Machine Learning Engineer Nanodegree

Capstone Project

By: Alvaro Azabal
Date: 18th June, 2020
Project Name: Starbucks Capstone Project

Project Definition

Project Overview

As the largest coffeehouse retailer in the world, Starbucks possesses vast amounts of data. Some of this data refers to targeted advertising. The business insights that can be obtained through proper analysis of targeted advertising are extremely valuable. This data contains information about which customers receive special offers, which customers actually use these offers, how much do they spend, etc. Machine learning offers a large variety of techniques that could be applied to this data to gain useful information about the targeted campaigns. One of these useful techniques is clustering. Using this technique one could group customers in different clusters depending on their spend behaviour, thus predicting how these customers will behave in the future when receiving new offers. Another possible machine learning technique that could be used is a classification algorithm to predict if a customer will respond when receiving an offer or not. There are several classification techniques (decision trees, neural networks, etc) that will be explored in the project.

The data used for this project 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

Problem Statement

As briefly discussed in the previous section, this project will aim to build a classifier to help Starbucks predict if a customer will positively respond (ie: will make a purchase) when receiving a special offer. This will be a very useful model for future more accurate targeted advertising campaigns. There are a large amount of possible classifiers to use. This project will investigate using deep neural networks for this problem, as well as decision trees.

A binary classification model will be built to predict if a customer will buy a product when receiving a special offer. Once this model is properly trained, the user will be able to predict if a customer will use the promotional offer depending on the demographics of this user and the type of offer received.

The main steps required to obtain such models are: first, cleaning and preprocessing the data. For the classification problem, the data will have to be splitted in training, validation and test datasets, to then be fitted into the model. Hyperparameter tuning and grid search cross-validation can be done to obtain the best model. This model will then be able to predict the customer behaviour, by outputting either a 0 (offer not used) or a 1 (offer used).

Metrics

To evaluate the binary classification model, the dataset will be divided in three sets (training, validation and test). The accuracy, precision, recall and f1-score will be used to evaluate how good the model is. To ensure that the model is learning properly and not overfitting, a validation dataset will be used, making sure that the evaluation metrics mentioned above are in the same order of magnitude for both the training and validation datasets. Once the model has been trained, it will be evaluated using the test dataset, using the evaluation metrics already discussed.

Data Analysis

Once the project has been defined, and the main steps involved briefly discussed, it is time to actually start exploring the data available, which will be the first step in the data wrangling process.

Data Exploration & Preprocessing

As discussed, there are three input datasets: *portfolio.json*, *profile.json* and *transcript.json*. Each dataset will first be explored independently, but the process followed will be the same.

Starting with the *portfolio.json* dataset, we can see that it contains 10 rows, one for each type of offer, and 6 columns, each for one feature of the data, as described in the Project Overview section. As part of the data pre-processing stage, several changes will be done to this dataset:

- The “channels” column will be one-hot encoded to create four columns (“email”, “mobile”, “social” and “web”) with values of 0 or 1
- The “duration” column will be transformed from days to hours, by multiplying it by 24
- The “offer_type” column will be one-hot enconded to cretate three columns (“bogo”, “info”, “discount”) with values of 0 or 1
- A dictionary will be created containing the long alphanumeric version of the offer ID as the key, and a unique numeric counter (from 0 to 9) as the value. This dictionary is then used to convert the “id” column to its numeric value.

The two figures below (Figure 1 and 2) show how the portfolio data looks before pre-processing and after pre-processing.

	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: Portfolio dataset before pre-processing

	reward	difficulty	duration	id	email	mobile	web	social	bogo	info	discount
0	10	10	168	0	1	1	0	1	1.0	0.0	0.0
1	10	10	120	1	1	1	1	1	1.0	0.0	0.0
2	0	0	96	2	1	1	1	0	0.0	1.0	0.0
3	5	5	168	3	1	1	1	0	1.0	0.0	0.0
4	5	20	240	4	1	0	1	0	0.0	0.0	1.0
5	3	7	168	5	1	1	1	1	0.0	0.0	1.0
6	2	10	240	6	1	1	1	1	0.0	0.0	1.0
7	0	0	72	7	1	1	0	1	0.0	1.0	0.0
8	5	5	120	8	1	1	1	1	1.0	0.0	0.0
9	2	10	168	9	1	1	1	0	0.0	0.0	1.0

Figure 2: Portfolio dataset after pre-processing

The next dataset to analyse and pre-process is the *profile.json* dataset, which contains information about the customer. When loading the data, one can see that it contains entries of 17,000 customers and 5 rows with different features about each customer. The first thing to notice when loading the data is the the “income” and “gender” columns have *NaN* and *null* values at some rows. A heatmap of the data can be plotted to identify how many missing values there are. This is shown in Figure 3a.

