Udacity MLE Nanodegree Starbucks Promotion Analysis Capstone Project

by

Ting Lu

May 2024

Table of Contents

Definition

Project Overview

The idea of the project falls under the domain of Marketing and recommendation systems. The Starbucks company has provided a simulated dataset of its reward application including user profiles, offer types and some event-based data representing the transactions of these users. The idea of the project was to first analyze the provided dataset then build a binary classifier using machine learning methods. The binary classifier can be used by the application where given the user's information, and the offer type and information, it can predict whether the user would complete that offer or not and provide a probability score for its prediction. This might help the application in customizing its offers and directing it to its suitable audience.

Problem Statement

The famous Starbucks company has provided a mobile rewards application for their customers. The goal of the mobile application is to deliver certain offers to the users to encourage them to buy products from Starbucks. Since Starbucks has a variety of customer segments it might be hard to select the best offer for each user based on his demographic data. So, the company provided simulated data that simulates the real users of the application given their demographic data, their transactions, and details about some of their provided offers. My goal is to use the demographic data and the offer info first to analyze the different customer segments and how they are affected by different offers then to build a machine learning model, a binary classifier, that predicts whether a user might accept an offer given the user's demographic data and the offer details.

Metrics

Since the problem is a binary classification problem, predicting whether an offer might be completed or not, the main metrics that are used to evaluate the produced model are the following:

- **Accuracy**: Measures how good the model performs given what the model predicts.
- **F1-Score**: Added to quantify the model accuracy in case of any imbalance in the dataset, the F1-score is calculated as the harmonic mean between precision and recall.

Analysis

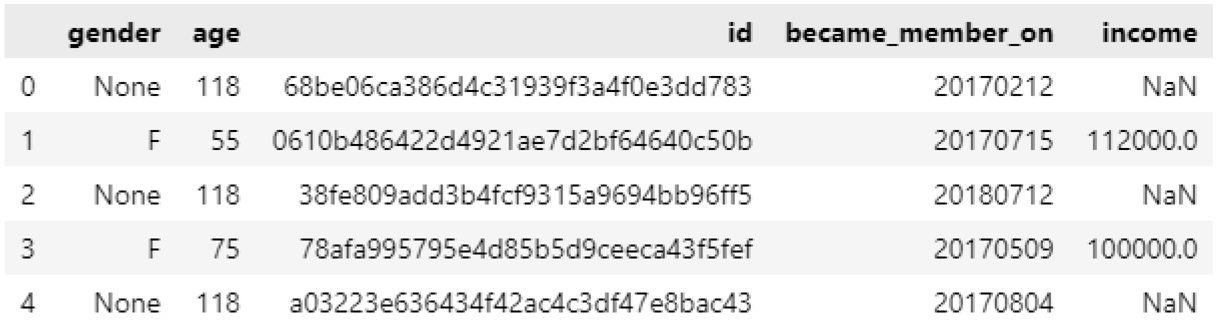
Data Exploration

The simulated dataset provided by the rewards application is given in the form of 3 separate JSON files as follows:

- **Profile JSON file**
  - Has demographic data about the users (age, gender, income, … etc.).
  - The simulated data has 17000 rows of different users with different demographic data.
  - A quick analysis of the dataset shows that there is some missing data, especially in the income and gender columns.
- **Portfolio JSON file**
  - Containing offer ids and metadata about each type of offer.
  - There are 3 main types of offers provided in the dataset.
    - Buy One Get One (BOGO): states that the user must pay a certain amount for a product to receive a reward of the same price.
    - Discounts: states that the user has a certain discount if he pays a certain amount during the offer duration.
    - Informational: Just advertising a certain product to influence the user to buy that product.
  - For each offer we have other information such as the duration of the offer, the difficulty (amount that needs to be paid) and the reward.
  - There are 10 combinations of offer types, durations, difficulty, and rewards provided in the dataset – only a subset of what the actual app offers.
- **Transcript JSON file**
  - Has data about the transactions and events made by different users.
  - For each user we have mainly 4 types of events.
    - Offer received: recorded whenever a user receives a certain offer.
    - Offer viewed: recorded whenever a user views the offer received.
    - Transaction: recorded whenever a user pays an amount of money through the app.
    - Offer Completed: recorded whenever a user completes an offer within the duration of that offer even if the user wasn't influenced by that offer.

Exploratory Visualization

- **Profile**
  The following figure shows the structure of the loaded data frame for the profile data before any preprocessing is done:

