Machine Learning Engineer Nanodegree

Capstone Project

Pedro Gabriel Salazar do Nascimiento
March 25th, 2020

I. Definition

Project Overview

Starbucks Corporation is an American coffee company and coffeehouse chain. Starbucks was founded in Seattle, Washington, in 1971. As of early 2019, the company operates over 30,000 locations worldwide.

Starbucks is a company with a lot of tough competition and that's why it depends a lot on the loyalty of its customers. For this, it is important to offer campaigns that, on the one hand, make the customer feel that it is valued, and on the other, offer a "win-win" environment in which the customer is encouraged to consume more.

The old "advertising mailshots campaigns" have been replaced by e-mail and social networks. These allow to reach more people more quickly, easily and economically. On the other hand, the overuse has caused that people get tired of receiving so many emails or communications and they end up immediately unattended or in the spam mailbox.

Therefore, there is a new trend towards personalization of campaign communication, to make each campaign reach only the people who are most likely to take advantage of it.

Starbucks joined the Capstone Project as a way to see how to profit from Machine Learning to create value from the data about customers and their consuming behavior recollected for one month.

The data collected by Starbucks was, basically (we will go deep in this matter in a further chapter):

- The profile of customers to whom different offers and communications were sent. This profile includes characteristics such as gender, age, income, etc.
- The characteristics of the different campaigns sent to the customers. Characteristics such as type of campaign (information, offer), value, means of communication.

- The reaction of the customers to the different campaigns. Whether it was received, read, completed.

Problem Statement

As introduced in the previous chapter, Starbucks has the need to change its customer communication strategy towards a more selective one where each offer is sent only to those who are more likely to take it and complete it. This way:

- The probability that the customer ends up considering communications from Starbucks as "spam" is drastically reduced.
- The general positive image of Starbucks is preserved.
- The attention paid to communications coming from Starbucks is maximized, what leads as well to a maximization of the number of customres completing any offer, what should lead in the end to more loyalty customers (and more customers as well) and an increase in sales and revenue.

Therefore, the problem that Starbucks is facing can be described as follows:

"How can I know if one customer is likely to accept one of my offers so I can decide whether to send him or not a communication about that offer?".

To deal with this problem, Starbucks has run a one-month-long research where information about customers and their behavior towards different offers was collected.

My approach to answer to the above question will make use of Machine Learning technology and will require some previous steps:

1. Analyze the data collected to determine what information could be relevant to my purpose.
2. Transform the original data and prepare the datasets that will be used by the machine learning algorithm.

From the introductory description of the data provided by Starbucks, we can already infer that our training and test data will consist of relevant customer characteristics and the propensity to take any of the different offers. Hence, we are in front of an example of supervised learning.

Supervised learning is the machine learning task of inferring a function that maps an input to an output based on example input-output pairs.

Scikit-learn library provides plenty of prediction models for supervised learning. Two of the most popular ones are Gaussian Naïve Bayes and C-Support Vector Classification.

It is not the scope of this project to find the best prediction model. Instead, this project will provide a framework to use and evaluate prediction models.

This framework consists of:

- Datasets that will be used to train and test a prediction model.
- Metrics calculation functionality for evaluating/benchmarking a prediction model.
- A couple of examples of training, testing and evaluating a prediction model.

Metrics

The performance of models will be evaluated upon two different metrics: accuracy and F1 score.

The trained models will provide predictions on the customers contained in the test dataset and their predictions will be compared with the corresponding propensity labels (also stored in the test dataset). There will be:

Propensity Label / **Prediction**	**True (customer tends to take offer)**	**False (customer tends to not take offer)**
True	True Positive (TP)	False Positive (FP)
False	False Negative (FN)	True Negative (TN)

Accuracy is used as a statistical measure of how well a binary classification test correctly identifies or excludes a condition. That is, the accuracy is the proportion of true results (both true positives and true negatives) among the total number of cases examined.[1]

Accuracy = (TP+TN)/(TP+TN+FP+FN)

Precision, also referred to as positive predictive value (PPV), is the fraction of retrieved documents that are relevant to the query:

Precision = (TP)/(TP+FP)

[1] Metz, CE (October 1978). "Basic principles of ROC analysis" (PDF). Semin Nucl Med. 8 (4): 283–98. PMID 112681

Recall, also referred to as the true positive rate (TPR) or sensitivity, is the fraction of the relevant documents that are successfully retrieved:

Recall = (TP)/(TP+FN)

F1-Score is the harmonic mean of Precision and Recall. It ensures that the model is predicting correct positive labels, without leaving behind any actual propensity from any customer.

F1-Score = 2*Precision*Recall/(Precision+Recall) = $2(TP)^2/(2TP+FP+FN)$

II. Analysis

Data Exploration and Exploratory Visualization

Starbucks & Udacity provide three data sources in three files:

Portfolio

It contains the list of all available offers to propose o the customer.

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.

The file contains the offers sent during the 30-day test period (10 offers x 6 fields)

- 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 [6]: portfolio.head(20)
```

Out[6]:

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

We will center our project on “real” offers. That is, informational offers will be discarded and only BOGO and discount offers will be considered for our Machine Learning prediction algorithms.

Profile

It contains information about the customers who participated in the reward program (17000 users x 5 fields):

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
- id (str) - customer id
- income (float) - customer's income

