Starbucks Capstone Project Report

Egor Makhov
george.mahoff@gmail.com

Udacity, MLE Nanodegree — May 24, 2020

Domain background

This Starbucks Capstone project is part of the Udacity Machine Learning Engineer Nanodegree. Udacity partnered with Starbucks to provide a real-world business problem and simulated data mimicking their customer behaviour.

Starbucks is an American coffeehouse chain. Once every few days, Starbucks sends out an offer to users via different ways such as mobile app. An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free). Some users might not receive any offer during certain weeks. An important characteristic regarding this capstone is that not all users receive the same offer. As part of marketing strategy, we always want to figure out if a customer will spend more by giving a sound offer. Providing right offer to right customer could help build loyalty of the brand and product and as a result increasing sales margins in the long run

Problem statement

The problem we are looking to solve here is conceptually easy to understand, albeit difficult to answer. We are looking to best determine which kind of offer to send to each customer segment based on their purchasing decisions. We'll touch more on what these offers are and data we'll be utilizing down in the next section. We will leverage traditional evaluation metrics to determine which model is most appropriate for our dataset. These evaluation metrics will be discussed in an upcoming section.

Datasets and inputs

As given by the Udacity's Starbucks Project Overview:

- The program used to create the data simulates how people make purchasing decisions and how those decisions are influenced by promotional offers.
- Each person in the simulation has some hidden traits that influence their purchasing patterns and are associated with their observable traits. People produce various events, including receiving offers, opening offers, and making purchases.
- As a simplification, there are no explicit products to track. Only the amounts of each transaction or offer are recorded.
- There are three types of offers that can be sent: buy-one-get-one (BOGO), discount, and informational. In a BOGO offer, a user needs to spend a certain amount to get a reward equal to that threshold amount. In a discount, a user gains a reward equal to a fraction of the amount spent. In an informational offer, there is no reward, but neither is there a requisite amount that the user is expected to spend. Offers can be delivered via multiple channels.
- The basic task is to use the data to identify which groups of people are most responsive to each type of offer, and how best to present each type of offer.

The data is divided in 3 files:

– profile.json: Rewards program users (17000 users x 5 fields)
- gender: (categorical) M, F, O, or null
- age: (numeric) missing value encoded as 118
- id: (string/hash)
- became_member_on: (date) format YYYYMMDD
- income: (numeric)

– portfolio.json: Offers sent during 30-day test period (10 offers x 6 fields)
- reward: (numeric) money awarded for the amount spent
- channels: (list) web, email, mobile, social
- difficulty: (numeric) money required to be spent to receive reward
- duration: (numeric) time for offer to be open, in days
- offer_type: (string) bogo, discount, informational
- id: (string/hash)

– transcript.json: Event log (306648 events x 4 fields)
- person: (string/hash)
- event: (string) offer received, offer viewed, transaction, offer completed
- value: (dictionary) different values depending on event type
- offer id: (string/hash) not associated with any “transaction”
- amount: (numeric) money spent in “transaction”
- reward: (numeric) money gained from “offer completed”
- time: (numeric) hours after start of test

Data cleaning

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 1: Complete Portfolio after cleaning

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 2: First 5 rows of cleaned Profile

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 3: Last 5 rows of cleaned Transcript

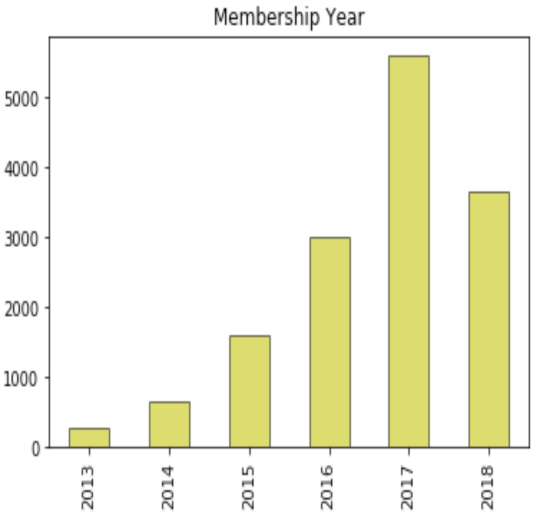
Preliminary data analysis

Key points observed when exploring the data:

