Capstone Project

Machine Learning Engineer Nanodegree

Hongxia(Sophie) Hou
January 6, 2022

I Definition

Project Overview

Nowadays, more and more shops, ecommerce are aiming to utilize the data in order to understand how the customers think, feel and decide, so the businesses can determine how best to market their products, services and which sales promotion strategies really work.

Starbucks, one of biggest chain of coffee shops, not only go through mounds of coffee beans to satiate its raving fans, but they also have mounds of data that they leverage in many ways to improve the customer experience and their business.

This project data is provided by Udacity in collaboration with Starbucks as the capstone project of Machine Learning Engineer Nanodegree. The data is simulated data that mimics customer behavior on the Starbucks rewards mobile app during a month experiment. It contains three files:

- Portfolio.json: Containing offer ids and meta data about each offer (duration, type, etc.)
- Profile.json: Demographic data for each user.
- Transcript.json: Records for transactions, offers received, offers viewed, and offers completed.

Problem Statement

People often say one thing but do another, and different people respond to the same thing in different ways. Starbucks rewards mobile app sends out an offer to users once every few days. An offer can be merely an advertisement for a drink or an actual offer such as a discount or BOGO (buy one get one free). Some users might not receive any offer during certain weeks. Not all users receive the same offer, and someone using the app might make a purchase through the app without having received an offer or seen an offer. From a business perspective if a customer is going to make a 10-dollar purchase without an offer anyway, you would not want to send a buy 10 dollar get 2 dollars off offer.

So, the goal of this project is to segment the customers and personalize the offer strategy by answering following questions:

1. *Which demographic groups are there and what the offers really excite them? In another word, what is the best offer, not just for the population as a whole but at an individual level?*

2. *Which demographic groups will complete an offer without opening or seeing the offer?*

3. *Can we build a machine learning model to predict what action a customer might take after an offer is received?*

To tackle the task, we will take the first step on data exploration and data understanding. Then data wrangling and build a function to determine if an offer completion was made by offer influence or not, which means if it's made after viewing the offer or not. Then merge and reshape the table to one time offer receive, corresponding actions (view or complete) in one row as final table for statistical analysis.

Question 1 and 2 will be answered by comparing view rate, completion rate (influenced and uninfluenced) in different groups. Those metrics will be presented in the next section. And questions 3 will be answered by using three machine learning classification algorithms: Logistic Regression, Random Forest, and K-Nearest Neighbor.

Metrics

To determine which offer is the winner, we will use:

- View_rate: which is the total views divided by total number of receive
- Influence_complete_rate: which is total number of completions with view record in advance divided by total number of receive
- Uninfluence_complete_rate: which is total number of completions without view record before divided by total number of receive

To determine which predictive model performs better, an evaluation of the machine learning model will be done with a validation data set. The output of the predictions can be checked through a confusion matrix, and metrics. Here is a quick description of those metrics:

- Confusion matrix to visualize TP (True Positive), TN (True Negative), FP (False Positive), FN (False Negative).
- Accuracy: is the correct values divided by total values
- Precision: is the True Positives divided by the sum of True Positives and False Negatives. So precision is the values of the true positives divided by the actual positive values.
- Recall: is the True Positives divided by the sum of True Positives and False Positives. So, recall is the value of the true positives divided by the positive guesses.
- F1-score: is a blended score of precision and recall which balances both values.
- Macro Avg: is the unweighted mean value of precision and recall.
- Weighted Avg: is the weighted mean value of precision and recall by the support values for each class.

- Support: is the number of observations in class to predict.

II. Analysis

Data Exploration & Exploratory Visualization

As mentioned in project overview, we have three files, let's take look one by one:

1. Portfolio:

   Containing offer ids and meta data about each offer. There are 10 different offers with 10 unique offer_ids, across 3 types of offer: Bogo, Discount and Informational.

   Each offer shows how much a user can get as reward (reward) if the total amount of the user spend reaches the minimum amount (difficulty) during the offer validity period(duration). There is also other info about which channels are in use for an offer promotion. (See the first 5 rows below)

```
portfolio
```

	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

   There is no missing value, but column channels need to be encoded for machine learning models later. Offer id is the key to join with other datasets, but it's not easy to interpret. We'll create another column by combining `offer_type` + `duration` + `difficulty` to improve the readability, for example: `bogo7+10` means Buy One Get One free with 7days validity and $10 minimum amount.

2. Portfolio:

   Containing demographic features for each user. The demographic features are gender, age, income and when a user registered as a member. (See the first 3 rows of the table below)

```
profile.head(3)
```

	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN

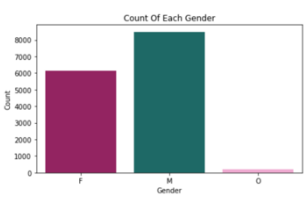
There are 17K users: Male 8484, Female 6129 and Oblivious 212 (Fig. 1). But 2175 users missing gender, income and age were encoded with 118. Since we need those features for analysis and group users, we also use it for predictive models later, therefore, we decided to drop those missing values. After drop, there are a total of 14825 users left in the data.

Users' age between 18 to 101, with mean 54 and median 55. 50% users are between 42 to 66, and the distribution is slightly left skewed. (Fig. 2).

Users' income between 30K to 120K, with mean 65405 and median 64000. 50% of user's income between 49K to 80K and the distribution is slightly right skewed. (Fig.3)

Users' membership started from 2013-07-29 to 2018-07-26. Majority joined in Y2016, Y2017 and Y2018.

We also observed that there are outliers in the female age group, male income group and membership. We will check outliers again after we get final cleaned data during the building model part, because some machine learning algorithms are sensitive to outliers in data.



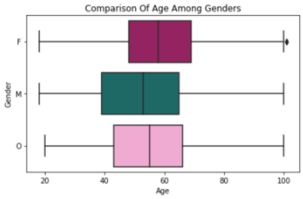
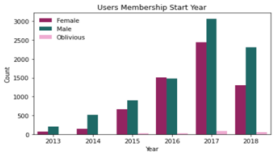


Fig. 1



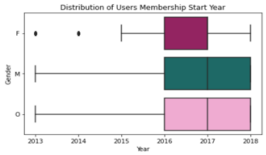
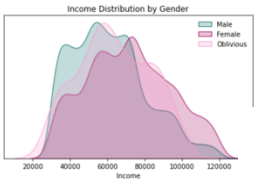


Fig. 4



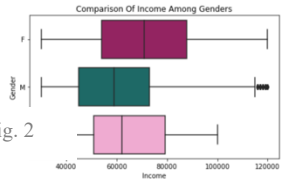


Fig. 2

Fig. 3

Since one of the tasks is to find which group likes which offer and which group is easily influenced by offer, which group is not, we will create buckets for age group, income level and membership year in order to group users.

