Starbucks Buy One Get One (BOGO)

Aekachai Tangratanavalee

Udacity Machine Learning Engineer Nanodegree , May 2020

I. Definition

Project Overview

Starbucks is one of the most well-known companies in the world: a coffeehouse chain with more than 30 thousand stores all over the world. It strives to give his customers always the best service and the best experience; as a side feature, Starbucks offers his free app to make orders online, predict the waiting time and receive special offers. This project aims at optimising the customers' experience using the app through user's behaviour analysis and one-to-one offers customisation.

Problem Statement

Starbucks wants to find a way to give to each customer the right in-app special offer. There are 3 different kind of offers: Buy One Get One (BOGO), classic Discount or Informational (no real offer, it provides informations) on a product.

Our goal is to analyse historical data about app usage and offers / orders made by the customer to develop an algorithm that associates each customer to the right offer type. We will develop 3 different models, one for each type of offer: each model will evaluate the customer's pro-pension towards that kind of offer returning a score. Given the 3 propensity scores, we will build an algorithm to make the final decision for each customer.

Metrics

Based on past data, we will compare the models prediction with the actual pro-pension/not- pro-pension of the customer through different performance metrics:

- balanced accuracy: it is the percentage of correctly classified records, corrected for unbalanced targets. The formula is

$$balacc = (\frac{TruePosition}{Position} + \frac{TrueNegative}{Negative})/2$$

where Positive or Negative are the actual propensity of the customer, and the True label means that the model prediction is correct.

Precision/Recall: this two measures focuses on the Positive labels:

$$precision = \frac{TruePositive}{PreictedPositive}, \quad recall = \frac{TruePositive}{Positive}$$

Basically, the first is the percentage of corrected positive labels on all the predicted positive labels, and answers to the question "How much of the predicted positive have actually the pro-pension?". The second one is the percentage of corrected positive labels on all the actual positive records, and answers to the question "How much of the positive records the model is recalling?".

- F1-score: this measure is the harmonic mean of the previous two:

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} = \frac{2TP}{2TP + FP + FN}$$

A high F1-score ensures that the model is predicting correct positive labels, without leaving behind any actual inclined customer.

II. Analysis

Data Exploration

For this project we have 3 available data sources.

Portfolio

It contains the list of all available offers to propose to the customer. Each offer can be a discount, a BOGO (Buy One Get One) or Informational (no real offer), and we've got the details about discount, reward and period of the offer. There are 10 available offers, for each one the features are:

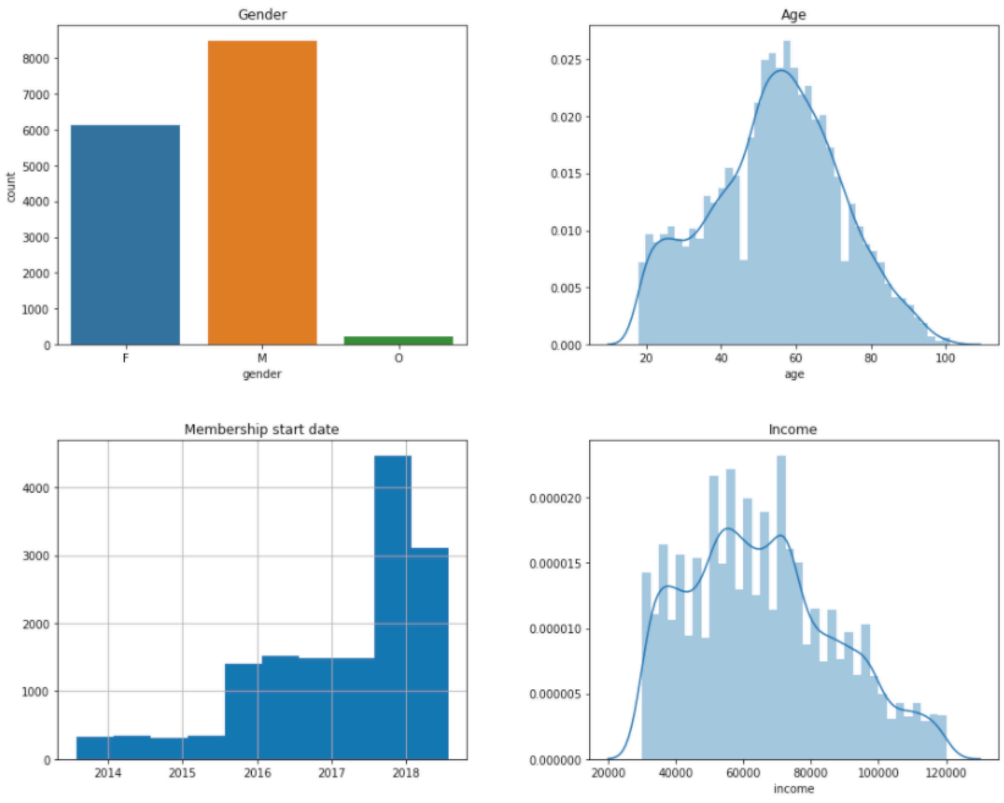
- id of the offer
- offer_type, it can be a discount, a BOGO (Buy One Get One) or Informational (no real offer)
- channels of the offer, a subset of web, email, mobile, social difficulty, the minimum amount the customer has to spend to activate the offer; for Informational it is always 0
- reward of the offer; for Informational it is always 0
- duration of the offer in days

