Machine Learning Engineer Nanodegree

Capstone Project

Harold Giovanny García Rodriguez

July 23, 2022

I. Definition

Project Overview

Companies that make an investment in marketing want to get as much profit as possible, so if the customer profile is not considered for the marketing campaigns the profit will decrease, a solution to this problem is the targeted marketing using machine learning techniques. The main idea of the targeted marketing is that we can obtain target groups of potential customers who have a high probability of acquiring the product or service that is being offered, to avoid wasting money offering it to people who are most likely not interested, basing the analysis and the model on demographic data of the clients, the type of offers available and the historic records of previous responds of the clients for different offers.

Problem Statement

It is important that each economic and operational effort made by the company to promote its offers has the highest possible return. That's why the main goal of this project is to use the data to identify which people would have a positive response to each type of offer that the company would send, and thus be able to send offers in a more personalized level and with better performance. The solution to the problem consist in making a binary classifier model, It is going to be a binary classifier because I am only considering two states, the person would complete an offer or not, to do this I am going to analyze and select the most relevant features with high predicting power to train several models, and then test it to get the metrics that will allow me to choose the model with the best performance. The output of the final model is going to be a prediction that indicates whether or not someone is going to respond to an offer.

Metrics

Considering that the final model will be a classification model, for evaluating it I am going to use four metrics: Accuracy, Precision, recall and the F1-Score, these are explained in more detail below:

- Accuracy: It is the fraction of predictions our model got right. Formally, accuracy has the following definition [1]:

  $$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

  For binary classification, accuracy can also be calculated in terms of positives and negatives as follows [1]:

  $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

- Precision and Recall:
  In this context these are two numbers which together are used to evaluate the performance of a binary classification model [2], based on the original and predicted condition (positive or negative), thus determining the values of true positives (true condition predicted as true), true negatives (false condition predicted as false), false positives (false condition predicted as true) and false negatives (true condition predicted as false).

Mathematically, precision and recall can be computed as follows:

Precision attempts to answer the following question: What proportion of positive identifications was actually correct? [3]

$$Precision = \frac{\sum True\ Positive}{\sum Predicted\ Condition\ Positive} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall attempts to answer the following question: What proportion of actual positives was identified correctly? [3].

$$Recall = \frac{\sum True\ Positive}{\sum Original\ Condition\ Positive} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

- F1-Score:
  The F-score is a measure of a test's accuracy. It is calculated from the precision and recall of the test. It is the harmonic mean of the precision and recall, it is important to know that the highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero [4]. The mathematical definition of the F1-Score is:

$$F_1 = \frac{2}{recall^{-1} + precision^{-1}} = 2 * \frac{precision * recall}{precision + recall}$$

II. Analysis

Data Exploration and Visualization

The data for this project is divided in three JSON files, and was proportioned by Udacity's Machine Learning Nanodegree:

- portfolio.json - containing offer ids and metadata 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

Below is the detailed description of each of the variables contained in the JSON 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

Exploring a little deeper the datasets I found the following relevant information:

- The profile dataset has the following statistics for the numeric features:

	age	income
count	17000.000000	14825.000000
mean	62.531412	65404.991568
std	26.738580	21598.299410
min	18.000000	30000.000000
25%	45.000000	49000.000000
50%	58.000000	64000.000000
75%	73.000000	80000.000000
max	118.000000	120000.000000

.

From where the maximum value of the age feature (118 years old) is not very common, another aspect to take into account is the count of rows with valid data of the features, while the age has 17000 data values, the income has only 14825.

- The portfolio dataset has the following statistics:

	reward	difficulty	duration
count	10.000000	10.000000	10.000000
mean	4.200000	7.700000	6.500000
std	3.583915	5.831905	2.321398
min	0.000000	0.000000	3.000000
25%	2.000000	5.000000	5.000000
50%	4.000000	8.500000	7.000000
75%	5.000000	10.000000	7.000000
max	10.000000	20.000000	10.000000

All the three numeric features have the same count of valid data, which is a good indicator. For the other hand, the mean value of the reward for the customer is 4.2 dollars, while the mean difficulty (money to expend in the offer in the duration time) for earning the reward is 7.7 dollars, finally the mean duration of the offers is 6.5 days.

- There are 2175 null values for the gender and income columns out of 17.000 total rows on the **profile** dataset.
- From the histogram in the Fig 1, we can see that it has the shape of a normal distribution, but also notice that at the right of the plot it has more than 2000 values for an age of 118 years old which doesn't seem very common. This may indicate that these are outliers. The other values of the histogram follows a normal distribution.

