UDACITY

aws

AWS Machine Learning Engineering Nanodegree

[Capstone Project]

STARBUCKS®

STARBUCKS PROMOTIONAL OFFERS

CONTENTS

Project Overview

Domain Overview

Besides being the largest coffeehouse company in the world, Starbucks is notoriously known for its rewards program.
Starbucks is a global and a leading brand. With the advent of the usage of Machine Learning(ML) in todays' world, the company can utilize the strength of ML to increase its revenue. This project is concerned with associating the various offer types to its customers.

So, When Starbucks launched its rewards program and mobile app, they dramatically increased the data they collected and could use to get to know their customers and extract info about purchasing habits.
The reason is that all users should not receive the same offer type. If we can understand the historical customer behavior/reaction to these offers, we can create personalized offer predictions which can target these customers to increase the revenue.
The data for this case simulates how people make purchasing decisions and how those decisions are influenced by promotional offers.
Each person in the simulation has some hidden traits that influence their purchasing patterns and are associated with their observable traits.

Project Problem Statement

Problem Statement

Starbucks collects the customer data to understand their behavior on the rewards and offers sent via the mobile-app. Once every few days, Starbucks sends the personalized offers to its customers. These customers can respond positively/negatively/neutrally. A key thing to note is that not all the customers receive the same offer.
Not only does Starbucks go through mounds of coffee beans to satiate its raving fans, but they also have mounds of data that they leverage in many ways to improve the customer experience and their business.
Starbucks benefits since they would not be wasting money contacting the wrong customers for offers, or sending offers to customers who actually would have purchased anyway (or who are actually put off by receiving promotional 'spam').
Customers benefit, in that they receive offers that they are actually likely to be interested in and glad to receive, rather than just a blanket sending of offers that are not relevant to many customers.
We will use the data to find a better promotion strategy by Identify which groups of people are most responsive to each type of offer,
How best to present each type of offer?
How many people across different categories actually completed the transaction in the offer window? We will also try to train a model to predict the amount that can be spent by an individual given the individual's traits and offer details.
This will help us decide which promotional offer best suits the individual and respond to the target audience with better accuracy.
The task of this project is to combine transaction, demographic and offer data of the past (which is already provided) to determine which demographic groups respond best to which
offer types.

Project Data

Data, Datasets and Inputs

Udacity and Starbucks as part of the Machine Learning Engineer Nanodegree program and provided source for the dataset. It contains simulated data that mimics customer behavior on the Starbucks rewards.
The program used to create the data simulates how people make purchasing decisions and how those decisions are influenced by promotional offers.
Each person in the simulation has some hidden traits that influence their purchasing patterns and are associated with their observable traits. People produce various events, including receiving offers, opening offers, and making purchases.

There are three types of offers that can be sent:
▪ Buy-one-get-one (BOGO), discount, and informational.

→ BOGO: a user needs to spend a certain amount to get a reward equal to that threshold amount.

▪ Discount: a user gains a reward equal to a fraction of the amount spent.
▪ Informational: there is no reward, but neither is there a requisite amount that the user is expected to spend.

Offers can be delivered via multiple channels: Email, Mobile, Social, Web

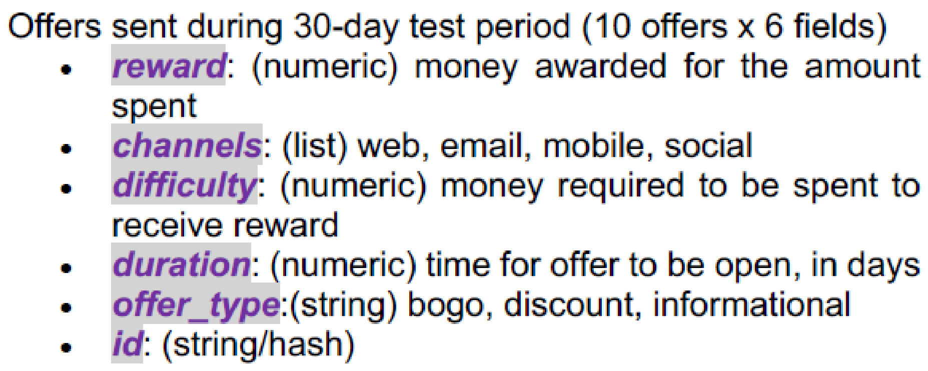
Project Data

Datasets provided:

Three json files were provided with simulated data:

▪ **portfolio.json :**

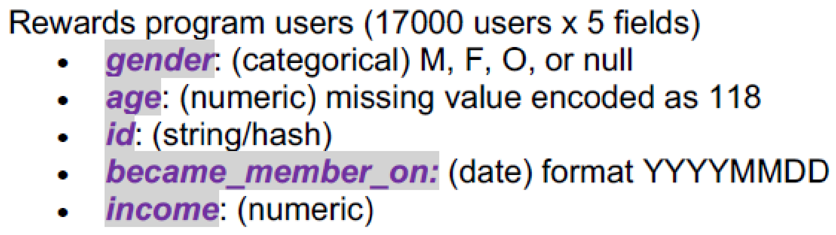
o This is a very short file giving details of the 10 unique offer ids used by Starbucks in this experiment. It provides details including reward (the dollar level of discount), difficulty (the dollar spend level required to trigger the offer), duration (the number of days for which the offer is valid), the channels (how the offer was sent to the customer) and the offer type


