Starbucks Capstone Challenge Report

Udacity Machine Learning Engineer Nanodegree Program - Capstone project
Essi Lehtola, April 2020

Development stages

1. Definition

In this Udacity Machine Learning Nanodegree Capstone project the assignment was to use datasets provided by Starbucks in a useful way. The data provided mimics customer behavior on the Starbucks rewards mobile app.

There were several interesting questions to answer, but I decided my goal was to answer to an important business question: is a customers going to buy something in the influence of a certain offer based on past customer behaviour?

During this project I builded a machine learning model that can answer to that problem. When customer and offer details are given to the model it predicts if the customer will make a purchase on Starbucks or not. It was easy to follow the progression of the model by checking the percentage of correctly predicted rows. Alongside the percentage I also used a simple confusion matrix.

2. Analysis

There were three data tables provided by Starbucks: Portfolio, Profile and Transcript. The first step was to read them from json files to data frames so that they are easier to read and modify.

When examining and processing the data I tried to keep in mind it is collected from 17 000 customers in 30 (29,75) days. In that way it was easier to understand the big picture.

1) Portfolio

The Portfolio table has the data about offers sent to customers in six columns.

- 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)

In total there were 10 different offers so the table was pretty easy to internalise and use. Bogo is ‘buy one get one’ offer, discount a discount and informational offer is just a message containing for example information about products.

	id	offer_type	duration	difficulty	reward	channels
0	ae264e3637204a6fb9bb56bc8210ddfd	bogo	7	10	10	[email, mobile, social]
1	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	5	10	10	[web, email, mobile, social]
2	3f207df678b143eea3cee63160fa8bed	informational	4	0	0	[web, email, mobile]
3	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	7	5	5	[web, email, mobile]
4	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	10	20	5	[web, email]
5	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	7	7	3	[web, email, mobile, social]
6	fafdcd668e3743c1bb461111dcafc2a4	discount	10	10	2	[web, email, mobile, social]
7	5a8bc65990b245e5a138643cd4eb9837	informational	3	0	0	[email, mobile, social]
8	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	5	5	[web, email, mobile, social]
9	2906b810c7d4411798c6938adc9daaa5	discount	7	10	2	[web, email, mobile]

Portfolio table

2) Profile

The Profile table contains all customer related details in 17 000 rows and 5 columns and the columns are:

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer ('F', 'M' and 'O' for other)
- id (str) - customer id
- income (float) - customer's income

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

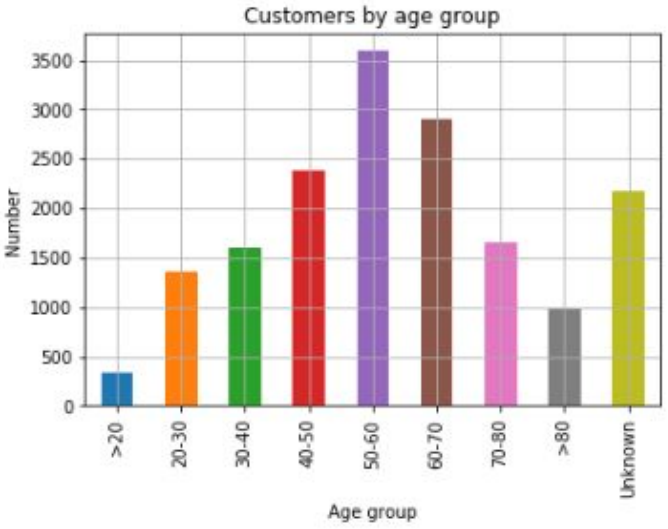
Example of the data in profile table

One strange detail quickly drew my attention - there were 2175 customers with age of 118. With more inspection I came to the conclusion it is just a placeholder for a null value as it seems that all customers with that age had their gender and income missing as well.

I also did some visualization using the profile table data.

