Machine Learning Engineer Nanodegree

Capstone Project

Ensiyeh Raoufi February 13th, 2022

I. Definition

Project Overview

Online advertising has gained significant attention in various platforms ranging from search engines, third-party websites, social media, and mobile apps. The prosperity of online campaigns is a challenge in online marketing and is usually evaluated by user response through different metrics, such as clicks on advertisement creatives, subscriptions to products, or purchases of items. Recent years have witnessed a significant increase in the number of studies using computational approaches, including machine learning methods, for user response prediction. User response prediction is considered an important task to evaluate the effectiveness of online advertising and recommender systems. Learning multiple factors regarding optimizing user preferences and commercial profits have the potential to shape the new trend in developing recommendation system algorithms [1]. Starbuck company also is using a variety of advertisement strategies like social media broadcasting, web inference, sending emails, and its mobile software's in-app advertising.

In this project, I have trained a model that predicts the customer cumulative transactions during a month, at the times that they receive, view, or complete a given Starbuck offer. It's a mini dataset and it is assumed that all offers are about purchasing only one product, although Starbucks has a varied range of products. Taking a good result from this model, the company can predict if they send a special offer to a customer with specified age, membership duration in months, and an estimated income, how much more the customer will spend to buy a product in that month. The result is directly related to the pure profit of the company, because we will understand that with certain monthly investment on special kinds of advertisements, how much more we will earn in that month. So I think it is worth solving this problem.

Problem Statement

My goal question is not like whether or not a customer would complete an offer, or would they view the offer and then buy some product or not (however I could design some questions like this), so I'm not faced with a classification problem. And also because I have the information to calculate the cumulative transactions of each customer, I should do a type of supervised learning. I'm using machine learning models to solve a regression problem that its result indicates:

How much more some customers will spend to buy a product, based on demographics and offer type they will receive/view during the time.

For solving this problem I first created a comprehensive table from all other data tables and worked on it as my data source. This table contains all the person ids with all of their offers and cumulative and total transactions information. Because I'm investigating the increasing amount of customer transactions when they interface an offer, I should remove the other rows that a customer does not receive/view/complete an offer. So a big part of this project consists of:

- Cleansing data and extracting information
- One-hot encoding of the necessary categorical data and Creating columns for each type of category
- Filling the NaN cells of each table with appropriate choices of amounts
- Concatenating columns and creating neat tables
- Merging all the tables to one comprehensive table that we can use as a source for our model training

When my source table is ready then we should solve the main problem. We should do the following jobs to get the results:

- Creating a Jupyter notebook instance in my AWS Sagemaker tab
- Open the Jupyter notebook
- Uploading the input files to the notebook and s3 bucket (when necessary)
- Creating a Python script .py file that does the training job
- Creating a Sagemaker Scikit_Learn estimator to fine-tune and fit the hyperparameters using our .py file as an entry point
- Training a Scikit_Learn Random Forest model based on the best hyperparameters
- Deploy the model to an endpoint
- Invoking the endpoint and making predictions on the test data
- Evaluating the model on some metrics like R-squared (R2), Mean squared error, and absolute error.
- Training the dataset with the auto-ML library Autogluon Tabular predictor to find a benchmark for our model
- Evaluating the model scores of this auto-ML predictor and comparing to our results
- Finding the best model for predicting our data

Metrics

I evaluated my models using metrics like the R2 score which is a suitable metric for my regression problem. We can use the R2 score that shows the performance of our model well. R2 (R-Squared) score is the coefficient of determination and calculation of R-squared requires several steps. This includes taking the data points (observations) of dependent and independent variables and finding the line of best fit, (here) from a regression model. From there you would calculate predicted values, subtract actual values and square the results. This yields a list of errors squared, which is then summed and equals the unexplained variance. In finance, an R-Squared above 0.7 would generally be seen as showing a high level of correlation, whereas a measure below 0.4 would show a

low correlation [2]. But because only R2 is not sufficient to judge a model's regression performance [3], I compared the mean squared and absolute errors of the training results of the models, too.

II. Analysis

Data Exploration

I read all the JSON files and made screenshots of all the raw data. Here you can their screenshots:

- reading transcript.json file:

```
transcript
```

	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
...	...	...	...	...
306529	b3a1272bc9904337b331bf348c3e8c17	transaction	{'amount': 1.5899999999999999}	714
306530	68213b08d99a4ae1b0dcb72aebd9aa35	transaction	{'amount': 9.53}	714
306531	a00058cf10334a308c68e7631c529907	transaction	{'amount': 3.61}	714
306532	76ddbd6576844afe811f1a3c0fbb5bec	transaction	{'amount': 3.5300000000000002}	714
306533	c02b10e8752c4d8e9b73f918558531f7	transaction	{'amount': 4.05}	714