Offers sent during 30-day test period (10 offers x 6 fields)

- ***reward***: (numeric) money awarded for the amount spent
- ***channels***: (list) web, email, mobile, social
- ***difficulty***: (numeric) money required to be spent to receive reward
- ***duration***: (numeric) time for offer to be open, in days
- ***offer_type***:(string) bogo, discount, informational
- ***id***: (string/hash)

▪ **profile.json :**

o This is a dataset containing 17,000 members, including demographic details such as gender, age, income and the date they became a member of the program.

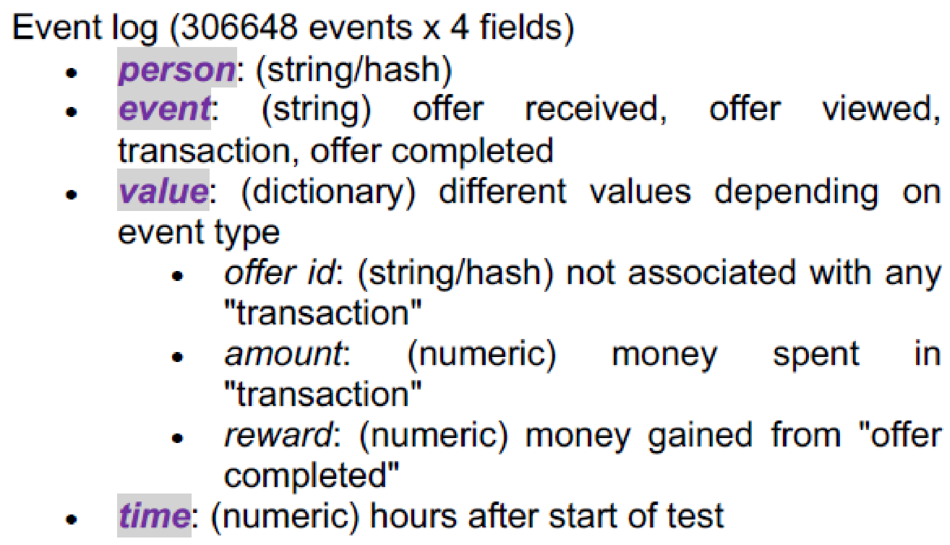

Rewards program users (17000 users x 5 fields)

- ***gender***: (categorical) M, F, O, or null
- ***age***: (numeric) missing value encoded as 118
- ***id***: (string/hash)
- ***became_member_on:*** (date) format YYYYMMDD
- ***income***: (numeric)

Project Data

▪ **transcript.json:**

o This is a large dataset containing over 300,000 rows of event data for four different event types: “offer received”, “offer viewed”, “offer completed” and “transaction”.



Event log (306648 events x 4 fields)

- ***person***: (string/hash)
- ***event***: (string) offer received, offer viewed, transaction, offer completed
- ***value***: (dictionary) different values depending on event type
  - *offer id*: (string/hash) not associated with any "transaction"
  - *amount*: (numeric) money spent in "transaction"
  - *reward*: (numeric) money gained from "offer completed"
- ***time***: (numeric) hours after start of test

Evaluation Metrics

Evaluation metrics are used to measure the quality of the model. When you build your model, it is very crucial to measure how accurately it predicts your expected outcome.
Evaluation metrics can help you assess your model's performance, monitor your ML system in production, and control your model to fit your business needs.
Our goal is to create and select a model which gives high accuracy on out-of-sample data.Classification Accuracy such as recall, precision, accuracy and ROC AUC.

- Precision: It is the fraction of relevant instances among the retrieved instances.
  - Precision = TP/ (TP + FP)
- Recall: It is the fraction of the total amount of relevant instances that were actually retrieved.
  - Recall = TP/ (TP + FN)
- F1: This metric is the harmonic mean of precision and recall. It is a point metric quite popular in evaluating the classifier.
  - F1 = 2 * (Precision * Recall) / (Precision + Recall)
- ROC-AUC Score: gives a trade-off between the true positive rate and false positive rate. It is the area under the ROC curve (Receiver Operating Characteristic).

