	2	2	20.80	10.40	5.0	5.0	5.0	...	0	1	0	0	0	0	0	1	0	0.00
3	2eeac8d8feae4a8cad5a6af0499a211d	3f207df678b143eea3cee63160fa8bed	1	0	0	0.00	0.00	0.0	0.0	4.0	...	1	0	0	0	0	0	0	1	0	0.00
4	aa4862eba776480b8bb9c68455b8c2e1	0b1e1539f2cc45b7b9fa7c272da2e1d7	1	1	0	12.33	12.33	5.0	20.0	10.0	...	0	1	0	0	0	0	1	0	0	0.00

Figure 18 - The Combined Dataframe after merging Transcript, Portfolio, and Profile.

2. **Preparing the Data for the Models**

At this point, all of our data is based on each ***id_customer*** / ***id_offer*** pair. In order for the data to be ready for the models, I added/removed the following columns:

- Added a column to indicate if the specific offer was “successful” or not. An offer is “successful” if the customer completed it at least once.
- Removed all “Informational” rows, since they are never completed.
- Removed rows where ***id_offer*** is NaN. These occur when a transaction ***event*** occurs outside of an offer.
- Removed the ***id_customer*** column, since we want the models to generalize to look at the offer as a whole.
- Removed the ***num_times_received***, ***num_times_viewed***, ***num_times_completed***, ***membership_year***, and ***amount_spent_not_in_offer*** as these were only used for creating graphs.

- Removed the ***email*** column, since every offer is sent via email.
- Removed ***avg_amt_spent_towards_offer*** and ***total_amt_spent_towards_offer*** since we only care that they completed the offer at least once.

My final Combined column list is:

```
combined.columns
```

```
Index(['id_offer', 'reward', 'difficulty', 'duration', 'mobile', 'social',
       'web', 'offer_bogo', 'offer_discount', 'income',
       'membership_total_days', 'age_[10, 20)', 'age_[20, 30)', 'age_[30, 40)',
       'age_[40, 50)', 'age_[50, 60)', 'age_[60, 70)', 'age_[70, 80)',
       'age_[80, 90)', 'age_[90, 100)', 'age_[100, 110)', 'gender_F',
       'gender_M', 'gender_O', 'offer_successful'],
      dtype='object')
```

Figure 19 - List of Columns in my Final Combined Dataframe.

An important data preparation step is scaling the data to be between 0 and 1. This is to prevent the model from incorrectly assigning importance to one column with extremely large values (such as income in the tens of thousands) over another column with much smaller values (such as duration which is 10 days or less).

To scale the data between 0 and 1, I used Scikit Learn's MinMaxScaler. I scaled the ***reward***, ***difficulty***, ***duration***, and ***income*** columns in my Combined dataset.

I then created my Training/Testing/Validation datasets. I used Scikit Learn's train_test_split to create a 60/20/20 split.

Evaluation Metrics

False negatives are the worst kind of error we can make for this project. To better understand why, look at the four possible scenarios below:

- **True Positive (TP):** Send offer and customer will likely use it.
- **False Positive (FP):** Send offer but customer doesn't want it or doesn't use it. ← ****Not as serious of an error.****
- **True Negative (TN):** Do not send offer and customer doesn't want it or doesn't use it.
- **False Negative (FN):** Do not send offer but customer would have likely used it if we sent it. ← ******Worst error******

The two possible errors are False Positives and False Negatives. If we produce a False Positive, the customer will likely just ignore our marketing effort and possibly result in some wasted effort in our part. In extreme cases, the user could view False Positives as harassment and be turned off by our brand. Because of this extreme case and our wasted effort, False Positives are still important but not as important as False Negatives.

False Negatives result in a missed opportunity to market to a receptive customer. As we saw in the above graphs *"Amount Spent towards Offers vs. Not towards Offers for Gender"* and *"Amount Spent towards Offers vs. Not towards Offers for Age Group"*, customers are more than twice as likely to spend money when offers have been sent than without any offers. Even if some customers may have still spent the same amount regardless of if an offer was sent or not, others definitely increase their spending when an offer is active (I know I do).