Profile

It contains the list of all customers that interacted with the app. There are 17,000 records. For each profile, the dataset contains some customer's information:

- id of the customer
- gender, it can be M, F or O
- age
- income
- became_member_on, date of when the customer became member of the app. The dates range from 2013-07-29 to 2018-07-26.

There are some missing values in *gender* feature: in the same rows, *income* is missing too, and *age* is always equal to 118. We can suppose that informations about these customer has been lost, so we can safely drop these records: they're remain 14,825 records.



Taking a quick look at the distribution of the features on these customers, we can see that

- the customer's age is normally distributed, with a mean of 60 years old
- the income is left skewed, so there are few high-income customers
- there's a peak of subscriptions from the second half of 2017: maybe a big update of the app?

TRANSCRIPT

This dataset has the list of all actions on the app relative to special offers, plus all the customers' transactions. There are 306,534 events. For each record, we've got:

- *person*, the id of the customer
- *event*, either offer received, offer viewed, offer completed or transaction
- *time* of when the event happened. It is measured in hours from a certain not specified **t0**. This value ranges from 0 to 714 (almost 30 days).
- *value*, a dictionary containing some additional data about the record. It can contains:

- **offer_id** (the same of *Portfolio* dataset) for all events except transaction
- **reward** of the offer for an "offer completed" event
- **amount** spent for a transaction

For a better analysis, we split the *value* feature into 3 different columns.

Analysing the event types we can see that

- for each offer completed, there is a transaction with the same *time* value
- for each offer viewed there is an offer received
- there's no clear relationship between offer viewed and completed: an offer can be viewed but not completed, and vice-versa (a customer completes an offer without knowing)

NEW DATASET - CUSTOMER JOURNEY

To understand a customer behaviour, we have to recreate the **customer journey**: we create a new dataset, *journey*, starting from *transcript*, in this way:

- for each **offer received** we concatenate the relative **offer viewed**: we must be sure that the view is the nearest AFTER the reception, otherwise we could join the view with a subsequent reception of the same offer (a customer could receive the same offer multiple times)
- for each **offer viewed** we join the eventual **offer completed**, with the same logic. In this way, we drop the completion without view: since they are casual, they do not represent the customer's behaviour
- for each **offer completed** we join the relative **transaction**
- finally, we concatenate all the remaining transactions that do not come from an offer completion