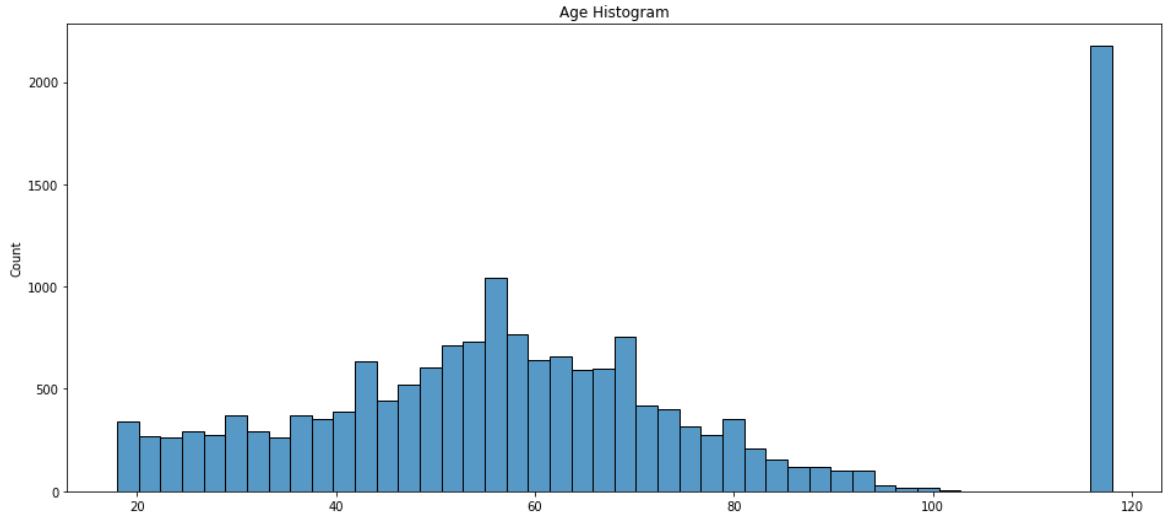

Fig 1. Age histogram

- On the Fig 2 shows that in 2015 and 2017 there are a significant increase of new user after the 8th month (August) of the year, meanwhile the rest of the years does not show any significant trend between months.

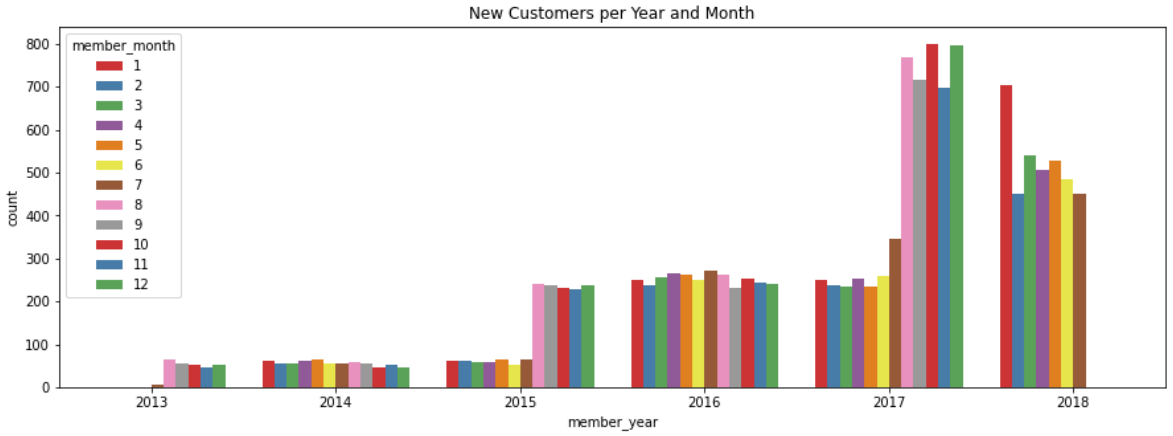

Fig 2. New customers per Year and Month

- From the count plot on the Fig 3 is clear that most observations at gender feature are male, and it is important to note that the number of observations for other genders are much less than those in the Male and Female, representing only the 1.4% of the total clients.

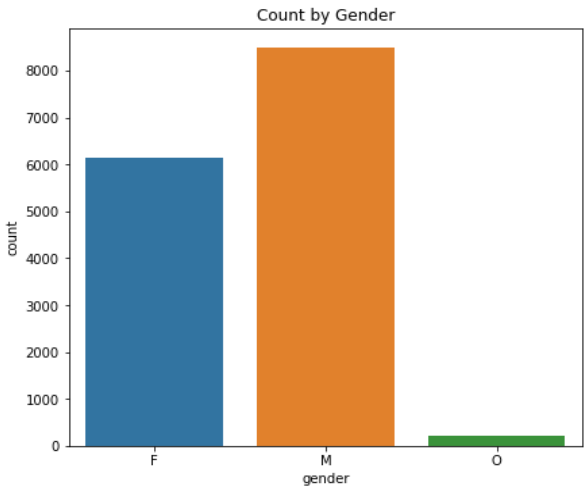

Fig 3. Count of Gender data

- On the transcript dataset there are 374 duplicated records considering all the columns, which doesn't make much sense that a person completed the same offer twice at the same time.
- For the informational offers the transcript dataset doesn't have any event for "offer completed" column. This is because the informational offers don't have a difficulty value, so we need a way to measure if a transaction was influenced by the informational offer before the end of the duration time of the offer.
- Taking the original transcript dataset, from above 12729 received offers weren't viewed and 27280 weren't completed.
- In a pivot table where I wanted to get at what time each event occurred for each combination of customer and offer, there are some offers that were sent more than once to some customers, and for this combination of offers and customers, the pivot table is having trouble because the time values of the events for each offer and person are being aggregated, which don't allow me to get the actual time of each event separately.