3. Transcript:

Containing all the records how a user reacts to an offer during the experiment. (See the first 3 rows of the table below)

```
transcript.head(3)
```

	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0

There are 306534 rows and 4 columns.

Column `person` contains 17K unique users with user id which can be used as the key to join the table `profile`.

Column `event` contains 4 different events: offer_recieved (76277), offer_viewed (57725), offer_completed (33579) and transaction (138953) (Fig. 5).

Column `value` records the offer_id, reward (if there is any) for each offer event, and amount for each transaction. We need to flatten this column and extract the offer_id in order to join the portfolio table on offer_id to get details of each offer.

Column `time` records time in hours when each event occurred, it's from 0 hour when the experiment started to the experiment end (at 714 hours), just about 1 month.

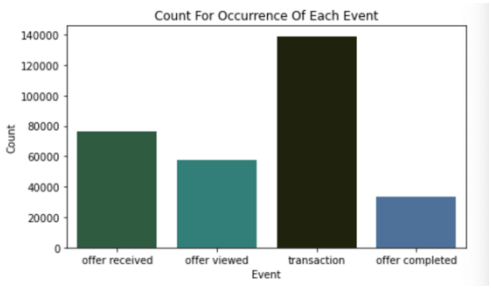


Fig. 5

There is no missing value in this table, but as mentioned in the problem statement section, the challenges are: 1) how to determine which users completed an offer influenced by the offer promotion and which users did not. 2)How to reshape the table and merge with other tables to get each time offer's status for each user in one row. Remember we want to calculate each time an offer's view rate, completed_influenced rate and complete_uninfluenced rate to compare offers performance.

Here is one data example of a user's behavior. (See below).

```
# randomly check another customer's transcript record
transcript[transcript.person == "aa4862eba776480b8bb9c68455b8c2e1"]
```

	person	event	value	time
8	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
20284	aa4862eba776480b8bb9c68455b8c2e1	offer viewed	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	18
53183	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	168
65845	aa4862eba776480b8bb9c68455b8c2e1	offer viewed	{'offer id': '4d5c57ea9a6940dd891ad53e9dbe8da0'}	168
83400	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 12.33}	210
83401	aa4862eba776480b8bb9c68455b8c2e1	offer completed	{'offer_id': '4d5c57ea9a6940dd891ad53e9dbe8da0...	210
110836	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	336
123541	aa4862eba776480b8bb9c68455b8c2e1	offer viewed	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}	336
150604	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}	408
174355	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 15.95}	426
174356	aa4862eba776480b8bb9c68455b8c2e1	offer completed	{'offer_id': 'ae264e3637204a6fb9bb56bc8210ddfd...	426
177264	aa4862eba776480b8bb9c68455b8c2e1	offer viewed	{'offer id': 'ae264e3637204a6fb9bb56bc8210ddfd'}	432
187138	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 23.3}	456
200087	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 10.08}	498
221906	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 11.33}	516
245129	aa4862eba776480b8bb9c68455b8c2e1	offer received	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	576
277105	aa4862eba776480b8bb9c68455b8c2e1	offer viewed	{'offer id': 'f19421c1d4aa40978ebb69ca19b0e20d'}	612
283960	aa4862eba776480b8bb9c68455b8c2e1	transaction	{'amount': 12.56}	630
283961	aa4862eba776480b8bb9c68455b8c2e1	offer completed	{'offer_id': 'f19421c1d4aa40978ebb69ca19b0e20d...	630

Those two offer_completed made after view. We regarded those as completed influenced by the offer promotion.

This offer_completed made before view. We regard this as completed uninfluenced by the offer promotion.

The method used to determine influenced vs uninfluenced and the way to reshape the table will be presented in the Data Processing section.

After we cleaned and reshaped the table, during the data preparing for the ML model, we confirmed there is no missing value. We observed the distribution of a single variable and relation between two variables by using pairplot and heatmap. It seems our data is not linear, so a linear classification may not fit. We also found there are collinearity among the features. Following heatmap (Fig. 6) shows high correlation between `difficulty` and `duration` which could impact accuracy of some linear machine learning algorithms', such as Logistic Regression.

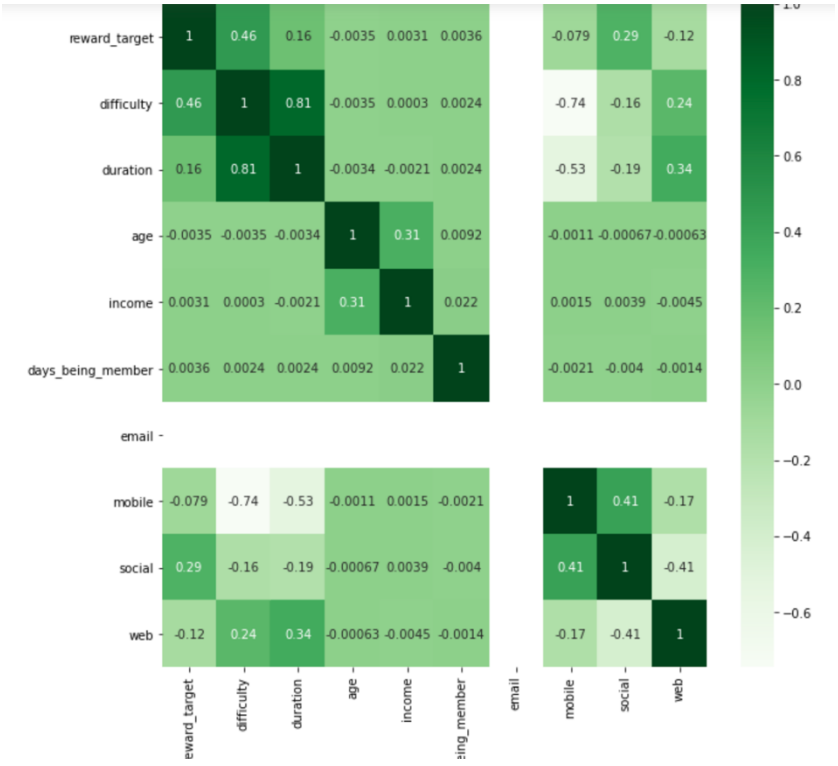


Fig. 6

We also checked outliers and observed there are outliers in `days_being_member` (see Fig. 7). We need to remove it before fitting Logistic Regression and KNN.