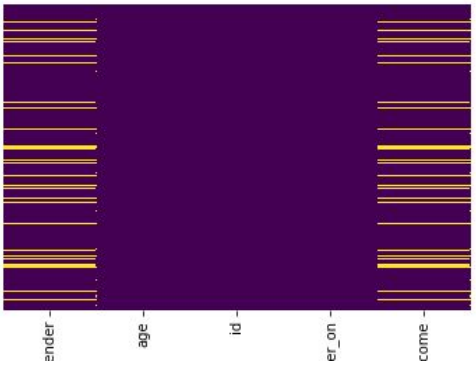


Figure 3a): Profile missing data

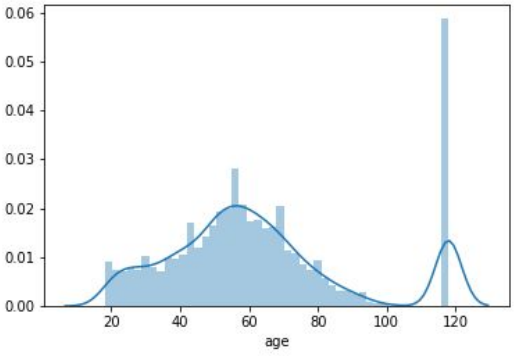


Figure 3b): Profile age distribution

This indicates that the amount of missing data is not insignificant, and has to be treated carefully. The next step will be to identify how much data is missing and decide what to do with this data. There are two options: remove the data or try to infer the missing data from the existing data. One technique is to look at the distribution of the data to understand if it will be easy to correlate the data to infer the age, gender and income of the customer based on other customer’s data. By doing so, Figure 3b) shows that there is what looks like an anomaly in the age of the customers, as a huge amount of passenger appear to be 118 years old. This is most probably the default value given by the app if the age is not entered properly when signing up. Therefore, we can conclude that there is missing data in the age, income and gender. Figure 4 shows the relation between these three features, which highlights the lack of correlation between them.

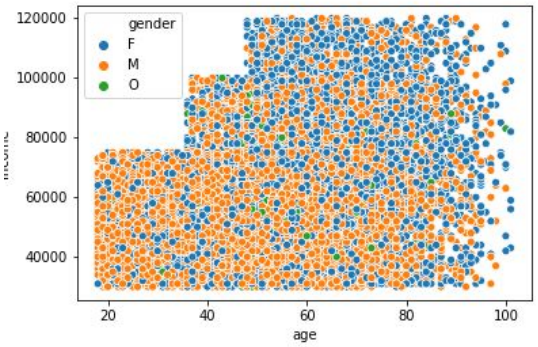


Figure 4: Age vs Income vs Gender profile data

Looking at the code and the output below shows that 12.8% of the data is missing, and that all of the missing data occurs for the same people (ie: every customer that has a missing gender also has a missing income and age), therefore this means that all rows containing a missing value can be deleted.

```
print("The total amount of missing 'gender' data is: ",len(profile[profile["gender"].isnull()]))
print("The total amount of missing 'income' data is: ",len(profile[profile["income"].isnull()]))
print("")
print("The age of the missing 'gender' data is: ",profile[profile["gender"].isnull()]["age"].unique())
print("The age of the missing 'income' data is: ",profile[profile["income"].isnull()]["age"].unique())
print("")
print("The income of the missing 'gender' data is: ",profile[profile["gender"].isnull()]["income"].unique())
print("The gender of the missing 'income' data is: ",profile[profile["income"].isnull()]["gender"].unique())
print("")
print("The percentage of missing data is: ",len(profile[profile["gender"].isnull()])/len(profile)*100)
```

```
The total amount of missing 'gender' data is:  2175
The total amount of missing 'income' data is:  2175

The age of the missing 'gender' data is:  [118]
The age of the missing 'income' data is:  [118]

The income of the missing 'gender' data is:  [nan]
The gender of the missing 'income' data is:  [None]

The percentage of missing data is:  12.794117647058822
```

This is a large amount of data. However, this data will be removed because the dataset is a large one that contains a lot of information, so even without this amount of data it will still contain a lot of information. Also, being able to properly predict the age, income and gender of a customer simply from the day it became a member will be a very difficult task that could add a lot of noise to the data, specially given the little apparent correlation between these features.

The next step is then to remove all rows containing missing values. Looking at the missing data heatmap in Figure 5a) now shows that there is no missing data in the dataset. One can then look at the distribution of age and income in the dataset, as done in Figures 5b) and 5c). The age resembles a normal distribution with an average of around 60, and varying from 18 to more than 100 years old, with a large concentration of people around 20-30, which would make sense as these are app users. The income shows a similar trend, with an average of around 70,000 and a larger concentration of people in the lower income range of approximately 40,000.

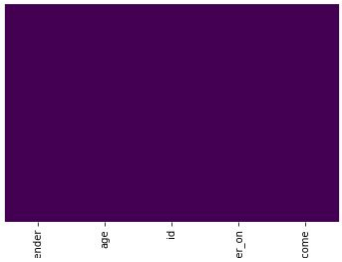


Figure 5a: Profile missing data

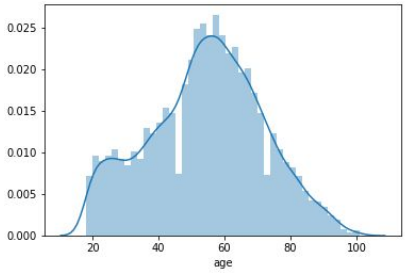


Figure 5b: Profile age distribution

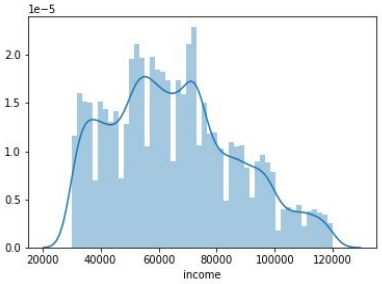


Figure 5c: Profile income distribution

The gender distribution from Figure 6b shows that 57% of the people are male, 41 % are female and the other 2% identify as other. Once that the missing data has been removed, the other profile features have to be formatted, as follows:

- The “gender” has been changed from *male, female* and *other* to 0, 1 and 2
- A dictionary is created containing the long alphanumeric version of the person ID as the key, and a unique numeric counter (from 0 to 14,824) as the value. This dictionary is then used to convert the “id” column to its numeric value.
- In order to use the "became_member_on" later during the training process, the format of the data will have to be changed from datetime to an actual number. Therefore, the column will be change from an actual date to the number of days the given person has been a member. This will be done by taking the most recent date someone signed up as day 0, so if someone signed up a day before that, they would have been a member for 1 day, and so on. To do so, first we'll have to find the date the last member signed up, and then find the difference between that date and the membership date of the other customers.

Figure 6a) shows that a large majority of the customers have been members for less than 500 days, and only a very low number of customers have been members for more than 1,000 days.

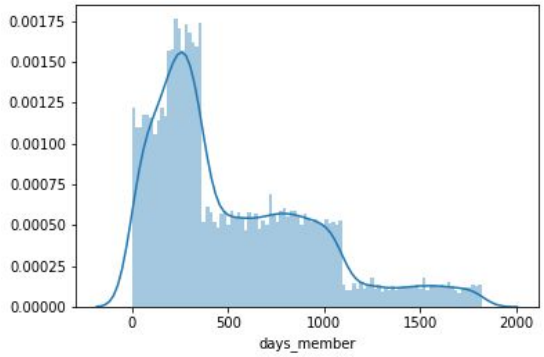


Figure 6a: Profile days as member distribution

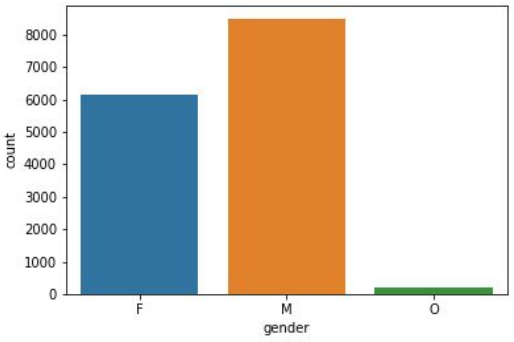


Figure 6b: Profile gender distribution

Once the data has been pre-processed, it is possible to look at the correlation between all of the parameters that define a customer, as shown in the table below. This shows that the largest correlation is between the income and age of the customers (this makes sense as typically older people have a larger income than young people and students). However even this correlation is quite small (0.3) which suggests that removing the missing data was a better choice than trying to infer it from the membership date, which is completely uncorrelated to any other parameter.

	gender	age	income	days_member
gender	1.000000	0.142886	0.209979	-0.022117
age	0.142886	1.000000	0.306703	0.012300
income	0.209979	0.306703	1.000000	0.025769
days_member	-0.022117	0.012300	0.025769	1.000000

The below pairplot is a more graphical way of showing the lack of correlation

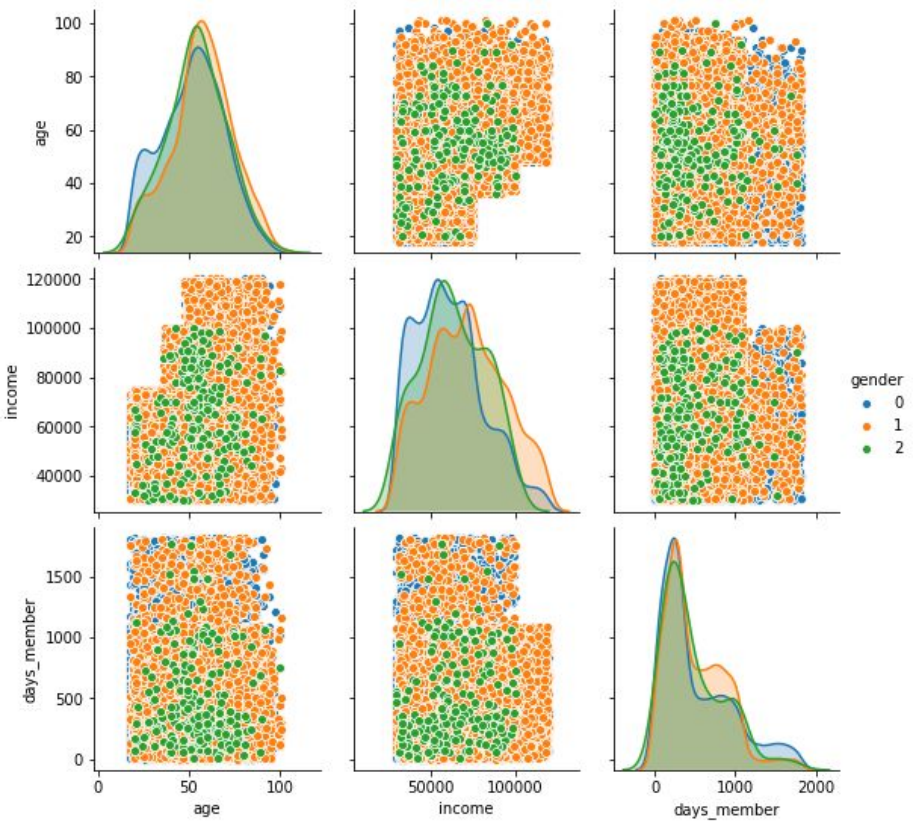


Figure 7: Correlation between the customer profile parameters

The final dataset to look at is the transcript dataset. Reading the data indicates that this dataframe has 306,534 rows (one for each transaction done by a customer) and four columns, as described in the project overview.