Algorithms and Techniques

For solving this problem I used multiple binary classification algorithms to choose the one with the best performance at the chosen metric. The three binary classification algorithms that I used are listed below, all takes as input two arrays: an array X, sparse or dense, of shape (n_samples, n_features) holding the training samples, and an array Y of integer values, shape (n_samples,), holding the class labels for the training samples [5], additionally each algorithm has a default parameter configuration for several parameters, I will briefly describe the main parameters of each one:

- Decision Tree Classifier [6]:
    - Criterion: The function to measure the quality of a split (default="gini").
    - Splitter: The strategy used to choose the split at each node (default = "best").
    - max_depth: The maximum depth of the tree (default=None).
    - min_samples_split: The minimum number of samples required to split an internal node (default=2).
    - min_samples_leaf: The minimum number of samples required to be at a leaf node. (default=1).

- Random Forest Classifier [7]:
    - n_estimators: The number of trees in the forest. (default=100)
    - Criterion: The function to measure the quality of a split (default=”gini”).
    - Splitter: The strategy used to choose the split at each node (default = “best”).
    - max_depth: The maximum depth of the tree (default=None).
    - min_samples_split: The minimum number of samples required to split an internal node (default=2).
    - min_samples_leaf: The minimum number of samples required to be at a leaf node. (default=1).
- XGBoost Classifier [8]:
    - n_estimators: Number of boosting rounds.
    - max_depth: Maximum tree depth for base learners
    - gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.

Benchmark

The benchmark I defined for this project is given by the performance of the Logistic Regression model, I implemented this model through the Scikit-Learn python module that allowed me to get a model with only three arguments: an array X with the training samples, an array Y with the training labels, and finally defining which solver to use on the algorithm. The chosen solver was 'liblinear' and the results obtained are below:

- The LogisticRegressionmodel got an accuracy score of: 0.8209403635271143
- The LogisticRegressionmodel got an F1_score score of: 0.7029900332225913

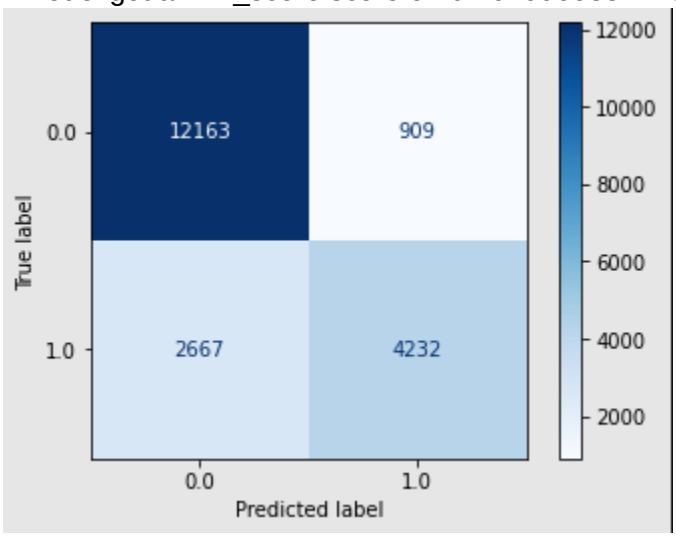

Fig 4. Logistic Regression Confusion Matrix

III. Methodology

Data Preprocessing

The data preprocessing was the key process in this project, because the data was not correctly structured for the modeling process, and because the data was divided in three different datasets, the preprocessing for each dataset will be described below:

- Profile dataset:
    1. From the dataset all the 2175 rows in wich age has a value of 118 (Fig. 1), also have null values at gender and income columns, this represents the 12.8% of the 17000 rows, and due to the lack of features where I can predict the gender, age and income (the only feature remaining is the date where the customer became a member) for these customers I decided to drop this rows.
    2. Store in a list the id of the deleted people in the step 1.
    3. Set the 'became_member_on' as a datetime type to be able to extract this values easily.
    4. Set a new column to include separately the day, month and year on when the user became a member.
    5. Transform a categorical variable to multiple numerical variables by getting dummy variables for gender column, and concatenate those dummy variables columns to the main "profile" dataframe
- Portfolio dataset:
    1. Create a new column for each channel of the "channels" column to indicate numerically if the offer was sent through that channel or if it wasn't.
    2. Add columns to the portfolio dataset that includes the dummy variables for the "offer_type" feature.
- Transcript dataset:
    1. Filter the transcript dataset to drop the events and transactions made by people deleted in the first preprocessing step of the profile dataset.
    2. Get the unique labels of the event column.
    3. Create a new dataframe *transaction_df* with only transaction events.

	person	event	value	time
12654	02c083884c7d45b39cc68e1314fec56c	transaction	{'amount': 0.8300000000000001}	0
12657	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	transaction	{'amount': 34.56}	0
12659	54890f68699049c2a04d415abc25e717	transaction	{'amount': 13.23}	0
12670	b2f1cd155b864803ad8334cdf13c4bd2	transaction	{'amount': 19.51}	0
12671	fe97aa22dd3e48c8b143116a8403dd52	transaction	{'amount': 18.97}	0