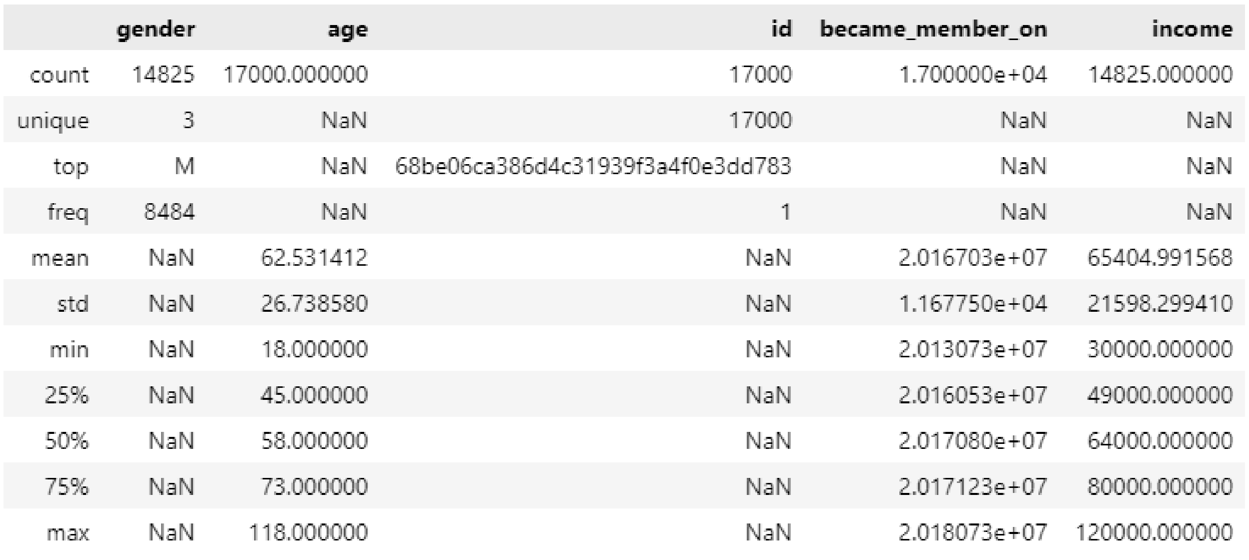


	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43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

The following is to be noted:

- The **gender** and **income** columns have some missing (None, Nan) values.
- The **became_member_on** column needs to be reformatted.
- The profile data frame has 17000 rows representing the information of 17000 unique users for the application.

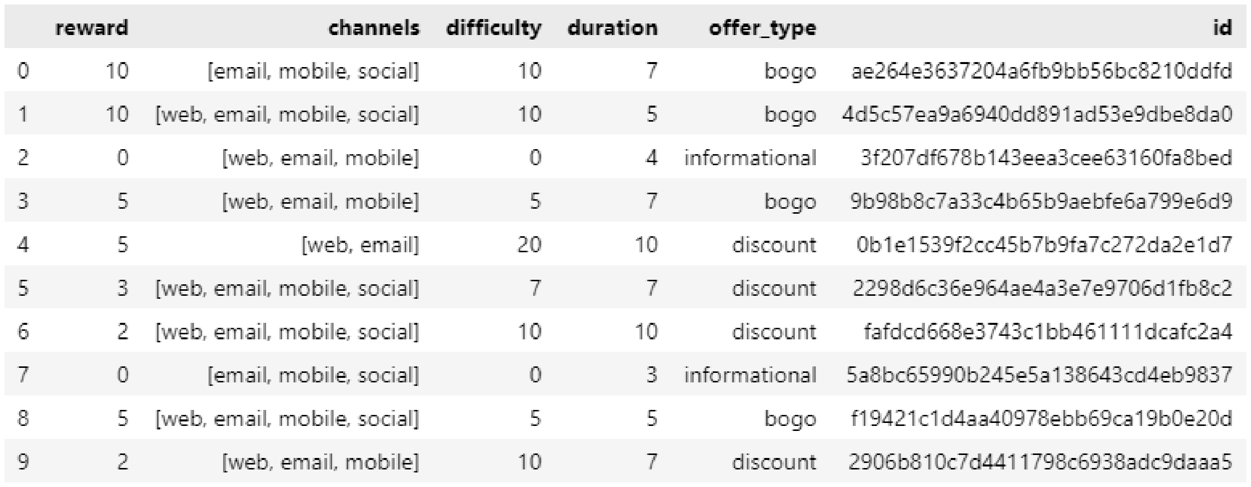
The following figure shows some of the statistics for the dataset columns:



	gender	age	id	became_member_on	income
count	14825	17000.000000	17000	1.700000e+04	14825.000000
unique	3	NaN	17000	NaN	NaN
top	M	NaN	68be06ca386d4c31939f3a4f0e3dd783	NaN	NaN
freq	8484	NaN	1	NaN	NaN
mean	NaN	62.531412	NaN	2.016703e+07	65404.991568
std	NaN	26.738580	NaN	1.167750e+04	21598.299410
min	NaN	18.000000	NaN	2.013073e+07	30000.000000
25%	NaN	45.000000	NaN	2.016053e+07	49000.000000
50%	NaN	58.000000	NaN	2.017080e+07	64000.000000
75%	NaN	73.000000	NaN	2.017123e+07	80000.000000
max	NaN	118.000000	NaN	2.018073e+07	120000.000000

The statistics show that precisely 2175 values are missing from both the age and income columns. These values might need to be imputed so that I don't lose data of many users of the application.