The first thing will be to change the "person" string value by its corresponding number created in the dictionary before. The only difference will be that some of the people doing transactions will be part of the missing data. In that case the key won't be identified. If that is the case, the person id should be set to -1, and then all rows with a -1 in the "person" column will be removed.

There are four unique events: receiving an offer, viewing an offer, completing an offer and making a transaction. Therefore the data will be split into four subsets, one for each event, and pre-processed individually.

The "value" column contains a dictionary with the offer id, in case of an offer being received, viewed or completed, with the reward received if the offer was completed, and a dictionary with

the amount spent in the case of a transaction. These dictionaries will be transformed to actual values and columns.

The end goal of the data processing step is to obtain a dataframe where each row is a unique offer sent to a customer and having columns which include the data about the customer who received the offer (from the profile dataset), information about the offer sent (from the portfolio dataset) and an identifier saying if the offer sent was actually viewed by the customer (using the offer_view dataset), if the offer was actually used (ie: completed) by the customer (using the offer_comp dataset), and finally the amount spent by the customer in case of completing the received offer.

To do so, first the dataset that contains all offers received by customers will be merged with the portfolio and profile datasets, to include information about the offer and customers. Then, an algorithm will be written to identify if the offer was viewed by the customer, if the offer was completed and if so, how much did the customer spend. Notice that if the customer completed the offer before viewing it, the customer actually didn't view the offer. This is taken into account. The informational offer types aren't linked to any transaction in particular, so initially they don't appear as "Completed". In order to identify if they were used or not, an algorithm is written to identify if the customer that received the informational offer viewed the offer and made transaction before the expiry date. If this is the case the offer will be considered as "Completed".

The final step in the data pre-processing is to create four one-hot encoded columns with the four possible outcomes:

1. Offer Viewed and Completed
2. Offer Not Viewed and Completed
3. Offer Viewed but Not Completed
4. Offer Not Viewed and Not Completed

Once the pre-processing steps are done, some data exploration can be done to get a better understanding of the effectiveness of the offers. Looking at Figure 8a) shows that 41.7% of the offers that were sent were actually viewed and used to make transactions. This represents the effectiveness of the promotion program. On the other hand, 26% of the offers were viewed but the customers weren’t interested in buying, 17.7% of the offers weren’t neither viewed nor used and 14.7% of the offers were used by the customers without being aware that the customer existed. This is a large value, and should be minimised for future promotions, as this implies that a large number of items have been sold at a discount to customers that would have bought the item even without the discount. Figure 8b) shows the same information but on an offer-by-offer basis. Offers 5 and 6 turned out to be the most effective ones (both of them “discount” offers) with an effectiveness ratio of 0.62 and 0.65 respectively. On the other hand, offer 4 was the least effective, and only 20% of them were actually used to make a purchase. Notice too that offers 2, 3, 4 and 9 had a large percentage of “Not Viewed”. These are the only offers that were not sent through social media, indicating that the channels are actually important, and that social media is a powerful channel to reach to your customers.

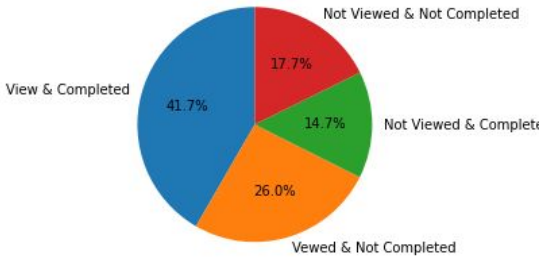


Figure 8a: Total promotion effectiveness

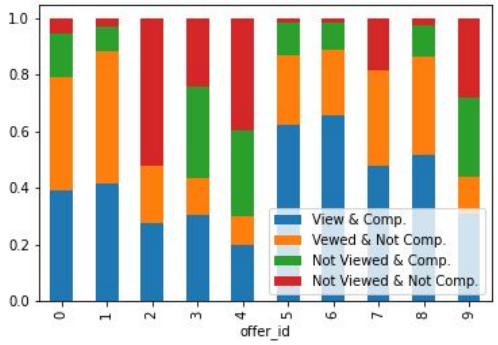


Figure 8b: Promotion effectiveness by offer ID

Looking at the informational offers, it turns out that sending an informational offers increases the customer spending by 4.5, from an average of $5.2 per customer to $23.8. This is better represented in Figure 9a, that shows the distribution of how much customers spend when receiving an informational offer depending on if they view and use the offer or not.