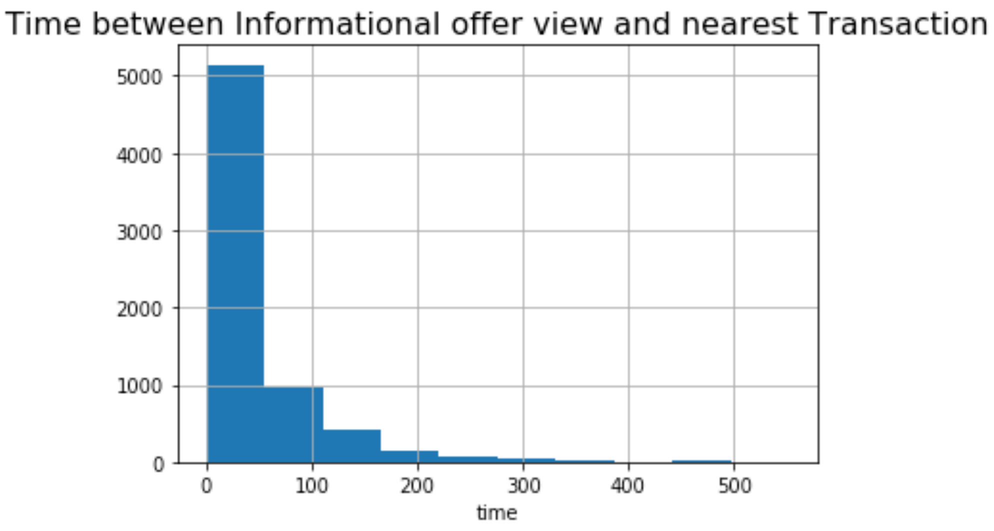
This data pipeline results in a data frame with 188,234 records; we can now join data about the relative offer from **Portfolio** and the relative customer from **Profile**.

FEATURE ENGINEERING

First of all, we build the 3 different targets. For **bogo** and **discount** we can put target = 1 (pro-pension) when the offer viewed (of the relative type) ends with a completion. We take only the viewed, not received, because we're interested in the customer's behaviour.

Since there's no real conversion (no *offer completed* record) for **Informational**, we develop a proxy. If we find a **transaction** right after an Informational offer view, we put that as pro-pension. But how much hours is *right after*?

```
mean   43.230928
min     0.000000
25%     6.000000
50%    24.000000
75%    54.000000
Max   552.000000
```



As we can see from the distribution of time between these two events, we can take the median value (24 hours) as *right after* value.

With these 3 targets, we have these Conversion percentages:

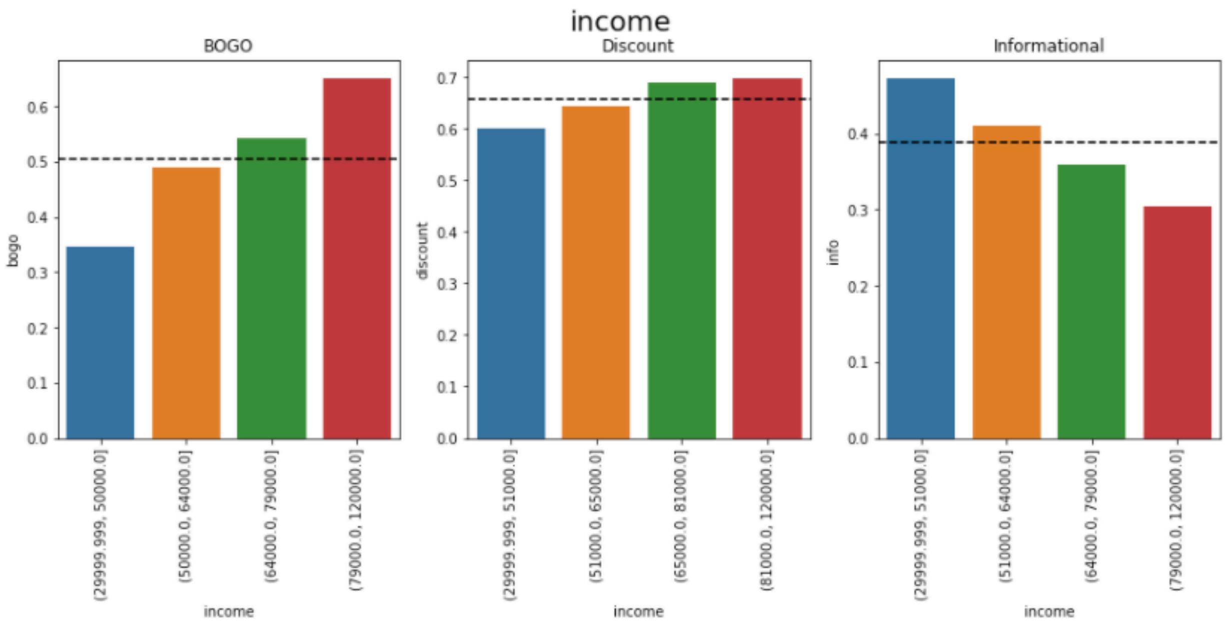
Bogo	50.47%
Discount	63.73%
Informational	38.82%

Starting from this dataset, we can enrich the data with new columns.