Precision is used when the cost of False Positives is high. Recall is used when the cost of False Negatives is high. The F_1 Score is the harmonic mean of Precision and Recall.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Figure 20 - Precision Equation

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Figure 21 - Recall Equation

In our case, we want to consider the cost of both Precision and Recall but focus more on False Negatives. To do this we can use the F_2 Score, which puts more emphasis on recall.

$$F_2 = (1 + 2^2) \cdot \frac{precision \cdot recall}{(2^2 \cdot precision) + recall}$$

Figure 22 - F_2 Score Equation

Accuracy is also a common metric focusing on % of guesses made correctly.

$$\text{Accuracy} = \frac{\text{Correct Guesses}}{\text{Total Population}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 23 - Accuracy Equation

I originally was going to only use the F_2 Score, but realized I needed to look at multiple metrics to get a full picture. I discovered that on some models the F_2 Score was really high (~0.90), but the model marked everything as Positive (i.e. send the offer). This resulted in all TP and FP and no TN and FN.

For each model I created a Confusion Matrix, and calculated the accuracy, F_1 Score, and F_2 Score. This allowed me to get a better picture of if the model was doing what I wanted.

A Confusion Matrix is a graph that shows the TP, TN, FP, and FN counts.

One exciting side-effect of this project was it resulted in me opening my first GitHub Pull Request. I used Scikit Learn's plot_confusion_matrix function to graph the confusion matrix for each model. I discovered a bug that forced the values to always be displayed in scientific notation (example: 3.2e+02 instead of 314). For this capstone project I implemented a work-around. For my pull request, I modified the source code, created a test to verify my changes, and modified the whats_new release notes. My pull request will be included in the upcoming Scikit Learn v0.22.1 release.

Models Used:

It is worth noting that Starbucks only had a 63.252% success rate during their trial with the offers they sent. So my models have to beat 63.252% to be considered successful.

1. My baseline/benchmark model was a logistic regression model.

   "[Logistic regression is] quite efficient in dealing with the majority of business scenarios [for propensity modeling]" -HG Insights (https://datascience.foundation/sciencewhitepaper/propensity-modelling-for-business)

   I used Scikit Learn's Logistic Regression model. I originally started with the default values (only setting random_state to get the same results every time), but found that the default max iterations value of 100 was too low. I increased the max iterations to 1000 and got better results.

The accuracy is 0.71208, F_1 Score is 0.79208, F_2 Score is 0.83016, and confusion matrix is:

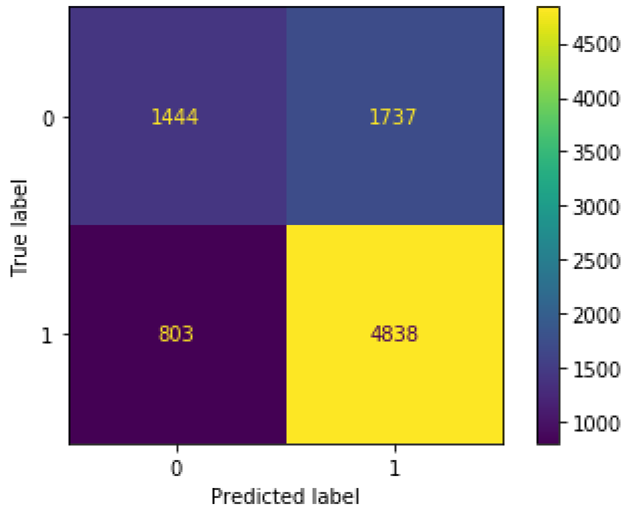

Figure 24 - Logistic Regression Confusion Matrix

2. My second model was a Support Vector Machine (SVM).

   I used Scikit Learn's SVM SVC implementation which is a C-Support Vector Classifier. C is a regularization parameter where the strength of regularization is inversely proportional to C (with C > 0). Gamma is the kernel coefficient for the Radial Basis Function (RBF) kernel.

   After Hyperparameter Tuning (described in detail in the below section) I settled on C=10,000 and gamma=1e-05.