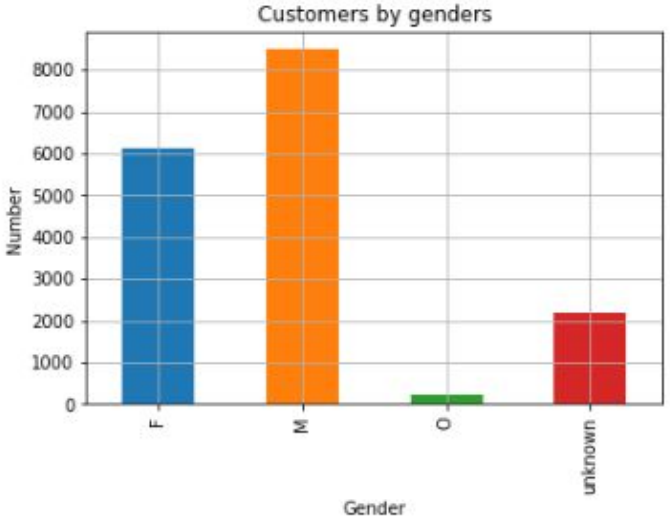
1) Age distribution

The bar chart below clearly indicates the customers are mostly over 50 years old and in my opinion the age distribution seemed peculiar. I was surprised to see the biggest age groups using the Starbucks mobile app are 50-60, 60-70 and 40-50 when customers on their 20's and 30's are not represented in these numbers that much.



2) Gender distribution

I also visualized the gender distribution. The customers with gender ‘unknown’ are mostly the same customers that have age 118. Otherwise seems like men use the mobile app more than ladies and there are also some customers that identify themselves as something else.



3) Transcript

From Transcript table you can find events connected to customers and offers. I found this table the most important but also hardest to handle.

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

event	person	time	value
offer received	0009655768c64bdeb2e877511632db8f	168	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
offer viewed	0009655768c64bdeb2e877511632db8f	192	{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
transaction	0009655768c64bdeb2e877511632db8f	228	{'amount': 22.16}
offer received	0009655768c64bdeb2e877511632db8f	336	{'offer id': '3f207df678b143eea3cee63160fa8bed'}
offer viewed	0009655768c64bdeb2e877511632db8f	372	{'offer id': '3f207df678b143eea3cee63160fa8bed'}

Example data from the Transcript table

The value column contains some relevant info (offer id, amount and/of reward) in a dictionary, so it was necessary to export that data from there somehow. The event column in turn has all interesting data regarding offer events in separate rows and that also made processing the data a bit complicated.

I also did some other observations:

- One offer can have three statuses: received, viewed and completed. Logically, an offer cannot be viewed or completed before it is received. However, it can be completed without being viewed. In this case you could say the offer did not affect the customer.

- A customer can get the same offer several times. In any case, each of them would be good to be considered separate to get better results when evaluating how an offer affected customers.

- Transactions cannot be connected to offers and one transaction can complete multiple offers. Than means you cannot tell how much a customer spent when completing an offer.

- If the offer sent is an informational offer there is no status ‘offer completed’. Instead, to know if an informational offer had any influence, I needed to check if customers have made transactions during the offer durations (the time frame the customer is feeling the influence of the offer).

In general examining the data was fun, but because of that it was also really easy to get lost in details and lose focus.

3. Processing

To make the data nice to analyze and for the future machine learning model to use, I collected a dataframe combining all interesting data from the existing source tables.

There were lots of interesting details I could have collected, but I decided to focus on the following:

- **Reactions to offers (was the offer sent viewed and/or completed)**

  First I wanted to see how each customer reacted to each offer. I collected customer actions from transcript data and checked if a customer saw an offer and made a purchase under influence of it.

  What made it harder was that it was necessary to handle informational and other kind of offers separately, since there was no ‘offer completed’ rows regarding informational offers. (Obviously there is nothing really to complete.)

  To work with the Transcript table the first step was to extract data from value field that was a dictionary.

value
{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
{'offer id': '5a8bc65990b245e5a138643cd4eb9837'}
{'amount': 22.16}
{'offer id': '3f207df678b143eea3cee63160fa8bed'}
{'offer id': '3f207df678b143eea3cee63160fa8bed'}

offer id	amount	offer_id	reward
9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	NaN	NaN
0b1e1539f2cc45b7b9fa7c272da2e1d7	NaN	NaN	NaN
2906b810c7d4411798c6938adc9daaa5	NaN	NaN	NaN
fafdcd668e3743c1bb461111dcafc2a4	NaN	NaN	NaN
4d5c57ea9a6940dd891ad53e9dbe8da0	NaN	NaN	NaN

  Offers that can be completed (bogo, discount) had three events: received, viewed and completed. If one person has offer with all those events, you can say the offer influenced the person. However, to understand if a customer was really influenced by an offer it was necessary to check if it was read before completion and completed within the offer time frame.