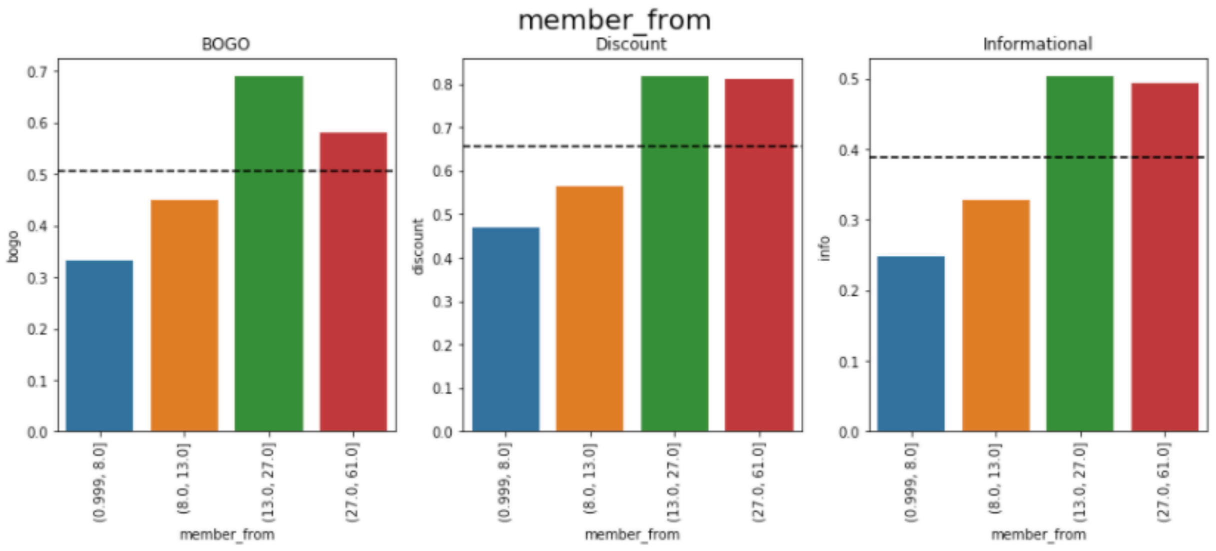
- *member_from* becomes number of months of membership
- *time* in hours is traduced in days; moreover, we count the days module 7 (days in the week), so the first 24 hours have label 2, the second 24 hours label 1 and so on
- we create a series of features based on past behaviour, such as
    - Number of transaction
    - Average amount of transaction
    - Number of completed / viewed offers
    - Average reward received from offers
    - Average time from reception to view, and from view to completion

All these feature aim at understanding how much a customer is active on the app and is responsive to offers.

