Udacity Nanodegree

Machine Learning Engineer

Capstone Project – Starbucks Marketing

Kevin Hubert - February 2021

Table of content

Definition....3
Intro....3
Starbucks & Machine Learning - Introducing....3
Problem Statement....4
Solution Statement....5
Benchmark Model....5
Evaluation Metrics....6
Analysis....8
Data exploration....8
portfolio.json....8
profile.json....8
transcript.json....9
Exploratory Visualization....9
Profiles – Data....9
Portfolio – Data....11
Transcript – Data....12
Merged data....14
Algorithem and Techniques....15
Methodology....18
Data preprocessing....18
Implementation....18
Refinement....20
Results....20
Model Evaluation, Validation and Justification....20
Conclusion....22
Reflection....22
Improvement....22

Definition

Intro

I this project I will apply my learned skills from the "Machine learning Engineer" Nanodegree and will demonstrate the potential of machine learning using a dataset provided by Udacity and Starbucks.

Starbucks & Machine Learning - Introducing

Small snacks such as cookies and cakes, free Wi-Fi and many good coffees in countless varieties - there are many things for which Starbucks, 50 years after its founding, is not only known but also incredibly successful and rightfully counts as the market leader[1] in this segment.
In this project, I demonstrate how this success story is linked to digitization and what additional potential there is for Starbucks in machine learning technology.

Starbucks employs approximately 346,000 people in moren than 30,000 stores in 80 countries worldwide[2]. Due to the frequency of Starbucks stores and the associated huge number of satisfied customers every day, Starbucks generates millions or even billions of data records annually, which can be collected, analyzed and integrated into the optimization of business models with the help of modern technology in the field of artificial intelligence.

For this project, a dataset provided by Starbucks is used as part of the Udacity "Machine learning Engineer" Nanodegree. In this dataset, the following data was provided:

File	Description
portfolio.json	JSON structured information about offers done to customs. Could be rewards, information or BOGO (Buy one & get one free)
profile.json	Demographic information about customers. Containing register-date, gender, income and age for the individual customers.
transcript.json	Information about fulfilled events like receiving-offer, viewed-offer, transaction done etc.

All files were formatted as JSON

- Entries of files are related - Relation is recognizable by the given ID

Source: https://de.wikipedia.org/wiki/Starbucks

1 https://www.statista.com/statistics/250166/market-share-of-major-us-coffee-shops/
2 https://de.wikipedia.org/wiki/Starbucks

Problem Statement

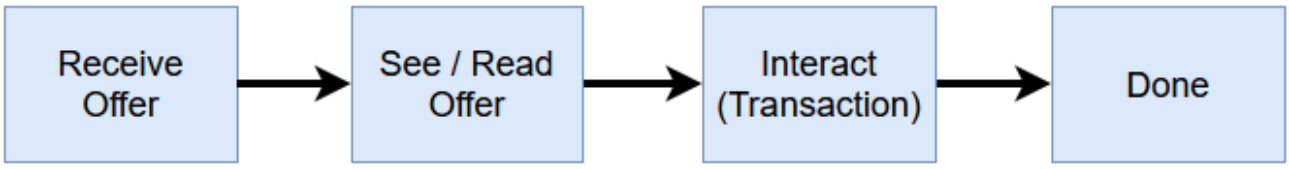
Customers are as individual as their needs. This makes addressing customers directly through marketing campaigns an extremely complex issue. There are various reasons why customers do not respond to offers from companies, a few examples are:

- The customer has not read/received the message - Wrong marketing channel.
- The offer does not meet the customer's needs
- The customer is not interested in the offer
- The offer does not match the type of products the customer is interested in
- The customer only looking forward specific type of offers e.g. BOGO[3] instead of discount
- The customer was approached at the wrong time

On the other hand, success through marketing campaigns depends on customers responding to the offers and going through the following steps to do so

1. Receive offer - Customer receives the offer
2. See / Read Offer - Customer sees/reads the offer
3. Transaction - Customer interacts with offer
4. Done - offer fulfilled

Visualized[4]:



As you can see there are several steps required until the offer is fulfilled (Done). Only if all steps are fulfilled then the custom will be happy and we will increase the companies profit

3 The acronym „BOGO“ stands for „Buy one get one“ and means the second product will be for free.
4 Visualization created using https://app.diagrams.net/

Solution Statement

My solution attempts to predict the propability of a offer to be successful[5] for a given customer. Based on this prediction it should be possible to find the best matching offer for each customer.

My attempts based on machine learning. Therefor I'll transform the data into my required format, analyse the data and visualize my results and then use my data to train a deep-neural-network which will be then used to estimate the probability of a given offer to be successful or not.

To train my model I'll use the data provided by Udacity and Starbucks for this project and based on this data I should be able to build/train a neural network with a good probability if a offer will be successful for a specific customer.

Keep in mind: The outcome of a predictions could also be "Make no offer" especially for customers which may think that offers are annoying or just don't interact with them.

The model then can be used either to plan the future marketing offers completly or as a additional instrument next to the existing markething actions.

Benchmark Model

To evaluate the results of my model, I will compare these predictions of my solution with that of the current marketing strategy using the given test data. For this, of course, only the results of BOGO (buy one get one) and of discounts will be compared, since offers of the "information" type do not lead to any transaction.

For this I'll train a very simple Linear Learner Model (logistic regression) to do a binary classification if a offer would be successfull or not.

5 A „successfull offers“ describes an offer which leads to a transaction (fulfillment) of the given customer.

Evaluation Metrics[6]

For the statistical evaluation of my model, I use the Confusion Matrix which divides the predictions into: TP[7], FP[8], TN[9], FN[10]

These prediction types are especially interesting for us because they describe the following actions:

Type	Description
TP	Offer with high probability to fit the customer needs which will be sent to customer. (good = Good offer for customer)
FP	Offer with low probability to fit the customer needs which is still been sent to customer (bad = most likely to be unnecessary).
TN	Offer with low probability to fit the customer needs which will not been sent to the customer. (good = Do not make bad offers)
FN	Offer with high probability to fit the customer needs which will not been sent to the customer (bad = Good offer not sent to customer)

I'll calculate the accurancy, precision and recall which are kindly explained by the graphic (Figure 1) below based on the given predictions:

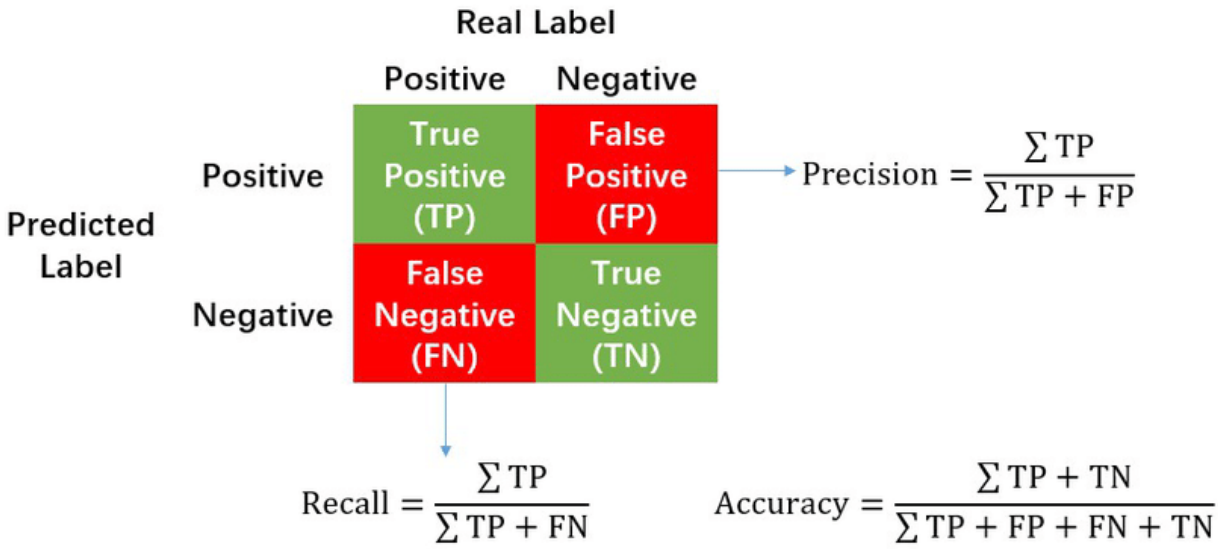


Figure 1

As described in the „Data exploration“-chapter (below) the data is good balanced because we have

- 48.79% of completed offers
- 51.21% of not completed offers.

This data is so good balanced so the accuracy (statistic of correctly classified entries divded by all classifications) has the most relevant meaning for my classification.

6 https://en.wikipedia.org/wiki/Confusion_matrix
7 TP ancronym for „True Positive“ describes a correctly classified prediction as positive
8 FP ancronym for „False Positive“ describes a negative result falsly predicted as positive
9 TN ancronym for „True Negative“ describes a correctly classified prediction as negative
10 FN ancronym for „False Negative“ describes a positive result falsly predicted as negative

Anyway I am also motivated to add the following metrics because each of them describe a slightly different aspect of the quality of the results preditions:

Metrics	**Description**
Recall	„Recall, also known as sensitivity, is the fraction of examples classified as positive, among the total number of positive examples. In other words, the number of true positives divided by the number of true positives plus false negatives.“ - Thomas Wood
Precision	„Precision is the fraction of true positive examples among the examples that the model classified as positive. In other words, the number of true positives divided by the number of false positives plus true positives.“ - Thomas Wood
F1-Score	„The formula for the standard F1-score is the harmonic mean of the precision and recall. A perfect model has an F-score of 1. We recall that the F-score is the geometric mean of precision and recall. Like the arithmetic mean, as a geometric mean the F-score is between the precision and recall.“ - Thomas Wood
Accurancy	„Accuracy is defined as simply the number of correctly categorized examples divided by the total number of examples. Accuracy can be useful but does not take into account the subtleties of class imbalances.“ - Thomas Wood

Sources: https://deepai.org/machine-learning-glossary-and-terms/f-score

Analysis

Data exploration

The data and files used by this project are provided by Udacity and Starbucks for the given nanodegree program. The given data contains simulated customers, transactions and offers similar as it is done by the Starbucks app.

portfolio.json

- JSON structured information about offers sent to customs. Could be „rewards“, „information“ or „BOGO“ (Buy one & get one (free)) |
- Shape (Rows x Columns): 10 x 6
- Column description:
    - id (string) - unique ID to identify offer referenced in other data
    - offer_type (string) - type of offer (one of: 'BOGO', 'discount', 'informational')
    - difficulty (int) - minimum required spend to complete an offer
    - reward (int) - reward given for completing that offer
    - duration (int) - time (in days) for offer to be open
    - channels (strings[]) - Channels the offer is sent out (i.e.: 'web', 'email', 'mobile', 'social')

profile.json

- Demographic information about customers. Containing register-date, gender, income and age for the individual customers.
- Shape (Rows x Columns): 17.000 x 5
- Column description:
    - id (str) - unique ID to identify customers referenced in other data
    - age (int) - age of the customer (value 118 indicates it is missing)
    - became_member_on (int) - date when customer created an app account (formatted: yyyymmdd)
    - gender (str) - gender of the customer ('O' = other, 'M' = male, 'F' = female, 'null' = missing)
    - income (float) - customer income (currency is not defined)

transcript.json

- Information about fulfilled events like receiving-offer, viewed-offer, transaction done etc.
- Shape (Rows x Columns): 306.648 x 4
- Column description:
  - event (str) - record description (ie 'transaction', 'offer received', 'offer viewed', 'offer completed')
  - 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

- All files were formatted as JSON[11]

Exploratory Visualization

The charts below show the quantity ratio of different data for the given example data provided for this project.
Because some of the given datasets also contained incomplete/invalid data I've cleaned out all unnecessary datasets which were incomplete or invalid.

Profiles – Data

```
getPandaTableInfo(profile, "Profile")
```

```
"Profile" has 17000 rows with 5 columns
```

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

(Screenshot from Jupyter Notebook)