   The accuracy is 0.72463, F_1 Score is 0.78873, F_2 Score is 0.80353, and confusion matrix is:

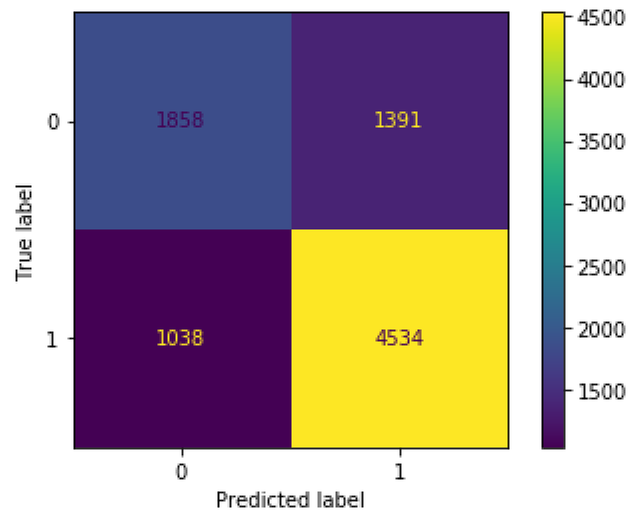

Figure 25 - SVM Confusion Matrix

3. My final model is a Neural Network. I used the TensorFlow 2.0 framework for building and training my model.

I will discuss the Hyperparameter Tuning in-depth in the next section. The hyperparameters selected are:

- 2 Hidden Layers
- Hidden Layer 1 has 128 nodes, with ReLU activation functions, and Dropout value of 0.3
- Hidden Layer 2 has 32 nodes, with ReLU activation functions, and Dropout value of 0.2
- I used the Adam optimizer with a learning rate of 1e-4
- My loss metric was Binary Cross Entropy
- I used an Early Stopping callback monitoring the validation loss. This stopped the training if the validation loss had not decreased within 20 epochs.
- Since I had the Early Stopping callback, I set the number of epochs equal to the number of training data rows (26,463). Because of the Early Stopping, I never exceeded 400 epochs.
- I also used a Model Checkpoint that saved the best model based on the minimum validation loss found.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 128)               3072
_________________________________________________________________
dropout_24 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_37 (Dense)             (None, 32)                4128
_________________________________________________________________
dropout_25 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_38 (Dense)             (None, 1)                 33
=================================================================
Total params: 7,233
Trainable params: 7,233
Non-trainable params: 0
```

Figure 26 - TensorFlow 2.0 Keras Model Summary

The accuracy is 0.71163, F_1 Score is 0.79726, F_2 Score is 0.84863, and confusion matrix is:

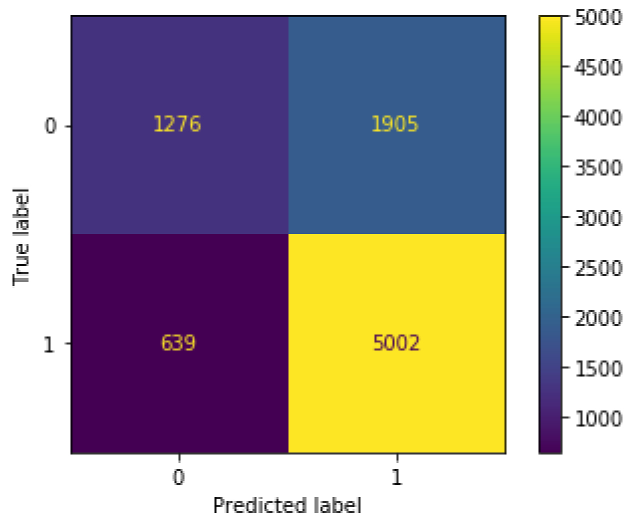

Figure 27 - Neural Network Confusion Matrix

Comparing the Models

Below is a graph showing the metrics for each model against the Test Set:

Model	Accuracy	F1 Score	F2 Score	TP	FP	TN	FN
Logistic Regression **[test set]**	0.71208	0.79208	0.83016	4838	1737	1444	803
Support Vector Machines **[test set]**	0.72463	0.78873	0.80353	4534	1391	1858	1038
Neural Network (Final) **[test set]**	0.71163	0.79726	0.84863	5002	1905	1276	639

Figure 28 - Final Metrics using the Test Set

The Neural Network performed the best out of the 3 models with an F_2 Score of 0.84863 and with the lowest False Negative count of 639. It also has the most True Positives. As stated previously, the False Negatives is the worst kind of error for us since it is a missed opportunity to market to a receptive customer.

The Logistic Regression (baseline model) was a close second to the Neural Network.

The Support Vector Machine performed the worst overall out of the 3 models.

Hyperparameter Tuning

We use the Validation Set when Hyperparameter Tuning.

SVM Tuning

For determining the C and gamma values for the SVM SVC model, I got my idea from here. I used grid search to look for 10 'C' values in log space between 10^{-2} and 10^{4}, and 10 gamma values in log space between 10^{-9} and 10^{3}. The best parameters found were C of 10,000 and gamma of 1e-05 which received an accuracy score of 0.73 on the validation set.

Neural Network (NN) Tuning

For determining hyperparameters for the NN I used TensorFlow's TensorBoard and the HParams Dashboard. One issue I encountered was my Jupyter Notebook kernel repeatedly crashed when trying to perform hyperparameter tuning for the neural network. I eventually wrote a separate Python program do the hyperparameter tuning.

I went through 4 iterations when finding the NN hyperparameters, each time narrowing the focus.

The first iteration tested the following:

- Hidden Layer 1 Number of Nodes: 32, 64, 128, 256
- Hidden Layer 1 Dropout: 0.1, 0.2, 0.3, 0.4, 0.5
- Number of Hidden Layers: 1 or 2

- Hidden Layer 2 Number of Nodes: 32, 64, 128, 256
- Hidden Layer 2 Dropout: 0.1, 0.2, 0.3, 0.4, 0.5
- Optimizer: Adam or SGD
- Learning Rate: 1e-5, 1e-4, 1e-3

This iteration eventually ran out of memory before fully completing, but it gave me enough of an idea to move onto other iterations. I knew from this first iteration that Adam received better results than SGD, so I removed SGD from later iterations.

The second iteration used 1 Hidden Layer and tested the following:

- Hidden Layer 1 Number of Nodes: 32, 64, 128
- Hidden Layer 1 Dropout: 0.1, 0.2, 0.3, 0.4, 0.5
- Number of Hidden Layers: **1**
- Optimizer: **Adam**
- Learning Rate: 1e-4, 1e-3

The third iteration used 2 Hidden Layers and tested the following:

- Hidden Layer 1 Number of Nodes: 32, 64, 128
- Hidden Layer 1 Dropout: 0.1, 0.2, 0.3, 0.4, 0.5