EXPLORATORY VISUALISATION

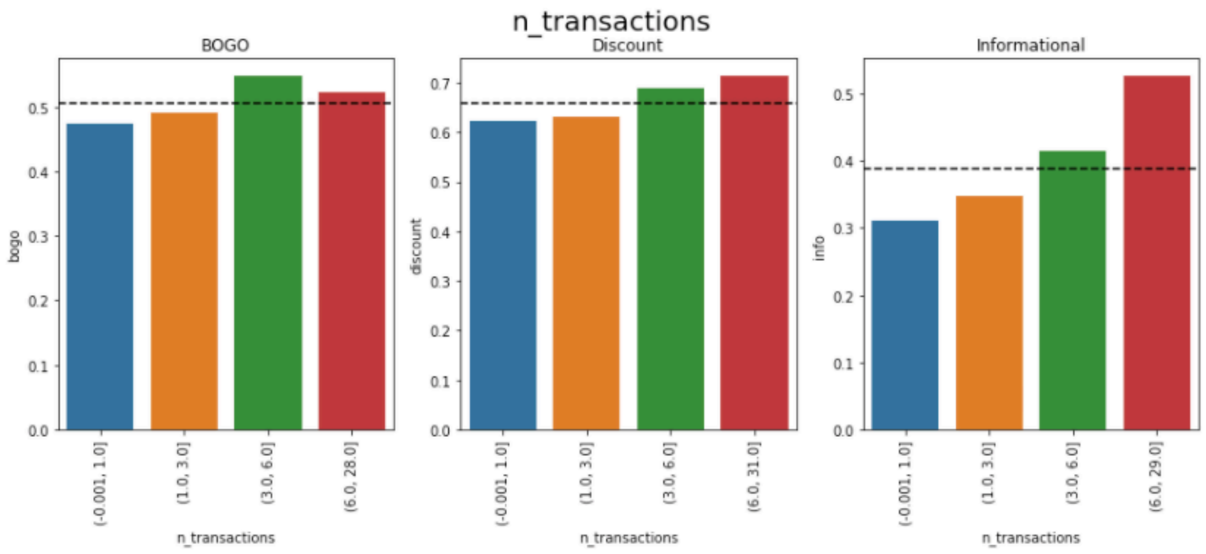


We cover now a subset of the input variables that has an interesting relation with the target(s). For each feature, we plot the average of the target for each category / bin; the black line is the target overall mean.

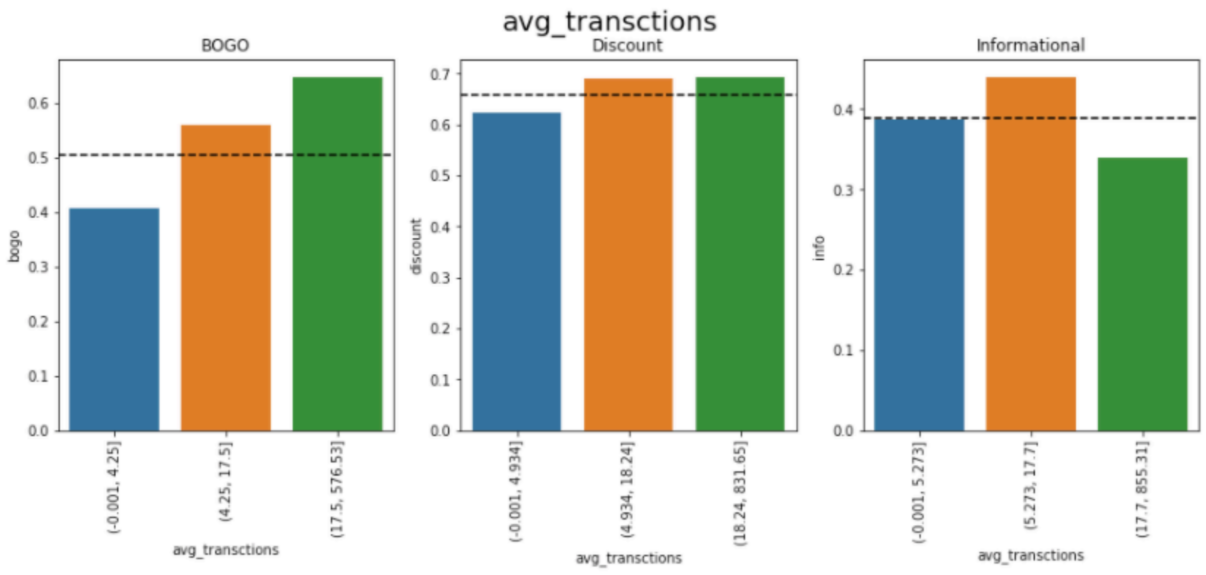


First of all, it seems like that higher incomes are more interested in special offers, especially **BOGO**. **Informational** offers instead have more effect on low income customers.

Being a senior member makes you more incline to complete each kind of offer. There's an interesting lower peak on the **BOGO** offering for the more senior customers.



This feature is relevant only for the **Informational** offers: it seems like more active customers are more interested in info on new products than discounts or special offers.
If we look at the average transaction, instead, we see that higher spending customers are more interested in special offers (especially **BOGO**) than information on products.



ALGORITHMS AND TECHNIQUES

To address the problem of understanding the best offer type for each customer we develop a series of Machine Learning models: taking as input past data, with the corresponding label of *pro-pension* or *not pro-pension*, the model will find relations between the input features and the event we want to predict (the offer completion) by building a mathematical infrastructure re-usable on new customers.