Positive = identified and negative = rejected,

TP- True Positives → correctly identified positive samples

FP- False Positives → incorrectly identified

FN- False Negatives → incorrectly rejected

TN- True Negatives → correctly identified negative samples

Data Analysis and Cleaning

During a 30-day test period, data was collected to measure user behavior upon promotional offers that were sent to them. Hence, there is a relational model available,
where transactional data reflects user's actions which can be easily related to metadata about users and offers.
There are three main files:

Profile Dataset: Rewards program users (17000 users x 5 fields)

Variable	Type	Observation
gender	Categorical	M, F, O, or null
age	Numeric	Missing value encoded as 118
id	String/Hash	Primary key
became_member_on	Date	YYYYMMDD
income	Numeric	Salary

Portfolio Dataset: Offers sent during 30-day test period (10 offers x 6 fields)

Variable	Type	Observation
reward	Numeric	Money awarded for the amount spent
channels	List	web, email, mobile, social
difficulty	Numeric	Money required to be spent to receive reward
duration	Numeric	Time for offer to be open, in days
offer_type	String	bogo, discount, informational
id	String/Hash	Primary key

Transcript Dataset: Event log (306648 events x 4 fields)

Variable	Type	Observation
person	String/Hash	Foreign key (FK) to Profile dataset
event	String	offer received, offer viewed, transaction, offer completed
value	Dictionary	• *offer_id*: (string/hash) FK to Portfolio dataset • *amount*: (numeric) money spent in "transaction" • *reward*: (numeric) money gained from "offer completed"
time	Numeric	Hours after start of test

Project Data Analysis and Cleaning

- In all the three tables, some column names were renamed to be more meaningful such as offer id, customer id.
- All the categorical variables were one-hot-encoded and the continuous variables were normalized.
- The transaction history of all the customers which are not present in the profile table were deleted.
- All the three tables were combined on the fact that the customers first should receive the offer, then he should view it and then if the offer is still valid, either it can be left or can be proceeded for the completion.
- Data Cleaning Main Changes:
  - Portfolio:
    - one-hot encode channels and offer_type.
    - remove underscore in column names.
    - rename id to offer id (more meaningful).
  - Profile:
    - remove null values.
    - filter the other gender for simplicity.
    - preprocess date column.
    - binarize the gender.
  - Transcript:
    - rename person to customerid.
    - remove all the customer ids which are not in the profile table.
    - convert hours into days and rename the column.
    - parse the offers as one-hot encoded columns.
    - create a separate dataframe for transaction
    - These transactions are all about the offers which a customer has first received, then viewed and if applicable, used and completed.
    - For this to happen first we check if still the respective offers stand valid.

- There were some offers which were never viewed by the customer. Such transactions were filtered out.

– After the three tables were combined, 'time','customerid','email','informational' attributes were dropped.

- Time: we are not concerned when the offer's start date is.
- Customerid: no need for this since it does not have any meaning.
- Email: all the offers were sent via email, hence there is no extra information conveyed by this feature.
- informational: all the offers which only convey some information about the offers were also not required because only 20% of offers were informational and out of which only 7% were marked as successfully converted.
- For continuous variables such as 'income','totalamount','duration', 'reward', we used min max scaler function.
- This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

Project Data Analysis and Cleaning

Exploratory Data Analysis

As mentioned before, we were provided with three datasets:

- Profile
- portfolio
- transcript.

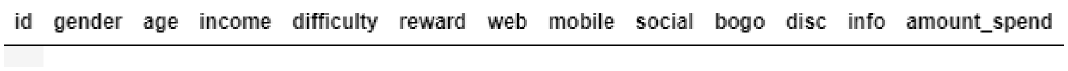
First we performed some structural analysis and Data Cleaning on the three datasets as mentioned below:

- Portfolio:
    - There are 10 records with 6 features given for 10 unique offer ids.
    - These offers can be sent via 4 different channels such as social, web, mobile and email.
    - There are 3 unique offer types such as bogo, discount and informational.
- Profile:
    - There are 17000 records with 5 features present.
    - One observation was that there were a total 2175 records present where age of the customers was mentioned as 118 with null income values.
    - We filtered out such records.
- Transcript:
    - This was the trickiest dataset with 306534 records with 4 features which comprised the information of various offers received, viewed and completed by any customer.

Data Exploration and Visualization

Data Preprocessing for Visualization Only

Data provided by Starbucks is preprocessed for visualization and converted into one table:



id	gender	age	income	difficulty	reward	web	mobile	social	bogo	disc	info	amount_spend

This is necessary for:

- **convenient data exploration**
- **visualization.**

Fields:

- **id, gender, age, and income are copied from profile.json**
- **difficulty, reward, web, mobile, social are from portfolio.json**
- **Finally, bogo, disc, info and amount_spend are calculated from transcript.json.**

Values in bogo, disc, info are calculated by matching customer's responses to each offer, particularly:

- **The offer was not given to a customer.**
- **A customer received the offer but never viewed it.**
- **The offer was viewed.**
- **The offer was viewed and completed (in the case of informational offers, it means that the offer was viewed and a transaction was made by the customer within the “influence” period specified by Starbucks).**