- Number of Hidden Layers: **2**
- Hidden Layer 2 Number of Nodes: 32, 64, 128
- Hidden Layer 2 Dropout: 0.1, 0.2, 0.3, 0.4, 0.5
- Optimizer: **Adam**
- Learning Rate: 1e-4, 1e-3

From this I was able to determine 2 Hidden Layers was better than 1, and smaller learning rate was better. I also found that 128 nodes in Hidden Layer 1 and 32 nodes in Hidden Layer 2 resulted in the best results.

The fourth iteration further refined the previous iterations and tested the following:

- Hidden Layer 1 Number of Nodes: **128**
- Hidden Layer 1 Dropout: 0.1, 0.2, 0.3
- Number of Hidden Layers: **2**
- Hidden Layer 2 Number of Nodes: **32**
- Hidden Layer 2 Dropout: 0.1, 0.2, 0.3

- Optimizer: **Adam**
- Learning Rate: 7.5e-5, 8.75e-5, 1e-4

Below is a table showing the best results from each iteration:

Model	Accuracy	F1 Score	F2 Score	TP	FP	TN	FN
Neural Network (1st iteration) **[validation set]**	*0.72222*	*0.79634*	*0.82741*	*4792*	*1602*	*1579*	*849*
Neural Network (2nd iteration) **[validation set]**	*0.7131*	*0.79682*	*0.84462*	*4963*	*1853*	*1328*	*678*
Neural Network (3rd iteration) **[validation set]**	*0.71911*	*0.805*	*0.86312*	*5115*	*1952*	*1229*	*526*
Neural Network (4th iteration) **[validation set]**	*0.70676*	*0.79966*	*0.86523*	*5163*	*2109*	*1072*	*478*

Figure 29 - Hyperparameter Tuning Scores for each Iteration against Validation Set

The best parameters from the fourth iteration were the final parameters used.

- Hidden Layer 1 Number of Nodes: **128**
- Hidden Layer 1 Dropout: **0.3**
- Number of Hidden Layers: **2**
- Hidden Layer 2 Number of Nodes: **32**
- Hidden Layer 2 Dropout: **0.2**
- Optimizer: **Adam**
- Learning Rate: **1e-4**

Below is a screenshot from the Tensorboard HParams Dashboard showing the best hyperparameters from the final iteration highlighted in green. The blue line has a higher F_2 Score but also classifies all data as Positive (send offer), TP or FP, and does not classify any as Negative (don't send offer), TN or FN. The colors range from bright red for highest accuracy to dark blue for lowest accuracy. The line in green is the one I selected.

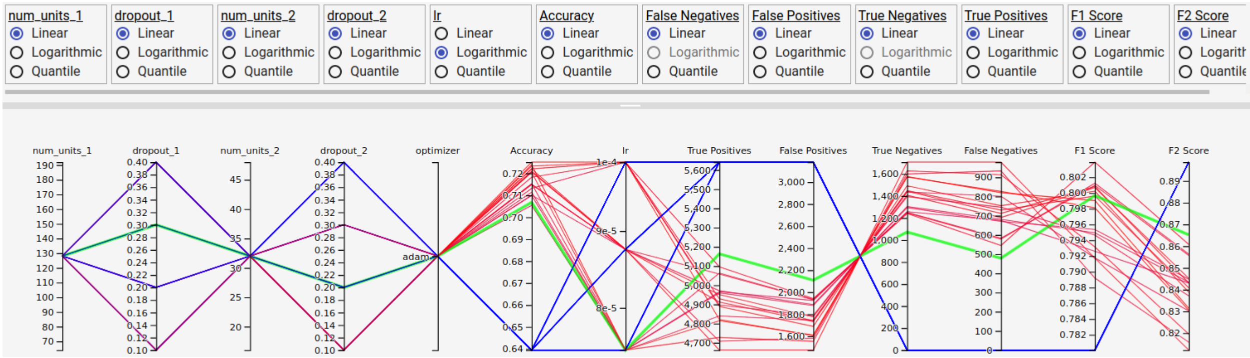

Figure 30 - HParams Dashboard from Final Iteration — Best Hyperparameters in Green

Conclusion

For this project I have analyzed, cleaned, performed feature engineering, and created 3 models using the Starbucks data. I have created a Neural Network model that successfully performs propensity modeling with an F_2 Score of 0.84863 on the Test Set and the lowest False Negative score out of all my models. As mentioned previously, Starbucks was only able to achieve a 63.252% success rate during their trial. My model definitely improves upon the trial results, and so it is a success!

Future Improvements

If I were to extend this project further, I would attempt the following:

- I would dig into the examples that were labeled as False Negatives to determine why the model incorrectly labeled them. I could then possibly apply additional feature engineering or tweak hyperparameters to reduce these.
- I would ensemble several models together to see if they improve the results.
- I would try a tree-based model like Random Forest.
- I am curious how the models would handle a new offer (other than the 10 existing offers). Would they handle this new offer with similar metric results, or would the models require additional training?
- I am also curious if certain demographics respond more to certain offer types. This information could be used to engineer new offers targeting the specific demographic.

References

- Matplotlib: Bar Chart Example

- Matplotlib: Pie Chart Example
- Matplotlib: Plotting Multiple Bar Charts
- Pandas: Move Column to Front of Table
- Pandas: Why Itertuples is Faster than Iterrows
- Propensity Model: Algorithms
- Propensity Model: What is it?
- Scikit Learn: Logistic Regression Model
- Scikit Learn: MinMaxScaler
- Scikit Learn: SVM SVC Model
- Scikit Learn: SVM SVC Hyperparameter Tuning Example
- Scikit Learn: train_test_split
- TensorFlow: Early Stopping and Saved Model Callback Examples
- TensorFlow: Effective TensorFlow 2.0
- TensorFlow: Hyperparameter Tuning with HParams Dashboard
- Wikipedia: Accuracy
- Wikipedia: Confusion Matrix
- Wikipedia: F_1 Score and F_2 Score
- Wikipedia: Precision and Recall