    4. Create a new dataframe *offer_events_df* with the offer events (offer received, offer viewed and offer completed)

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0
5	389bc3fa690240e798340f5a15918d5c	offer received	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	0
7	2eeac8d8feae4a8cad5a6af0499a211d	offer received	{'offer id': '3f207df678b143eea3cee63160fa8bed'}	0
8	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0

5. Extract the dictionary value from each row of the "value" column in the transaction_df dataframe with the "*amount*" key, and put this value in a new column named "*amount*".

	person	event	time	amount
12654	02c083884c7d45b39cc68e1314fec56c	transaction	0	0.83
12657	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	transaction	0	34.56
12659	54890f68699049c2a04d415abc25e717	transaction	0	13.23
12670	b2f1cd155b864803ad8334cdf13c4bd2	transaction	0	19.51
12671	fe97aa22dd3e48c8b143116a8403dd52	transaction	0	18.97

6. Extract the dictionary value from each row of the "value" column in the *offer_events_df* dataframe with the "*offer id"* or "*offer_id"* keys, and put this value in a new column named "*offer_id*".

	person	event	time	offer_id
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	0	9b98b8c7a33c4b65b9aebfe6a799e6d9
2	e2127556f4f64592b11af22de27a7932	offer received	0	2906b810c7d4411798c6938adc9daaa5
5	389bc3fa690240e798340f5a15918d5c	offer received	0	f19421c1d4aa40978ebb69ca19b0e20d
7	2eeac8d8feae4a8cad5a6af0499a211d	offer received	0	3f207df678b143eea3cee63160fa8bed
8	aa4862eba776480b8bb9c68455b8c2e1	offer received	0	0b1e1539f2cc45b7b9fa7c272da2e1d7

7. Drop the duplicated rows in the *offer_events_df* because doesn't have much sense that a person completed the same offer twice at the same time.
8. Then with a pivot table count the number of times that the same offer was sent to the same person.

	person	offer_id	offer received	offer viewed	offer completed
0	0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	1.0	NaN	1.0
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	1.0	1.0	NaN
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	1.0	1.0	NaN
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	1.0	1.0	1.0
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	1.0	1.0	1.0
...	...	...	...	...	...
55217	fffad4f4828548d1b5583907f2e9906b	f19421c1d4aa40978ebb69ca19b0e20d	2.0	2.0	2.0
55218	ffff82501cea40309d5fdd7edcca4a07	0b1e1539f2cc45b7b9fa7c272da2e1d7	1.0	1.0	1.0
55219	ffff82501cea40309d5fdd7edcca4a07	2906b810c7d4411798c6938adc9daaa5	3.0	3.0	3.0
55220	ffff82501cea40309d5fdd7edcca4a07	9b98b8c7a33c4b65b9aebfe6a799e6d9	1.0	1.0	1.0
55221	ffff82501cea40309d5fdd7edcca4a07	fafdcd668e3743c1bb461111dcafc2a4	1.0	1.0	1.0

With this information I realized that to confirm if the offers were really completed it was necessary to make a distinction for each offer that was sent more than once to the same person, in order to avoid aggregating the time value registered for each event.

9. A solution for the problem mentioned above is to generate a variable that indicates me if it is the first time, second time, and so on, that the offer event occurred for that person. First I split the dataframe in two new dataframes, one that contains offers that were sent to the corresponding customer just one time, and a second one that contains the relation of offers that were sent more than once to the customers.
10. The first dataframe was obtained by getting each person and offer_id pairs from the pivot table in the step 8 for the offers that were received only once by the customer, and with this I proceed to filter the offer_events_df (table in step 6) to put an integer 1 in a new column named “n_times”, which indicates that these pairs of person and offer_id only appears one time.

	person	event	time	offer_id	n_times
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	1.0
2	e2127556f4f64592b11af22de27a7932	offer received	0	2906b810c7d4411798c6938adc9daaa5	1.0
7	2eeac8d8feae4a8cad5a6af0499a211d	offer received	0	3f207df678b143eea3cee63160fa8bed	1.0
8	aa4862eba776480b8bb9c68455b8c2e1	offer received	0	0b1e1539f2cc45b7b9fa7c272da2e1d7	1.0
9	31dda685af34476cad5bc968bdb01c53	offer received	0	0b1e1539f2cc45b7b9fa7c272da2e1d7	1.0

11. The second dataframe was the result of the inverse operation of the previous step, this means, by getting the person and offer_id pairs of offers that were received by the customer more than once. With this relation I filtered the offer_events_df (from step 6) and got all the events related to each person and offer_id pair, then I passed this dataframe to a custom function to number the order of appearing of each event in a new column named “n_times” by considering the “time” feature, and this new feature in the returned dataframe of the function give me the information if it was the first, second, third, fourth or fifth time the offer was received by the same person.
12. As the previous step has a high computational consumption, the result was saved in a csv file to avoid having to run it every time.
13. With this final offer_events_df dataframe, I made a pivot table considering as index the columns ‘person’, ‘offer_id’ and ‘n_times’, and the columns of the pivot table were the

different events (‘offer received’, ‘offer viewed’ and ‘offer completed’) and the values were the times in which the events happen.