306534 rows × 4 columns

By watching carefully at this table we can see that the "event" column consists of categorical data with 'offer completed', 'offer received', 'offer viewed', and 'transaction' as its category names. So if we care about this feature to be one of our features for training the model or other future works on the dataset, we should generate a label encoding scheme for mapping each category to a numeric value. But because our category is not ordinal and for example, if we assign the number 1 to the 'offer completed' then it has no preference to the number 4 that we assigned to the 'transaction' category, so we should do the one-hot encoding scheme that encodes or transforms this feature into 4 binary features which can only contain a value of 1 or 0. Also, the "value" column contains some dictionary-type data that its keys should be extracted and turned into some other new columns. Using a Pandas function that is named "json_normalized", I've extracted the information on the 'value' column of this table, and automatically one-hot encoded each of the keys to a column. After obtaining the unique types of events on the 'event' column, I've done one-hot encoding by applying a lambda function on this column. Finally, I filled all of the remaining NaNs with zero, because it has no effect on any other information on this table and is sensible. For example, if a customer does not do any transaction, its related amount cell is NaN, which can simply be zero because They didn't buy anything. Here is a piece of code of mine for cleansing this table:

```
#transcript_value contains each of keys of value column in a separate column
transcript_value = pd.json_normalize(transcript['value'])

#Merging 'offer id' and 'offer_id' columns to 'offers' column, because by type of event we can see
transcript_value['offers'] = transcript_value['offer id'].fillna(transcript_value['offer_id'])

#Deleting 'offer id' and 'offer_id' columns
transcript_value = transcript_value.drop(['offer_id', 'offer id'], axis = 1)

#deleting the value column from transcript
transcript = transcript.drop(['value'], axis = 1)

#concatenating the separated columns of value in the transcript_value to the transcript
transcript = pd.concat([transcript, transcript_value], axis=1)

#observing categorical data in event and offers columns
print("event types:", np.unique(transcript['event']))
print("offers ids:", transcript['offers'].unique())

#One-hot encoding of the events column and deleting the event itself
for chan in ['offer_completed', 'offer_received', 'offer_viewed', 'transaction']:
    transcript[chan] = transcript.event.apply(lambda x: 1 if x==chan.replace('_',' ') else 0)
transcript.pop('event')

#Filling the NaN in transcript table
transcript['amount'] = transcript['amount'].fillna(0)
transcript['reward'] = transcript['reward'].fillna(0)
```

And at last, you can see the edited version of my table in the following:

```
transcript
```

	person	time	amount	reward	offers	offer_completed	offer_received	offer_viewed	transaction
0	78afa995795e4d85b5d9ceeca43f5fef	0	0.00	0.0	9b98b8c7a33c4b65b9aebfe6a799e6d9	0	1	0	0
1	a03223e636434f42ac4c3df47e8bac43	0	0.00	0.0	0b1e1539f2cc45b7b9fa7c272da2e1d7	0	1	0	0
2	e2127556f4f64592b11af22de27a7932	0	0.00	0.0	2906b810c7d4411798c6938adc9daaa5	0	1	0	0
3	8ec6ce2a7e7949b1bf142def7d0e0586	0	0.00	0.0	fafdcd668e3743c1bb461111dcafc2a4	0	1	0	0
4	68617ca6246f4fbc85e91a2a49552598	0	0.00	0.0	4d5c57ea9a6940dd891ad53e9dbe8da0	0	1	0	0
...	...	...	...	...	...	...	...	...	...
529	b3a1272bc9904337b331bf348c3e8c17	714	1.59	0.0	NaN	0	0	0	1
530	68213b08d99a4ae1b0dcb72aebd9aa35	714	9.53	0.0	NaN	0	0	0	1
531	a00058cf10334a308c68e7631c529907	714	3.61	0.0	NaN	0	0	0	1
532	76ddbd6576844afe811f1a3c0fbb5bec	714	3.53	0.0	NaN	0	0	0	1
533	c02b10e8752c4d8e9b73f918558531f7	714	4.05	0.0	NaN	0	0	0	1

534 rows × 9 columns

- reading portfolio.json file:

```
#View the raw table of portfolio
portfolio
```

	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

For this table, we should also do the one-hot encoding on the “channels” and “offer_type” columns. We should also notice that the data on the “channels” column is of a list type. So we should change our lambda function to apply on this column. Here is a piece of my code for editing this table:

```
#One-hot encoding of the channels column and deleting the channels column itself
for chan in ['web', 'email', 'mobile', 'social']:
    portfolio[chan] = portfolio.channels.apply(lambda x: 1 if chan in x else 0)
portfolio.pop('channels')

#One-hot encoding of the offer_type column and deleting the offer_type column itself
for chan in ['bogo', 'discount', 'informational']:
    portfolio[chan] = portfolio.offer_type.apply(lambda x: 1 if x==chan else 0)
portfolio.pop('offer_type')

#Renaming columns to avoid in common names with the profile table
portfolio = portfolio.rename(columns={'reward': 'offer_reward', 'duration': 'offer_duration',
                                      'difficulty': 'offer_difficulty'})
```

The edited portfolio table is like this:

We can understand from this table that there are 10 types of offers.

- 4 BOGO (buy one get one free) offers
- 4 types of discount offers
- 2 informational types of offers

Informational offers give no reward and rewards and durations of different BOGO offers vary. Offer difficulty means how much a customer should spend to receive the reward of the offer. Also, offer durations are presented in days.