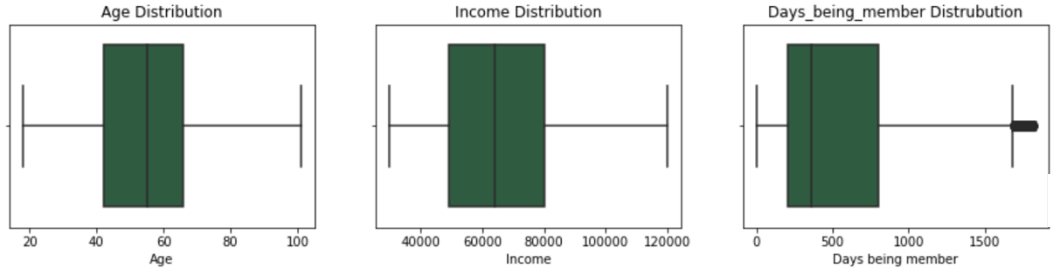


Fig. 7

Furthermore, we have an imbalance between classes, class “1” and “2” are much more than class “-1” and “0” in the dataset (See Fig. 8). We need to put it in consideration when use Logistic Regression and KNN.

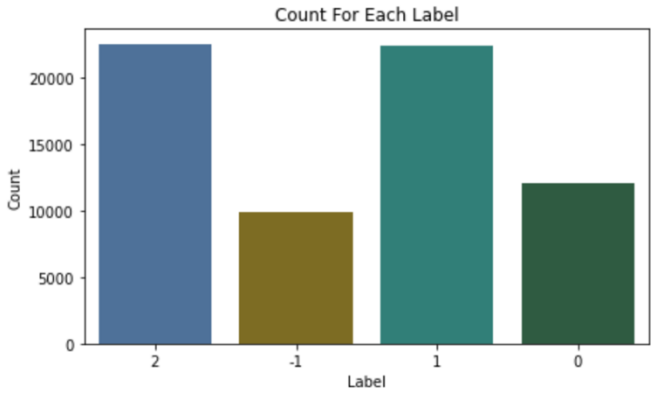


Fig. 8

Algorithms and Techniques

In the data cleaning part:

To determine if the offer_completed under influence or not, and to reshape the table, we create a function called sign_order_id. The function will take in a DataFrame and add two columns to the DataFrame. One column is called `order_id` and another column is called `influence_index`. The purpose of `order_id` is to remove the invalid offer_events and reshape the table by split table to different events and then join them together on order_id. The purpose of `influence_index` is to distinguish if an offer_view is valid, if an offer_completed made under influence or not, then we can use the influence_index to label how the user reacted to each time offer_received.

The whole function will be presented in the Data Processing section. To explain how it works, basically the function will check every offer event (offer_recieve, offer_viewed, offer_completed). If the event corresponding time is in the event corresponding validity period, then the event is valid and signed with an order_id by combining offer_id + person id + time for offer_received, otherwise will sign an order_id with combining offer_id + person id and ending with a fixed number 800. This way will make sure one time offer_received will have the same order_id as one time offer_viewed (if there is any valid) and one time offer_completed(if there is any valid). Note here, if there are two times offer_completed during an offer valid period, the two

times offer_completed will have the same order_id, which leads to a duplicated offer_received when we reshape the table. Meantime, the function will sign offer_received an influence_index as “0” and valid offer_viewed as “1”, invalid offer_viewed as “0”. Regarding offer_completed, the function will look for if a valid offer_viewed occurred before the completion or not. If yes, sign influence_index as “1”, if not, then the influence index is “0”. By this way, we will separate the offer_completeds into two groups, one made under influence, and another made without influence.

In the statistical analysis part:

We used `groupby` with `agg` function to calculate the number of received, viewed, completed in order to calculate the matrix we plan to use to determine which offer is the winner, which groups of users are offer_insensive and which are not. The code will be presented in the Implementation section.

In the model building part:

We wanted to predict which action would be taken when a user receives an offer. In the final cleaned dataset, we have 4 classes to predict. Therefore, we need a machine learning classification algorithm for multi-class.

To start with, we chose Logistic Regression because:

It’s strengths: good accuracy on small amounts of data, easy to interpret (we get feature importance), easy to implement, efficient to train (doesn’t need high compute power), can do multi-class.

We are also mindful of its weaknesses: tend to overfit on high dimensions (use regularization), can’t do non-linear classification (or complex relationship), not good with multicollinearity, sensitive to outliers, and requires linear relationship between log odds and target variable.

Concerning its hyperparameters: we chose to use are: `solver = 'saga'` which is the algorithm to use in the optimization problem; `multi_class = ‘ovr’` (one-vs-rest schema), `class_weight = ‘balanced’` to auto adjust the weight of classes. We also use GridSearch for best hyper-parameters: inverse of regularization strength `C` and `penalty` type.

Then we use Random Forest to build the second model. We chose Random Forest because:

It’s strengths: high accuracy, doesn’t need pruning, no overfitting, low bias with quite low/moderate variance (because of bootstrapping), can do both classification and regression, can do numerical and categorical, can classify non-linear data, data doesn’t need to be normalized nor scaled, not affected by missing values, not affected by outliers, performs well with unbalanced data (the nature of data distribution does not matter), can be parallelized (can use multiple CPUs in parallel), good with high dimensionality.

We are also aware of its weaknesses: long training time, requires large memory, and is non-interpretable (because there are hundreds of trees).

Regarding its hyperparameters: instead of taking the default number of trees 100, we firstly use the elbow method to find a reasonable number of trees `n_estimators`. Then do GridSearch for Best Hyper-Parameters, the parameters we will try adjusting are:

- ⇨ `n_estimators`: number of trees in the forest. With a large number of trees comes high accuracy, but high computational complexity.
- ⇨ `max_depth`: max number of levels in each decision tree
- ⇨ `max_features`: max number of features considered for splitting a node.
- ⇨ `bootstrap`:method for sampling data points (with or without replacement)
- ⇨ `oob_score`: Whether to use out-of-bag samples to estimate the generalization accuracy (# Note, oob_score only makes sense when bootstrap=True!)
- ⇨ `min_samples_leaf`: min number of data points allowed in a leaf node

Lastly, we built out the third model by using K-Nearest Neighbor (KNN). We picked KNN because:

It's strengths: simple to understand (intuitive), simple to implement (both binary and multi-class), handles non-linear data well, non-parametric (no requirements on data distribution), respond quickly to data changes in real time implementation, can do both classification and regression.

We are also familiar with its weaknesses: long training time, doesn't work well with high dimensional data, requires scaling, doesn't work well with imbalanced data, sensitive to outliers and noise, affected by missing values.