	person	offer_id	n_times	offer received	offer viewed	offer completed
0	0009655768c64bdeb2e877511632db8f	2906b810c7d4411798c6938adc9daaa5	1.0	576.0	NaN	576.0
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	1.0	336.0	372.0	NaN
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	1.0	168.0	192.0	NaN
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	1.0	408.0	456.0	414.0
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	1.0	504.0	540.0	528.0
...	...	...	...	...	...	...
66565	ffff82501cea40309d5fdd7edcca4a07	2906b810c7d4411798c6938adc9daaa5	1.0	336.0	354.0	384.0
66566	ffff82501cea40309d5fdd7edcca4a07	2906b810c7d4411798c6938adc9daaa5	2.0	408.0	414.0	414.0
66567	ffff82501cea40309d5fdd7edcca4a07	2906b810c7d4411798c6938adc9daaa5	3.0	576.0	582.0	576.0
66568	ffff82501cea40309d5fdd7edcca4a07	9b98b8c7a33c4b65b9aebfe6a799e6d9	1.0	504.0	534.0	504.0
66569	ffff82501cea40309d5fdd7edcca4a07	fafdcd668e3743c1bb461111dcafc2a4	1.0	0.0	6.0	60.0

14. Merge the portfolio information with the offer_times_df dataframe to get information about the relation of the offer type and the original offer completed events feature.
15. Classify the records that were completed correctly and which were not. The conditions to classify the records are the next ones:
    - The offer that was never seen by the customer (offer viewed with a null value) will be classified as not completed.
    - The offer where the customer failed to reach the transaction difficulty on the duration time (offer completed with a null value on discount and BOGO offers).
    - (offer viewed - offer received) > 0 and (offer completed - offer viewed) > 0: With this two conditions we can guarantee that the offer was not viewed after "completed" it, so we can classify it as completed correctly, and if at least one of these two conditions is false, it will be classified as not completed correctly.
    - For the informational offers, if the offer was received, viewed and there are transactions during the duration time of the offer, it will be classified as completed correctly, otherwise it will be classified as not completed correctly.
16. Drop the duplicates.
17. Drop the columns that are not going to be used anymore.
18. Rename the dataframe to “final.
19. Adding a column to the “final” dataframe with the total amount of money spent during the offer by each customer (this column will be removed in the implementation process because we cannot use it as a predictive feature due to the fact that before sending the offer to the client we don’t have this information).
20. Merge the customers profile dataframe with the final dataframe, then I have in the “*final*” dataframe all the necessary information to build the model (66570 rows and 25 columns).

Implementation

First of all I have a dataframe with all the preprocessing steps already done, and the columns of this final dataframe are the following

```
1 final.columns
```

```
Index(['n_times', 'reward', 'difficulty', 'duration', 'email', 'mobile',
       'social', 'web', 'bogo', 'discount', 'informational', 'true_completed',
       'age', 'income', 'member_day', 'member_month', 'member_year',
       'gender_F', 'gender_M', 'gender_O'],
      dtype='object')
```

The implementation was done considering four models, one of which was used as a benchmark model, so the first step in the implementation process was to define or features and our target variables from the final dataset.

```
1 X = final.drop(['true_completed'], axis = 1)
2 y = final.true_completed
```

Splitting the dataset into train and test datasets using the 'train_test_split' method that is provided by the scikit-learn library.

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, shuffle = True, random_state = 0)
```

Now I'm going to generate a benchmark model, this model will help me as a baseline to compare against another models and evaluate performance.

```
1 lr = LogisticRegression(solver = 'liblinear', random_state = 0)
```

```
1 lr.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

Then I need to define a function to compute the model performance.

```
def model_performance(model, X_test, y_test):
    '''Function to evaluate a model performance taking the estimator, the test features and the test labels'''
    y_predict = model.predict(X_test)
    return accuracy_score(y_predict, y_test), f1_score(y_test, y_predict)
```

Using the previous function to compute the baseline model performance I got the following result.

```
1 metrics_scores = model_performance(lr, X_test, y_test)
2 print('The ' + lr.__class__.__name__ + ' model got an accuracy score of: ' + str(metrics_scores[0]))
3 print('The ' + lr.__class__.__name__ + ' model got a F1_score score of: ' + str(metrics_scores[1]))
```

```
The LogisticRegression model got an accuracy score of: 0.6590556306644635
The LogisticRegression model got a F1_score score of: 0.23434161700213652
```

Considering the previous metrics results of the baseline model we can tell that it showed a very poorly performance, now we can evaluate if another model has better performance. So I'm going to define another classifiers with default parameters:

```
# defining the classifiers
dtc = DecisionTreeClassifier(random_state = 0)
rfc = RandomForestClassifier( random_state = 0)
xgbc = XGBClassifier(use_label_encoder = False, random_state = 0)
```

