Starbucks Reward Program Analysis

Introduction

Once every few days, Starbucks sends out an offer to users of the 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). The project represents a one-month experiment for Starbucks to test out the market and see what offers really excite different demographics of people.

The Github Link of this project can be found here:

https://github.com/pato0301/starbucks_capstone_udacity

Business Context

- The solution here aims to analyse how people make purchasing decisions and how those decisions are influenced by promotional offers.
- Every individual in the dataset has some hidden attributes that impact their buying patterns and are related to their discernible characteristics. Individuals produce different events, including accepting offers, opening offers, and making buys.
- 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 required amount that the user is expected to spend. Offers can be delivered via multiple channels.

Project Motivation

The purpose of this project is to determine which demographic groups respond best to which offer types, to help Starbucks make better decisions on sending out offers to targeted customers to increase sales and potentially save money.

By sending offers to the RIGHT customers:

- Starbucks can boost brand recognition, customer loyalty, increase sales activities, promote a new product, etc.

By NOT sending offers to the RIGHT customers:

- Starbucks can save costs on the promotion event on customers who would make purchases without offers
- Starbucks can prevent customers from reacting negatively to an offer by not sending the offer

Project Goal

In this project, we are going to try to classify the users to see if there is any type of promotion that adjusts better to each user, and therefore make them use it.

Dataset and Inputs

For this project, we will be leveraging the data graciously provided to us by Starbucks /Udacity. This is given to us in the form of three JSON files. Before delving into those individual files, let us first understand the three types of offers that Starbucks is looking to potentially send its customers:

- **Buy-One-Get-One (BOGO):** In this particular offer, a customer is given a reward that enables them to receive an extra, equal product at no cost. The customer must spend a certain threshold to make this reward available.
- **Discount:** With this offer, a customer is given a reward that knocks a certain percentage off the original cost of the product they are choosing to purchase, subject to limitations.
- **Informational:** With this final offer, there isn't necessarily a reward but rather an opportunity for a customer to purchase a certain object given a requisite amount of money. (This might be something like letting customers know that Pumpkin Spice Latte is coming available again toward the beginning of autumn.)

With that understanding established, let's now look at the three provided JSON files and their respective elements:

- portfolio.json — file describes the characteristics of each offer, including its duration and the amount a customer needs to spend to complete it (difficulty).
- profile.json — file contains customer demographic data including their age, gender, income, and when they created an account on the Starbucks rewards mobile application.
- transcript.json — file describes customer purchases and when they received, viewed, and completed an offer. An offer is only successful when a customer both views an offer and meets or exceeds its difficulty within the offer's duration.

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

Portfolio.json

- id (string) — offer id
- offer_type (string) — the type of offer ie BOGO, discount, informational
- difficulty (int) — the minimum required to spend to complete an offer
- reward (int) — the reward is given for completing an offer

- duration (int) — time for the offer to be open, in days
- channels (list of strings)

Profile.json

- age (int) — age of the customer
- became_member_on (int) — the date when the 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 the start of the test. The data begins at time t=0
- value — (dictionary of strings) — either an offer id or transaction amount depending on the record

Data Exploration

In order to analyze the problem better in the next sections, we first need to explore the datasets which include checking the missing value, visualizing the data distribution, etc. In that way, we can have a better understanding of how the dataset looks and how we can feature the data to make it ready for modelling.

```
portfolio.head(5)
```

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

```
portfolio.isnull().sum()
```

```
channels      0
difficulty    0
duration      0
id            0
offer_type    0
reward        0
dtype: int64
```

As shown above, there are no missing values in the portfolio dataset. The channels columns require to be one-hot encoded. we need to rename the id column name to offer_id.

```
profile.head(5)
```

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

```
]: profile.isnull().sum()
```

```
]: age                 0
   became_member_on    0
   gender           2175
   id                  0
   income           2175
   dtype: int64
```

By viewing the first several rows of the dataset, it apparently shows missing values in the age.

```
transcript.head(5)
```

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

```
transcript.isnull().sum()
```

```
event     0
person    0
time      0
value     0
dtype: int64
```

It seems that the transcript file has no missing values.

Data cleaning

1. Portfolio Dataset

The cleaning and preprocessing of the portfolio are performed as follow :

- Rename id column name to offer_id.
- Drop channels columns.
- The offer_type columns require to be one-hot encoded.

2. Profile Dataset

The cleaning and preprocessing of the profile are performed as follow :

- Rename id column name to customer_id.
- Create a readable date format in the became_member_on column
- Impute with most frequent rows with no gender, income, age data.
- Convert gender column values to dummies.