- **Portfolio**
  The following figure shows the structure of the loaded data frame for the portfolio data before any preprocessing is done:



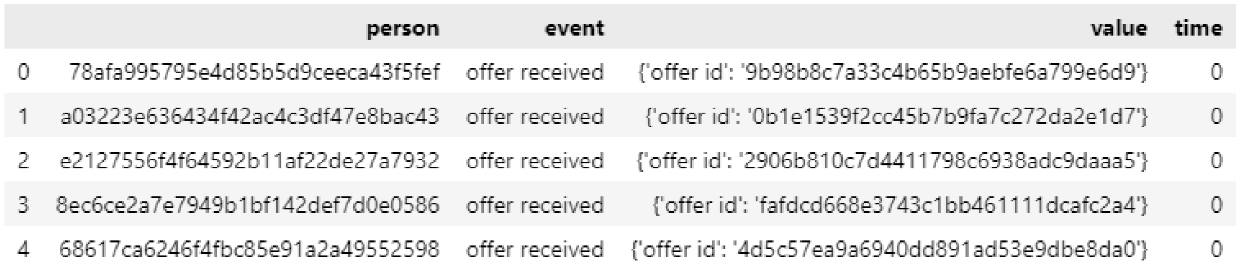
	reward	channels	difficulty	duration	offer_type	id
0	10	[email, mobile, social]	10	7	bogo	ae264e3637204a6fb9bb56bc8210ddfd
1	10	[web, email, mobile, social]	10	5	bogo	4d5c57ea9a6940dd891ad53e9dbe8da0
2	0	[web, email, mobile]	0	4	informational	3f207df678b143eea3cee63160fa8bed
3	5	[web, email, mobile]	5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

The following is to be noted:

- There are only 10 different offer types available in the dataset.
- The **channels** column is given as a list of channels, so it needs to be converted or spread across several columns.
- The **reward**, **difficulty** and **duration** columns are small integer numbers.
- The **offer_type** column is a categorical column with 4 unique values.
- There is no missing data in the Portfolio data frame.

- **Transcript**

The following figure shows the structure of the loaded data frame for the transcript data before any preprocessing

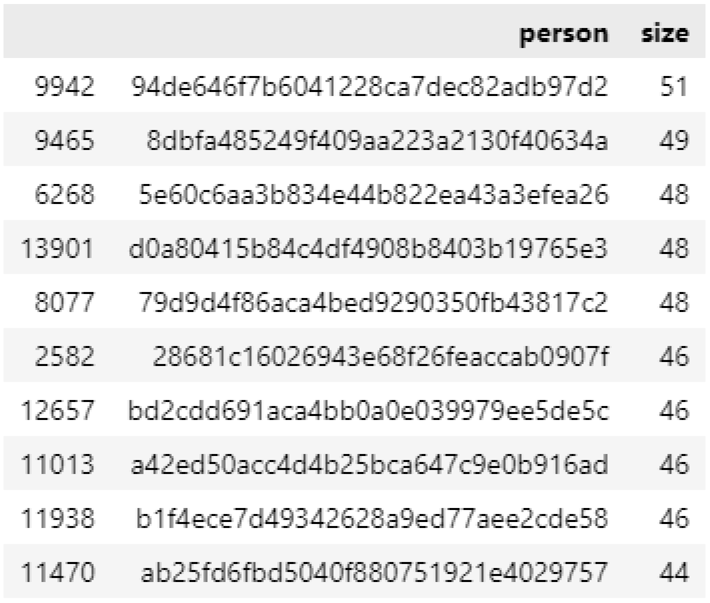


	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}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

The following is to be noted:

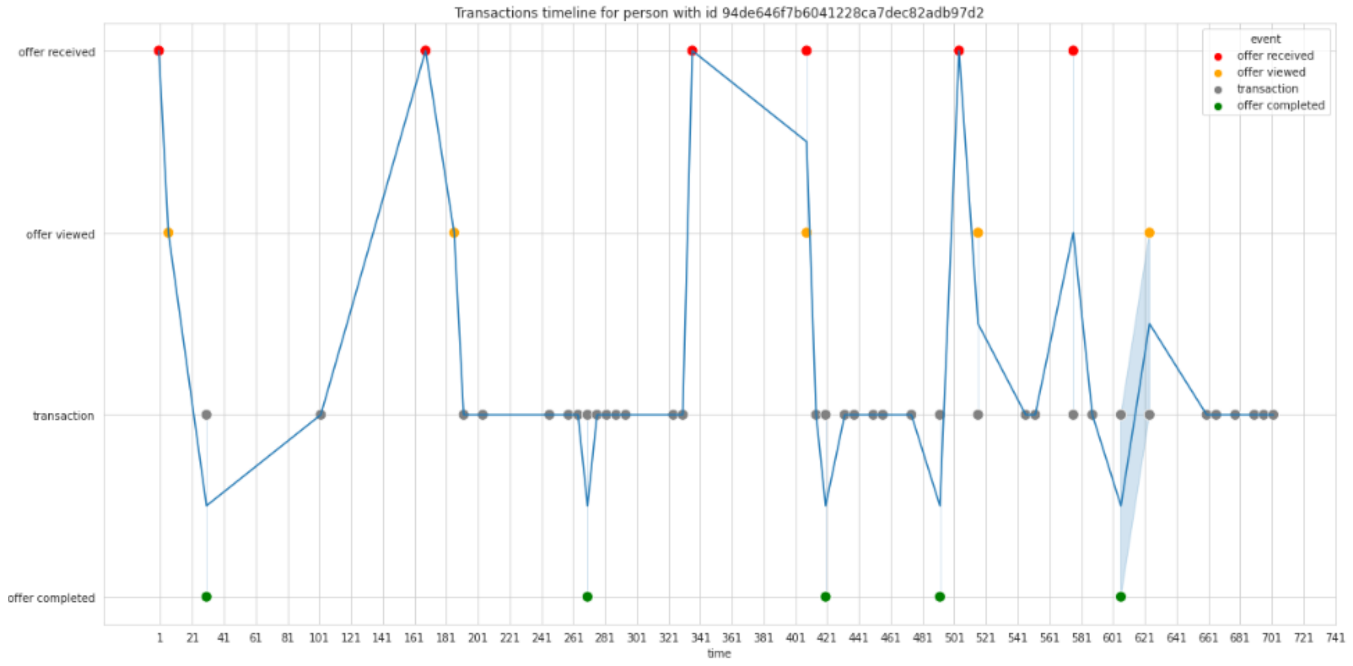
- The transcript data is an event-based data. I might need to convert it first to a tabular format to make the most out of it.
- The **value** column is given as a dictionary that has a different value according to the event. I might need to spread it across different columns.
- There is a time stamp for every event that seems to be recorded relative to a certain point in time starting at zero.