- reading profile.json file:

```
profile
```

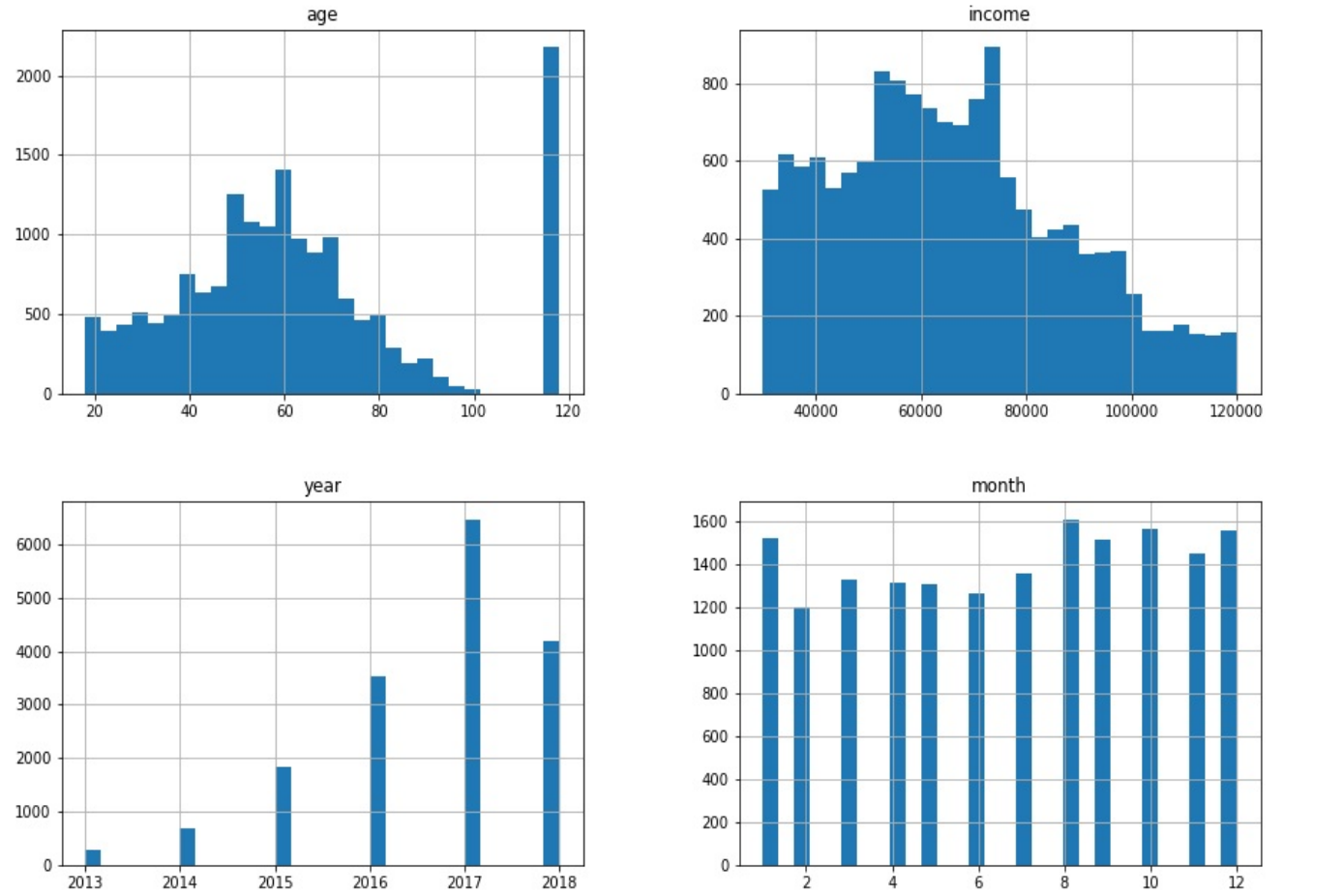
	gender	age	id	became_member_on	income
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	20170212	NaN
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	20180712	NaN
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
4	None	118	a03223e636434f42ac4c3df47e8bac43	20170804	NaN
...	...	...	...	...	...
16995	F	45	6d5f3a774f3d4714ab0c092238f3a1d7	20180604	54000.0
16996	M	61	2cb4f97358b841b9a9773a7aa05a9d77	20180713	72000.0
16997	M	49	01d26f638c274aa0b965d24cefe3183f	20170126	73000.0
16998	F	83	9dc1421481194dcd9400aec7c9ae6366	20160307	50000.0
16999	F	62	e4052622e5ba45a8b96b59aba68cf068	20170722	82000.0

17000 rows × 5 columns

By plotting histograms of this raw table we can see the following plots:

```
#View the raw data plots before filling the NaNs
profile.hist(bins=30, figsize=(15,10))
```

```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'income'}>],
       [<AxesSubplot:title={'center':'year'}>,
        <AxesSubplot:title={'center':'month'}>]], dtype=object)
```



Observing these plots we can see that the customers are between about twenty to 100 years old! The people with the age of 118, are those whose age is not entered in the database. The annual income of customers is almost in the range of 15000-120000. There are a high amount of customers who became a member of Starbucks in 2017. People tend to join the membership of Starbucks in

the second half of every year.