About its hyperparameters: we start to build a model with default `n_neighbor`: 5. Then we set up a machine learning pipeline, optimize and Tune the Pipeline to search for the best K value.

Benchmark

For the EDA (exploratory data analysis) part, the benchmark is just comparing among the groups percentage of received, viewed, and completed at each type of offer level.

For the predictive model part, we use 3 machine learning algorithms to build 3 models. Each model was trained on a training dataset and evaluated on a testing dataset to avoid data leaking and to see how the models would perform on unseen data. Then we compare the models and choose a better one. Overall, 70% accuracy could be decent for a multi-classification model.

III. Methodology

Data Preprocessing

To prepare the data for statistical analysis, all the data processing steps we took are listed as follows:

1. Added a column `offer_name` in the table “portfolio” by combining `offer_type`, `duration` and `difficulty`.
2. Dropped null values from the table “profile”, which caused 13% of total users to be removed.
3. Added a column `year` in the table “profile” by extract year from the column `became_member_on`
4. Added a column `days_being_member` in the table “profile” by subtracting the column `became_member_on` with reference date 2018-7-26.
5. Added a column `age_group’ by interval range (start=10, freq=10, end=110)
6. Added a column `income_level` by interval range (start=30000, freq=10000, end=120000)
7. Remove all transactions from the table “transcript” since we don’t need it in this project based on our project goals.
8. Flatten column `value` in the table “transcript” by using following method:

```
rows = []
df = transcript["value"]
for row, person, event, time in zip(df, transcript["person"],transcript["event"], transcript["time"]):
    row["person"] = person
    row["event"] = event
    row["time"] = time
    rows.append(row)

transcript = pd.DataFrame(rows)
```

9. Created table “offers” by merging the table “transcript” and “portfolio” based on `offer_id`
10. Signed `order_id` and `influence_index` to each time offer in the table “offers” by using following functions:

```
def get_time_for_last_step(lis, start, end):

    '''
    This function is to find a number in the list within a period (start, end),
    If there is match, will return the first match value.
    If there is no match at all, will return a customs number.

    lis: a list in decending order in order to catch the closest one in time
    start: start number
    end: end number
    '''

    for x in lis:
        for y in range(start, end+1):
            if x==y:
                return x
    return 800
```

```
def sign_order_id(df):

    '''
    This function to check each offer, and sign a new order_id and meantime determine influence_index.

    If event = offer_received:
                      order_id = offer_id + person + time, influence_index=0
    If event = offer_viewed, check if event within offer validity period, if in, then sign:
                      order_id = offer_id + person + corresponding offer_recieve time, influence_index=1
                      if not:
                      order_id = offer_id + person + 800, influence_index=0
    If event = offer_completed, check if event within offer validity period, if in, then check:
                      if there is offer_viewed between received and completed,
                             if yes
                             order_id = offer_id + person + corresponding offer_recieve time, influence_index=1
                             if not:
                             order_id = offer_id + person + corresponding offer_recieve time, influence_index=0
                      if offer_completed not within offer validity period:
                             order_id = offer_id + person + 800, influence_index=0
    '''

    order_id = []
    influence_index = []

    for i in range(len(df)):

        event = df.loc[i, 'event']

        if event == 'offer received':
            given_id = df.loc[i, 'offer_id']+df.loc[i, 'person']+"-"+str(df.loc[i, 'time'])
            order_id.append(given_id)
            influence_index.append(0)

        if event == 'offer viewed':
            offer = df.loc[i, 'offer_id']
            validity = df.loc[i, 'duration']*24
            end = df.loc[i, 'time']
            start = int(end-validity)
            lis = df[(df['event'] == 'offer received') & (df['offer_id']== offer)].time.to_list()
            lis = sorted(lis, reverse=True)

            time = get_time_for_last_step(lis, start, end)

            given_id = df.loc[i, 'offer_id']+df.loc[i, 'person']+"-"+str(time)
            order_id.append(given_id)

            if time != 800:
                influence_index.append(1)
            else:
                influence_index.append(0)

        if event == 'offer completed':
            offer = df.loc[i, 'offer_id']
            validity = df.loc[i, 'duration']*24
            end = df.loc[i, 'time']
            start = int(end-validity)
            lis = df[(df['event'] == 'offer received') & (df['offer_id']== offer)].time.to_list()
            lis = sorted(lis, reverse=True)

            time = get_time_for_last_step(lis, start, end)

            given_id = df.loc[i, 'offer_id']+df.loc[i, 'person']+"-"+str(time)
            order_id.append(given_id)

            if time != 800:
                lis = df[(df['event'] == 'offer viewed') & (df['offer_id']== offer)].time.to_list()
                lis = sorted(lis, reverse=True)

                time1 = get_time_for_last_step(lis, time, end)

                if time1 != 800:
                    influence_index.append(1)
```

```
                else:
                    influence_index.append(0)

            else:
                influence_index.append(0)


    df['order_id'] = order_id
    df['influence_index'] = influence_index

    return df
```

To improve function efficiency, we split data based on each user, then use the function check each offer's influence_index, after done for each user table, then concat them together, see following step:

```
offer_list = []
person_list = offers.person.unique()

for person in person_list:
    person_df = offers[offers.person == person]
    person_df = person_df.reset_index(drop=True)
    person_df = sign_order_id(person_df)
    offer_list.append(person_df)

offers = pd.concat(offer_list)
```

11. Split the table “offers” to three tables: “offer_received”, “offer_viewed” and “offer_completed” then join those three tables based on `order_id` and for the table called “offer_reshaped”.
12. Label each time offered with following criteria, then dropped no need columns.
    - `0` for `offer_recieved` but no `offer_viewed`
    - `1` for `offer_viewed` but no `offer_complete`
    - `2` for `offer_viewed` and followed with `offer_completed`
    - `-1` for `offer_completed` without `offer_viewed` before
13. Merged the table “offer_reshaped” and “profile” based on `person_id` and got the final table “offers_final”(see below first three rows). This is the table for statistical analysis to answer questions 1 and question 2.

```
offers_final.head(3)
```

	person	reward_target	channels	difficulty	duration	offer_type	offer_name	label	gender	age	income	age_group	income_level	year	days_being_member
0	78afa995795e4d85b5d9ceeca43f5fef	5	[web, email, mobile]	5	7	bogo	bogo7+5	2	F	75	100000.0	(70, 80]	(90000, 100000]	2017	443
1	78afa995795e4d85b5d9ceeca43f5fef	5	[web, email, mobile, social]	5	5	bogo	bogo5+5	-1	F	75	100000.0	(70, 80]	(90000, 100000]	2017	443
2	78afa995795e4d85b5d9ceeca43f5fef	10	[email, mobile, social]	10	7	bogo	bogo7+10	2	F	75	100000.0	(70, 80]	(90000, 100000]	2017	443

To prepare the data for ML model, based on the table “offers_final”, following steps has been taken:

14. Encoded the column `channels`
15. Dropped no need columns
16. Removed outliers (use 1.5IQR)
17. Got dummies for categorical variables
18. Split data to X_train, X_test, y_train and y_test on 70/30
19. Scale the data for Logistics Regression and KNN model.