The data can be grouped by the unique person ids to create a group per user/person representing his/her own transactions, doing so and showing the first 10 persons with the most number of transactions gives the following table:



	person	size
9942	94de646f7b6041228ca7dec82adb97d2	51
9465	8dbfa485249f409aa223a2130f40634a	49
6268	5e60c6aa3b834e44b822ea43a3efea26	48
13901	d0a80415b84c4df4908b8403b19765e3	48
8077	79d9d4f86aca4bed9290350fb43817c2	48
2582	28681c16026943e68f26feaccab0907f	46
12657	bd2cdd691aca4bb0a0e039979ee5de5c	46
11013	a42ed50acc4d4b25bca647c9e0b916ad	46
11938	b1f4ece7d49342628a9ed77aee2cde58	46
11470	ab25fd6fbd5040f880751921e4029757	44

It is apparent that the person with the most transactions in the transcript has 51 recorded events in the dataset. Plotting these events as a time series yields the following plot:



From the above plot I noted the following:

- A person doesn’t have to complete a previous offer to receive another offer.
- A person can receive two offers of the same type.
- A person can do transactions on the application without any offer received.

Algorithms and Techniques

Since the problem of predicting the completion of the offer was scaled down to a simple binary classification problem where a tabular dataset with a binary target column is given. I decided to move with the AutoML approach as taught in course 1 to explore the largest number of models possible. For AutoML I decided to use the autogluon package by Amazon.

For the basic usage of autogluon, a group of arguments need to be specified namely the presets. I decided to use the **'best quality'** preset that aims to produce the best possible models at the expense of increasing the training time. I also set the **'save space'** argument to be true since the data is to be saved to S3 and the final model is to be deployed to an endpoint so I need to optimize the final predictor size for deployment.

After experimenting with multiple tree models that will be presented along with their results in the next section, autogluon created a weighted ensemble of the best performing models and stored the model artifacts on S3.

Finally using the Sagemaker SDK, I deployed the model to an endpoint. The endpoint was later queried using the test set to evaluate the model.

Benchmarks

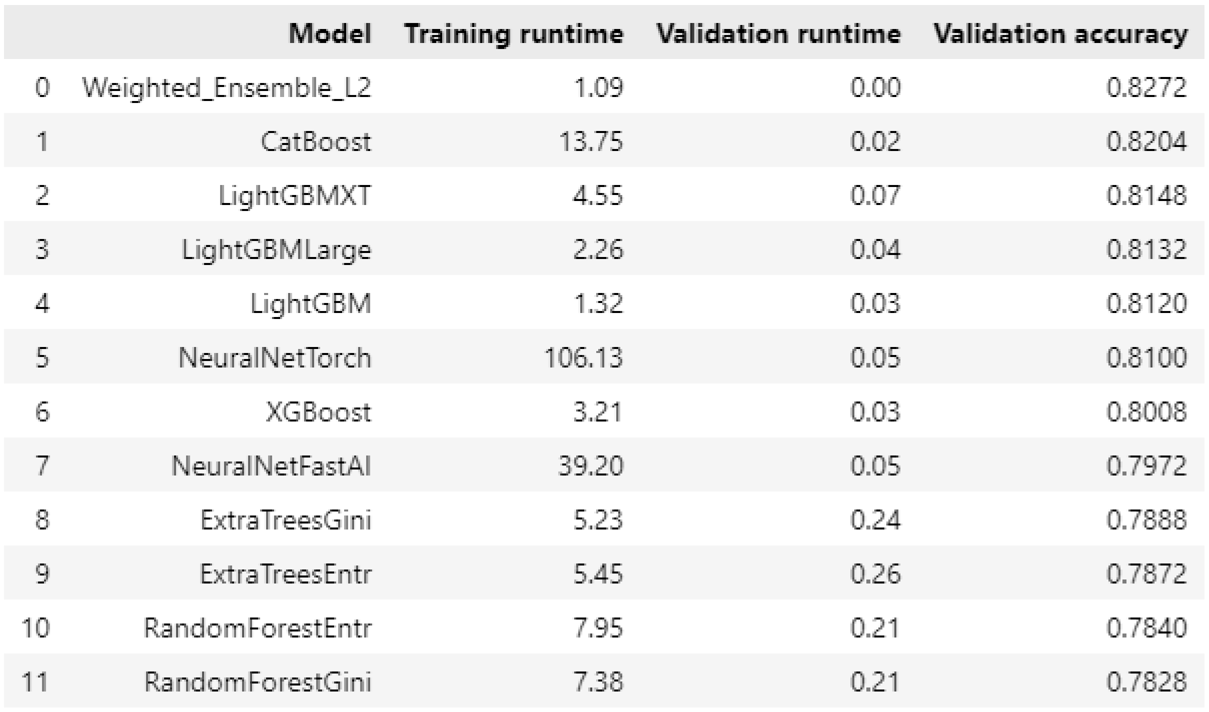
The following benchmark table's information was copied from the logs printed by the autogluon training job:

Model	Training runtime (sec)	Validation runtime (sec)	Validation accuracy
LightGBMXT	4.55	0.07	0.8148
LightGBM	1.32	0.03	0.812
RandomForestGini	7.38	0.21	0.7828
RandomForestEntr	7.95	0.21	0.784
CatBoost	13.75	0.02	0.8204
ExtraTreesGini	5.23	0.24	0.7888
ExtraTreesEntr	5.45	0.26	0.7872
NeuralNetFastAI	39.2	0.05	0.7972

XGBoost	3.21	0.03	0.8008
NeuralNetTorch	106.13	0.05	0.81
LightGBMLarge	2.26	0.04	0.8132
WeightedEnsemble_L2	1.09	0.0	0.8272

The validation set is extracted automatically by autogluon from the training set and kept as a held-out set, it is represented as a subset of 2500 rows out of 41723 rows with a percentage 6% of the training set. This is because the training data is a small dataset.

The following figure represents the same table sorted with respect to the accuracy score and represented as a pandas dataframe:



	Model	Training runtime	Validation runtime	Validation accuracy
0	Weighted_Ensemble_L2	1.09	0.00	0.8272
1	CatBoost	13.75	0.02	0.8204
2	LightGBMXT	4.55	0.07	0.8148
3	LightGBMLarge	2.26	0.04	0.8132
4	LightGBM	1.32	0.03	0.8120
5	NeuralNetTorch	106.13	0.05	0.8100
6	XGBoost	3.21	0.03	0.8008
7	NeuralNetFastAI	39.20	0.05	0.7972
8	ExtraTreesGini	5.23	0.24	0.7888
9	ExtraTreesEntr	5.45	0.26	0.7872
10	RandomForestEntr	7.95	0.21	0.7840
11	RandomForestGini	7.38	0.21	0.7828

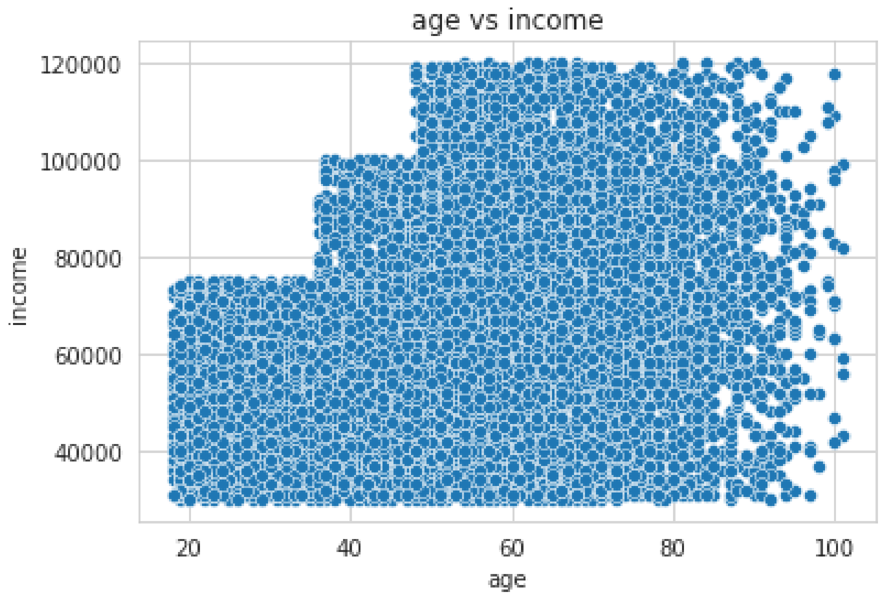
From the table, it appears that the models that have the most potential from the set of experimented models are the **CatBoost** and **LightGBM** models**.**

Methodology

Data Preprocessing

Profile

- Imputing values for income:
    - Age was plotted against income to visualize whether a relation exists between both as shown in the following figure:



    - From the figure, it appears that the users can be segmented into three different age groups: under 40 years, between 40 and 50 years and above 50 years.
    - For each group I can find the average (mean) income for the group and use it to impute the missing values for the income.
- Imputing values for gender:
    - After imputing values for the income column with the mean income according to the age group, only the gender column now has missing values.
    - To impute the missing values of a gender column, a simple decision tree algorithm is used to predict the gender of a user based on his age & income.
- Changing the format of the **became_member_on** column:
    - The became_member_on column was parsed as a datatime object and then spread to 3 separate columns namely the year, month and day columns.