In this table, we should be careful about NaN cells and choose a strategy for filling or removing the rows that contain NaN. I prefer to fill the NaN cells of a column with the help of the other cells of the column. For the gender column, I prefer to fill the NaN cells with a random choice of “F” or “M”. We can ignore the Other genders mentioned by “O” because of their minority in the dataset. I also prefer to fill the NaN cells of the “income” column with the average of the column. We should change the cells that are filled with 118 on the “age” column because they are representing the NaN cells. For this column I also choose the average amount of the cells that are not 118, to fill these cells. I’ve done all this job and here you can see some part of my code for modifying this table:

```
#Calculating the average of people whose ages are not 118
mean_age = profile['age'][profile['age']!=118].mean()
mean_age = int(mean_age)

#fill the NaN cells of the age column with average age
profile.loc[profile.age == 118, 'age'] = mean_age

#fill the NaN cells of the income column with the average income
mean_income = round(np.mean(profile['income']),0)
profile['income'] = profile['income'].fillna(mean_income)

O_count = profile[profile['gender']=='O']['gender'].count()
print(f"There are {O_count} persons having non obvious gender")

#fill the NaN cells of gender column in random, we don't Other genders for random choicing because of their minorit

num_na = profile['gender'].isna().sum()  # number of missing cases
profile.loc[profile['gender'].isna(), 'gender'] = random.choices(["M", "F"], k=num_na)
```

```
There are 212 persons having non obvious gender
```

```
#One-hot encoding of the gender column
print("gender types: ", profile.gender.unique())

for chan in ['M', 'F', 'O']:
    profile[chan] = profile.gender.apply(lambda x: 1 if x==chan else 0)
profile.pop('gender')

#Renaming to appropriate names
profile = profile.rename(columns={'F': 'Female', 'M': 'Male', 'O': 'Other'})
```

```
gender types:  ['F' 'M' 'O']
```

Our profile table looks like this after editing:

```
profile
```

	age	id	income	membership_duration	Male	Female	Other
0	54	68be06ca386d4c31939f3a4f0e3dd783	65405.0	72	0	1	0
1	55	0610b486422d4921ae7d2bf64640c50b	112000.0	67	0	1	0
2	54	38fe809add3b4fcf9315a9694bb96ff5	65405.0	55	0	1	0
3	75	78afa995795e4d85b5d9ceeca43f5fef	100000.0	69	0	1	0
4	54	a03223e636434f42ac4c3df47e8bac43	65405.0	66	0	1	0
...	...	...	...	...	...	...	...
16995	45	6d5f3a774f3d4714ab0c092238f3a1d7	54000.0	56	0	1	0
16996	61	2cb4f97358b841b9a9773a7aa05a9d77	72000.0	55	1	0	0
16997	49	01d26f638c274aa0b965d24cefe3183f	73000.0	73	1	0	0
16998	83	9dc1421481194dcd9400aec7c9ae6366	50000.0	83	0	1	0
16999	62	e4052622e5ba45a8b96b59aba68cf068	82000.0	67	0	1	0

17000 rows × 7 columns

After merging all the edited tables, applying sorting, deleting unnecessary columns, and resetting indexes here we achieve our source table that I named my_starbuck and its all columns are like this:

```
#View the edited table to work with
my_starbuck
```

	person	time	cumulative_transaction	reward	offers	offer_completed	offer_received
0	0009655768c64bdeb2e877511632db8f	7.0	0.00	0	5a8bc65990b245e5a138643cd4eb9837	0	1
1	0009655768c64bdeb2e877511632db8f	8.0	0.00	0	5a8bc65990b245e5a138643cd4eb9837	0	0
2	0009655768c64bdeb2e877511632db8f	14.0	22.16	0	3f207df678b143eea3cee63160fa8bed	0	1
3	0009655768c64bdeb2e877511632db8f	16.0	22.16	0	3f207df678b143eea3cee63160fa8bed	0	0
4	0009655768c64bdeb2e877511632db8f	17.0	22.16	0	f19421c1d4aa40978ebb69ca19b0e20d	0	1
...	...	...	...	...	...	...	...
167576	ffff82501cea40309d5fdd7edcca4a07	21.0	182.81	1	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	0
167577	ffff82501cea40309d5fdd7edcca4a07	23.0	182.81	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	0	0
167578	ffff82501cea40309d5fdd7edcca4a07	24.0	182.81	0	2906b810c7d4411798c6938adc9daaa5	0	1
167579	ffff82501cea40309d5fdd7edcca4a07	24.0	197.04	1	2906b810c7d4411798c6938adc9daaa5	1	0
167580	ffff82501cea40309d5fdd7edcca4a07	25.0	197.04	0	2906b810c7d4411798c6938adc9daaa5	0	0

167581 rows × 25 columns

```
#View the edited table to work with
my_starbuck
```