Implementation

1. **In statistical analysis part:**

To compare each other's performance, based on the final cleaned table “offers_final”, we did groupby `offer_name`, calculated the number of received, number of viewed, and number of completed with influence, number of completed without influence. Then we calculated `view_rate`, `influenced_complete_rate` and `uninfluenced_complete_rate`. By comparing those rates, we determine which offer was the winner.

```
# let's check each offer's performance
offer_result = offers_reshaped.groupby(['offer_type','offer_name', 'duration', 'difficulty', 'reward_target'])\
                              .label.agg(
                                         [('received', 'count'),
                                          ('viewed', lambda x: (x=='1').sum()+(x=='2').sum()),
                                          ('completed_influenced',lambda x: (x=='2').sum()),
                                          ('completed_no_influenced', lambda x: (x=='-1').sum())
                                         ]).reset_index()
```

```
# add view_rate, influence_complete_rate and uninfluence_complete_rate
offer_result['view_rate'] = np.round(offer_result['viewed']/offer_result['received']*100, 2)
offer_result['influenced_complete_rate'] = np.round(offer_result['completed_influenced']/offer_result['received']*10
offer_result['unfluenced_complete_rate'] = np.round(offer_result['completed_no_influenced']/offer_result['received']
```

We also did compare complete rates in different difficulty levels, different duration, and different reward amounts to see how those offer requirements impact an officer's performance.

Since we want to know which group of the users prefer which offer, and which group of users are more likely buy without view an offer (we call them offer_insensitive), which group of the users more likely buy when they see an offer promotion(we call them offer_sensitive). We did group by `gender`, `year`, `age_group` and `offer_name` and calculated the metrics, then split the user group to offer_sensitive group and offer_insensitive group.

```
# group each demografical feature and calculate the number of received, viewed, completion
demo_group_result = offers_final.groupby(['gender','year','age_group','income_level','offer_name'])\
                              .label.agg(
                                         [('received', 'count'),
                                          ('viewed', lambda x: (x=='1').sum()+(x=='2').sum()),
                                          ('completed_influenced',lambda x: (x=='2').sum()),
                                          ('completed_no_influenced', lambda x: (x=='-1').sum())
                                         ]).reset_index()
```

```
 add view_rate, influence_complete_rate and uninfluence_complete_rate
emo_group_result['view_rate'] = np.round(demo_group_result['viewed']/demo_group_result['received']*100, 2)
emo_group_result['influenced_complete_rate'] = np.round(demo_group_result['completed_influenced']/demo_group_result['
emo_group_result['unfluenced_complete_rate'] = np.round(demo_group_result['completed_no_influenced']/demo_group_resu
```

Split the demographical group to `offer_sensitive` or `offer_insensitive` by following criteria:

- offer_sensitive: influenced_complete_rate >= unfluenced_complete_rate & influenced_complete_rate > 50%
- offer_insensitive: influenced_complete_rate < unfluenced_complete_rate & uninfluenced_complete_rate > 50%

```
offer_sensitive = demo_group_result[(demo_group_result['influenced_complete_rate'] >= demo_group_result['unfluenced_
                                    (demo_group_result['influenced_complete_rate'] > 50)]

offer_insensitive = demo_group_result[(demo_group_result['influenced_complete_rate'] < demo_group_result['unfluenced
                                      (demo_group_result['unfluenced_complete_rate'] > 50)]
```

The result will be presented in the “Result” section.

2. **In building predictive model part:**

As we described in the previous Algorithms and Techniques section, we use three machine learning algorithms in this project.

NO.1: Logistic Regression Model.

```
# solver: Algorithm to use in the optimization problem. Default is 'lbfgs'.
# The training algorithm uses the one-vs-rest (OvR) scheme
log_model = LogisticRegression(solver='saga',multi_class='ovr',class_weight='balanced')
```

```
log_model.fit(scaled_X3_train,y3_train)
```

```
LogisticRegression(class_weight='balanced', multi_class='ovr', solver='saga')
```

After we trained the model, we used test dataset to test the model:

```
y3_pred = log_model.predict(scaled_X3_test)
```

Then we computed the confusion_matrix and printed the classification_report:

```
confusion_matrix(y3_test,y3_pred)
```

```
array([[1245,  536,  224,  900],
       [ 709, 2171,  491,  181],
       [ 401, 1228, 3245, 1701],
       [1253,  419,  892, 4152]])
```

```
print(classification_report(y3_test,y3_pred))
```

```
              precision    recall  f1-score   support

          -1       0.35      0.43      0.38      2905
           0       0.50      0.61      0.55      3552
           1       0.67      0.49      0.57      6575
           2       0.60      0.62      0.61      6716

    accuracy                           0.55     19748
   macro avg       0.53      0.54      0.53     19748
weighted avg       0.57      0.55      0.55     19748
```

The f1-score and accuracy are very low.

Logistic Regression Model Refinement

To improve the model's performance, we used GridSearch for best hyper-parameters: inverse of regularization strength C and penalty type:

```
# Penalty Type
penalty = ['l1', 'l2']

# Use logarithmically spaced C values
C = np.logspace(0, 4, 10)
```

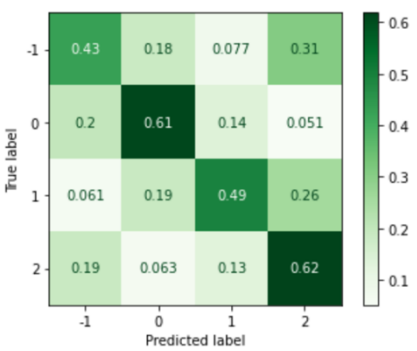
```
grid_lr = GridSearchCV(log_model,param_grid={'C':C,'penalty':penalty})
```

We got the best parameters value: C=1.0, penalty=L1

```
grid_lr.best_params_
```

```
{'C': 1.0, 'penalty': 'l1'}
```