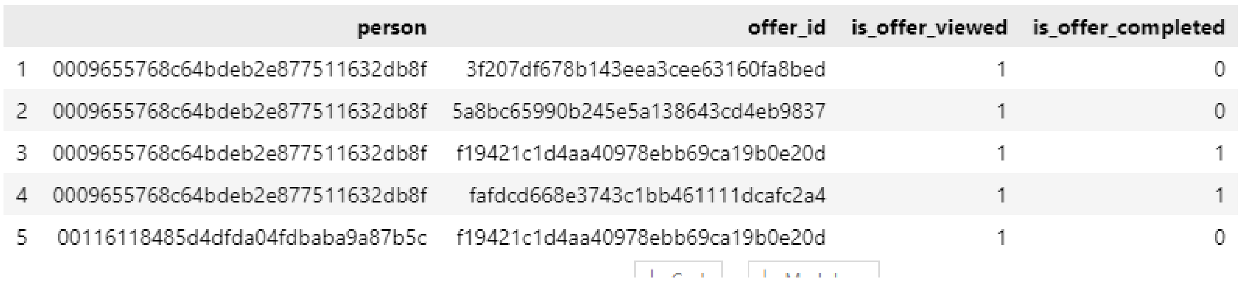
Portfolio

- The channels column list values were split into 4 separate Boolean type columns namely (email, mobile, web & social).
- These 4 columns replaced the original channels column.

Transcript

- The value column that had values in the forms of dictionaries with two unique and mutually exclusive keys according to the event type was split into 2 columns namely:
    - Offer Id if the event type is one of offer received, offer viewed or offer completed.

    - Amount if the event type is transaction.
- The transcript was grouped by the values of the person columns to create a group of transactions per user, then any user with no offers viewed had his transactions removed from the transcripts data since he wasn't influenced by any offer.
- The next step was to convert the event-based data to a tabular data with 4 columns (Person Id, Offer Id, Offer Viewed, Offer Completed.
- All transaction events were first removed from the data since they aren't relevant to the usecase.
- The data was then grouped by the person & offer columns to produce a group per person and offer. Each group was then iterated using the following algorithm:
    - If the event was an offer received event add a row to the tabular data table with the offer viewed and the offer completed columns set to False.
    - If the event was an offer viewed event associate the event with the closest offer back in time of the same offer id that wasn't viewed.
    - If the event was an offer completed event associate the event with the closest offer back in time of the same offer id that wasn't completed.
- Next All offers that were completed without being viewed were removed from the tabular data since they are assumed to not have influenced the user.



	person	offer_id	is_offer_viewed	is_offer_completed
1	0009655768c64bdeb2e877511632db8f	3f207df678b143eea3cee63160fa8bed	1	0
2	0009655768c64bdeb2e877511632db8f	5a8bc65990b245e5a138643cd4eb9837	1	0
3	0009655768c64bdeb2e877511632db8f	f19421c1d4aa40978ebb69ca19b0e20d	1	1
4	0009655768c64bdeb2e877511632db8f	fafdcd668e3743c1bb461111dcafc2a4	1	1
5	00116118485d4dfda04fdbaba9a87b5c	f19421c1d4aa40978ebb69ca19b0e20d	1	0

Joining the data frames together

- The three data frames were joined together after processing them where every person Id was replaced by the person information and every offer id was replaced by the offer details.
- The **is_offer_viewed** column was dropped from the dataset since it has only one value which is 1 (True)
- Duplicate rows were detected and removed from the dataset.
- Any rows having the same exact values for its features, but different values for the label (**is_offer_completed**) were completely removed to avoid any ambiguity in the data.
- The final dataset had 46358 rows and 15 columns including the target column.

Implementation

Model Training:

To implement autogluon, I used the Sagemaker SDK. A Sagemaker estimator is created and given the image Uri of the **'autogluon classification ensemble'** using the following configuration:

```
from sagemaker.estimator import Estimator


tabular estimator = Estimator(
    role=aws_role,
    image_uri=train_image_uri,
    source_dir=train_source_uri,
    model_uri=train_model_uri,
    entry_point="transfer_learning.py",
    instance_count=1,
    instance_type=="ml.m5.large",
    max_run=40*60,
    hyperparameters=hyperparameters,
    output_path=s3_output_location,
)
```

The configuration has the following arguments:

- **role**: the role assigned to the Sagemaker notebook instance,
- **image uri**: the downloaded image Uri for the **'autogluon classification ensemble'** , it has all the necessary dependencies to run the autogluon AutoML algorithm.
- **source dir**: the source code for running autogluon downloaded from the **'autogluon classification ensemble'** registry.
- **model uri**: is downloaded from the **'autogluon classification ensemble'** registry and is a dummy model in its case.
- **entry point**: the script used to start the training and the fitting process.
- **instance count**: the number of instances used to train the model, since the dataset is not too large, I used only 1 instance.
- **instance type**: The type of instance used to train the model, since the dataset is small, I used a relatively small instance that is the **'ml.m5.large',** that has the following specs, 2 vCPUs and 8 GiB of memory.
- **max run**: Is the maximum time the training job can run, to avoid extra costs I set the time to be 40 minutes maximum which proved more than enough.
- **hyperparameters**: are the hyperparameters used by the autogluon algorithm, I set the hyperparameters as follows