```
In [9]: profile.head(10)
```

Out[9]:

	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
5	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0
6	118	20170925	None	8ec6ce2a7e7949b1bf142def7d0e0586	NaN
7	118	20171002	None	68617ca6246f4fbc85e91a2a49552598	NaN
8	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0
9	118	20161122	None	8974fc5686fe429db53ddde067b88302	NaN

There are some records where age or gender or income is not provided. After researching deeper on these records:

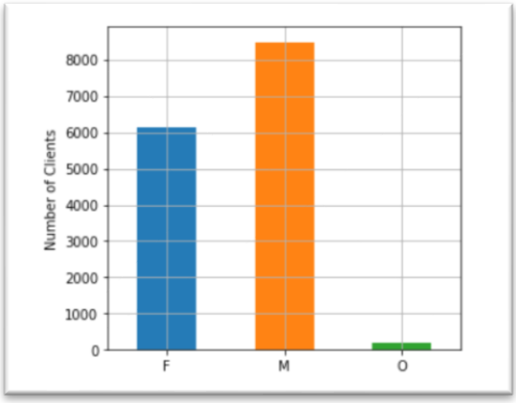
```
In [10]: print (f"Number of clients with no age informed: {profile[profile.age == 118].shape[0]}")
         print (f"Number of clients with no gender informed: {profile[profile.gender.isna()].shape[0]}")
         print (f"Number of clinets with no gender and age informed: {(profile[profile.gender.isna()])[profile.age == 118].shape[0]}")
         print (f"Number of clients with no income informed: {profile[profile.income.isna()].shape[0]}")
         print (f"Number of clients with no age+gender+income iformed: {((profile[profile.gender.isna()])[profile.income.isna()])[profile
Number of clients with no age informed: 2175
Number of clients with no gender informed: 2175
Number of clinets with no gender and age informed: 2175
Number of clients with no income informed: 2175
Number of clients with no age+gender+income iformed: 2175
```

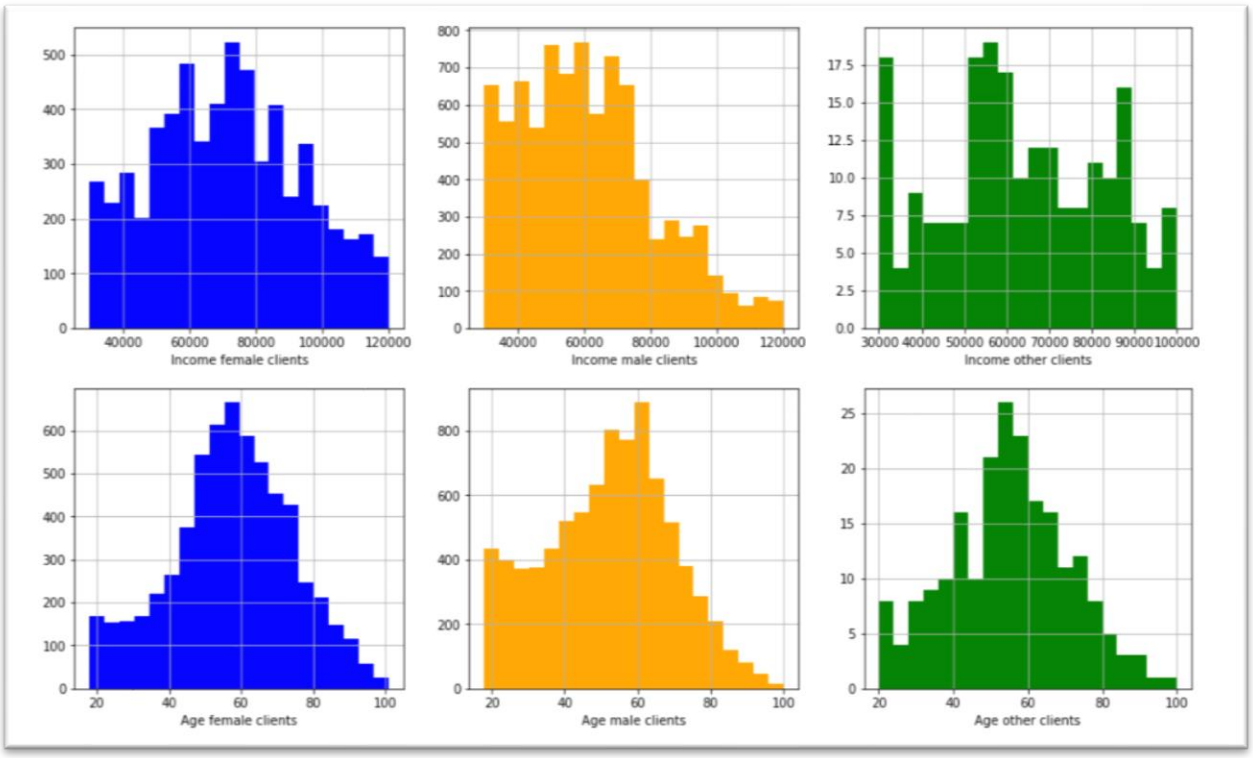
We can see that those records are all the same and represent only a 12,8% of the dataset.

Therefore, I will advance my first decision regarding data transformation that will be to dispose those clients with missing age, income and gender. This will reduce the data set from 17000 to 14825 rows, which is still a big amount of data.

Analyzing further the different features of the dataset:

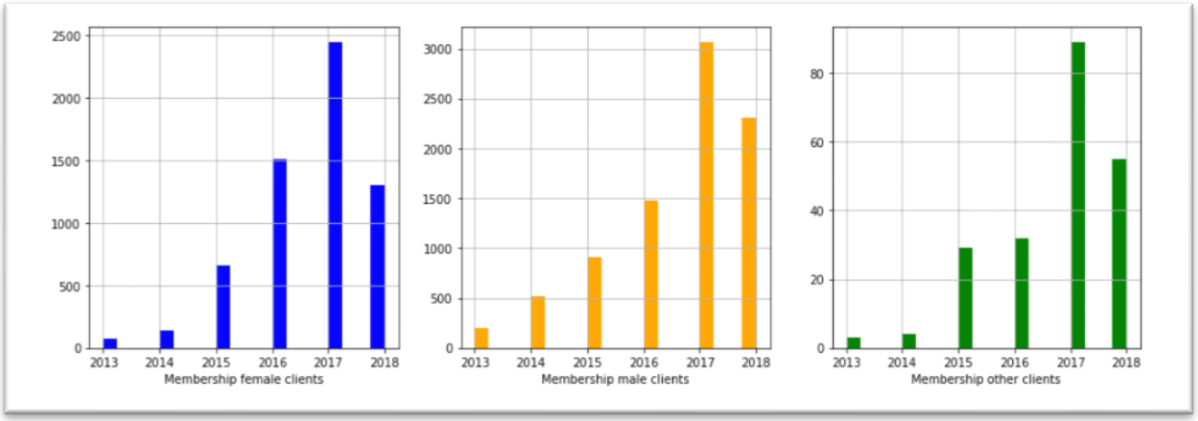
```
Number of female clients: 6129 (41.34 %)
Number of male clients: 8484 (57.23 %)
Number of other clients: 212 (1.43 %)
```





The distribution of male and female population in the age and income features are very similar. The same with the "other" population and the income feature (the age feature distribution is a little bit different but probably because of the small amount of "other" population).

Remarkable different could be here that there are more male clients than female.



The distribution of membership year is more or less the same for male and female clients, except for the last year where the decrease of new memberships is considerably bigger for female clients.

Transcript

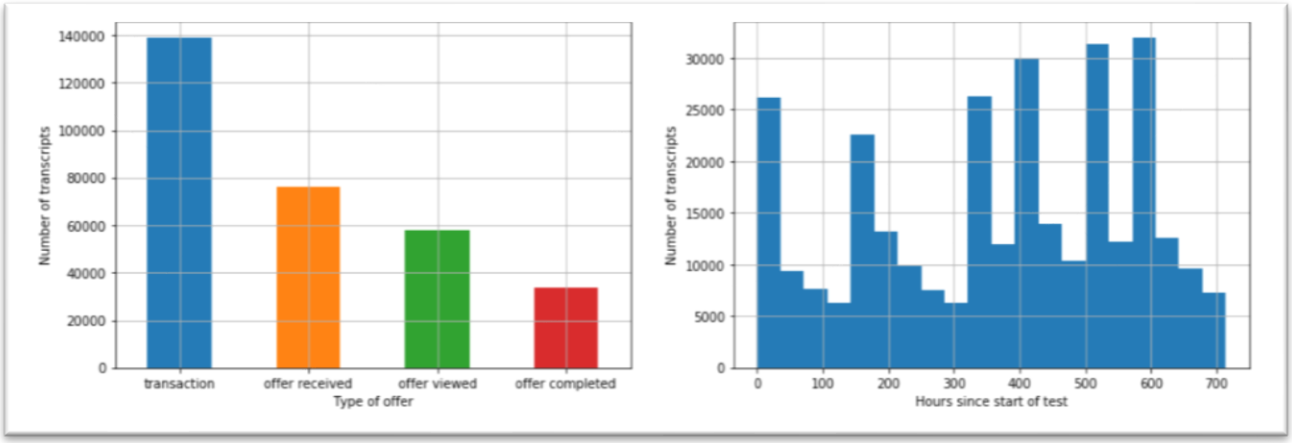
It contains the records of all events during this 30-day test period (306534 events x 4 fields):

- 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