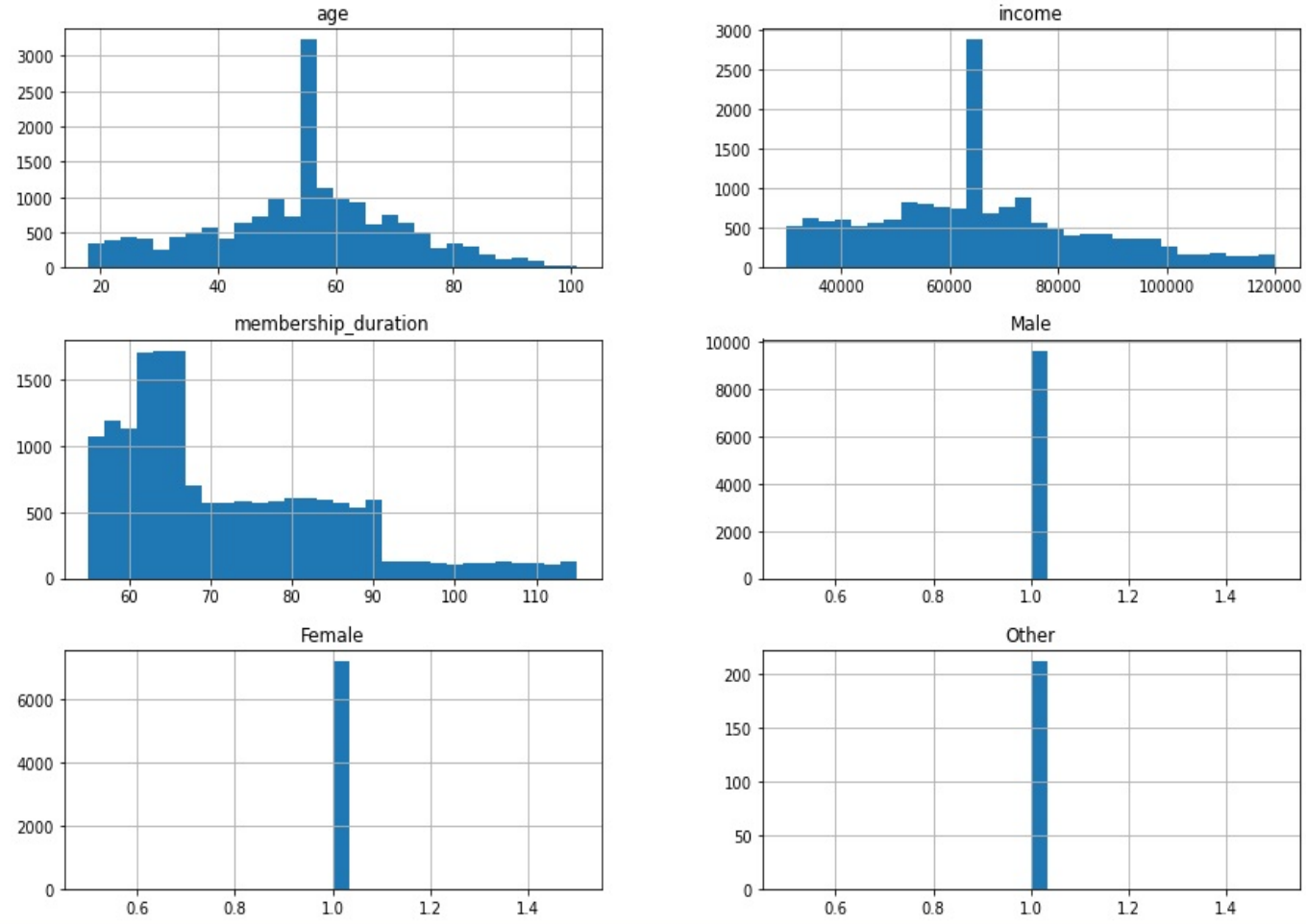
ted	offer_received	offer_viewed	age	income	...	offer_difficulty	offer_duration	web	email	mobile	social	bogo	discount	informational	total_transaction
0	1	0	33	72000.0	...	0.0	3.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	127.60
0	0	1	33	72000.0	...	0.0	3.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	127.60
0	1	0	33	72000.0	...	0.0	4.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	127.60
0	0	1	33	72000.0	...	0.0	4.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	127.60
0	1	0	33	72000.0	...	5.0	5.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	127.60
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	0	0	45	62000.0	...	5.0	7.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	226.07
0	0	1	45	62000.0	...	5.0	7.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	226.07
0	1	0	45	62000.0	...	10.0	7.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	226.07
1	0	0	45	62000.0	...	10.0	7.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	226.07
0	0	1	45	62000.0	...	10.0	7.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	226.07

Exploratory Visualization

We plot the histogram of our edited profile table and here is the result:

```
Edited profile table's feature histograms:

array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'income'}>],
       [<AxesSubplot:title={'center':'membership_duration'}>,
        <AxesSubplot:title={'center':'Male'}>],
       [<AxesSubplot:title={'center':'Female'}>,
        <AxesSubplot:title={'center':'Other'}>]], dtype=object)
```

