# Capstone Project: Starbucks use-case

## Project Overview

A company selling products aims to sell even more products, in order to increase its revenue. Various marketing strategies serve this purpose, and one of them is to send offers to customers, in order to encourage them buying more products, or more often.

No matter the strategy, that is always a good thing to understand better the purchasing behavior of your customers. And if you have enough customer and transaction data, machine learning can definitely help to discover patterns and structure into big amounts of data.

In this project, we train machine learning models encoding information about customer behavior regarding offer and personal characteristics. We then use these models to predict, based on unseen data (customer and offer characteristics) whether an offer will be completed or not. From a business point of view, such models can help to reduce the number of sent offers: indeed, models flag winning offer-customer pairs (pairs for which there is high chance of completeness) and Starbucks marketing services can then target more precisely the customers to contact.

## Problem Statement

Starbucks, the well-known coffeehouse chain, implements the offer-sending strategy through an app, on which customers can receive different types of offers. We implement here models in order to predict if an offer will be completed or not, based on offer characteristics and customer characteristics.

We went through different steps in order to solve our problem:

1. Data loading
2. Data exploration
3. Data formatting and preparation: we received raw data, that could have been used for many different use cases. Preparing the data to solve our use case specifically has been an important and heavy step in our project.
4. Modeling & hyperparameter tuning: we have trained 4 different models in order to reach the best performances we could.
5. Evaluation

The last (but not least) step of our project is model selection. Indeed, as we have trained multiple models, it is important to do a final selection, based on evaluation metrics and our business knowledge.

## Evaluation metrics

In order to evaluate our model, we will use accuracy, precision and recall.

Accuracy is widely used for evaluating binary classification tasks. As it takes into account both true positives and true negatives with equal weight, it is important to verify how balanced is our dataset before using it. In our case, our dataset is almost perfectly balanced, so using accuracy as an evaluation metric is fine.

Scikit-learn website defines recall and precision as this: “In information retrieval, precision is a measure of result relevancy, while recall is a measure of how many truly relevant results are returned.”

These evaluation measures give a clear and detailed overview of the performance of our model.

# Analysis

## Data Exploration

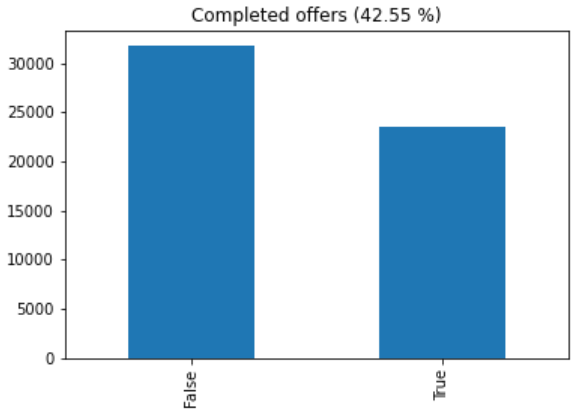
For this capstone project, 3 data sources are available: data about offers, data about people, and data about transactions. These datasets are nicely provided by Starbucks.

| | **Filename – entries** | **Fields** |
|---|---|---|
| **offers** | portfolio.json – 10 | **id** (string) - offer id<br>**offer_type** (string) - type of offer: BOGO, discount, informational<br>**difficulty** (int) - minimum required spend to complete an offer<br>**reward** (int) - reward given for completing an offer<br>**duration** (int) - time for offer to be open, in days<br>**channels** (list of strings) |
| **demographics** | profile.json – 17.000 | **age** (int) - age of the customer<br>**became_member_on** (int) - date when customer created an app account<br>**gender** (str) - gender of the customer<br>**id** (str) - customer id<br>**income** (float) - customer's income |
| **transactions** | transcript.json – 306.534 | **event** (str) - record description (ie. transaction, offer received, offer viewed, etc.)<br>**person** (str) - customer id<br>**time** (int) - time in hours since start of test. The data begins at time t=0<br>**value** (dict of strings) - either an offer id or transaction amount depending on the record |

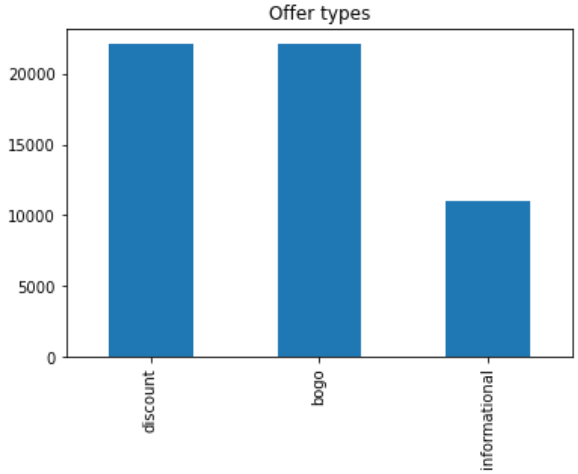
Each file is loaded into a Pandas Dataframe. For each file, different preprocessing steps are applied (see below).

