Starbucks Capstone Challenge

Introduction

This project is part of the Udacity Capstone project for Udacity data scientist Nanodegree course. We have been given data that contains customer information and behaviour on the mobile application of Starbucks. Indeed, Starbucks often propose an offer to the users, it can be advertising but also some gift such as a discount or a free drink when you buy one. Some users might not receive any offer in certain weeks.

The data are presented in 3 files:

-First, the Portfolio file that gives a description of each offer with information such as the duration, the type of the offer and the difficulty (amount a customer should spend to get it).

-Then, the Profile data file contains information about customer age, gender, income and account date creation.

-Last, the Transcript file contains element about client purchases and the reception, view and completion of the offer.

The objective of the project is to use the data in order to construct a model that predicts whether a client will respond to an offer, then find out the number of customers that received a specific offer and last provide an estimation of rate of success.

The project will be divided in the following steps:

-Cleaning of the data, this will help us to extract useful information from the data.

-Data exploration and Visualization

-Pre-processing the data: it will enable to analyse the main drivers of a transaction.

-Feature engineering: we will try to use some columns in order to create new features like finding the length of customer's membership, the number of offers received for each customer or the time lap between offers.

-Model Construction: Using response flag generated in earlier steps, we will build a model that predicts whether the client will respond to an offer or not.

-Improvement of the results by implementing a GridSearch method that will enable us to find the best parameters and therefore improve the model performance.

Metrics

The classification model will be based on accuracy as the model evaluation metric.
The reasons for this choice are that we only have a binary classification (offer viewed or offer completed), also, this metric allows us to judge the accuracy of the model by making a comparison of the number of correct predictions with the total predictions that were made.

Analysis

Data Exploration

We first need to manipulate the data and present them in an appealing form. This implies exploring the dataset, looking for the missing values, analysing the distribution of the data.
It will help us to know how to re-feature the data for the modelling part.

1.Portfolio Dataset

This dataset is composed of the following columns:
-id(string): offer id.
-offer_type: this shows the offer sent to the client, this can be BOGO, discount or informational.
-difficulty: this shows the minimum required amount to spend to complete the offer.
-reward: this is the reward given when the offer is complete.
-duration; it shows the time needed in days for an offer to be opened.
-channels: it includes mobile, social media, website or email.

	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
5	[web, email, mobile, social]	7	7	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	3
6	[web, email, mobile, social]	10	10	fafdcd668e3743c1bb461111dcafc2a4	discount	2
7	[email, mobile, social]	0	3	5a8bc65990b245e5a138643cd4eb9837	informational	0
8	[web, email, mobile, social]	5	5	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5
9	[web, email, mobile]	10	7	2906b810c7d4411798c6938adc9daaa5	discount	2

→The information we learnt from the portfolio dataset are that the dataset has no null or duplicates values. Also, the channels column needs to be one-hot encoded.

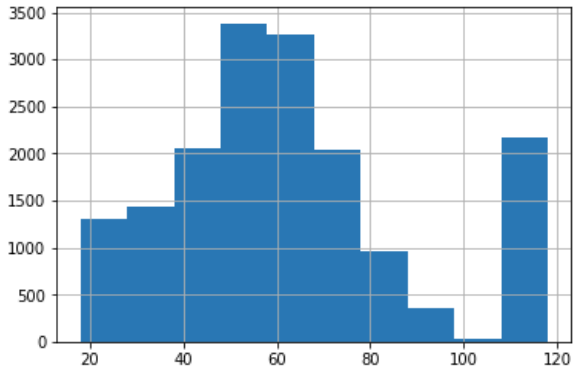
2. Profile Dataset

This dataset contains 17,000 rows meaning there are 17,000 unique customers information within the dataset. Moreover, the dataset has 5 columns:
-age
-became_member_on: this is the date at which the client created his account.
-gender of the customer
-id
-income of the client

	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

The dataset has 2175 missing values on each of 'gender', 'income' variables.
Also, the customers ages range from 18 to 101 years old. Nevertheless, the 2175 customers mentioned above were registered with an age of 118 years old, this shows that there is an issue to address with these 2175 rows of the dataset.
Last, the statistics summary of age and income features shows that the data are following a normal distribution.

Let's now explore the 'gender', 'age' and 'income' variable.
→First, the missing values for gender and income for the 2175 clients that were assigned the age of 118 has no assigned 'gender' and 'income' features. Therefore, we need to address this issue in the Data-processing section.