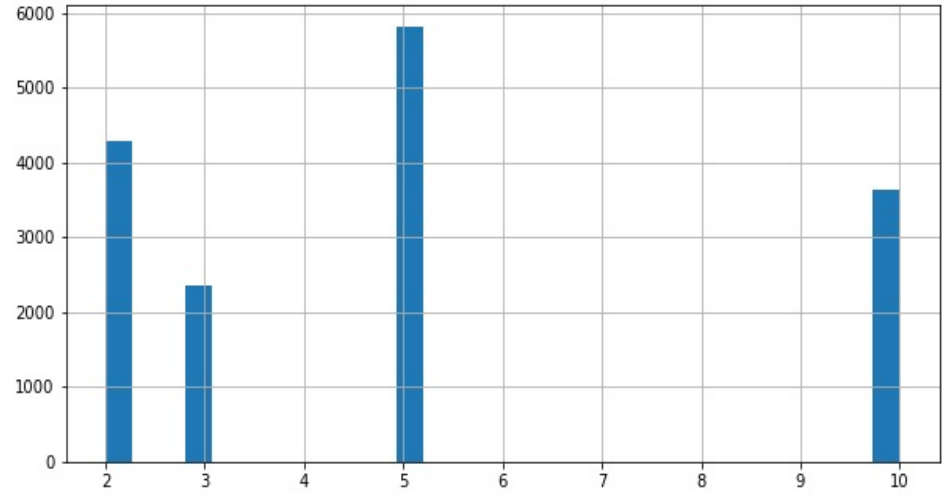


We can understand that there are only about 200 people of the Other gender. After randomly filling the NaN cells of the gender column, here we can see that there are almost 10000 men and less than 8000 women in the customers. The number of people of average age is about 3500 which is normal because we had about 3500 NaN cells in the age column that was filled with the average age of the other customers. Also, we have this phenomenon in the income column. The membership duration of the customers is between about 20 to 120 months.

We drew plots of given offer rewards to the customers, separately for men and women:

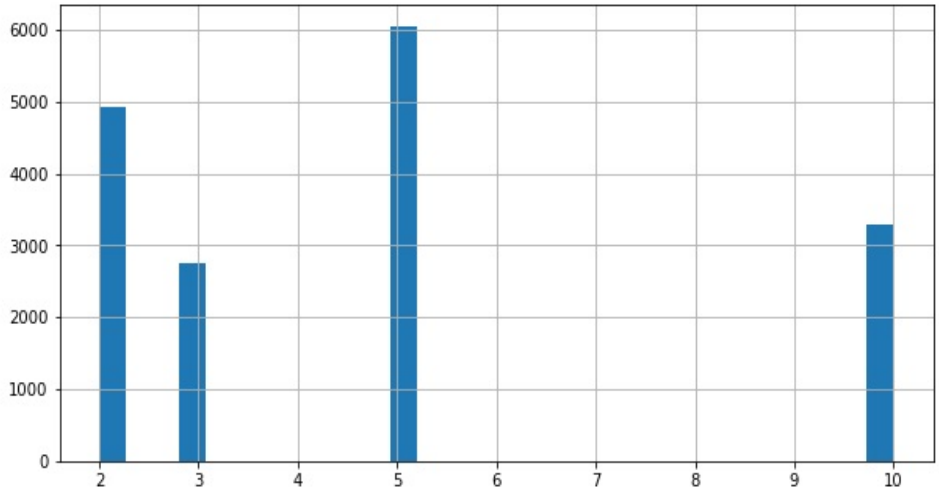
```
Histogram of female customer's completed rewards

<AxesSubplot:>
```