## Exploratory Visualisation

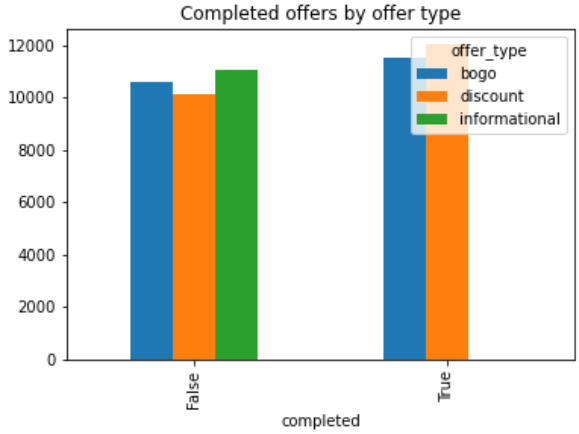
Plotting some visualisations allows us to understand better the data we are dealing with.



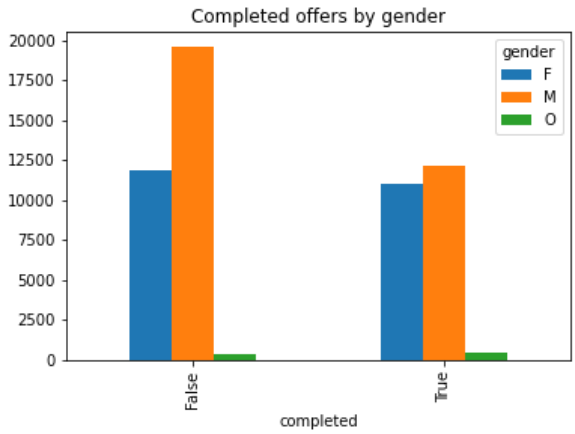

As shown on this plot, proportions of completed and uncompleted offers are almost equal. We can then consider our dataset as balanced.



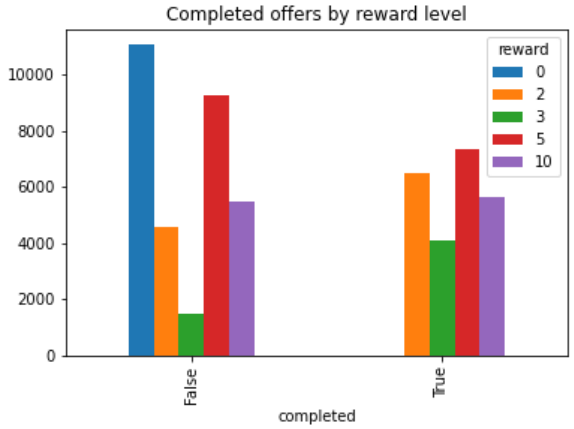

Discount and bogo offers are similarly distributed, while informational offers are a bit less represented in our dataset. This has no impact on the way we model, because we keep the 3 offer types in our dataset.




Based on this plot, we can see that an informational offer can not be completed. That is interesting to know, and that will be interesting to see if our models encode this information. Given that bogo and discount offers are more frequently completed than not completed, we can also deduce that informational offers are responsible for the imbalance between completed offers and not completed offers.




Starbucks has more male customers, but they are proportionally less responsive to offers.




Informational offers give no reward (0). Offers with lower reward (2-3) are proportionally more completed, probably because reward is correlated with difficulty. This is the opposite for offers with a reward of 5: our hypothesis is that the required effort to complete the offer is too high regarding the reward. For offers with high reward (10), proportion of completed VS not completed offers is similar.

## Algorithms and Techniques

### Train, validation and test sets

While the LinearLearner does not require a validation set, it is preferable to have one for training a XGBoost model: indeed, the validation set is used to tune and validate the hyperparameters of the model.

In order to use the same train set for both models, we split our full dataset into a train set (60% data; 33K rows), a validation set (20% data; 11K rows) and a test set (20% data; 11K rows) from the beginning.

These 3 datasets are stored on S3.

### Benchmark

Our problem is a classification problem. We will then use the LinearLearner algorithm as a benchmark model. This model is proposed by SageMaker, and able to solve regression and classification tasks.

### XGBoost model

We will also compare this model with an XGBoost model, also available on Amazon SageMaker. This model reaches very good performance on lots of modern problems, reason why we would like to compare it with the LinerLearner model.