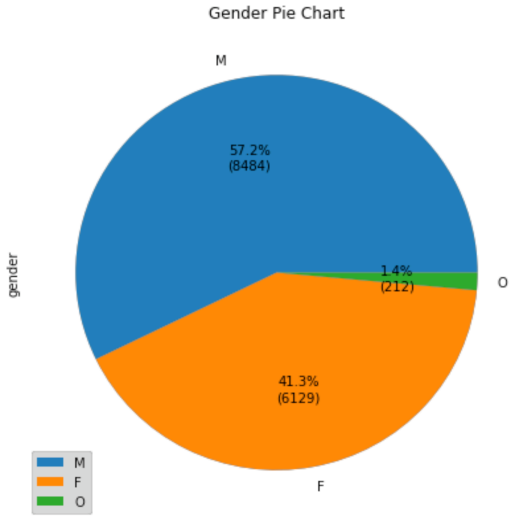
– Portfolio dataset exploration
- Every offer was sent via the email channel, in addition to other channels.
- Informational offer types are never "completed" since they have no difficulty.

– Profile dataset 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.

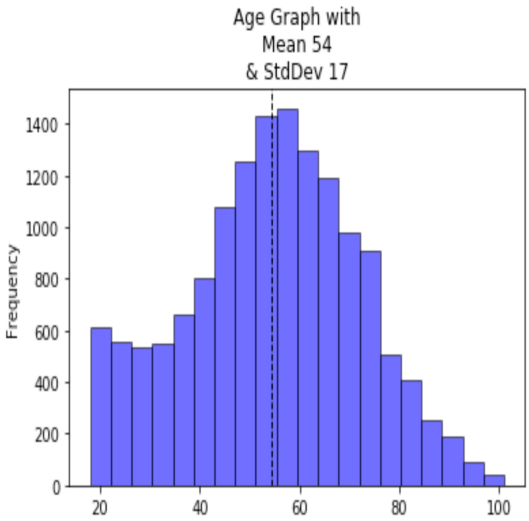
– Transcript dataset exploration
- This pie chart shows the distribution of events:
- Every customer received at least 1 offer, and some customers received up to 7 offers.
- Some customers received the same offer multiple times.
- No customers received multiple offers at the same timestamp.
- There were many instances where a customer would receive another offer before the previous offer expired, resulting in multiple offers being active at the same time.



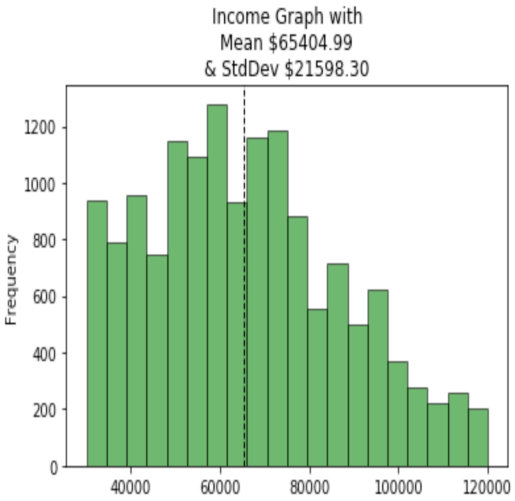
(a) Profile Memberships Per Year



(b) Profile Gender Percentages and Counts



(c) Profile Age Distribution



(d) Profile Income Distribution

Figure 4: Profile dataset explorations charts

Data preparation

Joining 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've 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:

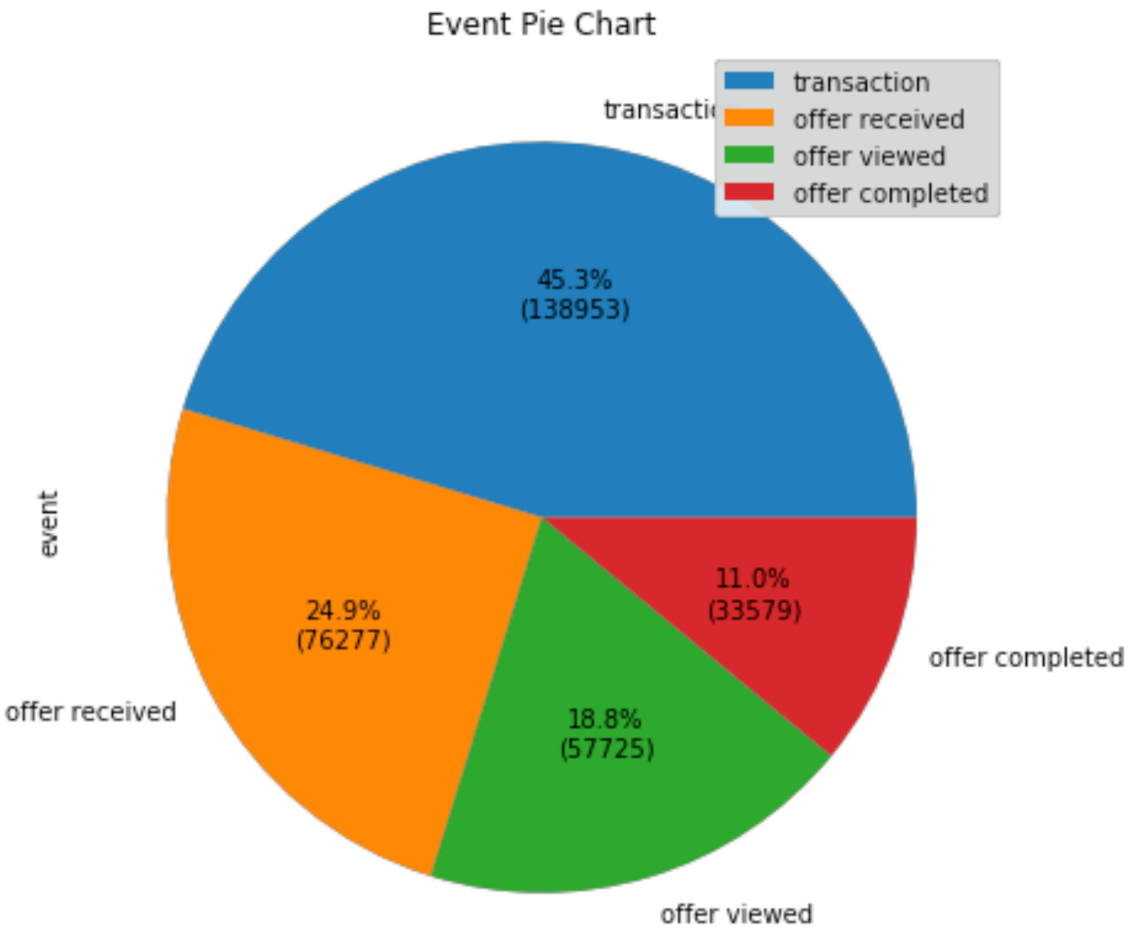


Figure 5: Transcript Event Percentages and Counts

- '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	2906b810c7d441798c6938adc9daaa5	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 6: The Combined Dataframe after merging Transcript, Portfolio, and Profile

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 7: 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.

Then I created my Training/Testing/Validation datasets with a 60/20/20 split.

Evaluation metrics

False negatives are the worst kind of error we can make for this project. If we produce a False Positive, the user will likely just ignore our marketing effort and result in possibly some wasted effort on 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, 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. This can result in the business not making sales that they would have otherwise made.

Precision is used when the cost of False Positives is high. Recall is used when the cost of False Negatives is high. In our case we want to consider the cost of both but focus more on False Negatives. To do this we can use the F2 score, which puts more emphasis on recall.

The F2 score is defined as:

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

Model training

Benchmark model - logistic regression model

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, F1 Score is 0.79208, F2 Score is 0.83016, and confusion matrix is:

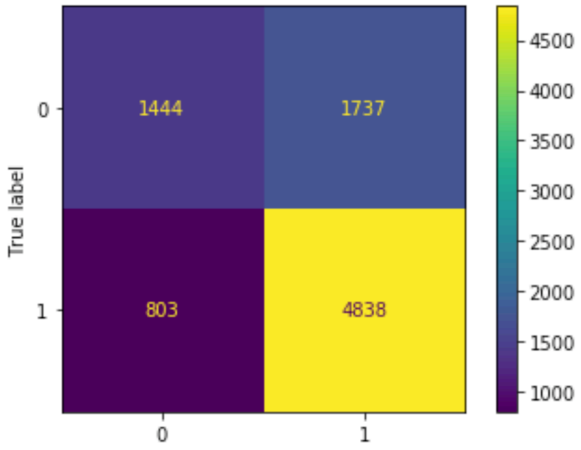


Figure 8: Logistic Regression Confusion Matrix

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, F1 Score is 0.78873, F2 Score is 0.80353, and confusion matrix is:

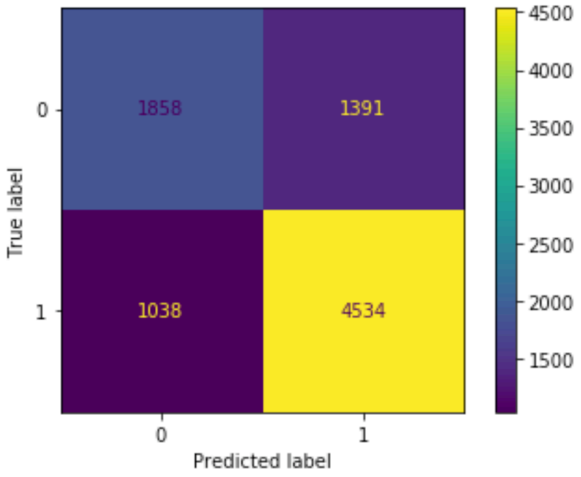


Figure 9: SVM Confusion Matrix

Neural Network model

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 10: TensorFlow 2.0 Keras Model Summary

The accuracy is 0.71163, F1 Score is 0.79726, F2 Score is 0.84863, and confusion matrix is:

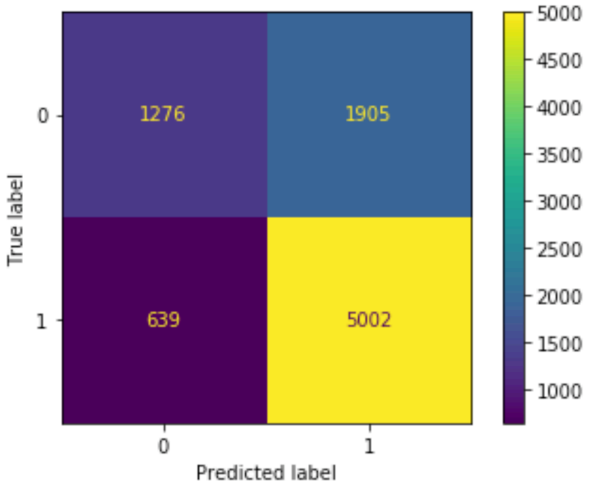


Figure 11: Neural Network Confusion Matrix

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 12: Final Metrics using the Test Set

The Neural Network performed the best out of the 3 models with an F2 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

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 102 and 104, and 10 gamma values in log space between 109 and 103. 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 13: 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 F2 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.

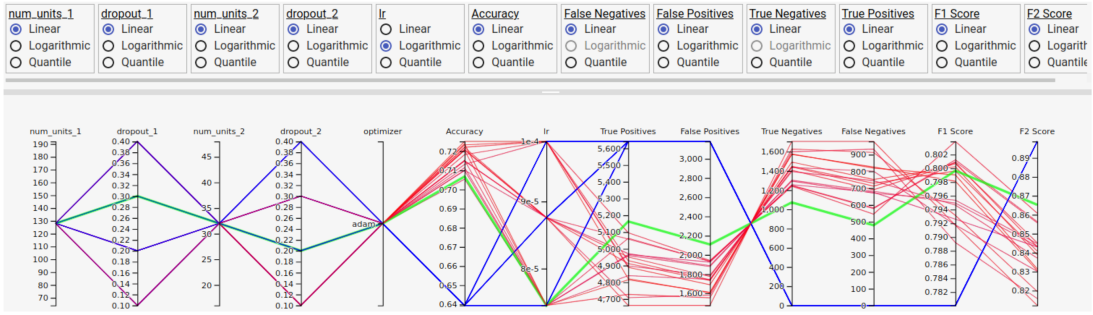


Figure 14: Params Dashboard from Final Iteration (Best Hyperparameters in Green)

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 F2 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 work

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.

Future work

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 hyper-parameters 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.