The three classifiers models correspond to a Decision Tree, Random forest and XGBoost, the next step is to fit the classifiers with the train data.

```
# Fitting the classifiers with the train data
dtc.fit(X_train, y_train)
rfc.fit(X_train, y_train)
xgbc.fit(X_train, y_train)
```

Now I need to measure the prediction performance of the previously fitted classifiers.

```
# Measuring the prediction performance of the classifiers with the default parameters
models = [lr, dtc, rfc, xgbc]
for model in models:
    metrics_scores = model_performance(model, X_test, y_test)
    print('The ' + model.__class__.__name__ + ' model got an accuracy score of: ' + str(metrics_scores[0]))
    print('The ' + model.__class__.__name__ + ' model got an F1_score score of: ' + str(metrics_scores[1]))

```

```
The LogisticRegression model got an accuracy score of: 0.6590556306644635
The LogisticRegression model got an F1_score score of: 0.23434161700213652
The DecisionTreeClassifier model got an accuracy score of: 0.6667167392719443
The DecisionTreeClassifier model got an F1_score score of: 0.5251141552511416
The RandomForestClassifier model got an accuracy score of: 0.7310099644484502
The RandomForestClassifier model got an F1_score score of: 0.5769412505906442
The XGBClassifier model got an accuracy score of: 0.7469330529267437
The XGBClassifier model got an F1_score score of: 0.6006637168141593
```

Comparing the prediction performance of the three models with the baseline model, it is important to note that the Random Forest and XGBoost classifiers got a very similar overall predicting performance, the accuracy was about 73%-74% and the F1 score was around 58%-60%, which is a very good enhancement in the score comparing it with the 23% F1 score that we got on the baseline model, and considering that all the models were defined with the default parameters.

Refinement

As an attempt to improve the models performance mainly for the F1 metric, I implemented a GridSearch [9] to find the parameters that generate the best performance for the F1 Score metric. So as a first step I need to create the dictionaries for each of the models where the keys are the names of the parameters and the values are lists with the values we want to iterate on.

```python
models_dict = {
    'Logistic Regression': lr,
    'Decision Tree Classifier': dtc,
    'Random Forest Classifier': rfc,
    'XGB Classifier': xgbc
}

parameters = {
    'Logistic Regression': {'penalty':['l2','l1'],'C':[0.01, 0.1, 1, 10, 100]
                            },
    'Decision Tree Classifier': {'criterion':['gini','entropy'],
                                 'max_depth': range(15,50,5)
                                },
    'Random Forest Classifier': {'n_estimators': range(50,500,50),
                                 'criterion':['gini','entropy'],
                                 'max_depth':range(15,50,5)
                                },
    'XGB Classifier': {'max_depth': range (2, 10, 1),
                       'n_estimators': range(60, 220, 40),
                       'learning_rate': [0.05, 0.01, 0.1]
                      }
}
```

Then to execute a GridSearch on each of the models I created a for-loop that iterates between the models in models_dict and instantiate the GridSearchCV class with the corresponding model and parameters, and finally print the best parameters combination for each model.

```python
best_params_dict = {}
for key in models_dict.keys():
    model = models_dict[key]
    params = parameters[key]
    gs = GridSearchCV(estimator = model, param_grid = params, scoring = 'f1', n_jobs = -1, refit = False,
                      error_score='raise')
    gs.fit(X_train, y_train)
    best_params_dict[key] = gs.best_params_
    print(f'The best score for the {key} model was {gs.best_score_}')
    print(f'The best parameters for the {key} model was {gs.best_params_}')
```

The following figure contain the result obtained after executing the previously described process.

```
The best score for the Logistic Regression model was 0.538975861436269
The best parameters for the Logistic Regression model were {'C': 10, 'penalty': 'l1'}
The best score for the Decision Tree Classifier model was 0.5655354717290131
The best parameters for the Decision Tree Classifier model were {'criterion': 'entropy', 'max_depth': 15}
The best score for the Random Forest Classifier model was 0.5974984427329264
The best parameters for the Random Forest Classifier model were {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 100}
The best score for the XGB Classifier model was 0.6119752332797902
The best parameters for the XGB Classifier model were {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 180}
```

Considering that the previous GridSearch process seeks to optimize the F1-Score, all 4 models were enhanced by tuning the parameters, and the best performance was obtained by the **XGBoost classifier** with a F1-Score of 61.2%, so applying this very computationally expensive method the model was optimized only by 2%.