### Hyperparameter tuning

A XGBoost model requires many hyperparameters to be set properly. As these hyperparameters can have a huge impact onto model performance, we perform here a hyperparameter tuning step, namely we ask SageMaker to train many different XGBoost models by combining different hyperparameter values (chosen into a range of values). Then we set our final XGBoost model by using the hyperparameter combination reaching the best performance.

## Methodology

### Data Preprocessing

In order to preprocess our data, we first preprocess each dataset separately, and then we merge the 3 different datasets into one big, clean dataset, directly usable by our models.

### Portfolio DataFrame

This small DataFrame (10 rows) contains information about the 10 different offers.

In this DataFrame, column “channels” contains a list. In order to make the information contained into the column easily available for a model, we split the list into different columns.

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

… becomes …

| email | mobile | social | web |
|---|---|---|---|
| True | True | True | False |
| True | True | True | True |
| True | True | False | True |
| True | True | False | True |
| True | False | False | True |

### Transcript DataFrame

This DataFrame contains information about when offers are sent, viewed and completed, and about transactions.

In this DataFrame, “value” column contains a dictionary. We also need to format this column to make its content available for a model.

By exploring this DataFrame, we notice interesting elements:

- One person can receive multiple times the same offer.

  For the sake of simplicity, we have decided here to keep each person/offer pair only once in the dataset. As we try to see if there is a relationship between person and offer characteristics and the fact that an offer is completed by a customer, it seems reasonable to keep in our dataset all the offers that have been completed **at least once** by a certain customer.

- Multiple transactions can be done in order to complete one offer.
- As mentioned in the project description, an offer can be completed (namely: the required amount can be spent) without having being seen: in that case, we don't consider the offer is completed, because this was not intentional and this can then bias our analysis.

  In order to make that clear in our final dataset, we have created an extra column, called "completed", and containing "True" values for actually completed offers, "False" otherwise.

### Profile DataFrame

This DataFrame contains information about 17.000 Starbucks customers. For 2.175 customers, there are some missing information (*gender, age* and *income*), while *id* and *became_member_on* are filled. In order to avoid biases in our analysis, we choose to clean rows with missing values. We end up with 14.825 valid profiles, which is already a good number to have solid conclusions.

### Clean dataset

After all the cleaning operations mentioned above, we merge information coming from the 3 original datasets into one big dataset. In this clean dataset, we keep only unique offer id/person id pairs, and only for customers without any missing information.

We end up with 55.222 usable rows, that we will use for modelisation. 15 columns are available: person, offer id, completed, reward, difficulty, duration, offer_type, email, mobile, social, web, gender, age, became_member_on, and income.

Last step we have to complete in order to have a totally usable DataFrame from a model point of view, is to transform non-numerical data into numerical data. For that, we transform Boolean columns into 0/1 columns and categorical columns into numbers (one number associated to each possible value). Regarding the column *became_member_on*, which is containing a date, we have decided to turn this column into the number of loyalty months of each customer. Namely, we computed the delta (in months) between the most recent date contained into *became_member_on* column and each entry date. This value is stored into *loyalty_in_months* column.

## Implementation

### Benchmark model: LinearLearner

In order to get a benchmark, we use the basic version of the LinearLearner model provided by SageMaker, namely we use the default parameters. The aim is only to get first performance, in order to compare the performance of our other models against it.

## Refinement

### LinearLearner, recall-oriented

According to us, what is important for this use-case is to successfully reach a maximum of Starbucks customers. Moreover, this is not a too costly if some offers are sent, but not completed. These 2 points let us think that we should favor recall, and not precision here.

A the LinearLearner implementation allows us to easily train the model by favoring the recall, we use this functionality and see if it can reach 85% recall.

### XGBoost model

We have trained 2 version of this XGBoost model:

- Version A: one version with some hyperparameters whom values are set manually, based on default values and suggested values into XGBoost documentation.

- Version B: one version for which we use a Hyperparameter Tuner in order to find the optimal combination of hyperparameters, namely the combination reaching best performance on the validation set.

# Results

## Model Evaluation and Validation

Once our various models are trained onto our training set, we can use our test set in order to evaluate their performance. As mentioned above, we use 3 different metrics in order to evaluate our models: accuracy, precision and recall.

| | LinearLearner; Basic Model | LinearLearner; Recall-oriented Model | XGBoost; Version A | XGBoost; Version B |
|---|---|---|---|---|
| **Precision** | 0.65 | 0.62 | 0.74 | 0.74 |
| **Recall** | 0.75 | 0.85 | 0.71 | 0.7 |
| **Accuracy** | 0.72 | 0.72 | 0.77 | 0.77 |

Our recall-oriented LinearLearner model reaches our 85% recall target, while the drop in precision remains totally acceptable.