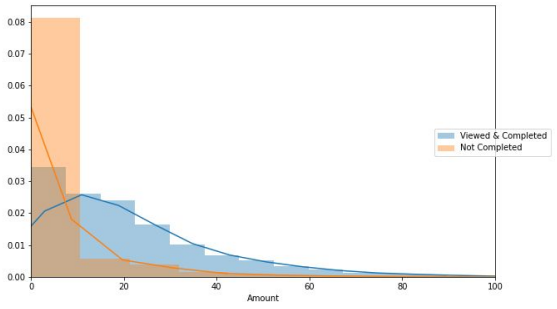


Figure 9a: Amount spent when receiving informational offer a

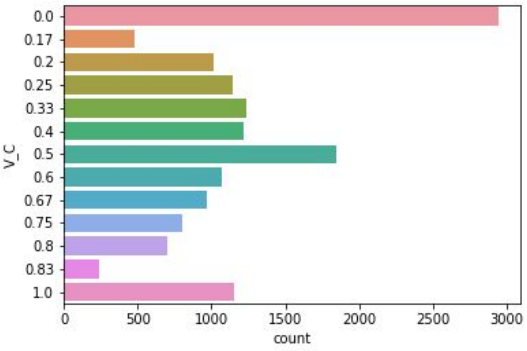


Figure 9b: Total number of customers grouped by average effectiveness

The consuming behaviour of each customer can be analysed by grouping all offers sent by customer, and then looking at the average number of offers that were viewed and completed. This is represented in Figure 9b. This shows that a large number of customers (approx. 3,000) never used an offer they received, 1,800 customers used 50% of the offers received and 1,200 customers used all offers they received. This looks like a normal distribution centered at 0.5 with the exception of the two most extreme values (0 and 1). This indicates that in general there are some customers which will always react to promotions and others that will never react to promotions.

There is a large number of analyses that can be done with this data, and infinite possible visualisations. However, the purpose of this project is to do machine learning, not to do data analytics. The next sections of this report will focus on the machine learning side of the project in more detail.

Methodology

This section will detail the steps required to achieve an optimal machine learning algorithm. Two main problem to be addressed is:

1. Algorithm to predict if a customer will positively respond (view and use) a promotional offer. A model will be created that receives as an input the demographic information about the customer (age, income, gender and days being a member), as well as the technical details about the offer (offer type, channels, duration, reward). The model will output a binary prediction (0 or 1) depending on if the customer will use the offer or not.

To address this problem, several steps will be followed:

1. Data pre-processing
    a. Split data in train, test and validation datasets
    b. Normalise data
2. Benchmark model
    a. Build a simple logistic regression model to be used as a benchmark for future, more complex models
    b. Identify key metrics to evaluate and validate model
3. Boosting Trees Classifier
    a. Build initial boosting tree classifier to assess improvement of model metrics versus benchmark model
    b. Hyperparameter tuning to obtain the best performance model
    c. Analyse predictions to assess limitation of model
4. Deep Neural Network Classifier
    a. Build initial deep neural network to expand the predictive capabilities of the model
    b. Hyperparameter tuning to obtain the best performance model
    c. Analyse predictions to assess limitation of model

Data Preprocessing

All of the initial data processing steps have been disclosed in the Data Analysis section. This section will explain the necessary steps that have to be done after the initial has already been pre-processed. As discussed above, the first step is to split the dataset into test and train datasets. As the dataset is large, a 10% test split has been used. The validation dataset will be prepared later. For the boosting tree classifier, a K-fold cross validation algorithm is used, using 10 different folds, which gives a good level of understanding of the overfitting in the model without taking too much time to run. For the deep neural network, a validation split of 1% will be used instead.

Once the datasets are split, the data is normalised using scikit-learn's MinMaxScaler class. This is done to make sure that all data is within the same order of magnitude. Furthermore, a

VarianceThreshold algorithm is used to remove the features with no variance, thus making the training process faster.

Benchmark Model

A simple logistic classifier is used as a benchmark model. This model has the following inputs and outputs:

- Inputs:
    - Customer demographics:
        - Person ID
        - Age
        - Gender
        - Days since becoming a member
        - Income
    - Offer Characteristics
        - Offer ID
        - Reward
        - Difficulty
        - Duration
        - Offer type - one hot encoded (info, bogo or discount)
        - Channels - one hot encoded (email, social, web, mobile)
- Output:
    - 1/0 - Customer will / will not use the offer

This benchmark model, and the following models, will be analysed using these metrics:

- Accuracy
- Precision
- Recall
- f1-score

Once the logistic regression model is trained, it can be evaluated using the test data left apart. This gives the following results, which are also summarised in the Table 2 at the end of the section.

	precision	recall	f1-score	support
0.0	0.68	0.78	0.73	3891
1.0	0.61	0.48	0.53	2760
accuracy			0.66	6651
macro avg	0.64	0.63	0.63	6651
weighted avg	0.65	0.66	0.65	6651

This shows that the benchmark model is already better than randomly guessing (65% accuracy) but still has margin for improvement. A more complex and accurate model is required.

Refinement

Boosting Trees Classifier

Using TensorFlow's BoostedTreeClassifier() estimator, one can build a more complex and accurate classification model, making use of gradient boosting applied to decision trees. The first step is to prepare the input and output data (features and labels) in the proper format, which is as a feature column.