IV. Results

Model Evaluation, Validation and Justification

Now I can evaluate other metrics like precision and recall, to do this first I need to define the classifiers with the parameters we got from the refinement process and then fit these classifiers with the train dataset.

```
# defining the classifiers
lr = LogisticRegression(solver = 'liblinear', C=10, penalty='l1' , random_state = 0)
dtc = DecisionTreeClassifier(criterion= 'entropy', max_depth= 15, random_state = 0)
rfc = RandomForestClassifier(criterion= 'gini', max_depth= 15, n_estimators= 100, random_state = 0)
xgbc = XGBClassifier(learning_rate= 0.1, max_depth= 7, n_estimators= 180, use_label_encoder = False,
                     random_state = 0)
```

```
# Fitting the classifiers with the train data
lr.fit(X_train, y_train)
dtc.fit(X_train, y_train)
rfc.fit(X_train, y_train)
xgbc.fit(X_train, y_train)
```

And to have some visual help to see the performance models when classifying the test dataset and the number of True Positives, True Negatives, False Positives and False Negatives on the classification it is a good idea to generate a confusion matrix for each of the models.

```
classifiers = [lr, dtc,rfc,xgbc]
fig, axs = plt.subplots(2,2, figsize =(15,10))

for cls, ax in zip(classifiers, axs.flatten()):
    plot_confusion_matrix(cls, X_test, y_test, ax=ax, cmap = 'Blues')
    ax.title.set_text(type(cls).__name__)

plt.show()
```

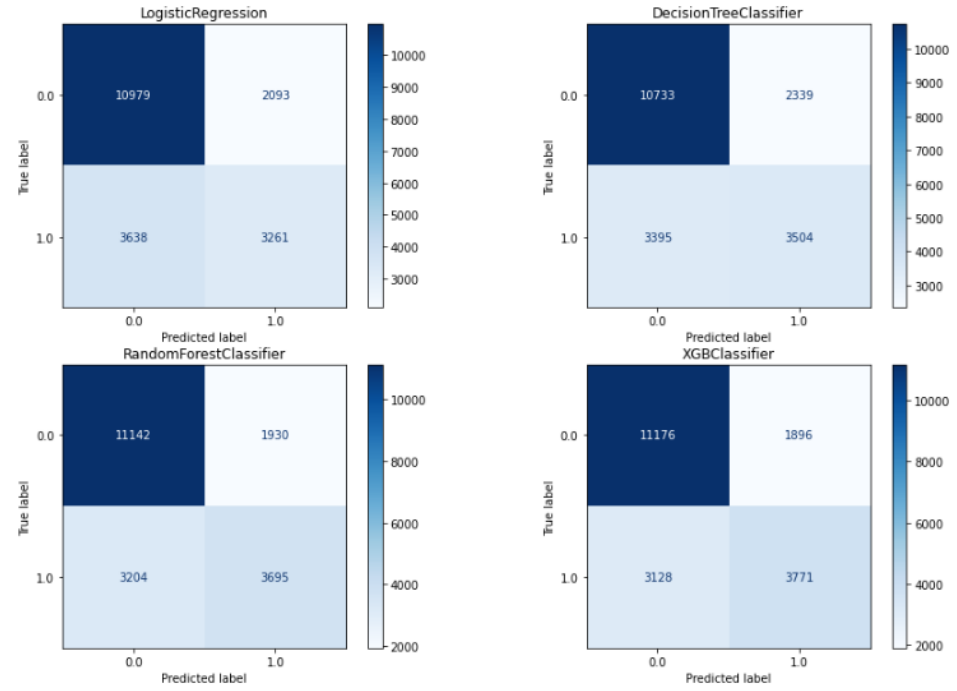

Fig 5. Models Confusion Matrix

From the above figure we can see that there are better models classifying the negative class, just as there are others that perform better classifying the positive class, and considering the goal of this project we want to achieve as few False Negatives as possible, this is because there is not much problem if we send additional offers to misclassified customers who may not respond positively to it, but if the model misclassify a customer as negative we would not send to the customer the offers to which he has responded positively, and we don't want that, so we need to choose the model with the best performance in the recall metric.

```
precision = []
recall = []
model_name = []
for model in classifiers:
    precision.append(precision_recall_fscore_support(y_test, model.predict(X_test), average='binary', pos_label=1)[0])
    recall.append(precision_recall_fscore_support(y_test, model.predict(X_test), average='binary', pos_label=1)[1])
    model_name.append(model.__class__.__name__)

pd.DataFrame({'Model': model_name,
              'Precision': precision,
              'Recall': recall})
```

	Model	Precision	Recall
0	LogisticRegression	0.609077	0.472677
1	DecisionTreeClassifier	0.599692	0.507900
2	RandomForestClassifier	0.656889	0.535585
3	XGBClassifier	0.665431	0.546601

After this process we can see from the table that the best classifier for the purpose of the project is the XGBoost Classifier outperforming the baseline model by 38.5% on the F1-Score metric.

As a final step it is useful to identify the more important features in the classification process that the model does, to do this I looked at the feature_importances_ attribute that the model has available and plot it.