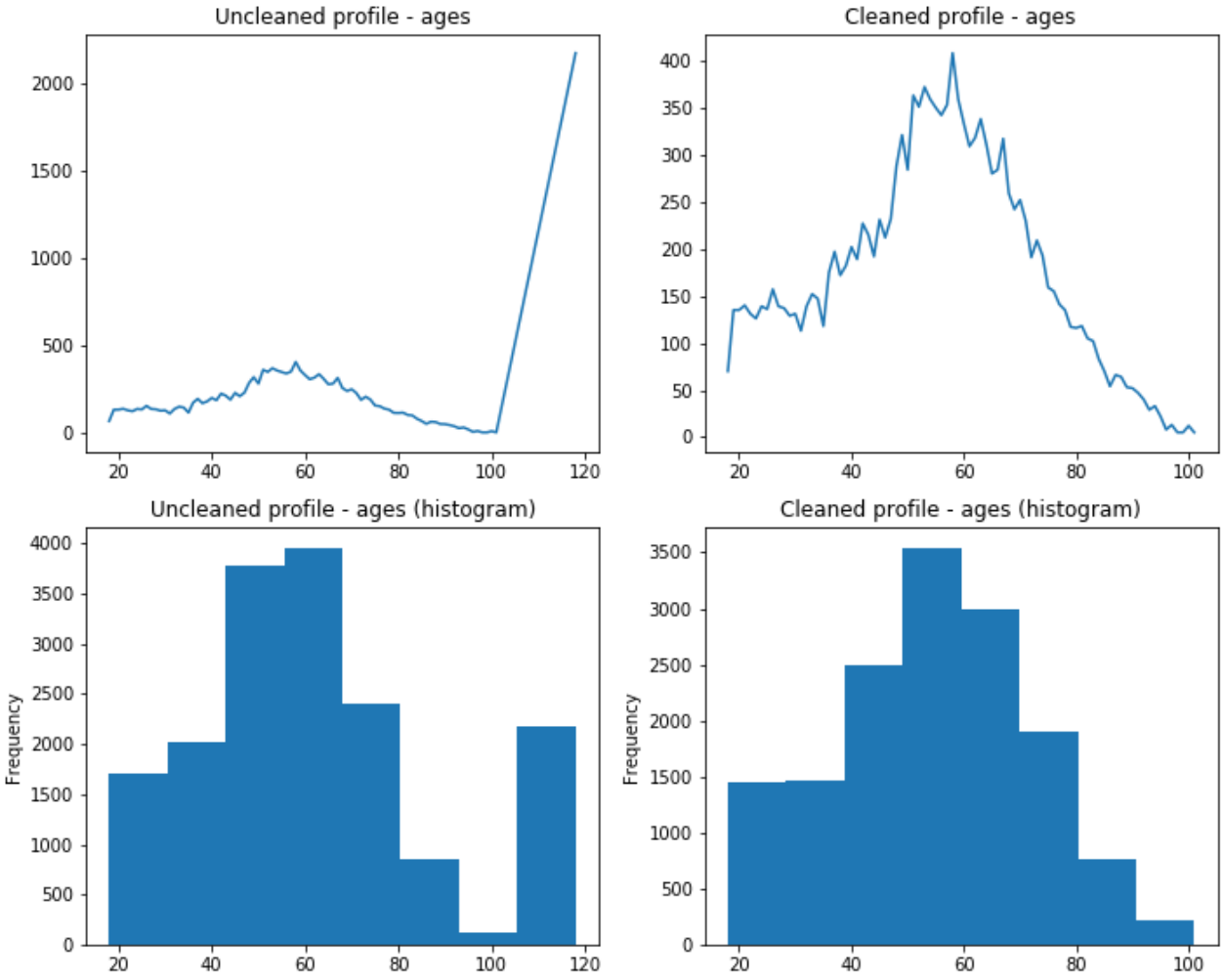
As you can see in the table above there are some people with obviously invalid or missing values

- income of "NaN"
- age "118"
- gender "None"

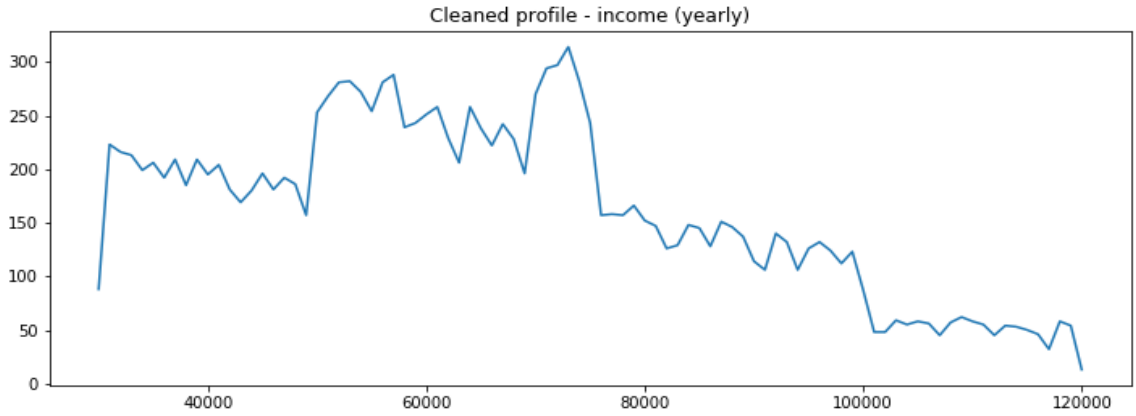
These affected a total of 2175 entries which were removed from the dataset.

Compare of quantity ratio of given ages before (left) and after (right) data cleaning.
In the barcharts you can clearly see that the average customer is ~45 - 60 years old

11 https://en.wikipedia.org/wiki/JSON



In the next chart I've visualized the given income of the cleaned customers. Because a income of „NaN“ in to uncleaned data won't be shown in the graph I've decided to only show the valid profiles in here. As expected the most customers have a yearly salary of 50k – 75k.

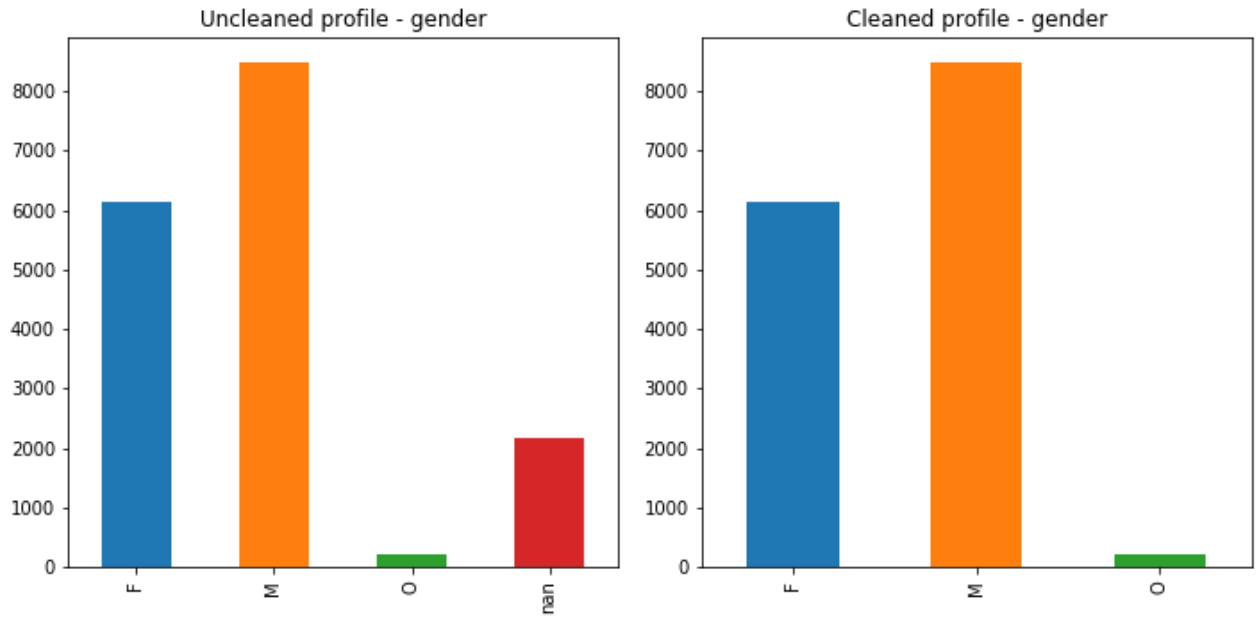


Also quantity ratio in relation to the selected gender is quite evenly.