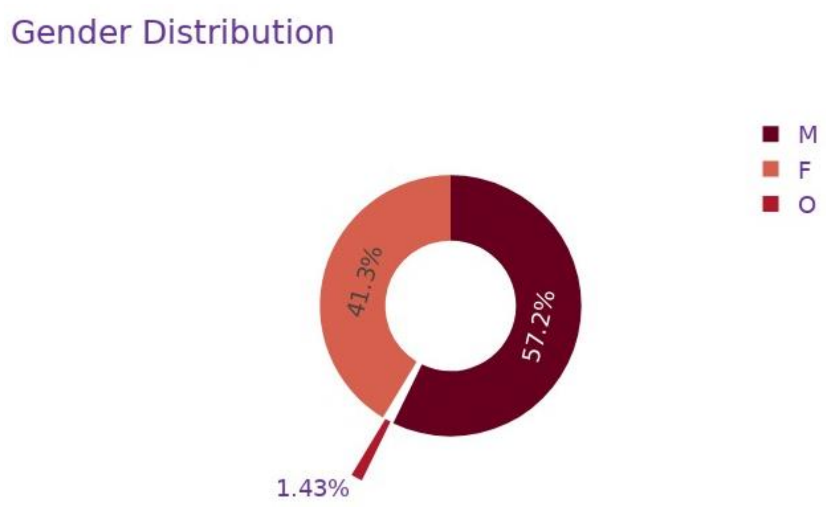
❖ **The Processed Dataset saved as pickle file as** `Data_cleaned_EDA.pkl.`

❖ **For Father using in Data Visualization.**

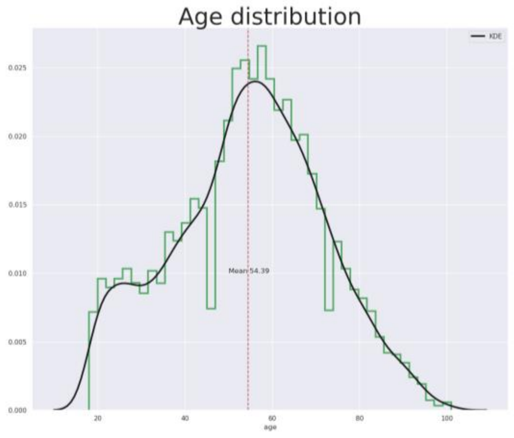
Data Exploration and Visualization

Data Visualization

Visualization and Distribution of Gender in the Data



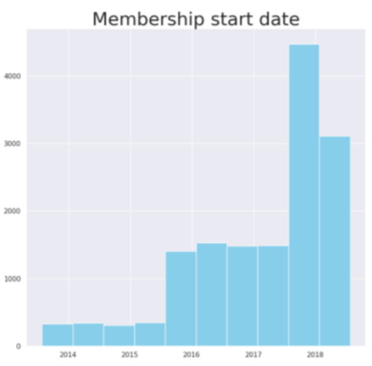
Visualization and Distribution Mean Age of the Customers



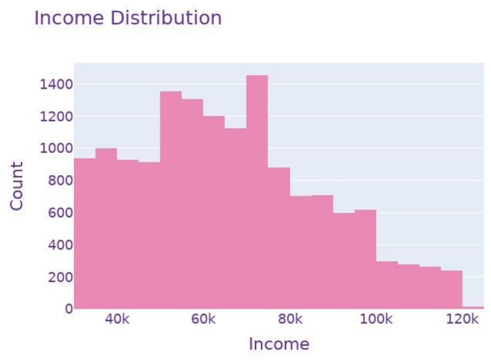
Data Exploration and Visualization

Data Visualization

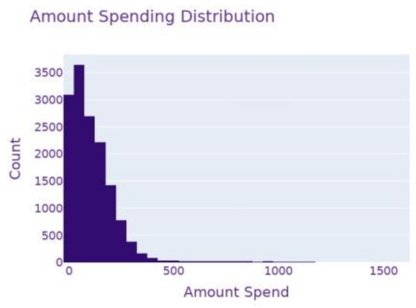
Visualization and Distribution of The year most of the customers started their membership with Starbucks.



Visualization of Average Income Distribution of the Customers.

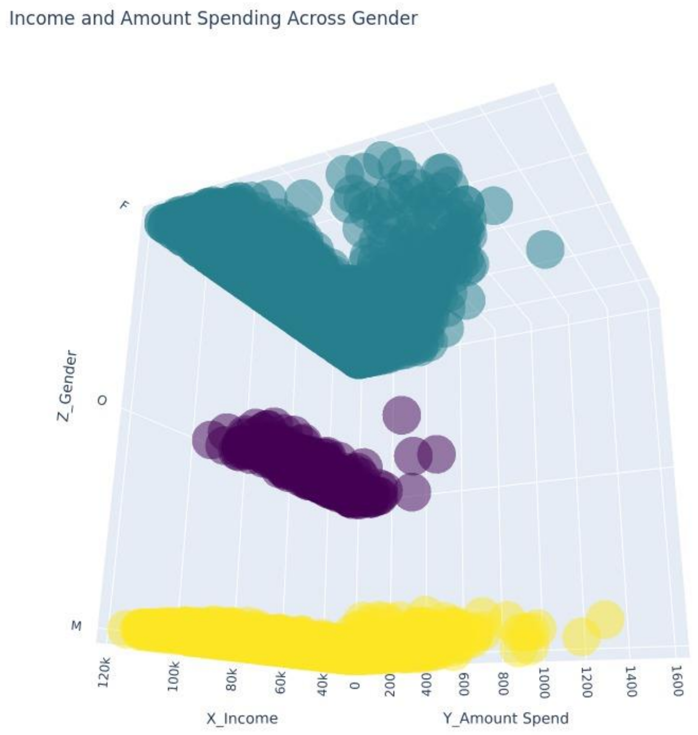


Visualization of Customers Amount Spending Distribution.