We used this grid_lr model to predict with test dataset, then plot the scaled confusion matrix and printed our classification_report:




```
print(classification_report(y3_test,y3_pred))
```

```
              precision    recall  f1-score   support

          -1       0.35      0.43      0.38      2905
           0       0.50      0.61      0.55      3552
           1       0.67      0.49      0.57      6575
           2       0.60      0.62      0.61      6716

    accuracy                           0.55     19748
   macro avg       0.53      0.54      0.53     19748
weighted avg       0.57      0.55      0.55     19748
```

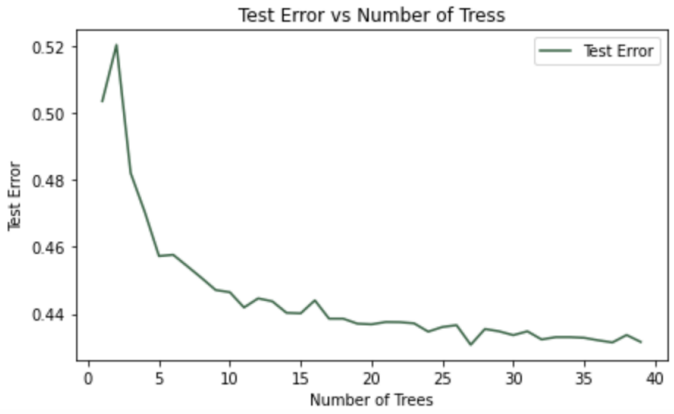
We can see by GridSearch for best hyper-parameters, we improved the model's both accuracy and f1-score.

NO.2: Random Forest Classification Model.

First we used elbow method to plot test error to find the reasonable number of trees:

```
test_error = []

for n in range(1,40):
    # Use n random trees
    model = RandomForestClassifier(n_estimators=n,max_features='auto')
    model.fit(X_train,y_train)
    test_preds = model.predict(X_test)
    test_error.append(1-accuracy_score(test_preds,y_test))
```

```
fig, ax = plt.subplots(figsize=(7,4))

plt.plot(range(1,40),test_error,label='Test Error', color='#29623F')
plt.xlabel("Number of Trees")
plt.ylabel("Test Error")
plt.title("Test Error vs Number of Tress")
plt.legend();
```



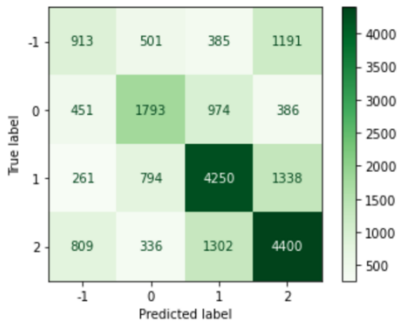
We can see from tree number 27, increasing the number of trees does not decrease the test_error much. So, we chose tree number 27 to build the RF model, then tested with test dataset, printed the classification report:

```
rfc27 = RandomForestClassifier(n_estimators=27,max_features='auto',random_state=101)
```

```
rfc27.fit(X_train,y_train)
```

```
RandomForestClassifier(n_estimators=27, random_state=101)
```

```
y_pred = rfc27.predict(X_test)
```




```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

          -1       0.38      0.31      0.34      2990
           0       0.52      0.50      0.51      3604
           1       0.61      0.64      0.63      6643
           2       0.60      0.64      0.62      6847

    accuracy                           0.57     20084
   macro avg       0.53      0.52      0.52     20084
weighted avg       0.56      0.57      0.56     20084
```

The f1-score and accuracy are very low.

Random Forest Model Refinement

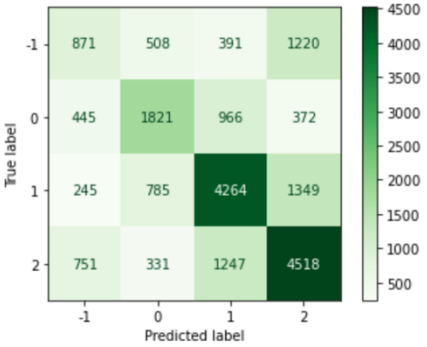
We try to adjust hyper-parameters for Random Forest by using GridSearch. We intended to adjust: bootstrap, oob_scire, max_depth, max_features, ,i_samples_lead, n_estimators, but due to computer memory limitation, we could not finish, but only did search for bootstrap, oob_score and n_estimators in list [100. 200. 300].

```
# Create the parameter grid

param_grid = {
    'bootstrap': [True, False],
    'oob_score': [True,False],
    'n_estimators': [100, 200, 300]
}
# Create a based model
rfc = RandomForestClassifier()

# Instantiate the grid search model
grid_rfc = GridSearchCV(rfc, param_grid)
```

```
grid_rfc.fit(X_train,y_train)
```

```
GridSearchCV(estimator=RandomForestClassifier(),
             param_grid={'bootstrap': [True, False],
                         'n_estimators': [100, 200, 300],
                         'oob_score': [True, False]})
```

We got best parameters, then tested grid_rfc model in test set, and printed out the classification report and plot the confusion matrix:




```
print(classification_report(y_test, y_preds))
```

```
              precision    recall  f1-score   support

          -1       0.38      0.29      0.33      2990
           0       0.53      0.51      0.52      3604
           1       0.62      0.64      0.63      6643
           2       0.61      0.66      0.63      6847

    accuracy                           0.57     20084
   macro avg       0.53      0.52      0.53     20084
weighted avg       0.56      0.57      0.57     20084
```

We can see that GridSearch did not help much to improve the model's performance. This is probably because we could not really run a full search as we intended to. Let's build the third model.

NO.3 K-Nearest Neighbor Model.

First, we built a model with default k value n_neighbor=5. Trained it with a scaled train set and tested it with scaled test set.

```
knn_model = KNeighborsClassifier()
```

```
knn_model.fit(scaled_X2_train,y2_train)
```

```
KNeighborsClassifier()
```

```
y2_preds = knn_model.predict(scaled_X2_test)
```

Then we printed the classification report:

```
print(classification_report(y2_test,y2_preds))
```

```
              precision    recall  f1-score   support

          -1       0.33      0.35      0.34      2905
           0       0.52      0.50      0.51      3552
           1       0.61      0.63      0.62      6575
           2       0.59      0.57      0.58      6716

    accuracy                           0.55     19748
   macro avg       0.51      0.51      0.51     19748
weighted avg       0.55      0.55      0.55     19748
```

The f1-score and accuracy are very low.