We have about:

- 6000 „Females“
- 8500 „Males“
- < 500 „Other“

The 2.100 entries without a gender given are only shown for quantity ratio compare but as already mentioned above has been returned during the data-cleaning phase.



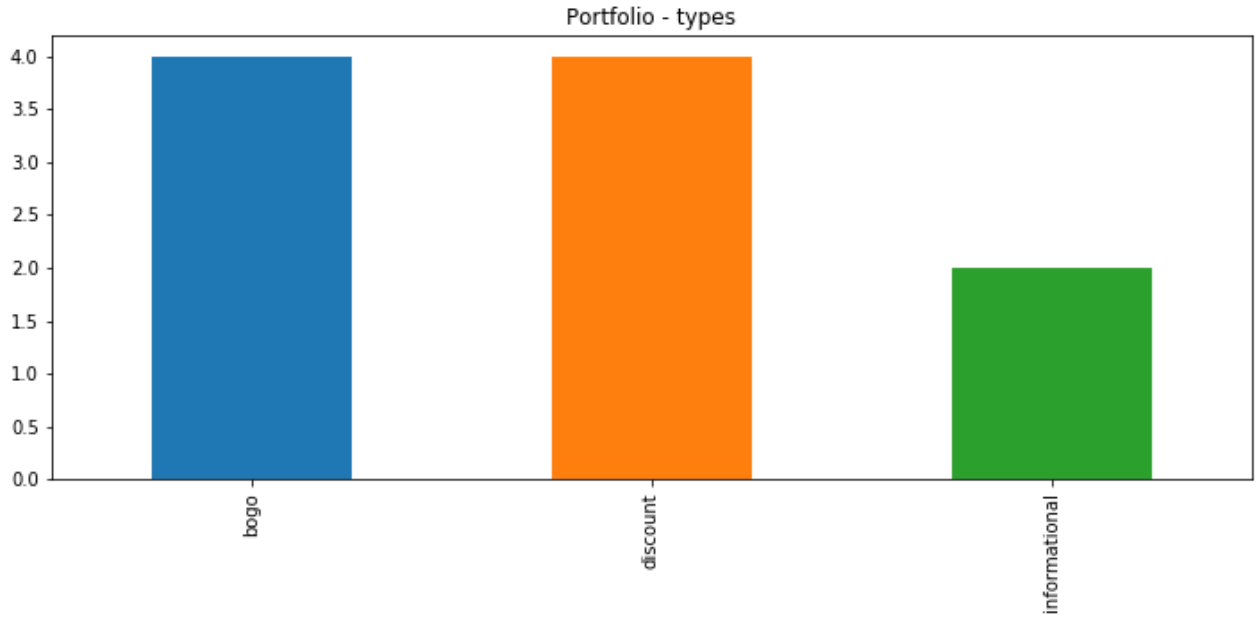
Portfolio – Data

The portfolio dataset only contained 10 rows with 6 columns which represent possible offers which can be sent to a customer. Because the data is quite small I've decided to add only some quantity-ratio visiualizations in form of small charts.

```
getPandaTableInfo(portfolio, "Portfolio")
```

```
"Portfolio" has 10 rows with 6 columns
```

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



All in all there are:

- 4 different types of „BOGO“-offers (buy one get one)

- 4 different types of discount offers

- 2 informational offers

As already described the informational offers will be removed from the dataset before training my neural-network and benchmark-models with it. This is useful because „informational“-offers are only meant to be read but do not lead to any kind of completion because they have no task to fulfill.

All in all we ca see that 8 / 10 offer-types are interesting for us (bogo + discount).

Transcript – Data

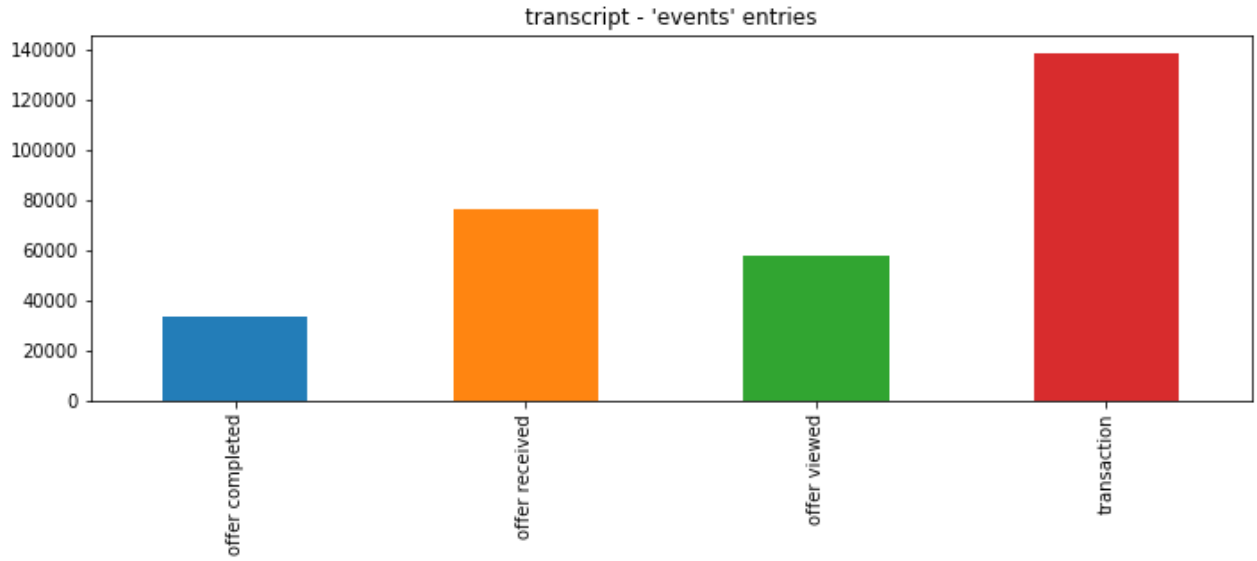
```
getPandaTableInfo(transcript, "Transcript")
```

```
"Transcript" has 306534 rows with 4 columns
```

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

(Screenshot from Jupyter Notebook)

The last provided datasets represents all the transcripts which describe all transaction done. This splits up into four different types of transcript-events.