  If the offer was not viewed, was viewed but not completed or viewed and completed outside the time frame I assumed the customer was not influenced by the offer.

Since it was possible that one customer gets one specific offer more than once and I wanted to view reactions to all of them I also had to keep on track how many times an offer is received.

In the end I managed to get a nice table containing all offers a customer had received and if each of them was viewed and/or completed and when.

person	offer id	offer_count	offer received	offer viewed	offer completed	reward	offer_duration	received time	viewed time	completion time
828548d1b5583907f2e9906b	f19421c1d4aa40978ebb69ca19b0e20d	0	1	1	1	5.0	5	0.0	6.0	36.0
		1	1	1	1	5.0	5	408.0	510.0	516.0
1cea40309d5fdd7edcca4a07	0b1e1539f2cc45b7b9fa7c272da2e1d7	0	1	1	1	5.0	10	168.0	174.0	198.0
	2906b810c7d4411798c6938adc9daaa5	0	1	1	1	2.0	7	336.0	354.0	384.0
		1	1	1	1	2.0	7	408.0	414.0	414.0
		2	1	1	1	2.0	7	576.0	582.0	576.0
	9b98b8c7a33c4b65b9aebfe6a799e6d9	0	1	1	1	5.0	7	504.0	534.0	504.0
	fafdcd668e3743c1bb461111dcafc2a4	0	1	1	1	2.0	10	0.0	6.0	60.0

After that I checked how each customer responded to each offer. Depending on the customer actions I separated each offer to following groups:

- Effected: customer viewed and completed the offer in time - marketing was successful

- Churn: customer did not view or complete the offer

- Disinterested: customer saw the offer but did not complete - the offer was not interesting enough

- Active: customer did not saw the offer but completed it anyway or saw it and completed it outside the time frame - an active customer, marketing not really needed

offer_duration	received time	viewed time	completion time	status
7	576.0	NaN	576.0	effected
4	336.0	372.0	NaN	disinterested
3	168.0	192.0	NaN	disinterested
5	408.0	456.0	414.0	effected
10	504.0	540.0	528.0	effected

The informational offers (that are not possible to ‘complete’) were also in that dataframe and that caused all customers receiving that kind of offers to be churn or disinterested. I had to handle those separately.

To know if an informational offer had any influence I had to check if customers had made transactions during the offer duration (= the time frame the customer is feeling the influence of the offer).

If the offer was not viewed, was viewed but not completed or viewed and completed outside the time frame I reckon the customer was not influenced by the offer.

To do that I wrote a small function that gives the status of the offer row.

```
def set_informational_offer_status(data):
    '''
    Checks if the offer is viewed.
    If not, set's status depending on if there are transactions or not.
    If yes, set's status depending on if there are transactions made during the offer duration.
    '''

    for index, row in data.iterrows():

        # not viewed
        if (str(row['viewed time'])=='nan'):

            # and no transactions
            if row['person'] not in customers_with_transactions:
                status = 'churn'
            # and transactions
            else:
                status = 'active'

        # viewed
        elif (row['viewed time'] >= 0):

            effect_until = row['received time'] + 24*row['offer_duration']

            # but too late
            if row['viewed time'] > effect_until:
                status = 'late'

            # but no transactions
            elif row['person'] not in customers_with_transactions:
                status = 'disinterested'

            # in time and has transactions
            else:
                cust_transactions_during_effect_time = transactions[
                    (transactions.person == row['person'])
                    & (transactions.time <= effect_until)
                    & (transactions.time > row['received time'])]

                # there's transaction(s) during the effective time
                if (len(cust_transactions_during_effect_time)>0):
                    status = 'effected'
                else:
                    status = 'disinterested'

        data.at[index,'status'] = status

    return data
```

In that way I ended up having a data table containing same columns as the other offers and was able to combine them.

offer_duration	received time	viewed time	status
4	336.0	372.0	effected
3	168.0	192.0	effected
4	0.0	6.0	disinterested
3	336.0	354.0	disinterested
3	504.0	660.0	late