K-Nearest Neighbor Model Refinement

We built a machine learning pipeline and did full cross validation Grid Search for best K value. Since we have imbalanced label, we put f1 score as the target score:

```
scaler = StandardScaler()
knn = KNeighborsClassifier()
operations = [('scaler',scaler),('knn',knn)]
```

```
pipe = Pipeline(operations)
```

```
k_values = list(range(1,40))
param_grid = {'knn__n_neighbors': k_values}
```

```
# we use f1 as scoring since we have imbalance labels
full_cv_classifier = GridSearchCV(pipe,param_grid,cv=5,scoring='f1_micro')
```

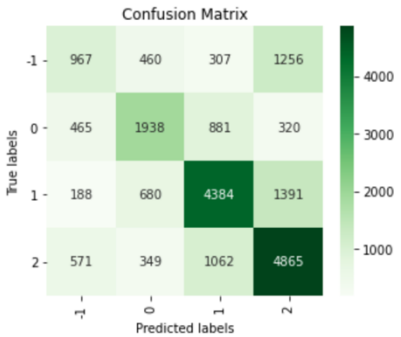
```
full_cv_classifier.fit(X2_train,y2_train)
```

We got the best parameter: K value = 39

```
full_cv_classifier.best_estimator_.get_params()
```

```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('knn', KNeighborsClassifier(n_neighbors=39))],
 'verbose': False,
 'scaler': StandardScaler(),
 'knn': KNeighborsClassifier(n_neighbors=39),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'knn__algorithm': 'auto',
 'knn__leaf_size': 30,
 'knn__metric': 'minkowski',
 'knn__metric_params': None,
 'knn__n_jobs': None,
 'knn__n_neighbors': 39,
 'knn__p': 2,
 'knn__weights': 'uniform'}
```

Then we used the pipeline again to build the last model `pipe_knn`. Trained and tested. The performance report:




```
print(classification_report(y_test,pipe_preds))
```

	precision	recall	f1-score	support
-1	0.44	0.32	0.37	2990
0	0.57	0.54	0.55	3604
1	0.66	0.66	0.66	6643
2	0.62	0.71	0.66	6847
accuracy			0.61	20084
macro avg	0.57	0.56	0.56	20084
weighted avg	0.60	0.61	0.60	20084

We can see both f1-score and accuracy got improved.

Comparing the above three models, the last model `pipe_knn` did a better job than the others not only accuracy, but also true positive and true positive prediction.

IV. Results

Analysis result

1. Offer performance in whole population level (Fig. 7)

`offer_result`

	offer_type	offer_name	duration	difficulty	reward_target	received	viewed	completed_influenced	completed_no_influenced	view_rate	influenced_complete_rate	unfluenced_complete_rate
0	bogo	bogo5+10	5	10	10	7623	6800	2759	572	89.20	36.19	7.50
1	bogo	bogo5+5	5	5	5	7605	6629	3529	767	87.17	46.40	10.09
2	bogo	bogo7+10	7	10	10	7711	6093	2606	1082	79.02	33.80	14.03
3	bogo	bogo7+5	7	5	5	7728	3564	2124	2230	46.12	27.48	28.86
4	discount	discount10+10	10	10	2	7684	6891	4653	664	89.68	60.55	8.64
5	discount	discount10+20	10	20	5	7775	2454	1339	2081	31.56	17.22	26.77
6	discount	discount7+10	7	10	2	7694	3574	2134	1883	46.45	27.74	24.47
7	discount	discount7+7	7	7	3	7700	6777	4390	766	88.01	57.01	9.95
8	informational	informational3+0	3	0	0	7618	6149	0	0	80.72	0.00	0.00
9	informational	informational4+0	4	0	0	7617	3798	0	0	49.86	0.00	0.00

We can see that all 10 offer types have quite a balanced number of release times, between 7600 to 7800 times. Among those times:

`Discount10+10` got highest view rate 89.68%, followed by `Bogo5+10` and `Discount7+7`

`Discount10+10` got the highest influenced complete rate 60.55%, followed by `discount7+7`

`Bogo7+5` go the highest uninfluenced completed rate 28.86%, followed by `discount10+20`

In general, **`Discount10+10` is the winner. (see Fig. 8)**

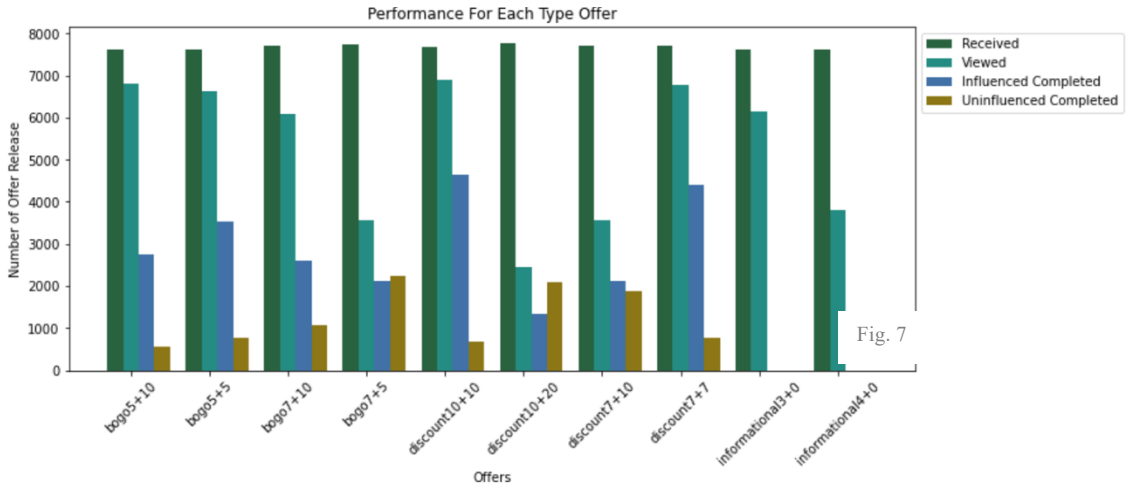


Fig. 7

2. How offer's duration, difficulty, or reward effects both influenced and uninfluenced completed rate (Fig. 8, Fig. 9, Fig. 10)

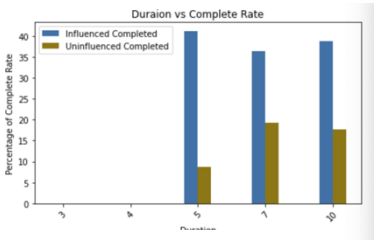


Fig. 8

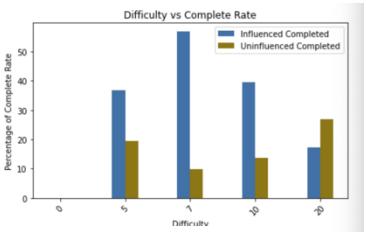


Fig. 9

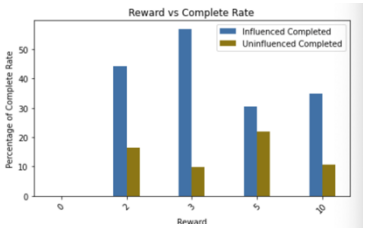


Fig. 10

For an influenced completion, duration does not play a big role, but longer duration leads to a higher uninfluenced complete rate. We need to be mindful of that when we make an offer strategy in future.
High difficulty generally lowers the influenced compete rate, but it does not affect uninfluenced rate, which further confirms our hypothesis some groups of users do not care about what the offer is about, they just buy when they want.
High reward did not generate a high influenced completed rate, one of possible reasons is high reward normally requests high minimum spend.