Once this is done, the algorithm can be run to train the model, first using the default hyperparameter values. This is a good way of starting, as it gives a rough idea of the time it takes to train a model, as well as the level of accuracy you're dealing with. Doing so, one can see that all metrics have improved considerably, when comparing it to the benchmark model, as seen in the Table at the end of the section and below.

	precision	recall	f1-score	support
0	0.71	0.82	0.76	3891
1	0.67	0.53	0.59	2760
accuracy			0.70	6651
macro avg	0.69	0.67	0.68	6651
weighted avg	0.69	0.70	0.69	6651

The next step is to do a hyperparameter optimisation, using TensorFlow and TensorBoard. This allows you to iterate through several hyperparameters and compute different metrics. Then the user can select the best combination of hyperparameters for the desired metric. Recall that, although *accuracy* is clearly a very important metric, so are *precision* and *recall*. *Precision* indicates how many false positives the model predicts, and *recall* is related to the number of false negatives. For this use case, we are trying to identify if a customer will use an offer when receiving it. If the model predicts that are customer will use it (positive) but then the customer doesn't use it, that is referred as a false positive. A low number of these leads to a large *precision.* On the other hand, if the model predicts that a customer won't use the offer (negative) but then the customer uses it, this is a false negative. A low number of these leads to a large *recall.* For this particular use case it is preferred to have the largest *recall* as possible (you don't want to not send an offer to customer because the model predicts the customer won't use it when in fact the customer would have used it, as this leads to a missing opportunity. On the other hand if you think a customer will buy something and in the end they don't, that won't have a negative effect on the revenues for Starbucks).
Having discussed, the hyperparameter tuning can then be done. This has been done by iterating through the hyperparameters shown in Figure 10.

```
HP_N_TREES= hp.HParam('n_trees', hp.Discrete([100, 150, 200]))
HP_MAX_DEPTH = hp.HParam('max_depth', hp.Discrete([10, 15]))
HP_LR = hp.HParam('learning_rate', hp.Discrete([0.1, 0.01, 0.001]))
HP_L2REG = hp.HParam('l2_regularization', hp.Discrete([0.01, 0.001, 0.0001]))
HP_BIAS = hp.HParam('center_bias', hp.Discrete([True, False]))


METRIC_ACCURACY = 'accuracy'
METRIC_PRECISION = "precision"
METRIC_RECALL = "recall"

with tf.summary.create_file_writer('logs/hparam_tuning').as_default():
    hp.hparams_config(
    hparams=[HP_N_TREES, HP_MAX_DEPTH, HP_LR, HP_L2REG, HP_BIAS],
    metrics=[hp.Metric(METRIC_ACCURACY, display_name="Accuracy"),
             hp.Metric(METRIC_PRECISION, display_name="Precision"),
             hp.Metric(METRIC_RECALL, display_name="Recall")],
  )
```

Figure 10: Boosted Tree Classifier Hyperparameter tuning

Several combination of hyperparameters reach values of accuracy and precision over 70% and 65% respectively. However, as discussed before, the key metric of interest is *recall*, so the hyperparameters selected have been the ones that, giving large values of *accuracy* and *precision*, give the largest value of *recall*. These hyperparameters are summarised in the Table 1 below.

Table 1: Hyperparameters chosen for boosting tree classifier

# Trees	Max. Depth	Learning Rate	L2 Regularization	Centre Bias
200	15	0.1	0.0001	False

	precision	recall	f1-score	support
0	0.73	0.77	0.75	3891
1	0.65	0.60	0.62	2760
accuracy			0.70	6651
macro avg	0.69	0.68	0.69	6651
weighted avg	0.70	0.70	0.70	6651

This gives a good increase in performance versus the benchmark model. To validate these results and make sure that there is no overfitting in the model, the next step is to use the KFold algorithm to do a cross-validation, using 10 folds. What this does is split the training dataset in 10 subsets, every time training on 9 of them and leaving the 10th out for validation. This is done on every subset, and the metrics for each subset are then analysed to make sure that they are all at around the same values. If this is the case, one can confirm that there is no overfitting. Looking at Table 2, which shows these results, indicates that the selected model architecture shows no signs of overfitting. In Table 2, the *recall* values refer to the recall of positive values only (model predicting that the customer will use an offer).

Table 2: K-Fold Cross-Validation Metrics

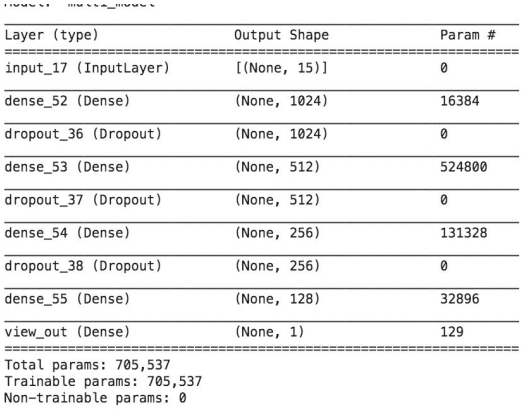
Fold #	Accuracy (%)	Precision (%)	Recall (%)
1	70.5	70.1	70.5
2	70.2	69.9	70.2
3	69.3	69.0	69.3

4	69.2	68.8	69.2
5	70.2	69.9	70.2
6	70.5	70.2	70.5
7	69.7	69.3	69.7
8	69.5	69.1	69.5
9	69.4	69.1	69.4
10	70.1	69.3	70.1

Deep Neural Network

Although the results of the boosted tree classifier are good (over 70% accuracy is much better than randomly guessing, which has a 50% accuracy) one could explore if building a deep neural network increases the performance of the model.