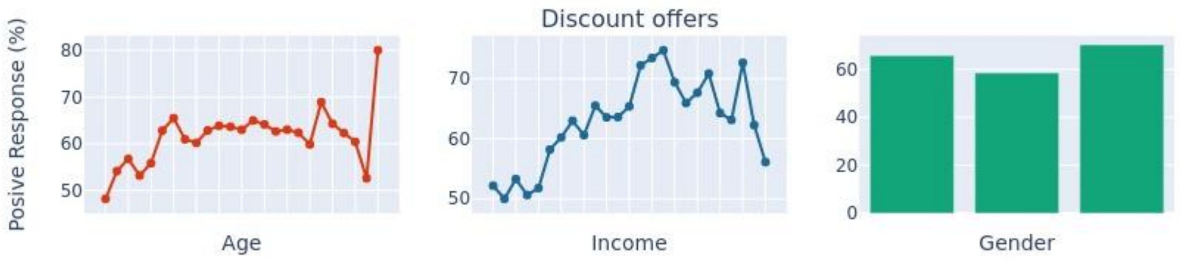


Data Exploration and Visualization

Visualization of Customers Income and Amount Spending Across Gender Distribution.

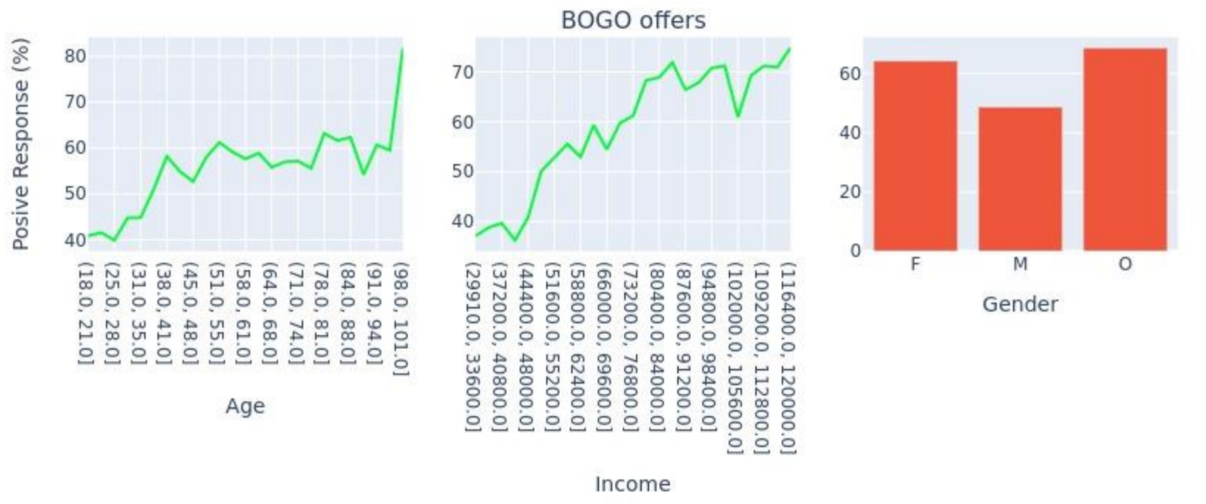


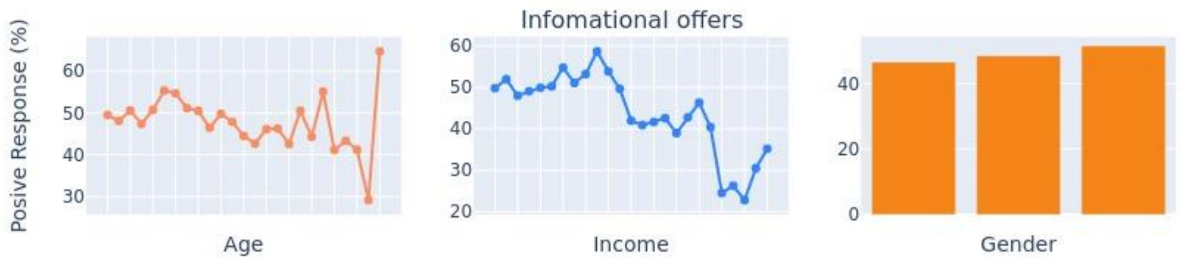
Visualization of how customers of different age, income and gender respond on different types of offers.



Data Exploration and Visualization

Visualization of how customers of different age, income and gender respond on different types of offers.