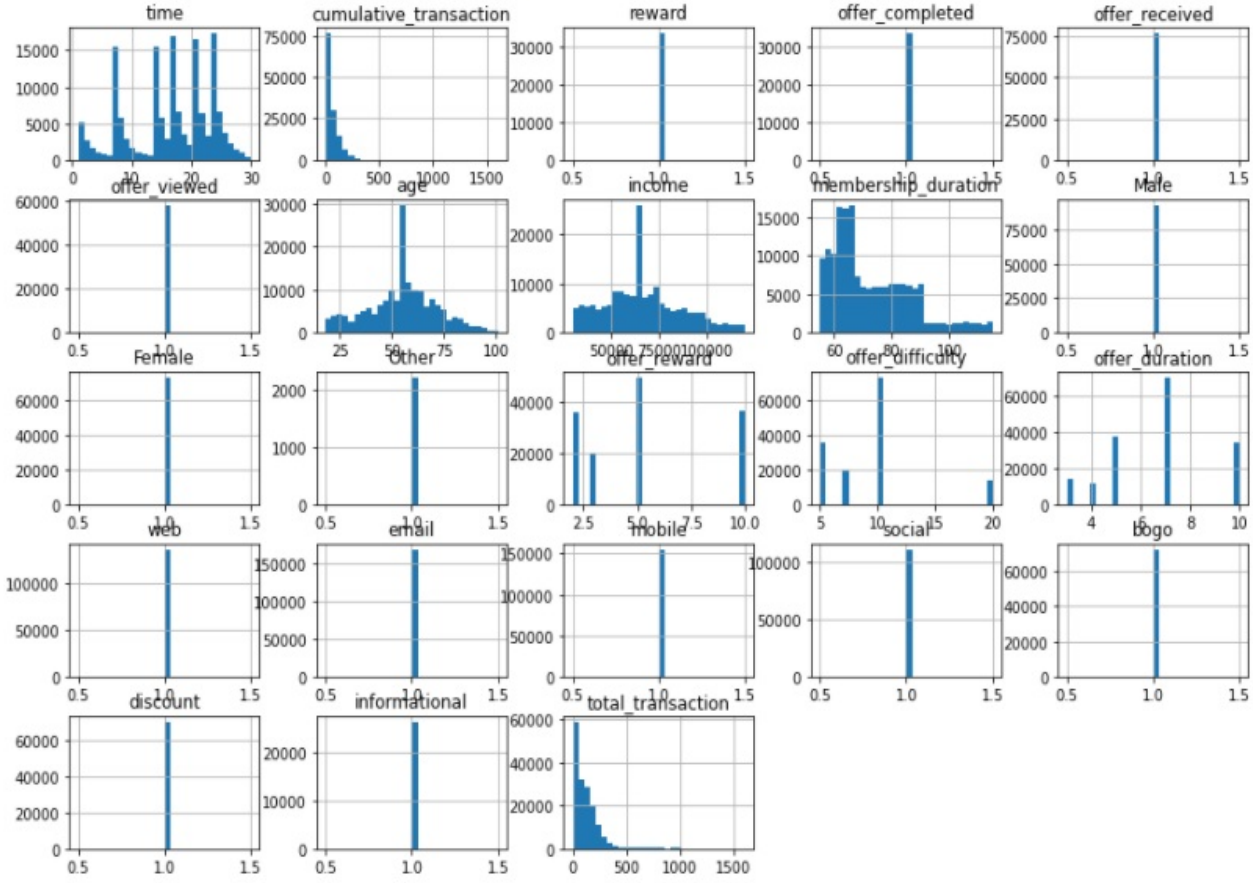

```
Histogram of male customer's completed rewards

<AxesSubplot:>
```

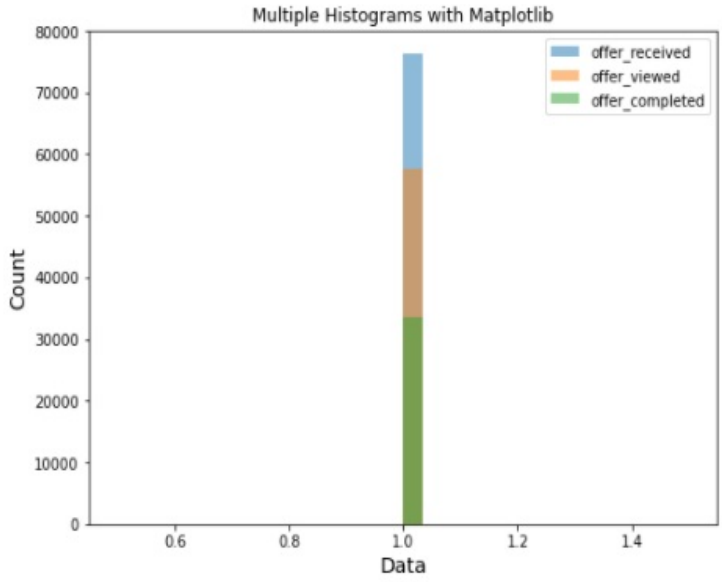


We can observe that both men and women first tend to complete the offers with 5 dollars reward (There are 3 offers matching this condition). At the second stage, they to finish the offers rewarding them 2 dollars (there are 2 offers with this condition) and so on. The reason that most of the customers tend to receive special rewards probably has a correlation with the offer difficulty and duration that should be considered.

Below you can see the histogram plot of my source data (my_starbuck) that shows how the data is distributed along with the dataset:



For example, we can see that a large number of customers have a low total amount of monthly transactions. So it's a good idea to spread the offer between this goal group to increase the sales among them. As the last exploratory plot, we can see below how the offers are received, viewed, and completed by customers. We can observe that people complete less than half of their received offers. This means that we should seriously think about how to spread every kind of offer between them.



Algorithms and Techniques

I want to predict the cumulative transaction amount of people at the times they receive, view or complete an offer. So my problem is a regression one and I need a model to predict my target amount by the least mean squared or absolute error. I trained a Scikit_learn random forest model using an Amazon Sagemaker estimator on the best hyperparameters I achieved from fine-tuning my estimator. We know that the hyperparameters in this model except for the "features" and "target", only include "the number of random records (samples) from the dataset" and "the number of trees in the algorithm". I calculated the number of estimators as 147 and the number of minimum samples leaf as 2. Then I created an endpoint and made predictions from it. I evaluated the model on R2 score and mean squared and absolute errors and checked how well my model is predicting on never-seen data (test data). Then I used an AutoGluon Tablular predictor to evaluate the same metric on it.

Benchmark

Because this problem is designed by myself, there's no definite source for benchmarking my model. However, if my problem was a Kaggle challenge, I could submit my results and give back a score. There are some online platforms like "Google's Vertex AI" that one can upload their CSV files and give back the prediction metrics. Unfortunately, because these kinds of sites incur costs, I can't use them. So I tried a different Auto-ML training like Autogluon Tabular predictor to compare the results with the first trained model on my dataset. We know that this predictor gives us the best model from its various models (that contains Sklearn random forest), and at the end, we can compare the R2 and other scores of both training jobs and choose what is better and more convenient.