We try 2 different type of algorithm for each target, then we choose the best one (comparing the performance measures listed in Section 1 - *Metrics*) for each kind of offer. The algorithms are:

XGBoost: this is an implementation of the *Gradient Boosting* algorithm, a widely used Machine Learning model. This model develops a series of weak learners called *Decision Trees* in subsequent fashion: the first tree tries to predict the given target, while the second one aims at modelling the error of the first one and so on.

AWS Sagemaker LinearLearner: this is the Sagemaker implementation of a classical linear model. It assigns a weight to each input feature to have an output score: if this score is higher of a certain *threshold*, the model predicts an event (offer completed), otherwise it predicts a non-event.

BENCHMARK

As benchmark measure, we decided to take the current **conversion rate** for each type of offer (described in this section's *Data Exploration*). After developing the models, we will compare these rates to the models' **precision scores**: since we will send the offer to all the customers that the model labels as "interested", the precision is exactly the expected conversion rate on that kind of offer.

III. Methodology

DATA PREPROCESSING

Starting from the *journey* dataset described in Section 2 - *Data Exploration* we create 3 different datasets, one for each type of offer. Each of these datasets contains only the data portion relative to viewed offers of the same type: for example. **bogo** dataset contains all the viewed BOGO offers with the relative target about offer completion. After that, we need to apply some data pre-processing steps to each dataset. These steps are:

- **data sampling:** since 2 out of 3 targets are unbalanced (*discount* is 66%, *info* 39%) we can sample data to have better performances on the model. In particular we apply the *under-sampling* method: we take all the records of the lowest frequent class (0 for *discount*, 1 for *info*) and a random subset of the most frequent, in order to have a balanced 50-50% ratio

- **train-valid-test split:** divide the input data into 3 parts, each one stratified by the target (the event percentage is the same both on the original dataset and the 3 derived ones). The *train* set is the "real" input dataset for the algorithm: all the weights and the formula will derive from the relations found on this set. The *valid* set is to tune the hyper-parameters of each model (more on this in the next paragraph). Finally, the *test* set is to compare the results of the several developed models to choose the best one

- **missing imputation:** substitute missing values in features, approximating the possible real value. The methods to impute are several: we use a constant value for the only categorical variable (missing *gender* is replaced by "O") and the *median value* for the numeric features. Another common way to address missing numerical values is using the *mean*, but we preferred the median because it is a robust statistics even with skewed data and with outliers

- **categorical encoding:** a Machine Learning algorithm can use only numerical data, so we must transform the *gender* variable into a numeric feature. Since we've got only one categorical input with only 3 categories, we decided to use *One Hot Encoding*: this method creates a binary feature for each category of the original variable, and each binary feature is equal to 1 if the relative category is present on that record. Here's an example:

Gender	Gender - M	Gender - F	Gender - O
M	1	0	0
F	0	1	0
O	0	0	1

For more complex categorical variables, with many different categories, this method could create too many binary flags, so other methodologies are preferred.

- **data standardisation:** for some algorithms an unbalanced distribution, with high skewness and outliers, could worsen the performance. For this reason we add a standardisation step: each numerical feature is transformed with the formula

Implementation

All the data pre-processing and the model building are developed under the Amazon Web Services infrastructure, specifically the Sagemaker component. This is a cloud Machine Learning platform to develop, tune, version, query and deploy Machine Learning models. In particular we will use the Python API of Sagemaker:

- The data processing component, via SKLearnProcessor
- The hyper parameter tuning with the HyperparameterTuner component
- The model development with the Estimator object
- Finally, the model query for result using the Batch Transformation part

Data processing

The SKLearnProcessor provides a way to run scikit-learn data-prep components on the Sakemaker cloud infrastructure, First, we have to create a Python script ./lib/preprocessing.py containing all the steps described in the previous paragraph. In particular we use a combination of Pipelines and ColumnTransformer to join several operations on different feature subsets.