```
In [18]: transcript.head()
```

Out[18]:

	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'}



An interesting result of the analysis above is that less than the half of the offers received are completed. That confirms the necessity for a personalization in the offers and, more important, in the communication of the offers.

Algorithms and Techniques

As already discussed in this document, there are two characteristics of Starbucks project to take into account:

1. The desired result is a “True” or “False” value (to the question of whether a client is likely to take an offer). That is, we are facing a **classification problem**.
2. We count with labeled examples (pairs of: <feature values ;True/False result>) to train and test our prediction models. That is, we are dealing with **supervised learning**.

In machine learning, support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other.

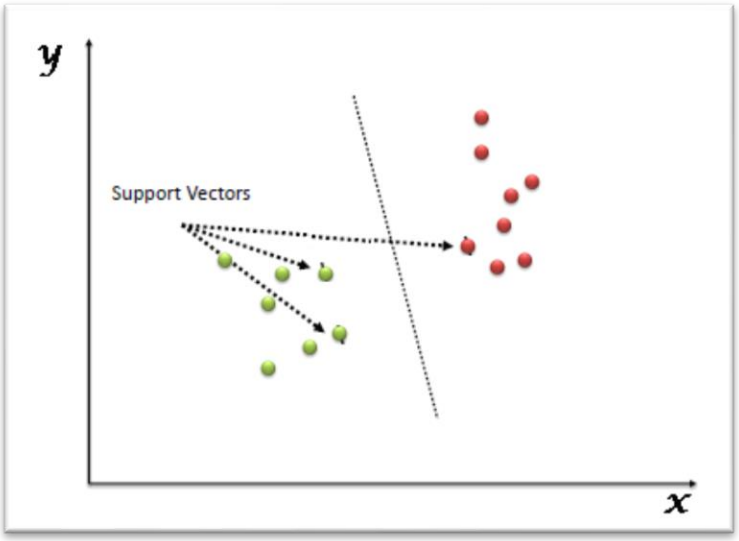
Scikit-learn is a free software machine learning library for the Python programming language. It features various classification and regression algorithms including support vector machines. One of the SVM algorithms offered is “C-Support Vector Classification”.

In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" with strong independence assumptions between the features. They are among the simplest Bayesian network models.

Once again, Scikit-learn provides several implementation of naïve Bayes classifiers. One of them is the "Gaussian Naive Bayes".

C-Support Vector Classification

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



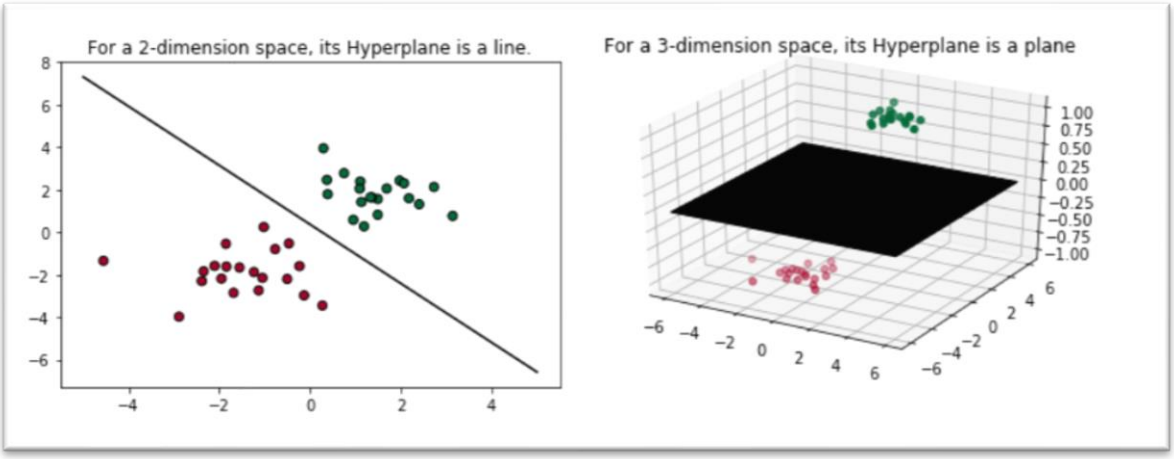
Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

How Support Vector Classifier works?

SVM needs to find the optimal line with the constraint of correctly classifying either class:

- Follow the constraint: only look into the separate hyperplanes (e.g. separate lines), hyperplanes that classify classes correctly
- Conduct optimization: pick up the one that maximizes the margin

What is a Hyperplane?



Hyperplane is an (n minus 1)-dimensional subspace for an n-dimensional space. For a 2-dimension space, its hyperplane will be 1-dimension, which is just a line. For a 3-dimension space, its hyperplane will be 2-dimension, which is a plane that slice the cube.

In the linearly separable case, SVM is trying to find the hyperplane that maximizes the margin, with the condition that both classes are classified correctly. But in real cases, datasets are probably never linearly separable, so the condition of 100% correctly classified by a hyperplane will never be met.

SVM address non-linearly separable cases by introducing two concepts: Soft Margin and Kernel Tricks.

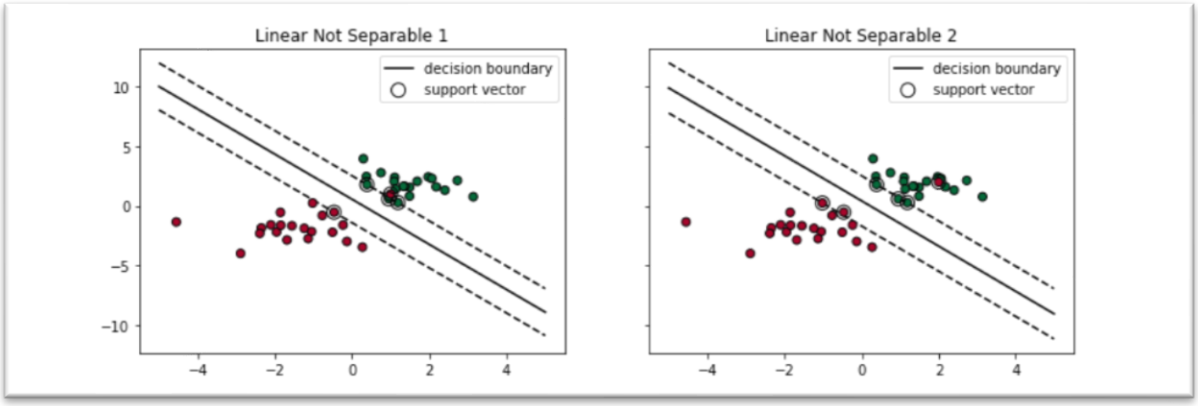
Soft Margin: try to find a line to separate, but tolerate one or few misclassified dots (e.g. the dots circled in red)

Kernel Trick: try to find a non-linear decision boundary

Soft Margin

Two types of misclassifications are tolerated by SVM under soft margin:

1. The dot is on the wrong side of the decision boundary but on the correct side/ on the margin (shown in left)
2. The dot is on the wrong side of the decision boundary and on the wrong side of the margin (shown in right)



Applying Soft Margin, SVM tolerates a few dots to get misclassified and tries to balance the trade-off between finding a line that maximizes the margin and minimizes the misclassification.

Kernel Trick

What Kernel Trick does is it utilizes existing features, applies some transformations, and creates new features. Those new features are the key for SVM to find the nonlinear decision boundary.

Pros:

- It is really effective in the higher dimension.
- Effective when the number of features are more than training examples.
- Best algorithm when classes are separable
- The hyperplane is affected by only the support vectors thus outliers have less impact.
- SVM is suited for extreme case binary classification.

Cons:

- For larger dataset, it requires a large amount of time to process.
- Does not perform well in case of overlapped classes.
- Selecting, appropriately hyperparameters of the SVM that will allow for sufficient generalization performance.
- Selecting the appropriate kernel function can be tricky.