The first step is to setup the model architecture, using TensorFlow's functional API. The model looks as indicated in Figure 11. This is a series of stacked dense neural networks, with a "relu" activation function. The input layer has a size of 15 (the total number of features) and the output has a size of 1 (binary classification). This layer has a sigmoid activation function, which is the one used for binary classification. A dropout layer with a 20% values is added between each dense layer to reduce overfitting in the training process. The model is then compiled, by defining the loss function (in this case binary cross-entropy loss), the metrics (accuracy) and the optimizer (in this case Adam, with a learning rate of 0.001). Once this is done, the model is fitted by defining the input and output, the number of epochs (100), the batch size (32) and the validation split (5%). A Keras checkpoint callback is used, which is done to store the iteration with the largest validation accuracy, used later to evaluate the model. Figures 12a) and b) show the evolution of the loss and accuracy for both the train and validation datasets during training. This indicates that there is no overfitting, as the train and validation losses are both in the same order of magnitude and the validation loss doesn't increase with the number of epochs. Also, one can see that an optimum accuracy is quickly reached, suggesting that more iterations won't improve the accuracy of the model.



Layer (type)	Output Shape	Param #
input_17 (InputLayer)	[(None, 15)]	0
dense_52 (Dense)	(None, 1024)	16384
dropout_36 (Dropout)	(None, 1024)	0
dense_53 (Dense)	(None, 512)	524800
dropout_37 (Dropout)	(None, 512)	0
dense_54 (Dense)	(None, 256)	131328
dropout_38 (Dropout)	(None, 256)	0
dense_55 (Dense)	(None, 128)	32896
view_out (Dense)	(None, 1)	129

Total params: 705,537
Trainable params: 705,537
Non-trainable params: 0

Figure 11: Deep Neural Network Binary Classifier

When evaluating this model, there is a slight improvement in the metrics when compared to the boosting tree classifier, as shown in the table at the end of the section.

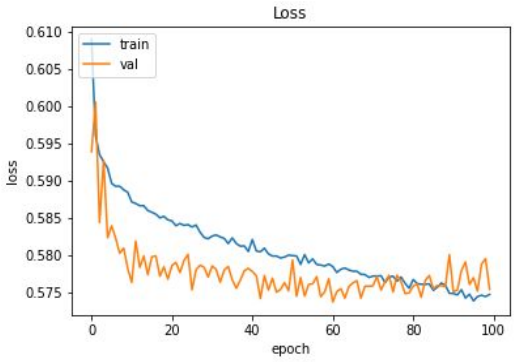


Figure 12a: Loss of training and validation

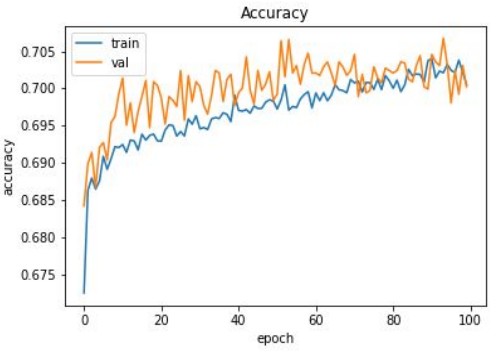


Figure 12b: Accuracy of training and validation

The confusion matrix for this model can be shown in Figure 13, which shows that for 69.24% of the cases the model accurately predicted either a 0 or a 1, for 18.87% of the cases the model predicted that the user wouldn't use the offer when they actually did and for 11.89% of the cases the model predicted that the use would use the offers but they didn't.

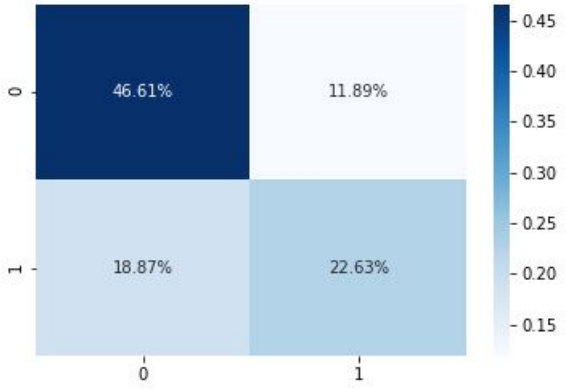


Figure 14: Deep NN Binary Classification confusion matrix

Using this neural network, the scope of the model can be increased, by analysing this as a multi-class classification problem rather than a binary classification problem. The dataset used has four classes, as follows:

1. Offer Viewed and Completed
2. Offer Not Viewed but Completed
3. Offer Viewed but Not Completed
4. Offer Not Viewed and Not Completed

The original model can be adapted to the multi-class model by doing several steps. First, the y-data that includes the labels are still a 1-D array, but rather that being binary (0 or 1) it has four possible values (0, 1, 2 or 3). The output layer of the model has a size of 4 and a "softmax" activation function, which outputs the predicted class in a one-hot encoded format (ie: [0 0 1 0] if the predicted class is 2). The model loss is changed to sparse categorical cross-entropy loss.