3. Transcript Dataset

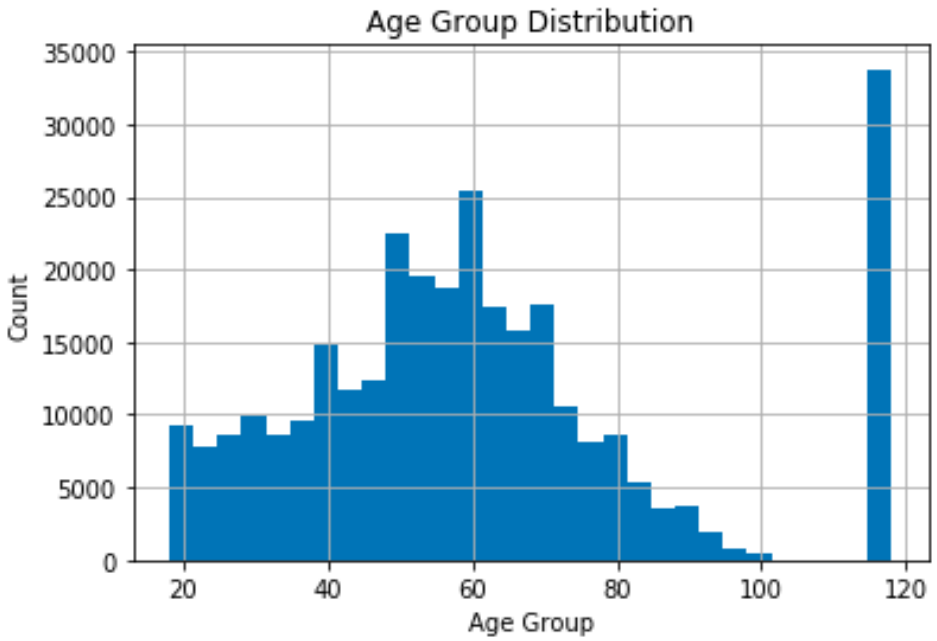
The cleaning and preprocessing of the transcript are performed as follow :

- Rename person column name to customer_id.
- Drop column value.
- Replace empty values in the event with '-' and then convert to dummy.
- Round amounts to two decimals.

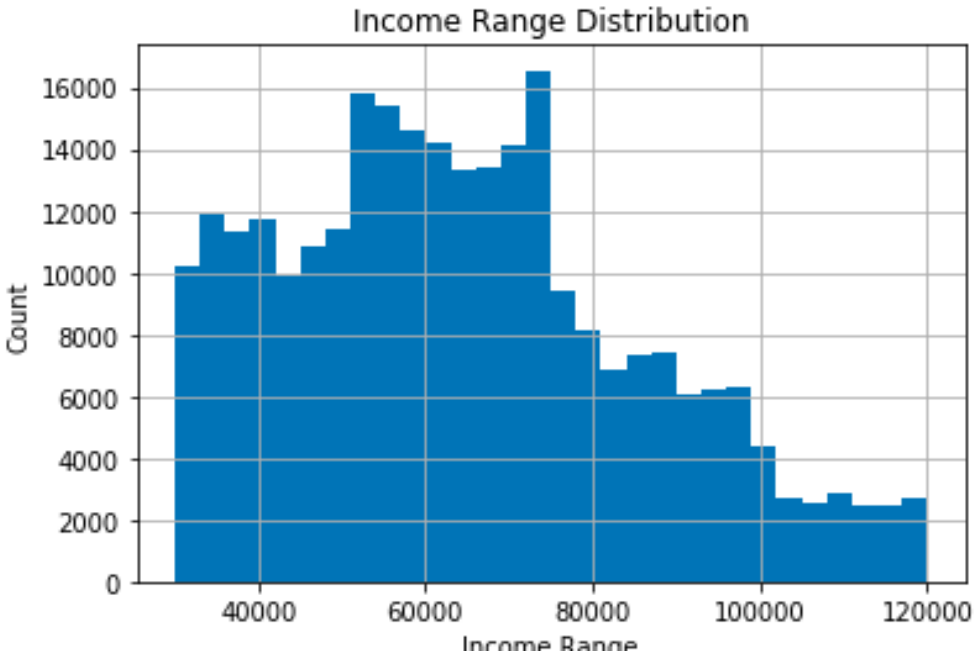
Exploratory Data Analysis

Profile General Distribution — Age, Income, Members

The age distribution plot depicts that the median age of a customer is 60 and most of the customers belong to the age range between 40 to 70.