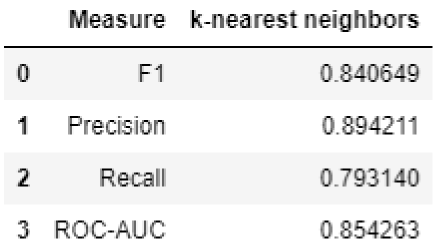


Model Building and Deployment

Algorithms and Techniques

Algorithm Model: [K-Nearest Neighbor(KNN) Classifier]

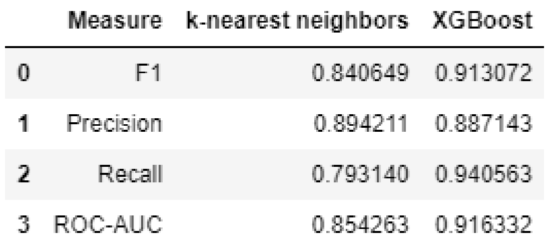
- **Using Our Training and Test Sets from Our Cleaned Dataset file we build K-Nearest Neighbor(KNN) classifier using Scikit-learn package.**
- **K Nearest Neighbor(KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms.**
- **KNN algorithm based on feature similarity approach.**
- **Evaluation Metric [KNeighborsClassifier]**



	Measure	k-nearest neighbors
0	F1	0.840649
1	Precision	0.894211
2	Recall	0.793140
3	ROC-AUC	0.854263

Algorithm Model: [XGBoostClassifier]

- **eXtreme Gradient Boosting XGBoost is an efficient and easy to use algorithm.**
- **XGBoost delivers high performance and accuracy as compared to other algorithms.**
- **It is also known as a regularized version of GBM (GradientBoosted Machines).**
- **It uses multiple CPU cores to execute the model (parallel processing).**
- **XGBoost model was picked (because of its high performance on all the metrics).**
- **Merging Both Evaluation Metric for [KNeighborsClassifier] and [XGBClassifier]**



	Measure	k-nearest neighbors	XGBoost
0	F1	0.840649	0.913072
1	Precision	0.894211	0.887143
2	Recall	0.793140	0.940563
3	ROC-AUC	0.854263	0.916332

Amazon SageMaker XGBoost Algorithm

Algorithm Model: SageMaker Algorithm [XGBoost]

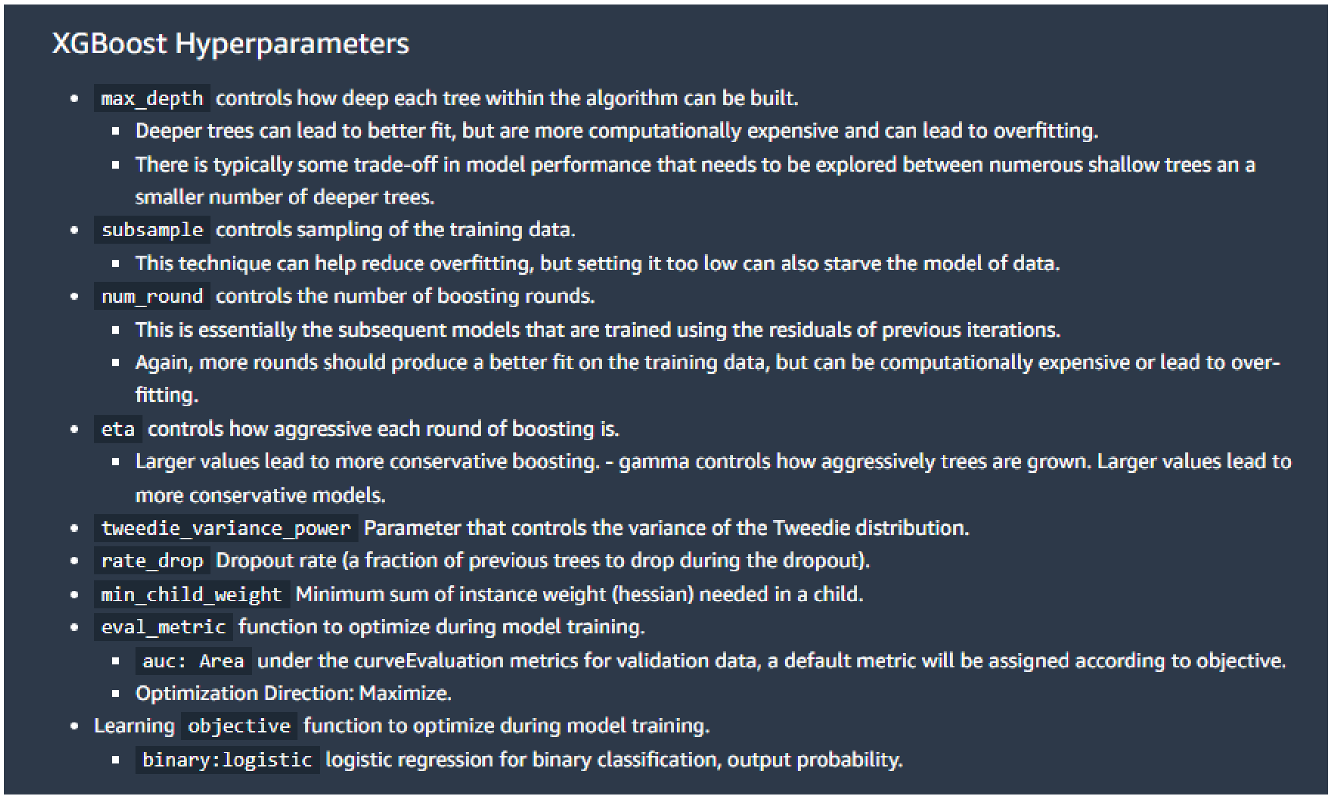
- Amazon SageMaker built-in XGBoost (eXtreme Gradient Boosting) algorithm performs well in machine learning because of its robust handling of a variety of data types, relationships, distributions, and the variety of hyperparameters that you can fine-tune.
- Using XGBoost as a built-in algorithm to run training scripts in your local environments.
- This is having a smaller memory footprint, better logging, improved hyperparameters validation, and an expanded set of metrics than the original versions.
- It also provides an XGBoost estimator that executes a training script in a managed XGBoost environment.