First we sample data and make the split: this parts are not included in the pipelines because at serving time we won't need them.

```
rus = RandomUnderSampler(random_state=seed)

x_tot, y_tot = rum.fit_sample(df.drop( tgt, 1), df([ tgt ])

# Split between train, validation and test data
X, X_val, y, y_val = train_test_split(

        X_tot, y_tot, test_size=args.val_split_ratio, stratify=y_tot, random_state=seed
)
X, X_test, y, y_test = train_test_split(
        X, y, test_size=args.test_split_ratio / (1 - args.val_split_ratio),
        stratify=y, random_state=seed
)
```

For under-sampling we are using the object from the imblearn library. The val_split_ratio and are two arguments, passed at run time, that specify the portion of data for validation

```
# Preprocessing pipeline for gender
# Imputing with "O", then One Hot Encoding
gender_pipe = Pipeline(
    [
        ('imputer', SimpleImputer(strategy='constant', fill_value='0')),
        ('ohe', OneHotEncoder())
    ] )
# Preprocessing pipeline for numeric features
# Imputing with median, then standardising distribution
num_pipe = Pipeline(
    [
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler( ))
    ] )
```

```
# Join the pipelines via a ColumnTransformer
preprocessing = ColumnTransformer(transformers=[
                ('gender_pipeline', gender_pipe, ['gender']),
                ('numeric_pipeline', num_pipe, keep_vars[1:])
                #all other vars are numeric
 ])
 # Fit the transformer
 X = preprocessing.fit_transform(X)
 X_val = preprocessing.transform(X_val) X_test = preprocessing.transform(X_test)
```

and test sets. After that we create two Pipelines, one for categorical data, the other for numeric data:
Finally, we parallelise the two pipelines with a ColumnTransformer and apply them to all datasets:

```
        pd.concat([y.reset_index(drop=True), pd.DataFrame(X)], axis=1).to_csv(
                os.path.join(f'/opt/ml/processing/output/{tgt}_train.csv'),
                header=False, index=False
        )
```

Keep in mind that to use the datasets with Sagemaker's estimators, the output `csv` file must have no header and the target variable before the others:
We can now launch the job with the SKLearnProcessor

```
        sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                             role=role,
                                             instance_type='ml.m5.xlarge',
                                             instance_count=1)
        sklearn_processor.run(
                code='lib/preprocessing.py', #entrypoint for processing
                inputs=[ProcessingInput(os.path.join('s3://', bucket,
                        f'Capstone_Starbucks/{tgt}.csv'), '/opt/ml/processing/input')],
                outputs=[
                        ProcessingOutput(source=f'/opt/ml/processing/output/',
                        output_name=f'{tgt}_data')
                        ],
                arguments=['--target', tgt]
```

This processor is run on each of the 3 starting datasets.

HYPERPARAMETER TUNING

This is a crucial part of the Machine Learning pipeline. Each algorithm has a set of *hyper-parameters* (different from the *parameters*, that are calculated at train time) and we have to set them before the training part. For example, for the XGBoost algorithm we can set the *number of estimators*, the *maximum depth* for each estimator, the *subsample* of the train data taken at each step.

We first define a base estimator of the chosen type, putting a default value for each hyper-parameter:

```
container = get_image_uri(session.boto_region_name, 'xgboost', '0.90-1')
xgb = sagemaker.estimator.Estimator( container,
                                     role,
                                     train_instance_count=1,
                                     train_instance_type='ml.c4.xlarge',
                       output_path=f's3://{bucket}/Capstone_Starbucks/{prefix}/model',
                       sagemaker_session=session,
                       base_job_name=prefix + '-'
)
xgb.set_hyperparameters(max_depth=4, eta=0.1,
                              gamma=4, min_child_weight=6,
                              colsample_bytree=0.5, subsample=0.6,
                              early_stopping_rounds=10, num_round=200,
                              seed=1123)
```

Then, we set a HyperparameterTuner object, specifying for each hyperparameter a range of values to try in the Hyperparameter_ranges argument, and we fit the object (here prefix is a placeholder for the target name, *bogo, discount or info).

The tuning works this way:

- Sagemaker trains a XGB instance applying a random choice (into the specified range) for each hyperparameter
- it assess the model performances on the validation set calculating the specified metric (in our case, the **F1-score**)

- using the Bayesian formula it constructs a probability space with the hyperparameters, understanding where the "right" hyperparameters (those which can maximize the performance) are

  at each iteration it modifies the probability space, with a new set of hyperparameters tried
- At the end, we take the model with the best F1-score.