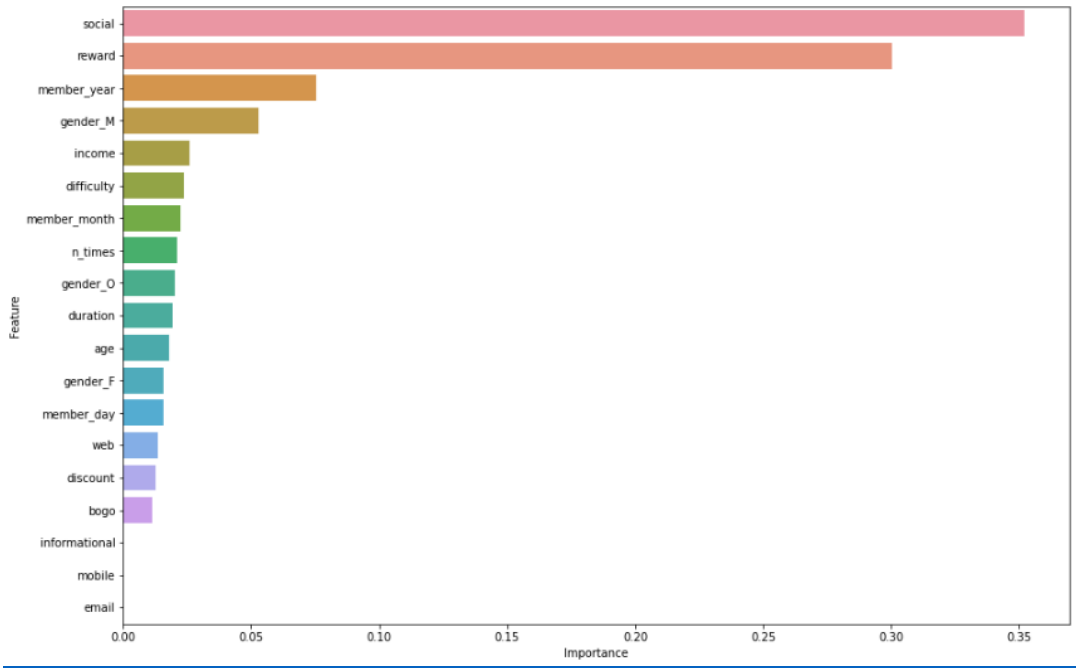

Fig 6. Feature Importances

From the above figure we can tell that the top five features with a predictive power on the model are:

- social: One of the channels through which the offer is sent.
- reward: Reward given for completing an offe
- member_year: Year in which the client created the account in the application
- gender_M: Identifies if the customer is a male.
- income: Customer's income.

V. Conclusion

Reflection

This was a very interesting project of targeted marketing, a technique that allows the companies to invest their budgets in a more reliable way to know if a user is going to buy their products or services, in regards of this project may I say that most of the time invested on it was on the data preprocessing, and this was the key part of the project due to the datasets were not structured in a way where we could continue with the modeling process very easily.

In this data preprocessing I created different dummy variables to split the data that was contained only in one column of the datasets (like the channels in the portfolio and the gender n the profile), and this preprocessing also made me able to identify some interesting trends in different features of the data and also identify outliers to remove them as appropriate. After this was necessary to merge the data from the different preprocessed datasets mainly based on the offer id and customer id, and here comes a very tricky part of the process where I found that some offers with some characteristics were sent to the same customer more than once, so to aggregate the time when the event of each offer occurred I had to create a custom function to separate each offer that was sent to the customer, and this was an important part because this made me able to re-classify the offers as completed or not.

When the preprocessing was done I started with the modeling process, first defining a logistic regression as a baseline model and then defining a Decision Three, a Random Forest and an XGBoost classifiers with the aim of outperforming the baseline model, and to enhance the performance of these models I used a GridSearch method to get the best combination of parameters that give me the best performance for each of the models, and I managed to determine that the best model performance was achieved by the XGBoost classifier basing the validation in F1-Score, accuracy, precision and recall metrics.

This final model achieved an F1-Score of 61.2% and a recall of 54.6%, and this is not a very good performance if we think that the goal of the project is to predict whether or not someone is going to respond to an offer, and considering that the idea is to get as few as possible of False Negative predictions, the result is almost a randomized one, very near to the 50%, so It needs more work and that’s why I don’t recommend using this model in a production environment.

Improvement

A way this model could be better and have a better performance relies on the feature engineering process, definitely this was a field that I should have explored more, like instead of just having the year, month and day where the customer became a member, It could be a good idea to add the amount of time the customer has been a member for example. And also another approach that might have worked would be to separate the modeling process for each type of offer (BOGO, discount and informational).

I would have liked to implement a deep learning algorithms in the modeling process to get an overall better performance, but to achieve this we need more data and in this project the data was very limited, because this lack of data is why I could not implement a model to predict some missing values in the dataset and I had to get rid of them.

There is an opportunity with the “total_amount” column that I computed but not used, and is to create another model but with a different goal, instead of trying to classify if a customer is going to respond to an offer, this column would allow us to train a model that predict the amount of money that a customer is going to spend in an offer.

VI. References

[1] Google, “Classification: Accuracy.” [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy.

[2] T. Wood, “Precision and Recall.” [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall.

[3] Google, “Classification: Precision and Recall.” [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

[4] “F-score.” [Online]. Available: https://en.wikipedia.org/wiki/F-score.

[5] “Decision Trees - Classification.” [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#classification.

[6] “sklearn.tree.DecisionTreeClassifier.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[7] “sklearn.ensemble.RandomForestClassifier.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[8] “Python API Reference.” [Online]. Available: https://xgboost.readthedocs.io/en/latest/python/python_api.html.

[9] “sklearn.model_selection.GridSearchCV.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.