3. Which group in gender, age and income like which type offer

We detected 975 groups which are more likely influenced by an offer promotion (we call them offer_sensitive). Their age between 30 to 70 and income between 40k to 100k. And the most popular offers among them are `discount10+10` and `discount7+7`.
Meantime, we detected 179 groups which more likely they buy anyway no matter if there is an offer or not. (we call offer_insensitive). Their average age is slightly older than the offer_sensitive group, and their income is much higher. Among the users whose income is over 100k, `discount 10+20` was the most one they have completed. And among the users whose income is less than 100k, the most they completed was `bogo7+5`.
Majority of offer_sensitive and offer_insensitive groups registered as members in Y2015 and Y2016.

Model Evaluation and Validation

We did model evaluation and validation during the section “Model Implementation and Refinement”. Here is a summary of those three models.

In each model, we evaluated the model with a validation data set, the output of the predictions was checked through a confusion matrix and a classification report with metrics such as accuracy, f1-score, etc. We compared all three models and KNN model is the better one and it’s the final model we could get in this project. (See comparison below).

Metrics		Logistic Regression model	Random Forest Model	KNN model
F1-score	Label -1	0.38	0.33	0.37
	Label 0	0.55	0.52	0.55
	Label 1	0.57	0.63	0.66

	Label 2	0.61	0.63	0.66
Accuracy		0.55	0.57	0.61
Macro avg		0.53	0.53	0.56
Weighted avg		0.55	0.57	0.60

Justification and Reflection

The final model KNN model is only a little bit better than guessing. It's not good enough as a multi classification model, the accuracy is below 70% and the f1 store for each class is low, especially for the class "-1".

What we could try next step is to get transaction data into the model instead of removing it as we did in this project, we also can try to get more data or collect more demographical features for each user. Or we could scale the data and deal imbalance label with different ways, then tune hyper-parameters in different range.

V. Conclusion

In this project, a 30 days experiment involved 10 different offers and 306534 transcript records among 17000 users which registered as members between 2013 to 2018. There are three genders (M, F, O) and their age between 18 to 101 and income between 30k to 120k.

By analyzing the data, we can answer our questions:

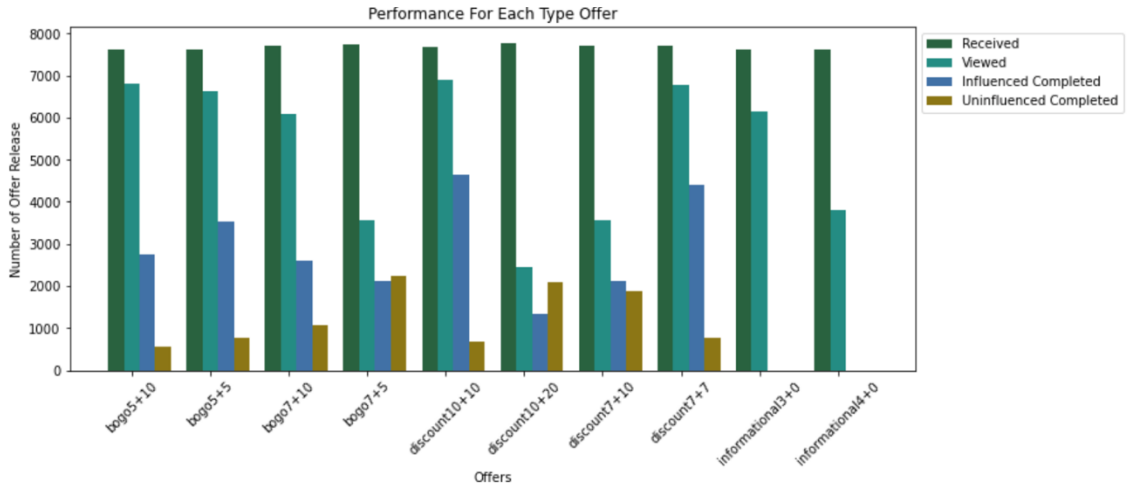
1. Which demographic groups are there and what the offers are that really excite them? In another word, what is the best offer, not just for the population as a whole but at an individual level?

- The offer which led most views and most offer_completion is `discount10+10` (discount with duration 10 and difficulty 10).

- Discount10+10 is also an offer which successfully influenced users and attracted users. Especially for the users who registered in Y2015 and Y2016, age between 30 to 70 and income between 40k to 100k.

- Following `discount 10+10`, `discount7+7` is also one of the most popular offers.

- Offer type `informational` led to no offer_completed, so suggest removing it.



2. Which demographic groups will complete an offer even if they don't open an offer?

Users who have a high income (over 100k) most likely do not care if there is an offer or not, they just buy whenever they need. They are also the groups who completed most `discount10+20`. From a business perspective, the better way is not to send them any offer or design other marketing strategies for them.

3. Can we build a machine learning model to predict which action a customer may take?

In this project, we built and tested three kinds of predict models by using three machine learning algorithms (Logistic Regression Random Forest Classification and K-Nearest Neighbor) to predict four kinds of reaction a customer might take:

* Label 0: customer will not check an offer after received
* label 1: customer will open the offer but not complete it
* label 2: customer will complete the offer after view it
* label -1: customer will complete without viewing it.

By tuning the parameters and evaluating the model, KNN model is better than others. Although it's not good enough as a multi classifier predictive model, there is some space we could do to improve the model in future, such as counting in the transaction record, increasing demographic features of each user, and trying to scale data with other types of scalers, etc.

Source:

https://bernardmarr.com/starbucks-using-big-data-analytics-and-artificial-intelligence-to-boost-performance/)

https://dwbi1.wordpress.com/2021/06/08/which-machine-learning-algorithms-should-i-use/

https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html