Steps for AWS SageMaker XGBoost (eXtreme Gradient Boosting) Model:

- Data upload to S3 bucket:
  - The XGBoost classifier requires the dataset to be written to a file and stored using Amazon S3.
  - So we first created training, validation and test set in a csv format with no headers and then uploaded to the s3 bucket.
  - Splitting the Data into training, validation, and test sets. This will help prevent us from over-fitting the model.
  - And allow us to test the model’s accuracy on data it hasn’t already seen.
  - The data format also should have the target variable in front followed by the rest of the feature variables.

- Creating XGBoost model:
  - Model artifacts:
    - The artifacts are the actual trees that are created during training i.e. the actual model itself.
  - Training code (container):
    - uses the training data that is provided and creates the model artifacts.
    - Using the XGBoost built-in algorithm to build an XGBoost training container `image_uris.retrieve`
  - Inference code (container):
    - uses the model artifacts to make predictions on new data.
    - The notebook was created and tested on a `ml.m4.xlarge` notebook instance.
  - After specifying the XGBoost image URI
    - We can use the XGBoost container to construct an estimator using the SageMaker Estimator and initiate a training job.
    - we can specify a few parameters
      - type of training instances `instance_type`
      - And how many instances `instance_type`

- **XGBoost Hyperparameters**



XGBoost Hyperparameters

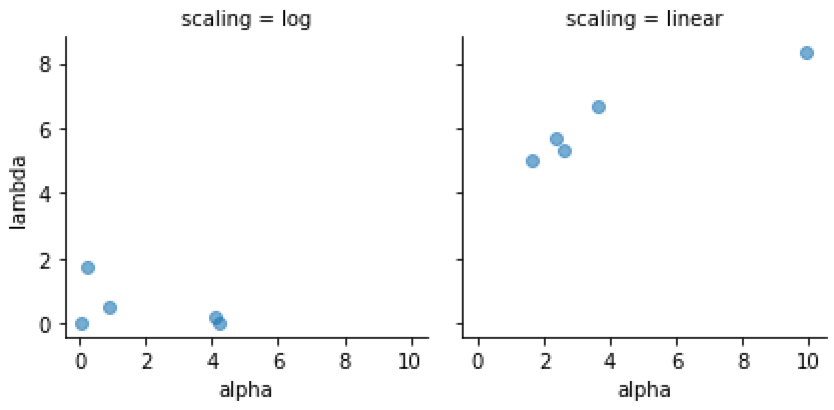
- `max_depth` controls how deep each tree within the algorithm can be built.
  - Deeper trees can lead to better fit, but are more computationally expensive and can lead to overfitting.
  - There is typically some trade-off in model performance that needs to be explored between numerous shallow trees an a smaller number of deeper trees.
- `subsample` controls sampling of the training data.
  - This technique can help reduce overfitting, but setting it too low can also starve the model of data.
- `num_round` controls the number of boosting rounds.
  - This is essentially the subsequent models that are trained using the residuals of previous iterations.
  - Again, more rounds should produce a better fit on the training data, but can be computationally expensive or lead to over-fitting.
- `eta` controls how aggressive each round of boosting is.
  - Larger values lead to more conservative boosting. - gamma controls how aggressively trees are grown. Larger values lead to more conservative models.
- `tweedie_variance_power` Parameter that controls the variance of the Tweedie distribution.
- `rate_drop` Dropout rate (a fraction of previous trees to drop during the dropout).
- `min_child_weight` Minimum sum of instance weight (hessian) needed in a child.
- `eval_metric` function to optimize during model training.
  - `auc: Area` under the curveEvaluation metrics for validation data, a default metric will be assigned according to objective.
  - Optimization Direction: Maximize.
- Learning `objective` function to optimize during model training.
  - `binary:logistic` logistic regression for binary classification, output probability.