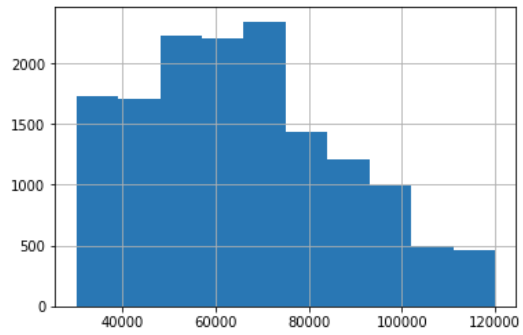


Age distribution

We can therefore confirm that 118 is an outlier and does not follow a normal distribution.

→The data shows three gender categories for the clients, Male, Female and a category labelled O. We have 57% of men, 41% of women and only 1,4% of clients in the 'O' category.

→Last, customer incomes lie between 30,000 and 120,000 as shown in the graph bellow.



Income distribution

3.Transcript Dataset

The four columns of this dataset are:
-event: this can be either transaction, offer received or offer viewed.
-person: customer_id.
-time: the number of hours since the start of the test.
-value: it can take the values of offer_id, reward and/or difficulty.

	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'}

There is some clearing that should be apply in this dataset. First, we should process the column of value and last, we will segregate the offer and transaction data.
Indeed, it appears as though the offer id column ended up being duplicates so we have to clean it up further to ensure there is only one offer id column.

Data processing

Cleaning and pre-processing of this dataset consist of Renaming id column name to offer_id, then we will make the channels columns one-hot encoded and last the offer_type column will be one-hot encoded as well.
We need first to identify the characteristics of an effective offer. We also process the data in order to merge the events of each specific offer that has been sent, this will allow us to determine which offer has been received, viewed and completed.
As offer_id is not linked with a transaction event, we must link each offer id to its transaction event. Last, the discount offers and the BOGO one will be marked as received, viewed and completed transaction and so such type of offer should be sent out. Also, we need to address a logistic point, even if there is no reward linked to the information offer, we should link a transaction to the usage of this offer.

```
# clean up the dataframe (removing the dummy columns)
transcript_processed['offer_id'] = np.where(transcript_processed['offer_id_x'].isnull(),\
                                            transcript_processed['offer_id_y'], transcript_processed['offer_id_x'])
transcript_processed.drop(columns=['offer_id_x','offer_id_y'], axis=1, inplace=True)
```

```
#merge portfolio dataset to get offer data
transcript_processed = transcript_processed.merge(portfolio, how = 'left',on='offer_id')
transcript_processed['duration'] = np.where(transcript_processed['duration_x'].isnull(), \
                                            transcript_processed['duration_y'], transcript_processed['duration_x'])
transcript_processed.drop(columns=['duration_x','offer_type_x','difficulty_x','channels_x','duration_y'],\
                          axis=1, inplace=True)
transcript_processed.rename(columns={'channels_y':'channels','reward_y':'reward','difficulty_y':'difficulty','offer_type_y':'o
ffer_type'},inplace=True)
```

```
# quick check on processed dataset
transcript_processed.head()
```

	event	person	time	value	amount	offer_id	channels
0	offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]
1	offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	NaN	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]
2	transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}	22.16	5a8bc65990b245e5a138643cd4eb9837	[email, mobile, social]
3	offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]
4	offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	NaN	3f207df678b143eea3cee63160fa8bed	[web, email, mobile]

After combining these data, we still face an issue to solve. In fact, we should remove the transactions which were completed after the offer was received and viewed. The solution would be to extract the transactions converted from offers by checking if the offer id before and after the transaction are matching.

```
# subset the dataset with only offer viewed, transaction, and offer completed events
transactions_after_viewed = transcript_processed[(transcript_processed['event']=='offer viewed')|\
                                                 (transcript_processed['event']=='transaction')|\
                                                 (transcript_processed['event']=='offer completed')].copy()

# generate the previous offer id
transactions_after_viewed['pre_offer_id'] = transactions_after_viewed.groupby(['person', 'offer_id'])['offer_id'].shift()

# create flag for responsed offer which competed after customer viewing the offer
transactions_after_viewed['completed_offer'] = np.where(transactions_after_viewed['pre_offer_id']==\
                                                         transactions_after_viewed['offer_id'],1,0)
```

```
# join back the 'offer received' events which was filtered out in the previous step
offer_received = transcript_processed[transcript_processed['event']=='offer received']

offer_received['pre_offer_id']=np.nan
offer_received['completed_offer']=np.nan

transcript_processed = offer_received.append(transactions_after_viewed).sort_values(['person','time'])
transcript_processed.head()
```