Evaluation metric	Accuracy
Presets	Best quality
Auto stack	True
Num bag folds	0
Num bag sets	1
Num stack levels	0
Refit full	False
Set best to refit full	False
Save space	True
Verbosity	2

Most of the hyperparameters were kept with their default values, I changed the following ones:

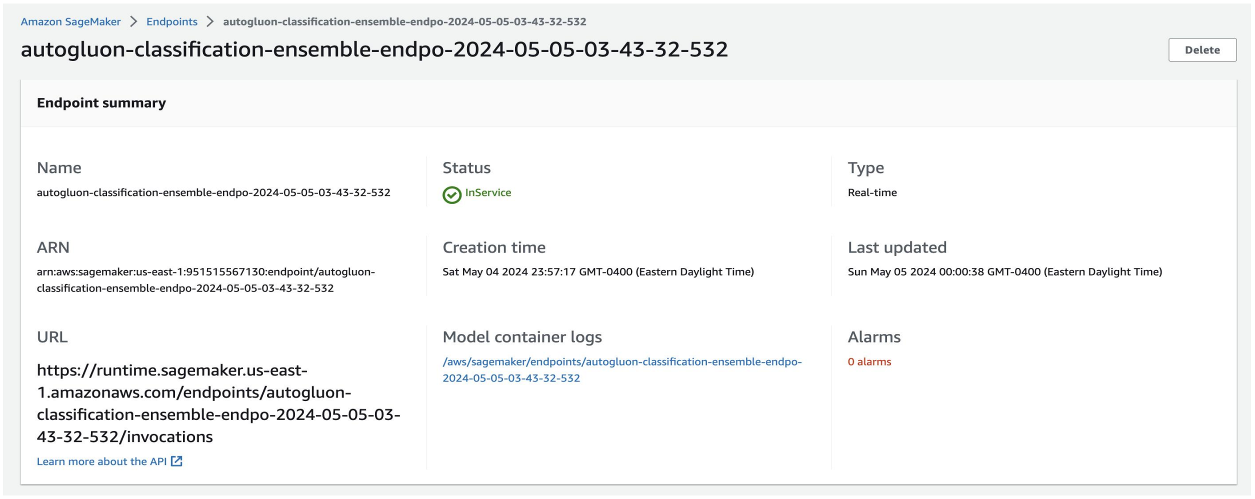
- **Evaluation metric**: Accuracy, since it is a binary classification problem, and the data is somewhat balanced based on the target column.
- **Presets**: Best quality, to attempt to achieve the best quality possible by applying ensemble and stacking techniques.
- **auto stack**: True, to utilize bagging and multi-layer stack to boost accuracy.
- **save space**: True, to optimize the model size for later deployment to an endpoint

Model Deployment:

I used the Sagemaker SDK, to deploy the model, the deployment was done based on the following configuration:

```
endpoint_name = name_from_base(f"{train_model_id}-endpoint")


# Use the estimator from the previous step to deploy to a SageMaker endpoint
predictor = tabular_estimator.deploy(
    initial_instance_count=1,
    instance_type=="ml.m5.large",
    entry_point="inference.py",
    image_uri=deploy_image_uri,
```

The fitted estimator was used to deploy the endpoint, the deployed model is automatically the best model, which is the Weighted Ensemble, An initial instance count of 1 was used since there is no much traffic at the deployment of the model, autoscaling can be later applied. An instance type of **‘ml.m5.large’** was chosen which is enough for the computations of the model in terms of CPU and memory requirements.


Amazon SageMaker > Endpoints > autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532
Delete

autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532

Endpoint summary

Name	Status	Type
autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532	InService	Real-time
ARN	Creation time	Last updated
arn:aws:sagemaker:us-east-1:951515567130:endpoint/autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532	Sat May 04 2024 23:57:17 GMT-0400 (Eastern Daylight Time)	Sun May 05 2024 00:00:38 GMT-0400 (Eastern Daylight Time)
URL	Model container logs	Alarms
https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532/invocations Learn more about the API	/aws/sagemaker/endpoints/autogluon-classification-ensemble-endpo-2024-05-05-03-43-32-532	0 alarms

Model Querying:

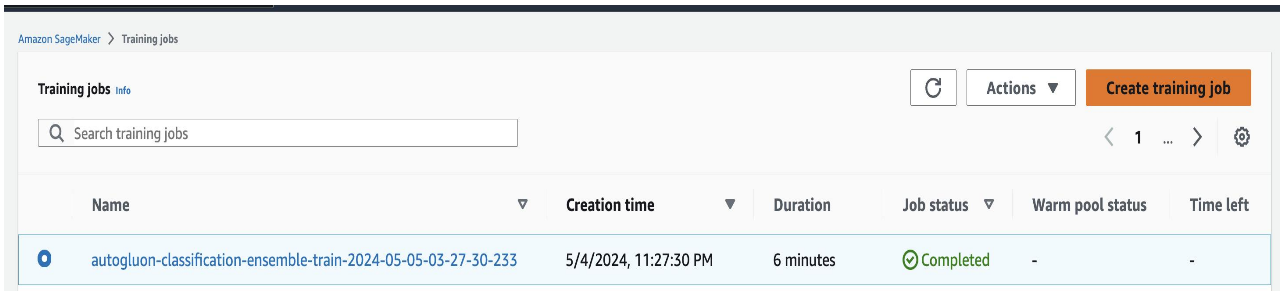
To query the endpoint after deployment, I needed to submit an HTTP request whose contents is a batch of rows from the test set in order to evaluate the model. The test set was divided into batches of 1500 rows each and each batch was used to invoke the endpoint as follows:

```
content_type = "text/csv"

def query_endpoint(encoded_tabular_data):

    # endpoint_name = endpoint_name

    client = boto3.client("runtime.sagemaker")

    response = client.invoke_endpoint(

        EndpointName=endpoint_name, ContentType=content_type,
Body=encoded_tabular_data

    )

    return response

def parse_response(query_response):

    model_predictions = json.loads(query_response["Body"].read())

    predicted_probabilities = model_predictions["probabilities"]

    return np.array(predicted_probabilities)

# split the test data into smaller size of batches to query the endpoint due to
the large size of test data.

batch_size = 1500

predict_prob = []

for i in np.arange(0, num_examples, step=batch_size):

    query_response_batch = query_endpoint(

        features.iloc[i : (i + batch_size), :].to_csv(header=False,
index=False).encode("utf-8")

    )

    predict_prob_batch = parse_response(query_response_batch)  # prediction
probability per batch

    predict_prob.append(predict_prob_batch)
```

Note: The scripts for training autogluon are adapted from the sagemaker example notebooks for autogluon tabular classification model.

Refinement

For autogluon algorithm refinement, I needed to submit multiple training jobs while changing some of the presets and hyperparameters either to optimize the space for deployment which resulted in about 100 MB of space saved or changing the evaluation metric from **'roc_auc'** to **'accuracy'** to achieve better results.



	Name	Creation time	Duration	Job status	Warm pool status	Time left
●	autogluon-classification-ensemble-train-2024-05-05-03-27-30-233	5/4/2024, 11:27:30 PM	6 minutes	Completed	-	-

Results

Model Evaluation and Validation

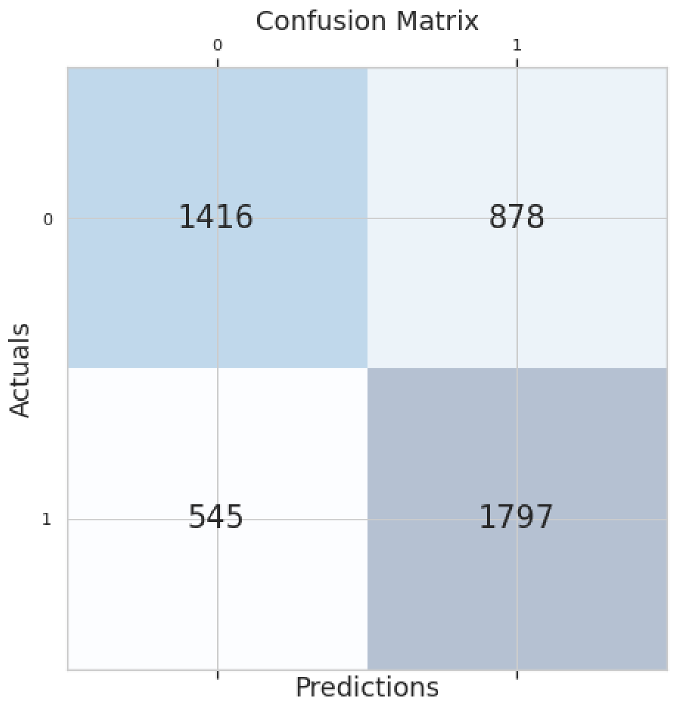
The dataset was split after preprocessing into a train, test split with a percentage 90%, 10% respectively. The test-set was held-out and not used by anyways during training the model. The test set was only used to evaluate the best model produced from the AutoML autogluon library.

During training the model, autogluon automatically extracted a held-out validation set of 2500 rows as mentioned in the Benchmarks section.

The following table shows the metrics used to evaluate the model on the held-out test set:

Metric	**Score**
Accuracy	69.3 %
F1-Score	71.63%

A confusion matrix was also plotted as follows which shows that most predictions resemble the actual labels:



Justification and future work

The model appears to achieve a decent accuracy, but not the highest possible accuracy, this might be for the following reasons:

- The dataset isn't very large, it consists of about 40,000 rows after processing the data, these rows might not cover all the possible information to train the model.
- The decisions taken during processing the dataset such as associating the offer completed and the offer viewed events to the closest non-completed and non-viewed received offer events of the same type and the same user as mentioned before.
- There were rows having the same set of features but different target labels. Although these rows were removed, it was still an indication that the features in the dataset weren't enough to represent the target function.
- AutoML has a lot of tuning possibilities. Although the best_quality preset was used, there might still be some better options and models that weren't explored.

As a conclusion, I think that the proposed problem of predicting whether or not a customer would complete an offer based on his demographic data and the offer details has a solution with machine learning that might have potential. Improving the data quality, feature engineering and trying more model options might achieve better results. The results produced automatically by autogluon can be further used to focus on certain types of models that seem to have better potential such as catboost and lightGBM models.