- **Because we're training with the CSV file format we'll create TrainingInputs that our training function can use as a pointer to the files in S3.**
- **Train the XGBoost Model To do this we specify our S3 bucket's location that is storing our training data and validation data and pass it via a dictionary to the fit method.**

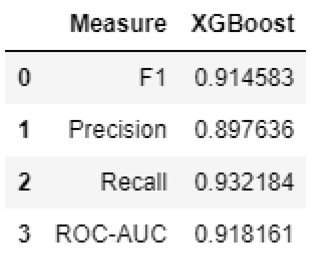
Amazon SageMaker Random Search and Hyperparameters Scaling with SageMaker XGBoost and Automatic Model Tuning.

- These features can improve our search for hyperparameters that perform well.
- Random search
  - Random search use to tell Amazon SageMaker to choose hyperparameters configurations from a random distribution.
  - The main advantage of random search is that all jobs can be run in parallel.
  - In contrast, Bayesian optimization, the default tuning method, is a sequential algorithm that learns from past trainings as the tuning job progresses. This highly limits the level of parallelism.
  - The disadvantage of random search is that it typically requires running considerably more training jobs to reach a comparable model quality.
  - In Amazon SageMaker, enabling random search is as simple as setting the Strategy field to Random when you create a tuning job.
- Using Random Search with Automatic Model Tuning allows us achieve faster results.
- Hyperparameters Scaling
  - Hyperparameters whose value can meaningfully span multiple orders of magnitude.
  - If I we manually tried a few different step sizes for a deep learning algorithm to explore the effect of varying this hyperparameter, we would likely choose powers of 10 (such as 1.0, 0.1, 0.01, …) rather than equidistant values (such as 0.1, 0.2, 0.3, …).
  - As know from experience that the latter is unlikely to change the behavior of the algorithm much.
  - For many hyperparameters, changing the order of magnitude yields much more interesting variation.

- <u>Logarithmic Scaling</u>
  - In Both cases we used logarithmic scaling.
  - Which is the scaling type that should be used whenever the order of magnitude is more important that the absolute value.
  - Used if a change, say, from 1 to 2 is expected to have a much bigger impact than a change from 100 to 101.
  - Due to the fact that the hyperparameters doubles in the first case but not in the latter.
- Analyze Tuning Job Results - After Tuning Job is Completed We can compare the distribution of the hyperparameters configurations chosen in the two cases.

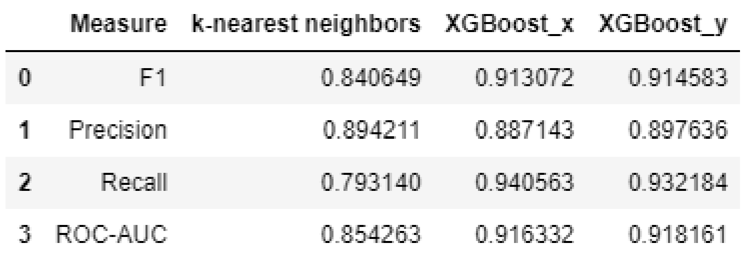


- Deploy the XGBoost Model:
  - We deploy the model on SageMaker by creating the endpoint.
- Evaluation Metric Make predictions:
  - we create predictions from the Tuned, trained and deployed model to evaluate it.



	Measure	XGBoost
0	F1	0.914583
1	Precision	0.897636
2	Recall	0.932184
3	ROC-AUC	0.918161

- **Clearing our AWS Resources:**
  - **Deleting Endpoints and Created Buckets for not causing any Additional Costs.**
- **Merging All Evaluation Metrics for [KNeighborsClassifier] and [XGBClassifier] with [Model Tuning XGBClassifier].**



	Measure	k-nearest neighbors	XGBoost_x	XGBoost_y
0	F1	0.840649	0.913072	0.914583
1	Precision	0.894211	0.887143	0.897636
2	Recall	0.793140	0.940563	0.932184
3	ROC-AUC	0.854263	0.916332	0.918161

Conclusion

The problem here was that we wanted to personalize sending the different kinds of offers to the customers because not all the offers should be sent to all the customers randomly. Hence, we turned this problem into a supervised learning task where we trained a classifier given the features related to the customer's profile, offer's portfolio and customer's transaction data, predict whether an offer will be converted/successful or not.
Amazon SageMaker has supported XGBoost as a built-in algorithm and as we took advantage of the open-source Amazon SageMaker XGBoost container, which has improved flexibility, scalability and extensibility. Also XGBoost to had a performance jump to 90% taking all the metrics together.
This is also because XGBoost is quite an efficient algorithm and known to outclass all other classifiers. It is capable of performing the three main forms of gradient boosting (Gradient Boosting (GB), Stochastic GB and Regularized GB) and it is robust enough to support fine tuning and addition of regularization parameters.
System-wise, the library's portability and flexibility allow the use of a wide variety of computing environments like parallelization for tree construction across several CPU cores; distributed computing for large models and Cache Optimization to improve hardware usage and efficiency.
Using random search with Automatic Model Tuning allows us to achieve faster results by running all tuning trials concurrently through random selection of hyperparameters combinations in the search space rather than the iterative approach used by default. Therefore, we should implement random search when speed is more important than obtaining the highest possible level of accuracy.