- **Person details**

  All person details were easy to extract from the Profile table. I also used the user ID's to find each person's transactions and in that way calculated the average purchase.

age	member_since	gender	income	avg_purchase
33	2017	M	72000.0	15.950000
33	2017	M	72000.0	15.950000
33	2017	M	72000.0	15.950000
118	2018	None	NaN	1.363333
118	2018	None	NaN	1.363333

- **Offer details**

  Offer details were also easily on the Portfolio table. The only small edit I wanted to make was to one-hot-encode channels. That means I moved all options to columns and gave a value 1 if the original value matches the column name and 0 if it doesn't.

channels
[email, mobile, social]
[web, email, mobile, social]

email	mobile	web	social
1	1	1	0
1	1	1	1

- **Adding label/result to each row**

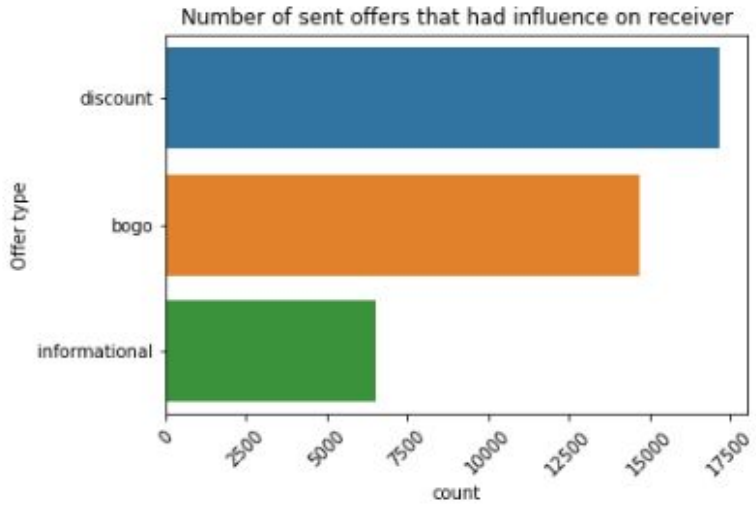
  The last step was to add a result (label) to each row so I added one more new column that has value 1 if the column 'status' had value 'effected' and 0 if it was something else.

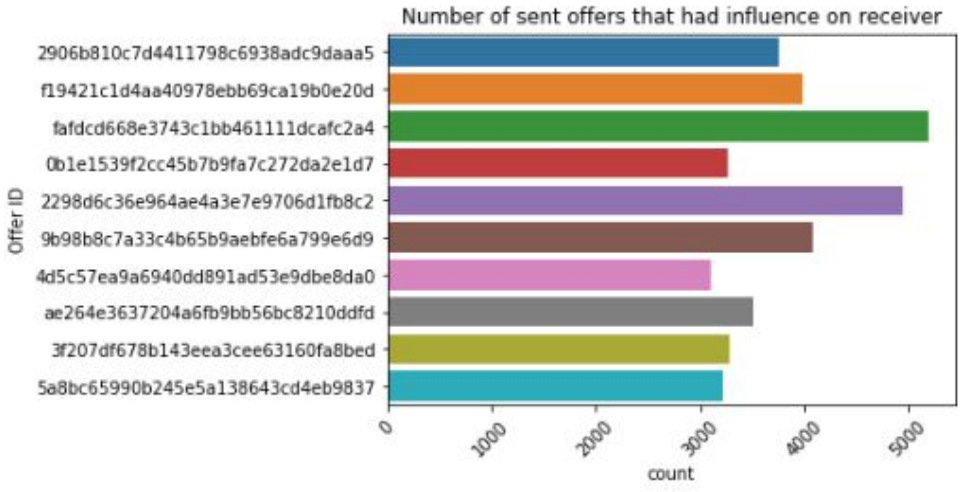
  The 'success' value defines a class the row belongs to and that kind of numerical value is something a model can provide us as a prediction.

status	success
effected	1
effected	1
effected	1
disinterested	0
disinterested	0
effected	1
effected	1
effected	1
effected	1
churn	0

On the following page there is a table with some example rows from the final data frame and some visualizations of how the offers influenced customers.