Scikit-learn implementation of Support Vector Classifier:

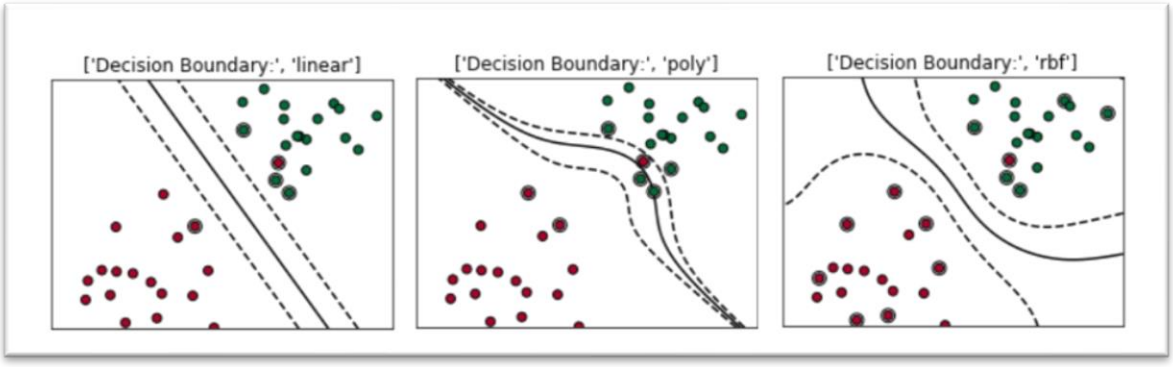
Scikit provides an implementation of SVC with the following main hyper-parameters:

C (hyper-parameter for the implementation of the Soft Margin)

How much tolerance(soft) we want to give when finding the decision boundary is an important hyper-parameter for the SVM (both linear and nonlinear solutions). In Scikit-learn, it is represented as the penalty term — 'C'. The bigger the C, the more penalty SVM gets when it makes misclassification. Therefore, the narrower the margin is and fewer support vectors the decision boundary will depend on.

Kernel (hyper-prameter for the implementation of the Kernel Trick)

In Sklearn — svm.SVC(), we can choose 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable as our kernel/transformation.

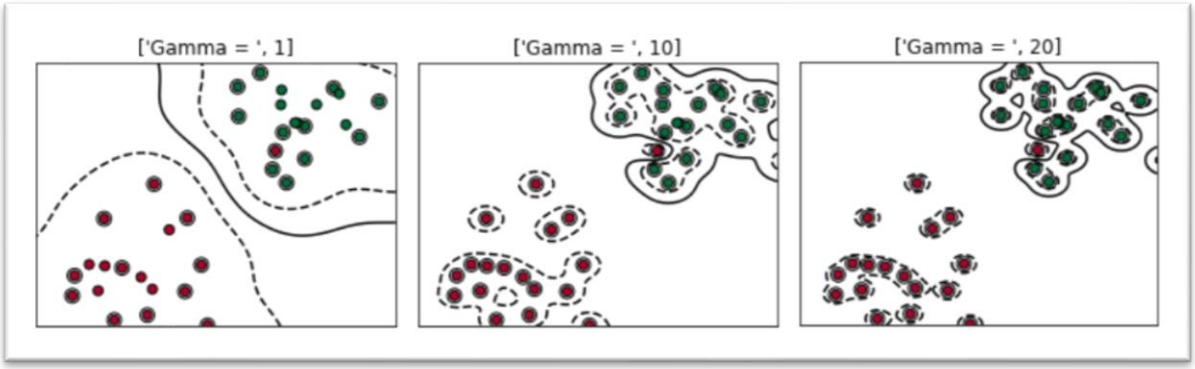


Gamma (hyper-parameter for the implementation of the Kernel Trick)

Think of the Radial Basis Function kernel ('rbf' kernel) as a transformer/processor to generate new features by measuring the distance between all other dots to a specific dot/dots — centers. The most popular/basic RBF kernel is the Gaussian Radial Basis Function:

$$\phi(x, center) = exp(-\gamma \|x - center\|^2)$$

gamma (γ) controls the influence of new features — Φ(x, center) on the decision boundary. The higher the gamma, the more influence of the features will have on the decision boundary, more wiggling the boundary will be.



Gaussian Naïve Bayes Algorithm

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution.

Gaussian Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Look at the equation below:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

P(h): the probability of hypothesis h being true (regardless of the data). This is known as the prior probability of h.

P(D): the probability of the data (regardless of the hypothesis). This is known as the prior probability.

P(h|D): the probability of hypothesis h given the data D. This is known as posterior probability.

P(D|h): the probability of data d given that the hypothesis h was true. This is known as posterior probability.

How Naive Bayes classifier works?

Naive Bayes classifier calculates the probability of an event in the following steps:

- Step 1: Calculate the prior probability for given class labels
- Step 2: Find Likelihood probability with each attribute for each class
- Step 3: Put these value in Bayes Formula and calculate posterior probability.
- Step 4: See which class has a higher probability, given the input belongs to the higher probability class.

Pros:

- It is easy and fast to predict class of test data set. It also performs well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Scikit-learn implementation of Gaussian Naïve Bayes:

The Scikit-learn implementation of this algorithm is very straightforward to use because it has only one parameter:

priors: ***array-like, shape (n_classes,)***
Prior probabilities of the classes. If specified, the priors are not adjusted according to the data.

It is highly recommended to let this parameter in its default value ("None") in order to adjust the prior probabilities according to the data.

Benchmark

80% of the model data will be used to train and the other 20% to evaluate the performance of the prediction model.

As one of the products from this project, an evaluation function will be developed to compare predictions over the customers in the test dataset with the real result to provide statistics about the accuracy and F1-score of a prediction model.

In order to establish a benchmark, we will use the “pre Machine Learning” situation as a baseline.

What is this preML situation? Well, before introducing ML, Starbucks wasn’t personalizing the offer communication. Therefore, we can assume Starbucks was considering that all the customers tend to take the offer.

Hence, we will create a dummy predictor that will return always a “true” propensity to take an offer.

The accuracy and f1-score of this dummy predictor will be our benchmark.

III. Methodology

Data Preprocessing

Profile Table

As advanced, first preprocessing step will be to discard those entries in Profile dataset that don’t have age, gender & income values:

```
my_profile = profile[profile.age < 118]
my_profile = my_profile.reset_index(drop=True)
my_profile.head()
```

	age	became_member_on	gender	id	income
0	55	20170715	F	0610b486422d4921ae7d2bf64640c50b	112000.0
1	75	20170509	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
2	68	20180426	M	e2127556f4f64592b11af22de27a7932	70000.0
3	65	20180209	M	389bc3fa690240e798340f5a15918d5c	53000.0
4	58	20171111	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0

I will now transform the 'became_member_on' field in a date format I can work with:

```
my_profile.became_member_on = pd.to_datetime(my_profile.became_member_on, format = '%Y%m%d')
my_profile.head()
```

	age	became_member_on	gender	id	income
0	55	2017-07-15	F	0610b486422d4921ae7d2bf64640c50b	112000.0
1	75	2017-05-09	F	78afa995795e4d85b5d9ceeca43f5fef	100000.0
2	68	2018-04-26	M	e2127556f4f64592b11af22de27a7932	70000.0
3	65	2018-02-09	M	389bc3fa690240e798340f5a15918d5c	53000.0
4	58	2017-11-11	M	2eeac8d8feae4a8cad5a6af0499a211d	51000.0

Transcript Table

We can see the value field contains different information depending of the type of event:

```
tr_tran=transcript[transcript.event == 'transaction']
tr_tran.head()
```

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

```
tr_rcvd=transcript[transcript.event == 'offer received']
tr_rcvd.head()
```

	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'}

The content of value has to be taken out and transformed in new columns of the table so they are better formatted to work with:

Moreover, for the shake of easier manipulation, the values of “event” are transformed in binary columns (that is, True-False columns):

```
my_transcript = pd.get_dummies(my_transcript , columns=['event'] , prefix='' ,prefix_sep='')
my_transcript.head()
```

	person	time	amount	offer_id	reward	offer completed	offer received	offer viewed	transaction
0	02c083884c7d45b39cc68e1314fec56c	0	0.83	NaN	NaN	0	0	0	1
1	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	0	34.56	NaN	NaN	0	0	0	1
2	54890f68699049c2a04d415abc25e717	0	13.23	NaN	NaN	0	0	0	1
3	b2f1cd155b864803ad8334cdf13c4bd2	0	19.51	NaN	NaN	0	0	0	1
4	fe97aa22dd3e48c8b143116a8403dd52	0	18.97	NaN	NaN	0	0	0	1

Merging...

...the three tables to have a data frame with the whole information available to work with.

	person	time	amount	offer_id	offer completed	offer received	offer viewed	transaction	age	became_member_on	gender	income	difficulty	d
0	02c083884c7d45b39cc68e1314fec56c	0	0.83	NaN	0	0	0	1	20	2016-07-11	F	30000.0	NaN	
1	02c083884c7d45b39cc68e1314fec56c	6	1.44	NaN	0	0	0	1	20	2016-07-11	F	30000.0	NaN	
2	02c083884c7d45b39cc68e1314fec56c	12	4.56	NaN	0	0	0	1	20	2016-07-11	F	30000.0	NaN	
3	02c083884c7d45b39cc68e1314fec56c	84	1.53	NaN	0	0	0	1	20	2016-07-11	F	30000.0	NaN	
4	02c083884c7d45b39cc68e1314fec56c	90	0.50	NaN	0	0	0	1	20	2016-07-11	F	30000.0	NaN	

As a result: a table with 20 fields and 272762 entries.

Preparing the data for the prediction model

As already stated, we are trying to predict if a client tends to take(complete) an offer (bogo or discount). Therefore, only those entries related to offers will be considered. Therefore, the transaction events and the informational offers will be discarded.

Moreover, as deciding features in the model data for the prediction machine I will choose: 'gender', 'age', 'income' and 'year_of_membership'.

There are labels of two types: 'offer_bogo' and 'offer_discount'.

As a result of the decisions taken above, we will have the following table, where the records will be grouped by every person and every type of offer:

```
model_data = my_merge_offer.groupby(['person' , 'offer_type']).agg({'offer completed':sum,
                                                                    'offer received': sum,
                                                                    'age':'last',
                                                                    'became_member_on':'last',
                                                                    'gender':'last',
                                                                    'income':'last'
                                                                    })
#model_data = model_data[my_model_data.offer_type != 'informational']
model_data = model_data.reset_index()
model_data.head(10)
```

	person	offer_type	offer completed	offer received	age	became_member_on	gender	income
0	0009655768c64bdeb2e877511632db8f	bogo	1	1	33	2017-04-21	M	72000.0
1	0009655768c64bdeb2e877511632db8f	discount	2	2	33	2017-04-21	M	72000.0
2	0011e0d4e6b944f998e987f904e8c1e5	bogo	1	1	40	2018-01-09	O	57000.0
3	0011e0d4e6b944f998e987f904e8c1e5	discount	2	2	40	2018-01-09	O	57000.0
4	0020c2b971eb4e9188eac86d93036a77	bogo	1	2	59	2016-03-04	F	90000.0
5	0020c2b971eb4e9188eac86d93036a77	discount	2	2	59	2016-03-04	F	90000.0
6	0020ccbbb6d84e358d3414a3ff76cffd	bogo	2	2	24	2016-11-11	F	60000.0
7	0020ccbbb6d84e358d3414a3ff76cffd	discount	1	1	24	2016-11-11	F	60000.0
8	003d66b6608740288d6cc97a6903f4f0	discount	3	3	26	2017-06-21	F	73000.0
9	00426fe3ffde4c6b9cb9ad6d077a13ea	discount	1	4	19	2016-08-09	F	65000.0

This table needs still a little bit of processing: take out “year_of_membership” from “became_member_on” and transform values of “gender” into binary columns:

```
my_model_data = model_data.copy()
my_model_data = pd.get_dummies(my_model_data , columns=['gender'])
my_model_data['year_of_membership'] = my_model_data.became_member_on.dt.year
my_model_data.drop(columns=['became_member_on'] , inplace=True)
my_model_data.head(10)
```

	person	offer_type	offer completed	offer received	age	income	gender_F	gender_M	gender_O	year_of_membership
0	0009655768c64bdeb2e877511632db8f	bogo	1	1	33	72000.0	0	1	0	2017
1	0009655768c64bdeb2e877511632db8f	discount	2	2	33	72000.0	0	1	0	2017
2	0011e0d4e6b944f998e987f904e8c1e5	bogo	1	1	40	57000.0	0	0	1	2018
3	0011e0d4e6b944f998e987f904e8c1e5	discount	2	2	40	57000.0	0	0	1	2018
4	0020c2b971eb4e9188eac86d93036a77	bogo	1	2	59	90000.0	1	0	0	2016
5	0020c2b971eb4e9188eac86d93036a77	discount	2	2	59	90000.0	1	0	0	2016
6	0020ccbbb6d84e358d3414a3ff76cffd	bogo	2	2	24	60000.0	1	0	0	2016
7	0020ccbbb6d84e358d3414a3ff76cffd	discount	1	1	24	60000.0	1	0	0	2016
8	003d66b6608740288d6cc97a6903f4f0	discount	3	3	26	73000.0	1	0	0	2017
9	00426fe3ffde4c6b9cb9ad6d077a13ea	discount	1	4	19	65000.0	1	0	0	2016

Now, it is time to split the model data into two different: the one to predict if bogo offer will be likely to be taken and the one to predict if discount offer will be likely to be taken.

This is needed also because, due to the fact that the events are independent (to take a bogo offer and to take a discount offer), we shouldn't use the data that shows the behavior towards a discount offer to train and test a machine learning model deployed to predict behavior towards a bogo offer and vice-versa.

Now, it is time to decide the condition under which we consider that the customer has a disposition to take an offer.

From the analysis of the data, we have seen that customers have received 3 offers of a type at the most, and completed 0, 1 or 2.

Even when 1 out of 4 seems to be a low rate, Starbucks wants to avoid excessive communication overload, but surely not to the cost of preventing this customer from accessing to this offer.

Hence, we will consider that a customer has a disposition to take an offer when he/she has completed at least a 25% of the offers received.

This applies to both types of offers:

```
def fill_taken (df):
    for i in df.index:
        if (df.at[i,'offer completed'] >= (df.at[i,'offer received']*0.25)):
            df.at[i,'offer_taken'] = 1
        else:
            df.at[i,'offer_taken'] = 0
    return df

my_md_bogo = fill_taken(my_md_bogo)
my_md_disc = fill_taken(my_md_disc)
```

As we can see...

```
print (my_model_data.shape)
print (my_md_bogo.shape)
print (my_md_disc.shape)
```

```
(26124, 10)
(13082, 11)
(13042, 11)
```

... the dataframes for bogo offers (13082 registries) and discount offers (13042) are similarly and pretty well populated.

We now need to split the dataframes into two: one with the features (characteristics of the customers) and other with the labels (True=1/False=0, propensity to take the offer).

```
my_md_bogo = my_md_bogo[['age', 'income', 'gender_F', 'gender_M', 'gender_O', 'year_of_membership', 'offer_taken']]
my_md_disc = my_md_disc[['age', 'income', 'gender_F', 'gender_M', 'gender_O' , 'year_of_membership', 'offer_taken']]
ybogo = my_md_bogo.offer_taken
Xbogo = my_md_bogo.drop(columns=['offer_taken'])
ydisc = my_md_disc.offer_taken
Xdisc = my_md_disc.drop(columns=['offer_taken'])
```

And then again, every couple of dataframes will be split in two sets: one with the 80% of the registries to train the prediction model and another one with the rest of the registries to test the model.

```
Xbogo_train, Xbogo_test, ybogo_train, ybogo_test = train_test_split(Xbogo, ybogo, test_size=0.2, random_state=0)
Xdisc_train, Xdisc_test, ydisc_train, ydisc_test = train_test_split(Xdisc, ydisc, test_size=0.2, random_state=0)
```

Implementation

As already stated, I will use a couple of prediction models and calculate accuracy scores in two ways:

1. using the available “score” method of the Scikit-learn model library.
2. calculating myself the % of success, false positives and false negatives (with the developed "evaluate_model" function)

Then, my evaluation function will provide the Precision, Recall and F1-score as well.

This will help to compare suitability of any prediction models, not only the two used as examples.

```
def evaluate_model (mod, X_test, y_test):
    model_success = 0
    model_fail = 0
    model_true_positive = 0
    model_true_negative = 0
    model_false_positive = 0
    model_false_negative = 0
    model_accuracy = 0
    model_precision = 0
    model_recall = 0
    model_f1score = 0

    model_total = X_test.shape[0]

    for i in range(model_total):
        pr = mod.predict(X_test[i:i+1])[0]
        val = y_test.values[i]
        if (pr == val):
            if (val == 1):
                model_true_positive += 1
            else:
                model_true_negative += 1
        else:
            if (val == 0):
                model_false_positive += 1
            else:
                model_false_negative += 1

    model_success = model_true_positive + model_true_negative
    model_fail = model_false_positive + model_false_negative

    model_accuracy = (model_success / model_total)
    model_precision = model_true_positive / (model_true_positive + model_false_positive)
    model_recall = model_true_positive / (model_true_positive + model_false_negative)
    model_f1score = (2 * model_precision * model_recall) / (model_precision + model_recall)

    print (f'Number of predictions: {model_total}')
    print (f'Number of true positives (TP): {model_true_positive} ({(model_true_positive/model_total)*100} %)')
    print (f'Number of true negatives (TN): {model_true_negative} ({(model_true_negative/model_total)*100} %)')
    print (f'Number of false positives (FP): {model_false_positive} ({(model_false_positive/model_total)*100} %)')
    print (f'Number of false negatives (FN): {model_false_negative} ({(model_false_negative/model_total)*100} %)')
    print (f'Number of failed predictions: {model_fail}')
    print (f'Number of successful predictions: {model_success}')
    print ('================================================================')
    print (f'ACCURACY  (TP+TN)/(TP+TN+FP+FN): {model_accuracy} ({(model_accuracy)*100} %)')
    print ('----------------------------------------------------------------')
    print (f'PRECISION  (TP)/(TP+FP): {model_precision}')
    print (f'RECALL  (TP)/(TP+FN): {model_recall}')
    print (f'F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): {model_f1score}')
    print ('================================================================')

    return
```

Benchmark (dummy prediction model):

We implement the dummy predictor so it always returns a “true” propensity to take an offer:

```
class AlwaysTrueModel:
    def predict(self,*args):
        return [1]
```

This dummy predictor doesn't have any "score" function, so we evaluate this model using the developed function:

```
print ('Evaluation of prediction of bogo offer propensity:')
print ('')
evaluate_model (my_ATModel, Xbogo_test, ybogo_test)
```

```
Evaluation of prediction of bogo offer propensity:

Number of predictions: 2617
Number of true positives (TP): 1753 (66.98509743981658 %)
Number of true negatives (TN): 0 (0.0 %)
Number of false positives (FP): 864 (33.014902560183415 %)
Number of false negatives (FN): 0 (0.0 %)
Number of failed predictions: 864
Number of successful predictions: 1753
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.6698509743981659 (66.98509743981658 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.6698509743981659
RECALL  (TP)/(TP+FN): 1.0
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.8022883295194507
================================================================
```

```
print ('Evaluation of prediction of discount offer propensity:')
print ('')
evaluate_model (my_ATModel, Xdisc_test, ydisc_test)
```

```
Evaluation of prediction of discount offer propensity:

Number of predictions: 2609
Number of true positives (TP): 1919 (73.55308547336145 %)
Number of true negatives (TN): 0 (0.0 %)
Number of false positives (FP): 690 (26.446914526638558 %)
Number of false negatives (FN): 0 (0.0 %)
Number of failed predictions: 690
Number of successful predictions: 1919
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.7355308547336145 (73.55308547336145 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.7355308547336145
RECALL  (TP)/(TP+FN): 1.0
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.8476148409893994
================================================================
```

Gaussian Naive Bayes model:

We instantiate the model for every type of offer and train them:

```
pred_model1_bogo = GaussianNB()
pred_model1_disc = GaussianNB()
pred_model1_bogo.fit(Xbogo_train, ybogo_train)
pred_model1_disc.fit(Xdisc_train, ydisc_train)
```

```
GaussianNB(priors=None)
```

Then, we evaluate the model.

- First, with the use of the built-in "score" method:

```
print(f'Prediction of bogo offer propensity: score = {pred_model1_bogo.score(Xbogo_test,ybogo_test)}')
print(f'Prediction of discount offer propensity: score = {pred_model1_disc.score(Xdisc_test,ydisc_test)}')
```

```
Prediction of bogo offer propensity: score = 0.7050057317539167
Prediction of discount offer propensity: score = 0.7447297815254887
```

- Then, with the use of our evaluation function:

```
print ('Evaluation of prediction of bogo offer propensity:')
print ('')
evaluate_model (pred_model1_bogo, Xbogo_test, ybogo_test)
```

```
Evaluation of prediction of bogo offer propensity:

Number of predictions: 2617
Number of true positives (TP): 1524 (58.23461979365686 %)
Number of true negatives (TN): 321 (12.26595338173481 %)
Number of false positives (FP): 543 (20.748949178448605 %)
Number of false negatives (FN): 229 (8.750477646159725 %)
Number of failed predictions: 772
Number of successful predictions: 1845
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.7050057317539167 (70.50057317539166 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.737300435413643
RECALL  (TP)/(TP+FN): 0.8693667997718197
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.7979057591623037
================================================================
```

```
print ('Evaluation of prediction of discount offer propensity:')
print ('')
evaluate_model (pred_model1_disc, Xdisc_test, ydisc_test)
```

```
Evaluation of prediction of discount offer propensity:

Number of predictions: 2609
Number of true positives (TP): 1813 (69.49022614028362 %)
Number of true negatives (TN): 130 (4.9827520122652365 %)
Number of false positives (FP): 560 (21.46416251437332 %)
Number of false negatives (FN): 106 (4.0628593330778076 %)
Number of failed predictions: 666
Number of successful predictions: 1943
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.7447297815254887 (74.47297815254888 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.7640117994100295
RECALL  (TP)/(TP+FN): 0.9447628973423658
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.8448275862068966
================================================================
```

C-Support Vector Classification model:

We instantiate the model for every type of offer and train them:

```
pred_model2_bogo = SVC()
pred_model2_disc = SVC()
pred_model2_bogo.fit(Xbogo_train, ybogo_train)
pred_model2_disc.fit(Xdisc_train, ydisc_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

Then, we evaluate the model.

- First, with the use of the built-in “score” method:

```
print(f'Prediction of bogo offer propensity: score = {pred_model2_bogo.score(Xbogo_test,ybogo_test)}')
print(f'Prediction of discount offer propensity: score = {pred_model2_disc.score(Xdisc_test,ydisc_test)}')
```

```
Prediction of bogo offer propensity: score = 0.6683225066870463
Prediction of discount offer propensity: score = 0.729781525488693
```

- Then, with the use of our evaluation function:

```
print ('Evaluation of prediction of bogo offer propensity:')
print ('')
evaluate_model (pred_model2_bogo, Xbogo_test, ybogo_test)
```

```
Evaluation of prediction of bogo offer propensity:

Number of predictions: 2617
Number of true positives (TP): 1533 (58.57852502865877 %)
Number of true negatives (TN): 216 (8.253725640045854 %)
Number of false positives (FP): 648 (24.76117692013756 %)
Number of false negatives (FN): 220 (8.406572411157814 %)
Number of failed predictions: 868
Number of successful predictions: 1749
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.6683225066870463 (66.83225066870463 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.7028885832187071
RECALL  (TP)/(TP+FN): 0.874500855675984
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.7793594306049824
================================================================
```

```
print ('Evaluation of prediction of discount offer propensity:')
print ('')
evaluate_model (pred_model2_disc, Xdisc_test, ydisc_test)
```

```
Evaluation of prediction of discount offer propensity:

Number of predictions: 2609
Number of true positives (TP): 1803 (69.10693752395554 %)
Number of true negatives (TN): 101 (3.8712150249137602 %)
Number of false positives (FP): 589 (22.5756995017248 %)
Number of false negatives (FN): 116 (4.446147949405903 %)
Number of failed predictions: 705
Number of successful predictions: 1904
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.729781525488693 (72.9781525488693 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.7537625418060201
RECALL  (TP)/(TP+FN): 0.9395518499218343
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.836464857341684
================================================================
```

Refinement

As explained before while describing the GaussianNB model, this predictor model doesn't allow refinement through hyper-parameter manipulation. This could only be achieved by increasing the model dataframes.

For the SVC model, we have three hyper-parameters: **Kernel**, **C** and **Gamma**.

Kernel:

It has been impossible to work with this hyper-parameter due to the limited computational power of the jupyter running environment. Everytime I tried to set it to another value, the thread got stuck.

Anyway, the default value (Radial basis function kernel (RBF)/ Gaussian Kernel) is a very good choice attending to the experts' reports about SVC model.