We just have to make few other adjustments in order to be able to prepare and implement the model.
First, we separate the transcript data by offer type in order to ease the processing.
Also, by using responded_offer flagged, we are able to filter out the offers which were viewed and completed by the customers. As mentioned before, discount and BOGO offer type are considered as completed offer, however, informational offers are considered as transaction.
We then identify clients who ‘viewed’ the offers without transaction and ‘completion’ in contrast to the users who received an offer without viewing it. We are then in a position to differentiate who only viewed an offer to the ones who didn’t open an offer.
We apply the same for Bogo and discount offers type.

Building-Model

The aim is to build three different models with accuracy and F1-score being the principal metrics that we are looking for in order to predict if a user will respond to an offer or not.
The implementation of the model consists of preprocessing the dataset by removing missing values, merge the dataset and last make a categorical column of dummy variables.

Firstly, we define our target and features variables.

```
def data_prep(df,col_drop):
    '''
    inputs:
    - df: prepared dataframe for modeling and columns to be dropped

    outputs:
    - Returns 2 dataframes - features and target dataframes
    '''
    # Split the data into features and target label
    target = df['effective_offer']
    features = df.drop(columns = col_drop,inplace=False,axis=1)
    return features,target
```

The following step is to separate data into training and test sets.

```
def model_pipeline(features,target):
    '''
    inputs:
    - features & target dataframe

    outputs:
    - Splits features and target dataframe to train and test sets, performs feature scaling on both datasets.
    - Outputs X_train, X_test, y_train and y_test dataframes
    '''

    #split into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(features,target, test_size=0.20, random_state=42)

    #fit and transform scaling on training data
    scaler=StandardScaler()
    X_train=scaler.fit_transform(X_train)

    #scale test data
    X_test=scaler.transform(X_test)
    return X_train,X_test,y_train, y_test
```

Then, we implement Grid Search matrix to determine what would be the optimal parameters for the model. For all three offers, the Random Forest model has relatively good performance, so we use Grid Search on this to determine the best parameters.

```
#define Grid Search function
def rand_forest_param_selection(X,y):
    '''
    input:
    - X,y: training datasets for X and y
    output:
    - dictionary with best parameters for random forest model
    '''

    param_grid={'max_features': ['auto', 'sqrt'],
                'max_depth' : [5,10,15,20],
                'n_estimators': [25,30,40,50],
                'min_samples_split': [2, 10, 20],
                'min_samples_leaf': [2, 10,15, 20],
                }
    grid_search = GridSearchCV(RandomForestClassifier(random_state=2), param_grid)
    grid_search.fit(X, y)
    grid_search.best_params_
    return grid_search.best_params_
```

Last, we just have to find the best model results for each type of offer.

```
#define function to find best model results for each offer type
def best_model(offer_type):
    '''
    input:
    - offer_type: string of offer type name
    output:
    - dataframe containing results of best model so far

    '''
    print('For ' + offer_type + ' RF model:')
    return results.transpose()[results.transpose()['testing_score']==results.transpose()[results.transpose().index.str.contain
s("RandomForestClassifier_"+offer_type)]['testing_score'].max()]
```

Results

We have refined the three models and we can assess the results for our best models for all 3 and check the feature importance to see the top drivers of effectiveness of offers.