	person	age	gender	income	member_since	avg_purchase	offer id	offer_type	offer_duration	email	mobile	web	social	difficulty	reward	status	success
0	0009655768c64bdeb2e877511632db8f	33	M	72000.0	2017	15.950000	2906b810c7d4411798c6938adc9daaa5	discount	7	1	1	1	0	10	2.0	effected	1
3	0009655768c64bdeb2e877511632db8f	33	M	72000.0	2017	15.950000	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	1	1	5	5.0	effected	1
4	0009655768c64bdeb2e877511632db8f	33	M	72000.0	2017	15.950000	fafdcd668e3743c1bb461111dcafc2a4	discount	10	1	1	1	1	10	2.0	effected	1
5	00116118485d4dfda04fdbaba9a87b5c	118	None	NaN	2018	1.363333	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	1	1	5	0.0	disinterested	0
6	00116118485d4dfda04fdbaba9a87b5c	118	None	NaN	2018	1.363333	f19421c1d4aa40978ebb69ca19b0e20d	bogo	5	1	1	1	1	5	0.0	disinterested	0
7	0011e0d4e6b944f998e987f904e8c1e5	40	O	57000.0	2018	15.892000	0b1e1539f2cc45b7b9fa7c272da2e1d7	discount	10	1	0	1	0	20	5.0	effected	1
8	0011e0d4e6b944f998e987f904e8c1e5	40	O	57000.0	2018	15.892000	2298d6c36e964ae4a3e7e9706d1fb8c2	discount	7	1	1	1	1	7	3.0	effected	1
11	0011e0d4e6b944f998e987f904e8c1e5	40	O	57000.0	2018	15.892000	9b98b8c7a33c4b65b9aebfe6a799e6d9	bogo	7	1	1	1	0	5	5.0	effected	1
12	0020c2b971eb4e9188eac86d93036a77	59	F	90000.0	2016	24.607500	4d5c57ea9a6940dd891ad53e9dbe8da0	bogo	5	1	1	1	1	10	10.0	effected	1
14	0020c2b971eb4e9188eac86d93036a77	59	F	90000.0	2016	24.607500	ae264e3637204a6fb9bb56bc8210ddfd	bogo	7	1	1	0	1	10	0.0	churn	0





The last preprocessing step was to make the table machine readable. Machine learning models can only consume numerical data, so I removed all rows with empty values (mostly the customers with age 118), replaced empty average purchases with zero and changed all values to numbers in two ways:

Encoding text and group features

The columns with text values (and also the 'member since' column) I encoded to numbers. That means counting the number of unique values in the column and changing texts with corresponding values. In that way each text value is a number between 0 and the maximum number of values.