The XGBoost tuned version performance is worse than the initial version of our XGBoost model. This can be explained by the fact that the hyperparameter ranges we set in our implementation are numerous and quite large, while the number of models to try is low. Consequently, SageMaker has less chance to find an hyperparameter combination beating our initial version.

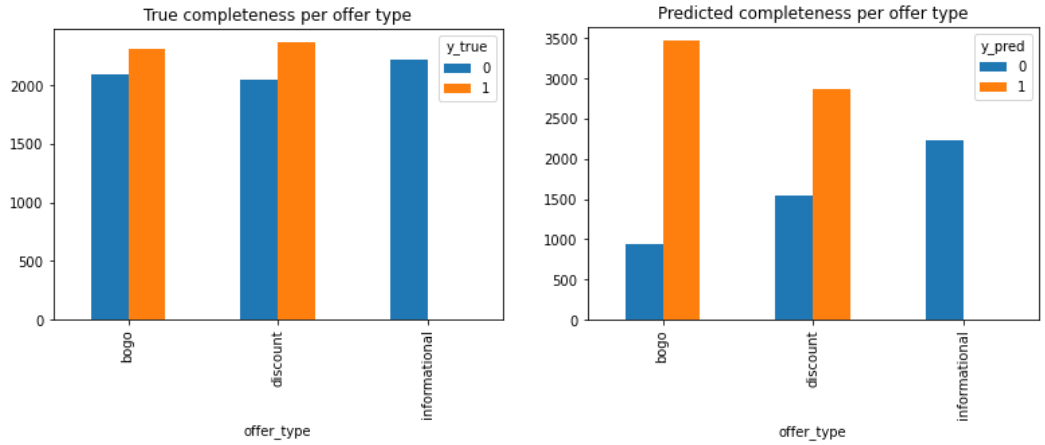
## Justification

For a use case such as the one solved here, having a better recall is, according to us, what we would recommend. Indeed, sending offers is not so expensive for Starbucks, so this is not a too big problem if they send offers to people not completing them, while they are reaching a maximum of people completing sent offers (higher recall). Consequently, we would recommend Starbucks to use our recall-oriented LinearLearner model, which is reaching 85% recall.

Personal note: The advantage with use cases as the one we have chosen to solve here is that we can test them effectively, in the real world: indeed, by doing an A/B test, this is very easy. Namely, Starbucks could split its customers into two groups: to group A, they continue sending offers as before, and to group B, they send offers by following the recommendations of our model. After a predefined amount of time, they can easily see if our model brings added value or not. Such an exogenous evaluation is the best evaluation we could make.

# Conclusions

## Free-Form Visualization




Plots above show that our selected model understood that informational offers are never completed, which is already a good point.

The plots show also the impact of favoring the recall: indeed, we can see that much more bogo and discount offers are predicted as completed compared to the true completeness status. This seems logical to us: indeed, by predicting more offers to be completed, the model has less chance to miss some completed offers, which is increasing its recall.

We can finally conclude that the bogo offer type is the weakness of our model: indeed, the ratio of misclassified offers is the highest for this offer type. Looking more carefully to this type is then a possible improvement.

## Reflection

As in any real-world machine learning use case, data preprocessing step has been the most heavy step in our project. Regarding how raw were the data when we received them, everything was possible: defining our research question and preparing the data regardingly has been, according to us, the most challenging part of this project.

## Improvement

### Feature Engineering

Some feature transformations could have improved the performance of the models, especially the LinearLearner. We suspect that bucketing features like *income* or *age* could help. Though, we think that the XGBoost model is able to deal with “raw” features.

Studying more carefully the features is also something to try if we want to improve our performance. It could be valuable to look at correlations between features, in order to remove redundant features for instance (especially for the LinearLearner model).

Finally, based on our results, we think that removing informational offers from the dataset would not hurt the performance for the use case we solved.

### Datasets

Dividing the full dataset between train, validation and test sets “more cleverly” could help to reach higher performance. Indeed, we could reasonably think that some people are more responsive to

offers than others. Based on this hypothesis, we could divide the full dataset by taking into account people receiving the offers, namely put all the offers received by one person into a same dataset. Here, we do not do anything to ensure that: it is then possible that two offers received by person_A are splitted between train and validation sets.

### Hyperparameter Tuning

Regarding the XGBoost model, testing even more hyperparameters combinations could be valuable, in order to gain some percents in terms of performance. As this takes time and costs computing resources, we haven't done that here.

### Targets

Defining a more precise target could bring more added value to the customer. Indeed, here, we "only" try to predict if an offer will be completed or not. We could break this target into a more granular one, for example by trying to predict if an offer will be completed quickly (in 2 days) or not (in 5 days).