C:

Different values of C were tested:

```
for i in range(3,11):
    iC = i/10
    pred_model3_bogo = SVC(C=iC)
    pred_model3_disc = SVC(C=iC)
    pred_model3_bogo.fit(Xbogo_train, ybogo_train)
    pred_model3_disc.fit(Xdisc_train, ydisc_train)
    print (f'C = {iC}')
    print(f'Prediction of bogo offer propensity: score = {pred_model3_bogo.score(Xbogo_test,ybogo_test)}')
    print(f'Prediction of discount offer propensity: score = {pred_model3_disc.score(Xdisc_test,ydisc_test)}')
```

```
C = 0.3
Prediction of bogo offer propensity: score = 0.6698509743981659
Prediction of discount offer propensity: score = 0.733997700268302
C = 0.4
Prediction of bogo offer propensity: score = 0.6713794421092855
Prediction of discount offer propensity: score = 0.7347642775009582
C = 0.5
Prediction of bogo offer propensity: score = 0.6687046236148262
Prediction of discount offer propensity: score = 0.733997700268302
C = 0.6
Prediction of bogo offer propensity: score = 0.6690867405426061
Prediction of discount offer propensity: score = 0.7401303181295515
C = 0.7
Prediction of bogo offer propensity: score = 0.6675582728314864
Prediction of discount offer propensity: score = 0.7385971636642392
C = 0.8
Prediction of bogo offer propensity: score = 0.669468857470386
Prediction of discount offer propensity: score = 0.7351475661172863
C = 0.9
Prediction of bogo offer propensity: score = 0.6702330913259458
Prediction of discount offer propensity: score = 0.7324645458029897
C = 1.0
Prediction of bogo offer propensity: score = 0.6683225066870463
Prediction of discount offer propensity: score = 0.729781525488693
```

We got a better score for C=0.6

Gamma:

Different values of Gamma were tested:

```python
for i in [0.01, 0.1, 0.3, 0.5, 0.8, 1, 10, 100]:
    pred_model3_bogo = SVC(C=0.6,gamma=i)
    pred_model3_disc = SVC(C=0.6,gamma=i)
    pred_model3_bogo.fit(Xbogo_train, ybogo_train)
    pred_model3_disc.fit(Xdisc_train, ydisc_train)
    print(f'gamma = {i}')
    print(f'Prediction of bogo offer propensity: score = {pred_model3_bogo.score(Xbogo_test,ybogo_test)}')
    print(f'Prediction of discount offer propensity: score = {pred_model3_disc.score(Xdisc_test,ydisc_test)}')
```

```
gamma = 0.01
Prediction of bogo offer propensity: score = 0.6667940389759266
Prediction of discount offer propensity: score = 0.7316979685703334
gamma = 0.1
Prediction of bogo offer propensity: score = 0.6641192204814673
Prediction of discount offer propensity: score = 0.7343809888846301
gamma = 0.3
Prediction of bogo offer propensity: score = 0.6717615590370654
Prediction of discount offer propensity: score = 0.7393637408968954
gamma = 0.5
Prediction of bogo offer propensity: score = 0.672907909820405
Prediction of discount offer propensity: score = 0.7428133384438482
gamma = 0.8
Prediction of bogo offer propensity: score = 0.6725257928926252
Prediction of discount offer propensity: score = 0.7393637408968954
gamma = 1
Prediction of bogo offer propensity: score = 0.6713794421092855
Prediction of discount offer propensity: score = 0.7389804522805673
gamma = 10
Prediction of bogo offer propensity: score = 0.6713794421092855
Prediction of discount offer propensity: score = 0.7370640091989268
gamma = 100
Prediction of bogo offer propensity: score = 0.6713794421092855
Prediction of discount offer propensity: score = 0.7370640091989268
```

We got a better score for Gamma=0.5.

Therefore, the final evaluation for SVC is the following:

```
pred_model3_bogo = SVC(C=0.6, gamma=0.5)
pred_model3_disc = SVC(C=0.6, gamma=0.5)
pred_model3_bogo.fit(Xbogo_train, ybogo_train)
pred_model3_disc.fit(Xdisc_train, ydisc_train)
print ('Evaluation of prediction of bogo offer propensity:')
print ('')
evaluate_model (pred_model3_bogo, Xbogo_test, ybogo_test)
print ('-----------------------------------------------------')
print ('Evaluation of prediction of discount offer propensity:')
print ('')
evaluate_model (pred_model3_disc, Xdisc_test, ydisc_test)
```

```
Evaluation of prediction of bogo offer propensity:

Number of predictions: 2617
Number of true positives (TP): 1687 (64.46312571646924 %)
Number of true negatives (TN): 74 (2.8276652655712646 %)
Number of false positives (FP): 790 (30.187237294612153 %)
Number of false negatives (FN): 66 (2.5219717233473444 %)
Number of failed predictions: 856
Number of successful predictions: 1761
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.672907909820405 (67.2907909820405 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.6810658054097699
RECALL  (TP)/(TP+FN): 0.9623502567027952
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.7976359338061466
================================================================
----------------------------------------------------
```

```
----------------------------------------------------
Evaluation of prediction of discount offer propensity:

Number of predictions: 2609
Number of true positives (TP): 1887 (72.32656190111155 %)
Number of true negatives (TN): 51 (1.9547719432732849 %)
Number of false positives (FP): 639 (24.492142583365272 %)
Number of false negatives (FN): 32 (1.2265235722499042 %)
Number of failed predictions: 671
Number of successful predictions: 1938
================================================================
ACCURACY  (TP+TN)/(TP+TN+FP+FN): 0.7428133384438482 (74.28133384438482 %)
----------------------------------------------------------------
PRECISION  (TP)/(TP+FP): 0.7470308788598575
RECALL  (TP)/(TP+FN): 0.9833246482542991
F1-SCORE 2*PRECISION*RECALL/(PRECISION+RECALL): 0.8490438695163104
================================================================
```

IV. Results

Model Evaluation and Validation

The evaluation has been already documented in the previous chapter as we needed the results of the evaluation to perform the refinement of our models.

Comparing the evaluation of our models with the Benchmark:

For the BOGO offer:

	Benchmark	**GaussianNB**	**SVC**
Accuracy	66.99%	70.50%	67.29%
F1-Score	0.80	0.80	0.80

For the discount offer:

	Benchmark	**GaussianNB**	**SVC**
Accuracy	73.55%	74.47%	74.28%
F1-Score	0.85	0.85	0.85

Justification

Even when it was not the scope of this project, we have achieved to improve the performance of the offer communication from Starbucks by means of implementing a personalization of the offer communication through Machine Learning.

The improvement is perhaps not remarkable, but it represents a solid starting point to prove the benefits of Machine Learning and, more important (and moreover, the real scope of this project), it offers a framework to delve into the matter, trying other models, trying mixed models, etc.

V. Conclusion

Reflection

I think this project has offered a valuable overview about the Machine Learning technology and its utility to generate value from data.

This project has created a framework to apply Machine Learning technique to solve a real problem from Starbucks. This framework consists of:

- Datasets that will be used to train and test a prediction model.
- Metrics calculation functionality for evaluating/benchmarking a prediction model.
- A couple of examples of training, testing and evaluating a prediction model.

Even though it was out of the scope of this project to find the better prediction model, the logic applied to select the two examples used has brought as a consequence very good scores (both Accuracy and F1-score) during the evaluation.

Improvement

After reading the chapter IV, anyone can see how easy is to train and evaluate a prediction model.

This way, we could use the framework and material generated by this project to try further with other algorithms and find a better (or even the best) prediction model.

Another aspect in which we could go further is by exploring the use of different/more features to train the prediction model like the duration of the offer and/or the difficulty of the offer.