gender	gender
M	1
F	2
F	2

Scaling numbers

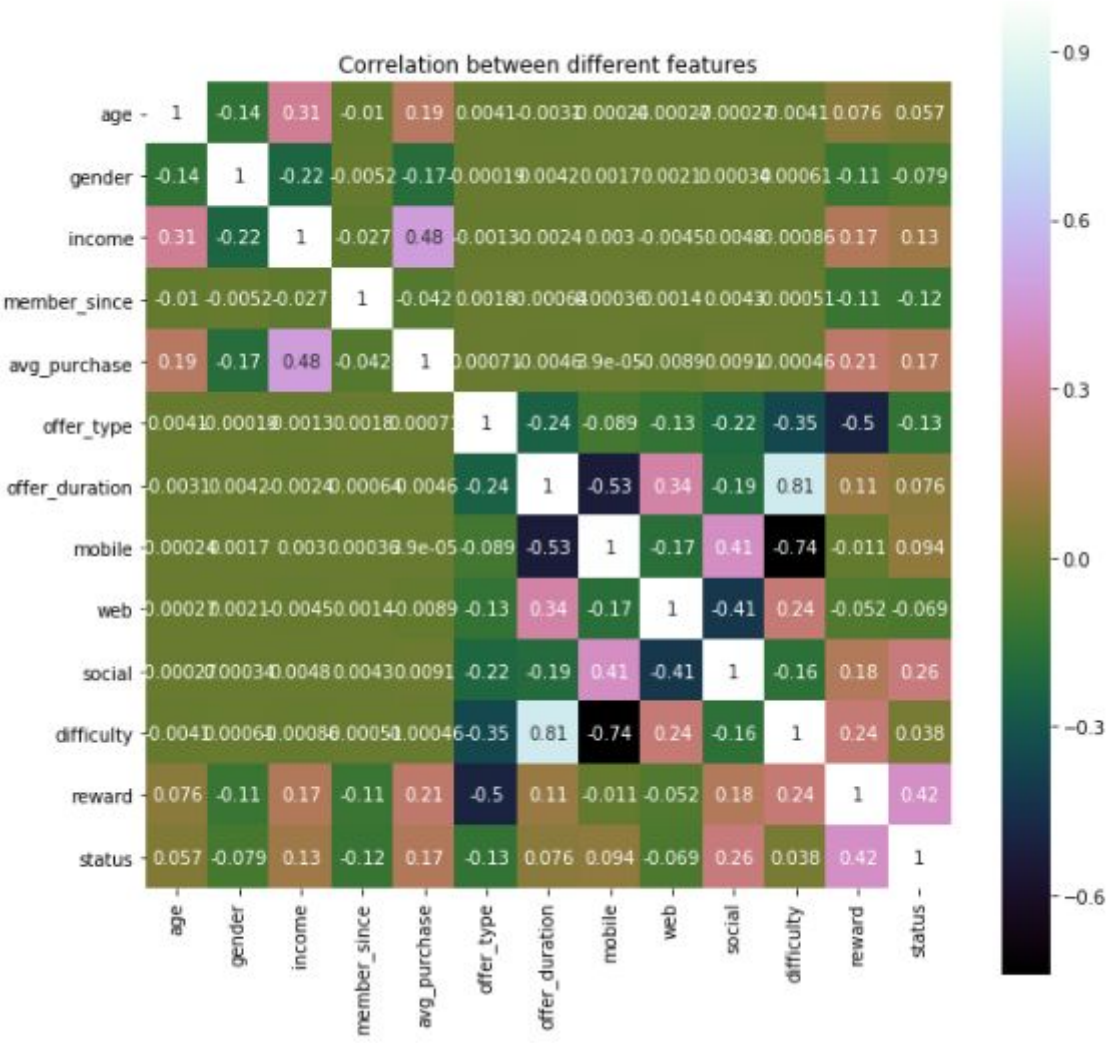
Scaling numbers means you change all values to be between 0-1 so that the max value is 1 and min value is 0. Scaling is a way to help machine learning models to work better because the differences in values are not that big.

At this point I also removed person and offer id columns as those are not scalable. In the eyes of a ML model they are also redundant.

age	gender	income	member_since	avg_purchase	offer_type	offer_duration	email	mobile	web	social	difficulty	reward	status	success
0.180723	0.5	0.466667	0.8	0.035329	0.5	0.571429	0.0	1.0	1.0	0.0	0.50	0.2	0.75	1.0
0.180723	0.5	0.466667	0.8	0.035329	0.0	0.285714	0.0	1.0	1.0	1.0	0.25	0.5	0.75	1.0
0.180723	0.5	0.466667	0.8	0.035329	0.5	1.000000	0.0	1.0	1.0	1.0	0.50	0.2	0.75	1.0
0.265060	1.0	0.300000	1.0	0.035201	0.5	1.000000	0.0	0.0	1.0	0.0	1.00	0.5	0.75	1.0
0.265060	1.0	0.300000	1.0	0.035201	0.5	0.571429	0.0	1.0	1.0	1.0	0.35	0.3	0.75	1.0
0.265060	1.0	0.300000	1.0	0.035201	0.0	0.571429	0.0	1.0	1.0	0.0	0.25	0.5	0.75	1.0
0.493976	0.0	0.666667	0.6	0.054505	0.0	0.285714	0.0	1.0	1.0	1.0	0.50	1.0	0.75	1.0
0.493976	0.0	0.666667	0.6	0.054505	0.0	0.571429	0.0	1.0	0.0	1.0	0.50	0.0	0.25	0.0
0.493976	0.0	0.666667	0.6	0.054505	0.5	1.000000	0.0	1.0	1.0	1.0	0.50	0.2	0.75	1.0
0.493976	0.0	0.666667	0.6	0.054505	0.5	1.000000	0.0	1.0	1.0	1.0	0.50	0.2	0.75	1.0

The table after changing it to machine readable

After the table was ready to use I checked some correlations just because I was interested. However, I noticed email column had always the same value and would have added unnecessary noise to the model so I removed it.