Model development

The other algorithm we choose to use, the Sagemaker's LinearLearner , does not have so many hyper-parameters to tune, so we decided to directly train it. Similarly as before, we first instantiate the estimator, then we fit it

Querying results

We finally want to assess the models' performances with the test set. In order to do so, we use the transformer method of a Sagemaker estimator, that computes a one-time query of the model on a given set of records.

Refinement

In a first time, we did not choose to sample the data: since the unbalance is not so strong, we tried to develop models with the original target distribution. However, we noticed that there were unbalanced predictions: the Discount model over-predicted the positive event and the Informational one. The predictions had the same unbalance as the target. For this reason, we chose to go for *under-sampling* and, as we'll see in the next section, the performances improved quite a lot.

IV. Results

MODEL EVALUATION AND VALIDATION

BOGO

Algorithm	F1-Score
XGBoost	0.7366
LinearLearner	0.7135

The XGBoost tuning resulted in F1-scores on the validation set ranging from 0.7099 to 0.7366. The best one has these hyper-parameters:

- colsample_bytree: 0.665418398434084
- eta: 0.3024061838045679
- gamma: 0.15467677184991543
- max_depth: 3
- min_child_weight: 8
- subsample: 0.7285079986427208

We apply the model to test set with these results:

Metric	Value
Accuracy	62.24%
Precision	60.11%
recall	72.80%
F1	**65.85%**

Confusion Matrix - normalised by row

a	Predicted 0	Predicted 1
True 0	51.68%	48.32%
True 1	27.20%	72.80%

Discount

Algorithm	F1-Score
XGBoost	**0.7282**
LinearLearner	0.7092

The XGBoost tuning resulted in F1-scores on the validation set ranging from 0.6980 to 0.7282. The best one has these hyper-parameters:

- colsample_bytree: 0.778622530979818
- eta: 0.4561709659187462
- gamma: 4.844954586874898
- max_depth: 5
- min_child_weight: 2
- subsample: 0.9165187239747279

We apply the model to test set with these results:

Metric	Value
Accuracy	64.98%
Precision	64.01%
recall	68.44%
F1	**61.15%**

Confusion Matrix - normalised by row

a	Predicted 0	Predicted 1
True 0	61.51%	38.49%
True 1	31.56%	68.44%

Informational

Algorithm	F1-Score
XGBoost	0.6660
LinearLearner	**0.6915**

In this case the best model is the LinearLearner, with these hyper-parameters:

- epochs: 100
- learning_rate: 0.01
- loss: auto
- mini_batch_size: 1000
- optimiser: adam

We apply the model to test set with these results:

Metric	Value
Accuracy	56.12%
Precision	53.47%
recall	93.39%
F1	**68.00%**

Confusion Matrix - normalised by row

a	Predicted 0	Predicted 1
True 0	18.96%	81.04%
True 1	6.61%	93.39%

Justification

We can see that the precisions of 2 out of 3 models are higher than our benchmark, the current conversion rate:

Offer	Current CR	Model precision	Delta
BOGO	50.47%	60.11%	9.64%
Discount	65.73%	64.01%	-1.72%
Informational	38.82%	53.47%	14.65%

The only negative case, Discount, have a small negative delta; the other two cases have a real positive delta. I think that the Informational is especially important, since on these type of offer Starbucks does not have to give money as reward.

V. Conclusion

Let's summarise all the different steps followed in this process.

1. We analyse the 3 different datasets containing information about offers in Starbucks app Then
2. we reconstruct the customer's journey through offer view, completion and transaction
3. After that, we create new input features to better understand one customer's behaviour
4. We made some pre-process on input data
5. We develop different Machine Learning models, comparing them and choosing one champion model for each type of offer

I think that wrangling the data and understanding all the different special cases to reconstruct customer' journey was the most difficult and time consuming part. I personally took several iteration, founding every time some edge case not taken under consideration. Also the model development through the Sagemaker platform was challenging, but very rewarding.

Improvement

We could achieve further improvements in several ways:

- the most relevant thing could be have more input features. Data about app usage and about which types of products a customer buys could really help understanding one's behaviour
- also having more records at our disposal could improve the model's performances
- we could set a minimum precision for the Discount model in order to have a non-negative delta with the current benchmark
- we could try other algorithms, such as Neural Networks, SVM and Random Forests, as they could better suit this particular use case