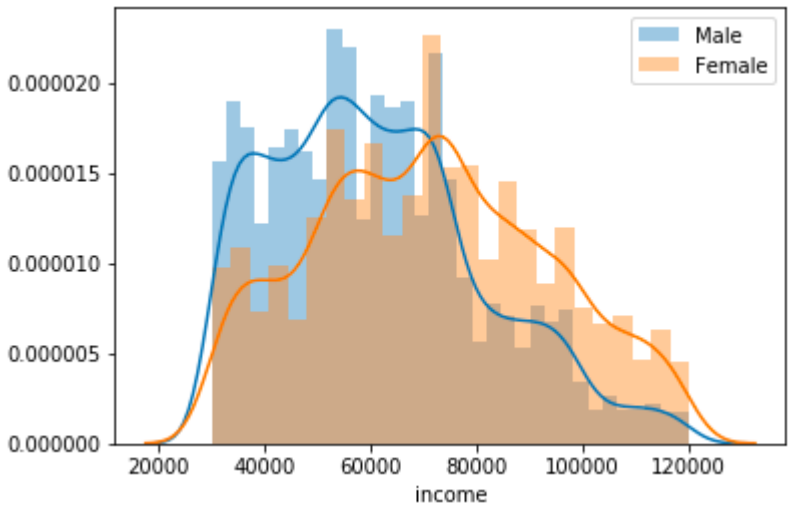


The income distribution plot shows that the number of customers whose average salary is less than 70K is higher than the other side considering 70K to be the median of the income distribution.



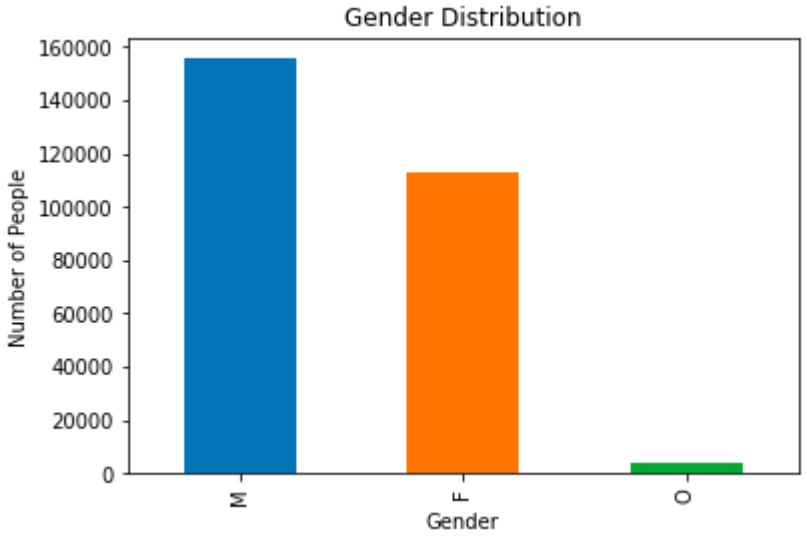
Income distribution as a function a gender

Below plots conclude that minimum and maximum income for both male and female are approximately the same but the count of male customers in low-income level is slightly higher than that of female customers



Gender distribution

The plot shows that there are more male customers joining the program every year.



Combine Data

Offer portfolio, customer profile and transaction data are combined to form a single dataset and train models. In the combined dataset, each row will describe an offer's attributes, customer demographic data, and whether the offer was successful and through which channels they offer was advertised. An offer is said to be successful only if it is viewed and completed within a given duration.

The output of the clean, combined data looks like this

```
]: data.head()
]:
```

	event	customer_id	time	offer-completed	offer-received	offer-viewed	transaction	offer_id	amount	age	...	income	difficulty	duration	offer
0	offer-received	78afa995795e4d85b5d9ceeca43f5fef	0	0	1	0	0	0.0	NaN	75	...	100000.0	5.0	7.0	
1	offer-viewed	78afa995795e4d85b5d9ceeca43f5fef	6	0	0	1	0	0.0	NaN	75	...	100000.0	5.0	7.0	
2	transaction	78afa995795e4d85b5d9ceeca43f5fef	132	0	0	0	1	NaN	19.89	75	...	100000.0	NaN	NaN	
3	offer-completed	78afa995795e4d85b5d9ceeca43f5fef	132	1	0	0	0	0.0	NaN	75	...	100000.0	5.0	7.0	
4	transaction	78afa995795e4d85b5d9ceeca43f5fef	144	0	0	0	1	NaN	17.78	75	...	100000.0	NaN	NaN	