- offer received – Offer was sent to the specific customer
- offer viewed - Offer was viewed by the customer
- offer completed – The offer was successful completed by the customer.
- transcation – Customer did a payment, can be caused based on a offer but does not have to.

The quantity ratio of offer-events is visualized here:

As you can see on the charts above the conversion rate of a completed offer after receiving/viewing it is quite high.

The detailed conversion-rate for offers viewed/-completed after a received offer are shown below:

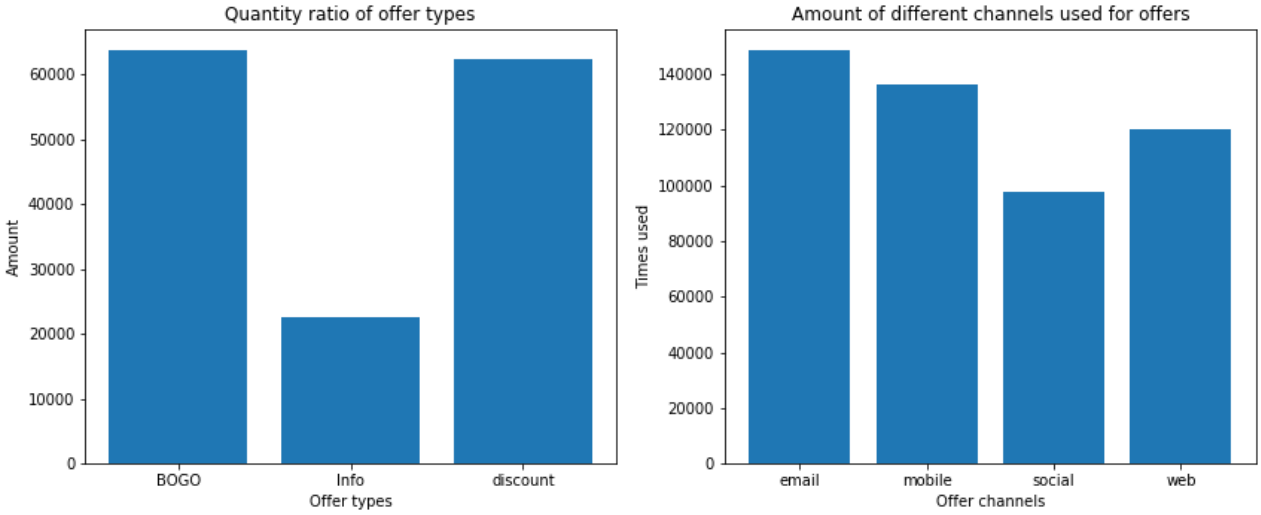
```
receivedOffers = merged[merged['offer_received'] == 1]
viewedOffers = merged[merged['offer_viewed'] == 1]
completedOffers = merged[merged['offer_completed'] == 1]
transactions = merged[merged['transaction'] == 1]

receivedOffers_count = receivedOffers.shape[0]
viewedOffers_count = viewedOffers.shape[0]
completedOffers_count = completedOffers.shape[0]
transactions_count = transactions.shape[0]

viewedOffersPercent =  viewedOffers_count / receivedOffers_count * 100
completedffersPercent =  completedOffers_count / receivedOffers_count * 100

print('receivedOffers:  {0}'.format(receivedOffers_count))
print('viewedOffers:    {0} ({1}%)'.format(viewedOffers_count, round(viewedOffersPercent, 2)))
print('completedOffers: {0} ({1}%)'.format(completedOffers_count, round(completedffersPercent, 2)))
print('transactions:    {0}'.format(transactions_count))
```

```
receivedOffers:  66501
viewedOffers:    49860 (74.98%)
completedOffers: 32444 (48.79%)
transactions:    123957
```

With a offer-completion rate of ~48.79% we can already see that the offers sent out to our customers are kind good and we can say that about 1 / 2 offers are completed by our customers based on starbucks existing marketing campaign.

Merged data

Since within the given datasets entries refer to each other with the help of foreign keys, some interesting information only comes to light after the datasets have been resolved on the basis of their IDs and thus a large amount of data with all metadata is displayed.
For this purpose, I used the "Transcript" records to replace the linked "Portfolio" records as well as "Profile" records based on the stored IDs and was thus able to find out even more information:



The most offers sent out to customer where either of type „discount“ or „BOGO“.
„Info“-offers were only given ~20.000 times and those will not be used for my machine-learning approaches.
Additional we can see that most of offers were sent out using e-mail or the mobile-app but also web and social are often used marketing-channels[12] to sent the given offers to the starbucks-customers.

12 Keep in mind that multiple marketing-channel can be used at the same time for an offer..

Algorithem and Techniques

For my solution to predict the probability if a offer will be completed or not I create a Classification-Neural-Network which is widely used for classification tasks.

Based on the huge amount of datasets I should be able to have a quite good working solution to predict if a given offer will be completed or not.
Based on this predictions the marketing campaigns can be adjusted and especially offers which are less likly to be completed can be removed.

In the end, the classification neural network I trained will give a value in the range of 0.0 - 1.0 to what extent your offer is completed or not. Here a value close to 0 corresponds to "No" and 1 to "Yes". This value is then rounded using the following formula to get a unique result.

(X >= 0.5) ? 1.0 : 0.0

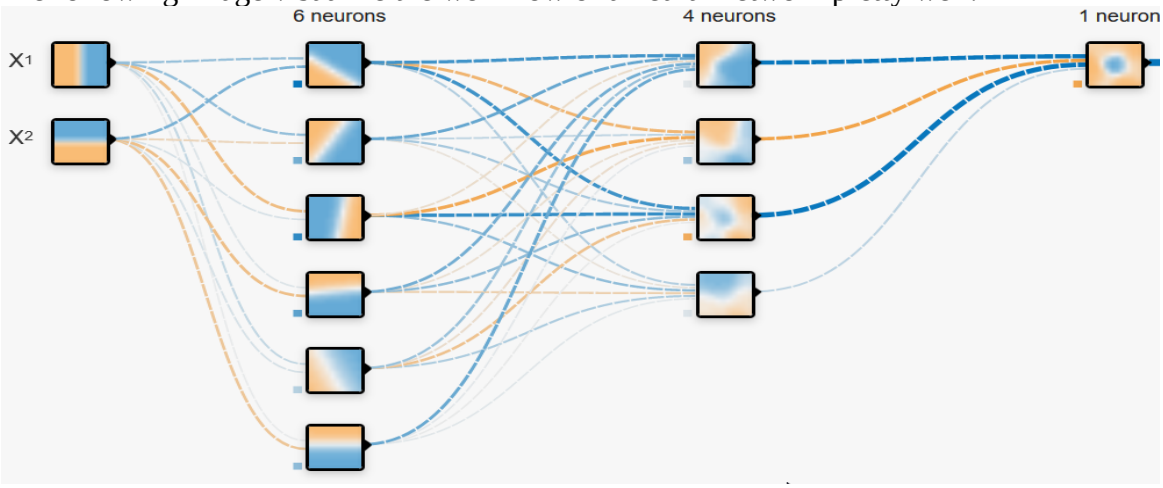
For the optimization of the classifier neural network, the following parameters can be adjusted:

- Numbers of epochs to train
- Batch size (amount of training-datasets to use during one single training step)
- Learning rate (factor to increase/decrease learning rate (weight adjustment))

Also the whole architecture of the Classifier-Neural-Network can be adjusted. This contains:

- Input-Dimension (amount of parameters used as input for training)
- Hidden-layer/-neurons (amount of fully connected hidden layers/neurons)
- Activation-Function used by each layer
- Dropouts (percentage amount of dropouts after each layer to avoid overfitting)

Neural-networks and deep-neural-network are especially effective for classification task with high dimensionality of data. To do the classification a neural network uses at least a input-layer which represents the amount of input-parameters with a single neuron for each of them and a output-layer which is in my approach a single neuron with a value between 0.0 (prediction „false“) to 1.0 (prediction „true“). Additional to the input-/output-layer there can be a theoretically infinit amount of hidden-layers with theoretically unlimited amount of neurons. Because i just a so called fully-connected-neural-network each neuron from the previous layer is connected with the neurons of the next layer and so has a high amount of adjustable weights which will be tuned during the network-training to be able to learn make predictions based on the data given.
The following image visualize the workflow of a neural network pretty well:



Source: https://playground.tensorflow.org/

On the other hand (deep-)neural-network are quite complex and may not lead to significantly better results than conventional classification algorithms.

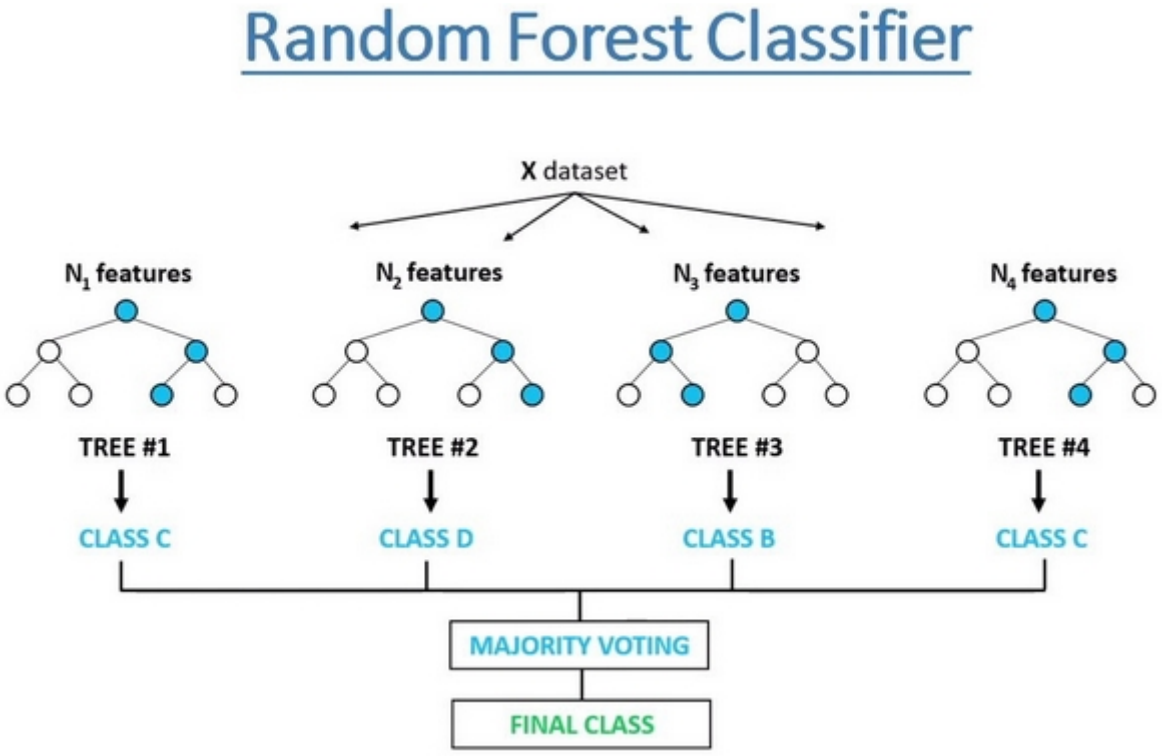
To have a compare of my neural network to more conventional machine learning techniques I'll also use two different approaches to see how they perform with the data given: Logistic Regression and Random Forest.

Random Forest: A random forest is a algorithm which can be used for both classification- and regression-tasks. Random forest combines the results of multiple decision-trees to get the best classification for the given tasks. The benefits of a random forest and decision trees in general are:

- Short training time
- Easy to use (libraries work out of the box)
- Relative simple procedure in direct compare to neural networks

The created decisions trees in a random forest are individual responsible for their classification and based on the results of multiple decisions tree a final prediction/classficiation is returned by the random forest.

Visualization of a decision-tree:



Source: https://www.youtube.com/watch?v=goPiwckWE9M

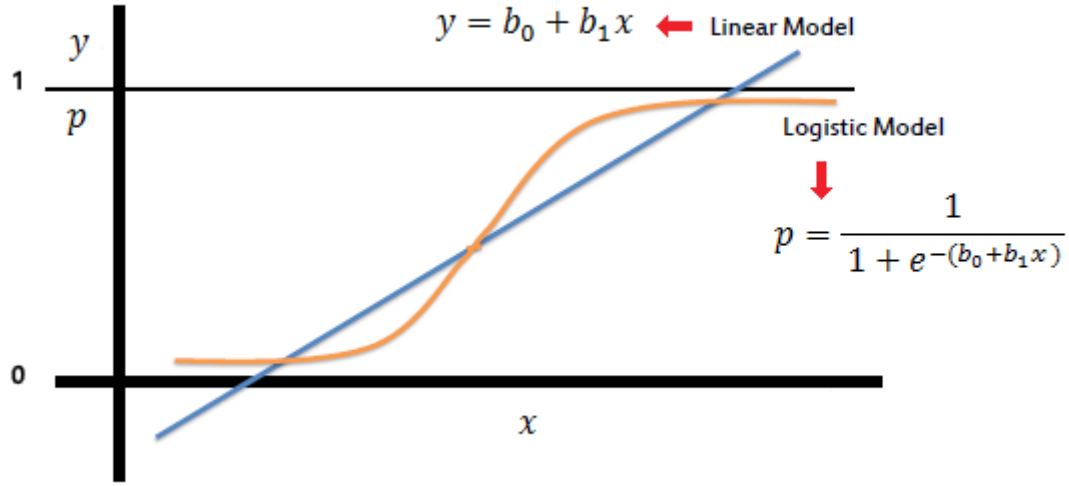
Logistic Regression: Also used for binary classification there is a classifier called „Logistic Regression“. A logistic regression analyzed a given set of datapoints of one or more independent variables and calculate the best fitting coeffients to get the expected result.
The equation for a logistic regression:

$$Y_i = \frac{1}{1-e^{-X_i'\beta}} + \varepsilon_i \text{ where } -\infty < x < \infty, y = 0,1$$

The strengths of a logistic regression are for sure the ease of implementation, traceability and especially the efficients classification for well known input variables.

The weaknesses of logistic regression classifier are the problems with the acquisition of high-dimensional correlations of input-data and problems during the prediction for partially unknown/new data which was not given in the training-dataset.

A visualization comparing a logistic regression and linear model:



Source: https://saedsayad.com/logistic_regression.htm

For a in-detail explanation on how logistic-regression works I highly recomment the „Statquest Logistic regression“ Youtube Tutorial by Josh Starmer: https://www.youtube.com/watch?v=yIYKR4sgzI8

Methodology

Data preprocessing

The whole preprocessing of the given datasets is completly implemented and well documents/structured in the attached jupyter-notebook. To fufill the data-processing I've realised the following steps:

1. Remove invalid dataset entries as described in „Data exploration“
2. Complete data sets by resolving the referencing.
3. Normalize data using MinMaxScaler[13] for numeric data or One-Hot-Encoding for strings[14]
4. Extract label for each dataset entry which indicates if a offer was completed (1) or not (0).
    1. If the same offer was sent to the same person multiple time I've calculated the propability of how often a offer was completed and rounded the result to a binary-classification-result (1 or 0)

After preprocessing all interesting datasets I had 13 input-parameters as inputs for my classification-neural-network. Visualized:

	age	gender_male	gender_female	gender_other	income	bogo	informational	discount	web	email	social	mobile	days_member	label
0	0.265060	0	0	1	0.300000	0	0	1	1	1	0	0	0.108612	1
1	0.096386	0	1	0	0.477778	0	0	1	1	1	0	0	0.219419	1
2	0.012048	0	1	0	0.388889	0	0	1	1	1	0	0	0.392759	0
3	0.433735	1	0	0	0.677778	0	0	1	1	1	0	0	0.386177	1
4	0.481928	0	1	0	0.988889	0	0	1	1	1	0	0	0.126714	1

(Screenshot from Jupyter Notebook)

Also the given dataset was splitted into three types of sub-datasets:

Name	Description	Percentage	Entries
Train	Used for training	60%	26.508
Validation	Used for training/validation of the network during the training phase	20%	8.837
Test	Used as an indicator after the model-training. Important: This data will not be used for training and is so completly new for the model.	20%	8.837

Implementation

For the creation of my classification-neural-network I've used the Keras[15] library which allows me

13 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
14 https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
15 https://keras.io/

to create a fully-connected-neural-network using a high-level-API.