Correlation matrix

At this point I also separated the data to training and testing sets. Training sets are used to 'train' the model so that it finds correct weights to be able to make predictions and testing sets are used to see if the model is working as it should.

In this case 75% of the data was used for training and 25% for testing.

4. Finding the best model

As the first machine learning model (benchmark model) I tested how a basic **decision tree classifier** handles the task. The model predicted 90,86% of the test set rows correctly.

The other models I tested were:

- **Random Forest Classifier**
  Many decision tree classifiers together - accuracy score 90,85%

- **KNeighborsClassifier**
  Groups rows with similar features together - accuracy score 90,31%

- **Multilayer Perceptron Classifier**
  Neural network with 1 hidden layer (8 nodes) - accuracy score 91,24%

5. Fine tuning and using the best model

The best model of the ones I tested was Multilayer Perceptron Classifier. To make it the best possible I ran a code that tested the hyperparameter options I gave to it and found the combination giving the best results.

```
0.9155 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 16, 'learning_rate': 'constant', 'solver': 'adam'}
0.9154 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 16, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9152 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 32, 'learning_rate': 'constant', 'solver': 'adam'}
0.9136 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 32, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9151 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 64, 'learning_rate': 'constant', 'solver': 'adam'}
0.9154 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 64, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'constant', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 16, 'learning_rate': 'constant', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 16, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9158 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 32, 'learning_rate': 'constant', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 32, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 64, 'learning_rate': 'constant', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 64, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'constant', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9157 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 16, 'learning_rate': 'constant', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 16, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 32, 'learning_rate': 'constant', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 32, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9116 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 64, 'learning_rate': 'constant', 'solver': 'adam'}
0.9150 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 64, 'learning_rate': 'adaptive', 'solver': 'adam'}
0.9155 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'constant', 'solver': 'adam'}
0.9156 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (8, 16), 'learning_rate': 'adaptive', 'solver': 'adam'}

Best parameters found:
 {'activation': 'relu', 'alpha': 0.002, 'hidden_layer_sizes': 32, 'learning_rate': 'constant', 'solver': 'adam'}
```

When the best parameters were found I gave the test set to the model and got the final accuracy score 91,26%.

To me, looked great that over 91% of the predictions were correct. I also managed to improve the score from the benchmark model (90%) by 1%. It sounds small, but that's around 120 correct predictions more.

However, when looking at the confusion matrices we can see that the decision tree classifier did not give as many false negatives as the final model did. It was also a bit better when it comes to finding true positives. The multilayer perceptron classifier found true negatives better and provided less false positives.

Nevertheless, the final model is also working in a nice way: false positives (meaning a customer that was predicted to be influenced buys nothing) are super low and that means there wouldn't be offers sent for nothing. False negatives (customers that were predicted not to be influenced by an offer buy something) are not as bad, but

obviously you wouldn't send offers for them so it would be good to improve the model more.

The benchmark model confusion matrix

The final model confusion matrix

Finally, I was also able throw some random numbers to the model so that it can predict the outcome.

```
test_person_row = [[0.421687, 0.0, 0.485664, 0.2, 0.032458, 1.0, 0.142857, 1.0, 0.0, 0.0, 0.0, 0.0, 0.75]]
clf.predict(test_person_row)
```

```
array([ 0.])
```

With the numbers I gave to it the model predicted the customer would not buy anything in the influence of the offer sent. That also means I found pretty nice and usable model.

In the end I was happy with the project outcome but also had some improvements in my mind. This time I decided to leave them for the future.

Examples of improvements that could be done:

- Trying to find out why false negatives happen more often than false positives
- The model could also predict the probability if offer will be viewed, not just completed
- More features could be added, f.ex. minimum and maximum purchases, offer opening percentage, offer completion percentage, how did a customer react to previous offers sent to them, did offer types matter, would it be enough to send an informative offer to someone...
- Features could be selected better to have less correlation between them