Building Models

Four different models with accuracy are built to predict whether a customer will respond to an offer. Before building models, our dataset should be split into train and test datasets so a model doesn't overfit the data and also a test dataset is used to evaluate how well our models are performing.

```
#split the dataset into test and train sets.
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((214573, 18), (91961, 18), (214573,), (91961,))
```

Also, before building our model let scale the data and build a pipeline. Two scale the date we need to split the data first, what we already did, so there is no data leakage.

```
std = StandardScaler()
X_train.income = std.fit_transform(X_train.income.values.reshape(-1, 1))
# X_train.age = std.fit_transform(X_train.age.values.reshape(-1, 1))

X_train.reset_index(inplace=True)
X_train = X_train.drop(['index'], axis=1)
```

```
X_test.income = std.transform(X_test.income.values.reshape(-1, 1))
# X_test.age = std.fit_transform(X_test.age.values.reshape(-1, 1))

X_test.reset_index(inplace=True)
X_test = X_test.drop(['index'], axis=1)
```

Now we can build our pipeline:

```
numerical_transformer = SimpleImputer(strategy='mean')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[ ('num', numerical_transformer, num_avg_list),
                   ('cat', categorical_transformer, num_ofent_list)])
```

Logistic Regression Model

Logistic Regression's instance is created with liblinear classifier and will serve as our base model.

Model built is as below

```
# instantiate a logistic regression classifier object
lr_clf = LogisticRegression(random_state=42, solver='liblinear')

# fit train data to the model
lr_clf_pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', lr_clf) ])
lr_clf_pipeline.fit(X_train, y_train)
```

The metrics of LogisticRegression are as below

```
preds_lr = lr_clf_pipeline.predict(X_test)

# Evaluate the model
score = accuracy_score(y_test, preds_lr)
print('Accuracy:', score)
```

```
Accuracy: 0.699938017203
```

```
print(classification_report(y_test, preds_lr, labels=[0,1, 2, 3]))
```

```
              precision    recall  f1-score   support

           0       0.46      0.94      0.62     23051
           1       0.34      0.07      0.12     17315
           2       1.00      1.00      1.00     41412
           3       0.34      0.00      0.01     10183

   micro avg       0.70      0.70      0.70     91961
   macro avg       0.54      0.50      0.44     91961
weighted avg       0.67      0.70      0.63     91961
```

Accuracy

- Logistic regression: 0.699

F1-score

- Logistic regression: 0.63

Random Forest Classifier Model

A Random Forest Classifier's instance is created and parameters like n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf are tuned to fit the training data using RandomizedSearchCV.

Model built is as below

```
# instantiate a random forest classifier obj
rf_clf = RandomForestClassifier(random_state=42)

# Number of trees in random forest
n_estimators = [10, 50, 100]

# Number of features to consider at every split
max_features = ['auto', 'sqrt']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.arange(3, 11)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]

# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]

# Create the random grid
grid_params = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

# tune the classifer
rf_random = RandomizedSearchCV(estimator = rf_clf,
                               param_distributions = grid_params,
                               n_iter = 10,
                               cv = 3,
                               verbose=2,
                               random_state=42,
                               n_jobs = -1)

# fit train data to the classifier
rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', rf_random) ])
rf_pipeline.fit(X_train, y_train)
```

The metrics of RandomForestClassifer are as below

```
# Preprocessing of validation data, get predictions
preds_rf = rf_pipeline.predict(X_test)

# Evaluate the model
score = accuracy_score(y_test, preds_rf)
print('Accuracy:', score)
```

```
Accuracy: 0.700981937995
```

```
print(classification_report(y_test, preds_rf, labels=[0,1, 2, 3]))
```

```
              precision    recall  f1-score   support

           0       0.46      1.00      0.63     23051
           1       0.00      0.00      0.00     17315
           2       1.00      1.00      1.00     41412
           3       0.00      0.00      0.00     10183

   micro avg       0.70      0.70      0.70     91961
   macro avg       0.36      0.50      0.41     91961
weighted avg       0.56      0.70      0.61     91961
```

Results suggest that a random forest model's accuracy and f1-score is better than the naive predictor

Accuracy

- Base predictor: 0.699
- Random forest: 0.700

F1-score

- Base predictor: 0.63
- Random forest: 0.61

KNN Model

A K Nearest Neighbors model.