For the architecture of my neural-network I've used the amount of neurons/hidden-layers which fits my data best after multiple attempts of try-and-error. The final architecture I end up with is the following:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 13)                182
_________________________________________________________________
dense_29 (Dense)             (None, 26)                364
_________________________________________________________________
dropout_16 (Dropout)         (None, 26)                0
_________________________________________________________________
dense_30 (Dense)             (None, 13)                351
_________________________________________________________________
dropout_17 (Dropout)         (None, 13)                0
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 14
=================================================================
Total params: 911
Trainable params: 911
Non-trainable params: 0
```

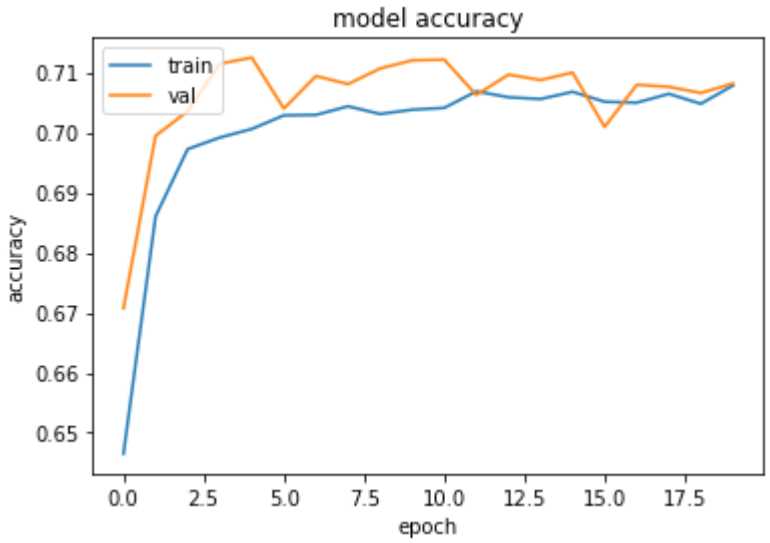
For the training/validation/test of my neural network I came out with 13 input parameters:

Name	Value Range	Description
age	0.0 – 1.0	MinMaxScaled age
gender_male	0 or 1	One-Hot-Encoded if gender 'male'
gender_female	0 or 1	One-Hot-Encoded if gender 'female'
gender_other	0 or 1	One-Hot-Encoded if gender 'other'
income	0.0 – 1.0	MinMaxScaled income
bogo	0 or 1	One-Hot-Encoded if type 'bogo'
informational	0 or 1	One-Hot-Encoded if type 'informational'
discount	0 or 1	One-Hot-Encoded if type 'discount'
web	0 or 1	One-Hot-Encoded if channel 'web'
email	0 or 1	One-Hot-Encoded if channel 'email'
social	0 or 1	One-Hot-Encoded if channel 'social'
mobile	0 or 1	One-Hot-Encoded if channel 'mobile'
days_member	0.0 – 1.0	MinMaxScaled amount of days the person was registered

To view the training performance of my model I've created an additional plot to visualize the accuracy for training and validation data during the training.

The graph especially visualize the accuracy for the training and validation data during the training

pretty well and make overfitting/underfitting pretty good recognizable:



Refinement

At the beginn I've started with a quite simple model which had no hidden layer/neurons.
After multiple attempts of adjustments I've did the following steps to ensure the model will learn stable and become better at predicting:

1. Add two hidden layers with a small amount of neurons to allow the model to learn more complexe relations between parameters given
2. Added a dropout after each hidden layer to reduce the probablity of over-fitting on the training data. I've just a dropout-rate of 0.2 (20%)
3. Increase the batch-size from 1 → 20 which should also ensure that the weight-adjustment is done based on multiple entries instead of a single one.
4. Ensure train-data is shuffled before each training-epoch
5. Reduced the amount of epochs: At the begin I've started with a very high amount of epochs (+50) but realised that after the first 20epochs the accurancy does not increase anymore.

Results

Model Evaluation, Validation and Justification

Next to my approach to solve the task with a neural network I've also decided to do the prediction using a simple LogisticRegression and DecisionTree (RandomForest). Both the LogisticRegression and RandomForest are already part of sklearn and so can easily been used and tested.

For my surprise both of them did a really great job on the data given and also the difference between them and the neural networks are very small, in some kind they also outperformed the neural network.
To keep the results comparable I've trained all (neural network, logistic-regression and RandomForest) with the exact same data and used the previously extracted test data which was unknown during the training for the evaluation of all given approaches.

The different models were created and trained with the following configuration:

LogisticRegression:

- Random State 0

RandomForest:
- Max-Depth: 8
- Random State: 0

Neural Network #1:
- Input Neurons: 13 (Relu)
- Output Neurons: 1 (Sigmoid)
- Type: Fully connected
- Optimizer: „Adam“
- Loss „Binary crossentroyp“
- Metrics: „accuracy“
- Epochs: 20
- Batch size: 500

Neural Network #2:
- Input Neurons: 13 (Relu)
- Hidden Layer 1 Neurons: 26 (Relu)
- Dropout 0.2 (20%
- Hidden Layer 2 Neurons: 13 (Relu)
- Dropout 0.2 (20%
- Output Neurons: 1 (Sigmoid)
- Type: Fully connected
- Optimizer: „Adam“
- Loss „Binary crossentroyp“
- Metrics: „accuracy“
- Epochs: 20
- Batch size: 500

For the comparsion of all models I've used the following metric-scores:
1. Accurancy
2. Precision
3. Recall
4. F1-Score
5. Confusion-Matrix

All of these are already part of the sklearn-lib. I found a excellent descriptions of them and how they are calculated here: https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/ which I highly recommend to read.

Model	Accuracy	Precision	Recall	F1 score	True Negative	False Positive	False Negative	True Positive
Random Forest	0.708385	0.792160	0.730494	0.760078	2178	1071	1506	4082
Logistic Regression	0.670363	0.796041	0.687793	0.737969	1822	1051	1862	4102
Neural Network #1	0.675116	0.831748	0.681399	0.749104	1680	867	2004	4286
Neural Network #2	0.704085	0.721910	0.758874	0.739930	2502	1433	1182	3720

All in all the results are pretty mixed, there are a lot of benefits and disadvantages for each model

	Benefit	Disadvantage
Random Forest	- Highest F1 Score - Good TP and TN. values	- No especially good value
Logistic Regression	- Very reduced amount of complexity - Easy to create - High TP	- High FN
Neural Network #1	- Highest precision - High F1 score - Low FP - High TP	- High FN - Low TN - Low accurancy
Neural Network #2	- High recall - Highest TN - Lowest FN	- Lowest TP - Lowest Precision - Highest FP

Conclusion

Reflection

Doing this project I was able to process completly unknown dataset with a high complexity an amount of entries. I was able to visulize quantity ratio and fulfilled the task of identify and clean incomplete data and anomalies.

I especially liked the step of data preparation, because here existing data could be taken over 1 to 1 but also additional data like the "member_days" could be derived based on other data (e.g.: "member_since"), which gave the models additional usable training parameters and thus could solve the task even more precisely.

Even if my attempt of a deep neural network had no extraordinary advanteges agains simple statistical-method like DecisionTrees or LogisticRegression I was able to build up neural networks of different complexity.

In the end I think it is a very existing knowledge that normal statistic sometimes can perform similar good to complex neural networks.

Based on my solution it should be possible to improve the marketing campaign of starbucks by decide if a offer would be used by a customer based on the calculated probablity of the given models.

Improvement

I think there are several ideas how to improve the model. I'll list some here and describe my expected benefits from them:

1. Evaluate and increase input-parameters: Based on additional input-parameters which are not yet given it could be possible to improve the predictions based on more information given. This could include e.g. the „difficulty“ or „reward“ of a offer from the portfolio-dataset or even additional derived information based on existing ones.
2. Highly increase the amount of neurons and epochs. Based on this it might be possible to

gain a benefits from detailed factors which were not learned by the model based on the limited neurons/training-epochs

3. Get more data: May it would be possible to collect/receive even more data by the starbucks app to get more knowledge from.
4. Use a Recurrent Neural Network which may benefits from the historical information in predicting if a offer will be completed or not

,

Sources / References:

Starbucks Logo	https://de.wikipedia.org/wiki/Starbucks	16.02.2021
Figure 1	https://www.researchgate.net/figure/Calculation-of-Precision-Recall-and-Accuracy-in-the-confusion-matrix_fig3_336402347	16.02.2021
Udacity capstone project - Report example by Martin Bede	https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf	19.02.2021
Udacity captstone project – Report example by Naoki Shibuya	https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-3.pdf	
Metrics used	https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/	19.02.2021
Binary Classification with keras	https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/	19.02.2021
One Hot Encoding	https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/	19.02.2021
MinMaxScaler	https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html	19.02.2021
Keras	https://keras.io/	20.02.2021
Udacity	https://www.udacity.com/course/machine-learning-engineer-nanodegree--nd009t	20.02.2021
Scikit-learn	https://scikit-learn.org/stable/	20.02.2021
Deepai Metric definition	https://deepai.org/machine-learning-glossary-and-terms/f-score	20.02.2021
Metrics for Classification	https://de.wikipedia.org/wiki/Beurteilung_eines_Klassifikators#Kombinierte_Ma%C3%9Fe	20.02.2021
Random Forest	https://www.bigdata-insider.de/was-ist-random-forest-a-913937/	20.02.2021
Logistic regression	https://missinglink.ai/guides/neural-network-concepts/classification-neural-networks-neural-network-right-choice/	20.02.2021