	RandomForestClassifier_bogo_2	RandomForestClassifier_bogo_2	RandomForestClassifier_discount_1	RandomForestClassifier_info_3
pred_time	0.043968	0.045300	0.034291	0.024378
testing_score	0.829943	0.829943	0.873477	0.753042
train_time	0.171545	0.175694	0.151919	0.090394
training_score	0.845050	0.845050	0.872876	0.755953

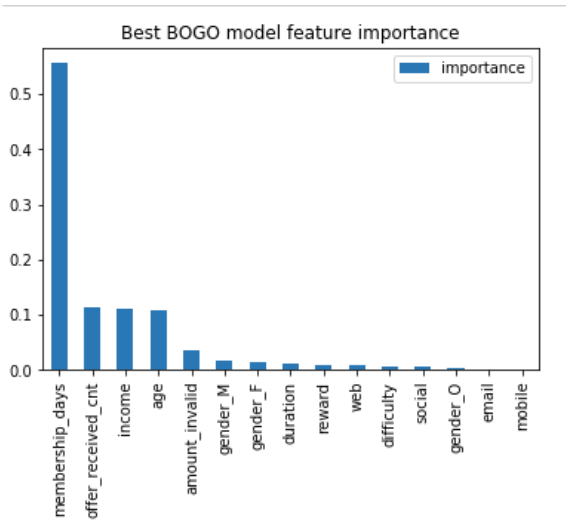
Overall, we can see that the top performing models are the 2nd model (with GridSearch to find optimal model parameters and removing amount_invalid column) for predicting effectiveness of BOGO and discount offers, and informational offers.

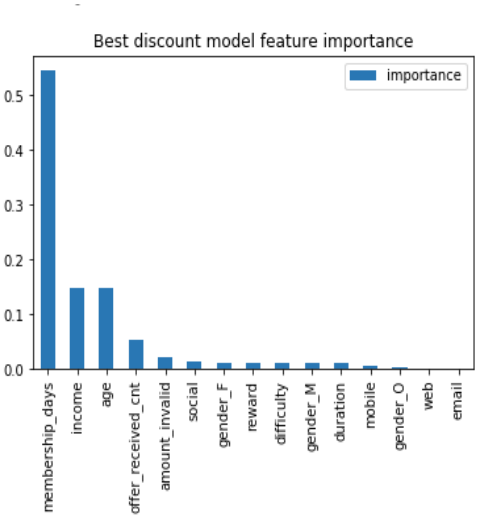
In order to find the most influential drivers of an effective offer, we can check the feature importance of our best models above.

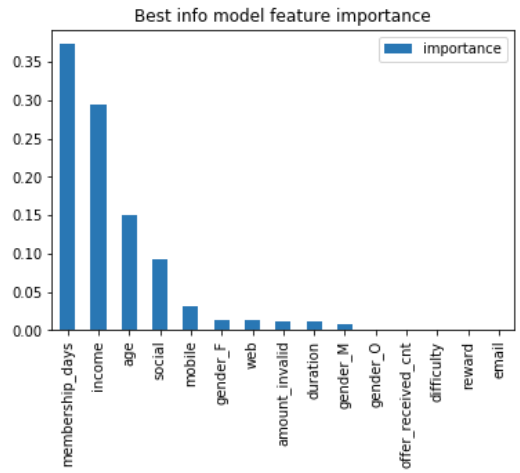
The table shows that the tune parameter allowed us to get an accuracy of 83% for BOGO, which is much higher than the BOGO baseline conversion rate (48%). We also get similar great improvement for the discount offer and informational offers.

We can therefore affirm that we managed to get a great improvement on the capability to predict if a user will respond to an offer sent or not.

Last, as we perform a feature analysis, we find out that the principal drivers of an effective offer is the length of a membership, indeed, the longer the client is a member of the app, the more likely the user will respond to a received offer. Also; age and income are important features that determine if the customer will give suite to a received offer.







Conclusion

The aim of this project was to construct a prediction model of user respond to an offer.
First, we merged offer portfolio, customer profile and transaction dataset and we assessed the accuracy and F1-score of a dummy model that assumed all offers to be successful.
The results obtained were very encouraging, with model getting a good performance, for instance, Bogo getting an accuracy of 0.838, discount 0.875 and informational 0.756.
We also derived that the length of membership was a major feature affecting whether the user will respond to an offer or not. The other important factors were age and income of the customer.

Improvement

There is lot of room for improvement in this model.
For instance, the random forest model can be more performant on detecting factors that impact an offer's success rate as a function of time and reward.
These additional features could help the random forest classifier to take more accurate decision boundary between successful and unsuccessful offers.

Also, a way to perform the model would have to get more data, this would have helped the classification models to be more accurate and would also lead to better F-1 score results.

Last, an essential element that would have increased the models performance is more details and information about the users.
Indeed, if we had more metrics about the client like address, ethnicity, job status etc. we would have been able to get a more performant classification model.