The absolute level of accuracy of this model is reduced when compared to the binary classification, but this is simply due to the fact that there are four classes, rather than two. The level of accuracy equivalent to guessing is 25% rather than 50%, so a level of accuracy of around 56%, as obtained, is still much better than just guessing. Furthermore, this new model gives a better understanding of the predictions and of how to target customers, as more information is provided by the predictions. These results are seen in Figures 15a) and b). The former shows that the largest levels of *f1-score*, which is a measure that takes into account both *precision* and *recall*, is class 0, which is the offer being viewed and used. This is good news, as it is the most important metric to predict future purchases. On the other hand, the hardest measure to accurately predict is a customer not viewing the offer but using it (which implies missing on potential customers). It makes sense that this is the toughest to predict, as it is the one with the lowest number of cases, and this model suffers slightly from class imbalance. Figure 15b) shows that the most common mistake made by the model is by confusing 0 and 2 (ie: offer viewed and completed and offer used but not completed). This is a good measure of the limitations of the model . Class 1, as discussed before, is also a challenging class to predict.

	precision	recall	f1-score	support
0.0	0.61	0.66	0.63	2760
1.0	0.46	0.31	0.37	1018
2.0	0.56	0.51	0.53	1721
3.0	0.53	0.64	0.58	1152
accuracy			0.56	6651
macro avg	0.54	0.53	0.53	6651
weighted avg	0.56	0.56	0.56	6651

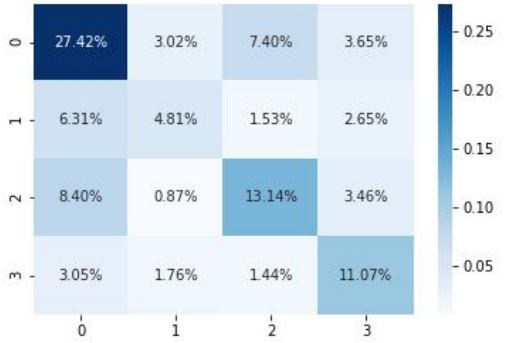
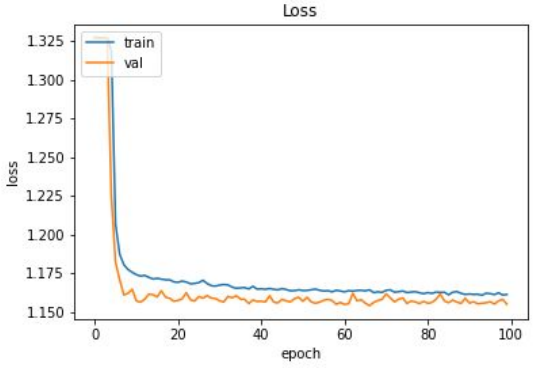


Figure 15a: Classification Report for multi-class deep NN **Figure 15b: Confusion matrix from multi-class deep NN**

As a final validation, the loss and accuracy of the training dataset can be compared versus the validation dataset, as done in Figure 16, to confirm that there is no overfitting.



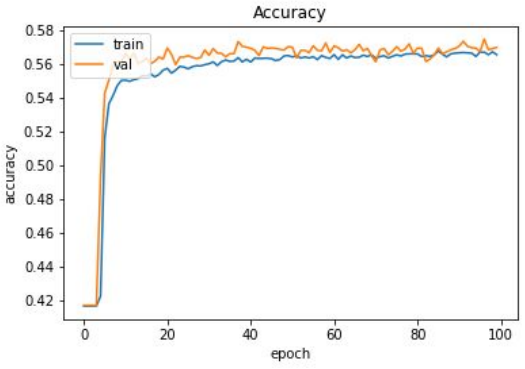


Figure 16a: Loss of training and validation of multi-class deep NN a **Figure 12b: Accuracy of training and validation of multi-class deep NN**

Model	Accuracy (%)	Precision (%)	Recall (%)	f1-score (%)
Logistic Regression	65.6	64.9	65.5	64.6
Boosted Tree Classifier	69.7	69.4	69.7	69.0
Hyperparameter Tuned Boosted Tree Classifier	**69.9**	**69.5**	**69.9**	**69.6**
Deep Neural Network	69.2	69.2	69.6	69.1

Conclusion & Future Steps

In conclusion, a machine learning algorithm has been created to predict if a Starbucks customer will use a promotional offer when receiving it. This has been done by using a combination of customer demographic data, information about the offers, and transactions data.

A simple logistic regression classifier model already provides prediction that are better than guessing. These predictions can be further improved with a boosting tree classifier. By doing a proper hyperparameter tuning, this boosting tree classifier can be optimised to increase the accuracy, precision and recall. A deep neural network can be also used to give results with the same level of accuracy, precision and recall. These networks can be used to predict four possible outcomes (rather than just using the offer or not) therefore giving a more detailed information about the offers, being more helpful more Starbucks. **Overall, a hyperparameter tuned boosted tree classifier is the model that gives the best results when looking for a binary classifier.**

Nonetheless, although the results are good, they are not good enough to be used as the only strategy to identify potential customers or assess the effectiveness of new promotional offers. There are several reasons for this. The first one could be the “low” amount of data. Even if there are more than 50,000 entries, this is still a small number when comparing this to high-performing deep learning models. The benefit of deep learning is that its performance constantly improves when more data is added. Therefore, a solution would be to increase the amount of data used to train the model. This has been partially done in the project by lowering the size of the test and validation datasets without reducing the scope of them. A second reason why the accuracy of the models are not too high is because the data itself is very difficult to predict. Machine learning is not magic, so sometimes it’s difficult to extract patterns because there are no patterns. Although not so extreme, this seems to be partially happening. A good way of knowing this is happening is by looking at the correlation matrices shown in the Data Analysis section. This indicated that there is little-to-no correlation between most of the features in the data. A possible solution would be to do

feature engineering by doing dimensional reduction to try to increase the variance in the data.