Model built is as below

```
# instantiate a KNN regression classifier object
knn_clf = KNeighborsClassifier(n_neighbors=4)

# fit train data to the model
knn_clf_pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', knn_clf) ])
knn_clf_pipeline.fit(X_train, y_train)
```

The metrics of KNN are as below

```
preds_knn = knn_clf_pipeline.predict(X_test)

# Evaluate the model
score = accuracy_score(y_test, preds_knn)
print('Accuracy:', score)
```

```
Accuracy: 0.679896912822
```

```
print(classification_report(y_test, preds_knn, labels=[0,1, 2, 3]))
```

```
              precision    recall  f1-score   support

           0       0.48      0.66      0.56     23051
           1       0.34      0.24      0.29     17315
           2       1.00      1.00      1.00     41412
           3       0.25      0.16      0.19     10183

   micro avg       0.68      0.68      0.68     91961
   macro avg       0.52      0.52      0.51     91961
weighted avg       0.66      0.68      0.66     91961
```

Accuracy

- Base predictor: 0.699
- Random forest: 0.67

F1-score

- Base predictor: 0.63
- Random forest: 0.66

Artificial Neural Networks Model (ANN)

A Keras ANN model has an input layer with 6 nodes and 2 hidden layers, all of which with ReLU activation function. Finally an output layer, with four nodes and a softmax activation function.

Model built is as below

```
ann = keras.models.Sequential()
```

```
input_shape = (X_train_keras.shape[1],)
ann.add(keras.layers.Dense(6,input_shape=input_shape, activation='relu'))
```

```
ann.add(keras.layers.Dense(6, activation='relu'))
ann.add(keras.layers.Dense(6, activation='relu'))
ann.add(keras.layers.Dense(4, activation = 'softmax'))
```

```
ann.compile(optimizer = 'adam',
            loss = 'sparse_categorical_crossentropy',
            metrics = ['accuracy'])
```

```
ann_history = ann.fit(X_train_keras, y_train_keras, validation_data=(X_test_keras, y_test_keras), epochs=15, batch_size
```

```
# Summary of our model
ann.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6)                 114
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 28
=================================================================
Total params: 226
Trainable params: 226
Non-trainable params: 0
```

The metrics of KNN are as below

```
ann.evaluate(X_test_keras , y_test_keras)
```

```
91961/91961 [==============================] - 2s 23us/step
```

```
[0.50758576635422614, 0.73645349658964987]
```

```
y_pred_ann = ann.predict_classes(X_test_keras)
```

```
91961/91961 [==============================] - 1s 15us/step
```

```
print(classification_report(y_test_keras, y_pred_ann, labels=[0,1, 2, 3]))
```

```
              precision    recall  f1-score   support

           0       0.51      1.00      0.68     23051
           1       0.58      0.16      0.25     17315
           2       1.00      1.00      1.00     41412
           3       0.60      0.06      0.11     10183

   micro avg       0.74      0.74      0.74     91961
   macro avg       0.67      0.55      0.51     91961
weighted avg       0.75      0.74      0.68     91961
```

Accuracy

- Base predictor: 0.699
- ANN: 0.736

F1-score

- Base predictor: 0.63
- Random forest: 0.68

Conclusion

The problem that we chose to solve was to build a model that predicts which offer will a customer respond to better.

My strategy for solving this problem has mainly two steps:

- First, we combined offer portfolio, customer profile, and transaction data.
- Second, we assessed the accuracy and F1-score of a base model and then we compared the performance of the random forest models, the KNN model and the ANN model.

This analysis suggests that an ANN model has the best training data accuracy and F1-score with 0.736 and 0.68 respectively.

So, in conclusion, we can say that the best model to classify a customer is the ANN model.

Improvements

The performance of an ANN can be still improved by analysing features that impact an offer's success rate as a function of offer difficulty, duration, and reward. These additional features should provide the model with the opportunity to construct a better decision boundary to cluster the users.

Moreover, initially, it seemed like we had a lot of data to work on, but once NaN values and duplicate columns were dropped and the data were combined into one single dataset, it felt as though the models might have benefited from more data. With more data, the classification models may have been able to produce better accuracy and F1-score results.

Additionally, better predictions may have been deducted if there were more customer metrics. For this analysis, I feel we had limited information about the customer available to us — just age, gender, and income. To find optimal customer demographics, it would be nice to have a few more features of a customer. These additional features may help in providing better classification model results.