III. Methodology

Data Preprocessing

Because there are 25 columns in my source table (my_starbuck), and all of these columns don't have a very significant effect on predicting my target (cumulative_transaction column) so we remove some of the columns at first. For example, the number of customers having the "Other" gender is only 212 persons, while the total number of customers is 17000. So we can consider this group as outliers and delete the rows that contain them (2190 rows). We should remove the columns that contain person or offer id, because all the information about offers and customers are now combined in the table. We should also ignore total_transaction column of every person, because we want it not to affect our predictions and our model should be able to predict the cumulative transactions based on the general information of the customers like age, income, etc. We should neglect the offer_received, offer_viewed, and offer_completed columns because just receiving offer messages is important for us and we are not focusing on the viewing of the offers. Here is the screenshot of this part's relating codes:

```
len(my_starbuck)
```

```
167581
```

```
print("Number of rows containing Other genders:")
my_starbuck[(my_starbuck['Other']==1)]['person'].count()
```

```
Number of rows containing Other genders:

2190
```

```
#Reading my source table
my_starbuck = pd.read_csv("my_starbuck.csv")

#Deleting the rows containing Other genders
my_starbuck = my_starbuck.drop(my_starbuck[(my_starbuck['Other']==1)].index)

#drop unimportant and unnecessary columns
my_starbuck = my_starbuck.drop([ 'person','offers', 'Other',  'email', 'web', 'mobile', 'social',
                                'informational', 'bogo', 'discount', 'offer_completed', 'offer_received',
                                'offer_viewed', 'total_transaction',], axis=1)
```

Then because the range of the numbers especially in the income column varies a lot with the other columns and this may lower the speed of gradient convergence, we should normalize our columns to be in the range of [0, 1]. We do this using a sklearn package as shown below:

```
#Do some preprocessing if necessary
from sklearn import preprocessing

#Normalizing some columns that contain big numbers
x = my_starbuck[['income', 'age', 'time','membership_duration', ]].values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
normalized_columns = pd.DataFrame(x_scaled)

data = my_starbuck.copy()
data['income'] = normalized_columns[0]
data['age'] = normalized_columns[1]
data['time'] = normalized_columns[2]
data['membership_duration'] = normalized_columns[3]
```

```
from sklearn.model_selection import train_test_split

#splitting the data to train and test
train, test = train_test_split(
    data, test_size=0.25, random_state=42)
```

Implementation

First, we should install the required packages. In this project I've upgraded the NumPy package to ensure that it can read JSON files correctly and, I've installed autoguon that can use its tabular predictor at the end of the program.

Installing necessary packages:

```
In [1]: !pip install numpy --upgrade
        !pip install autogluon --no-cache-dir
```

```
Requirement already satisfied: numpy in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (1.19.
5)
Collecting autogluon
  Downloading autogluon-0.3.1-py3-none-any.whl (9.9 kB)
Collecting autogluon.extra==0.3.1
  Downloading autogluon.extra-0.3.1-py3-none-any.whl (28 kB)
Collecting autogluon.mxnet==0.3.1
  Downloading autogluon.mxnet-0.3.1-py3-none-any.whl (33 kB)
Collecting autogluon.vision==0.3.1
  Downloading autogluon.vision-0.3.1-py3-none-any.whl (38 kB)
Collecting autogluon.tabular[all]==0.3.1
  Downloading autogluon.tabular-0.3.1-py3-none-any.whl (273 kB)
     |████████████████████████████████| 273 kB 80.4 MB/s
```

The main parts of the implementation can be split into 3 steps:

1. Cleansing data, modifying the tables, and merging them to create a source table
2. Train a sklearn random forest model and evaluate the model on the test data
3. Using an AutoGluon Tabular predictor to compare our model metric scores with

As I described before and as I shared my code scripts, preparing the data to become a neat, usable data frame is the toughest and the most time-consuming job. As a Forbes report indicates: "Data scientists spend 60% of their time on cleaning and organizing data." [4] and I clearly experienced this.

For the second step, I first wrote a code that creates a script.py file that contains codes of training a random forest model. I used the source code in the reference [5] and specialized it for training my dataset. For fine-tuning my sklearn estimator, I gave an integer range of (20, 200) to the number of estimators, and also I set an integer range of (2, 10) to the minimum samples leaf. I ran the tuner for 10 jobs allowing 2 maximum parallel jobs. It took about 30 minutes to fit the tuner. Here you can see screenshots of this step:

Sample code of a fitted fine-tuning job:

```
# we use the Hyperparameter Tuner
from sagemaker.tuner import IntegerParameter

# Define exploration boundaries
hyperparameter_ranges = {
    "n-estimators": IntegerParameter(20, 200),
    "min-samples-leaf": IntegerParameter(2, 10),
}

# create Optimizer
Optimizer = sagemaker.tuner.HyperparameterTuner(
    estimator=sklearn_estimator,
    hyperparameter_ranges=hyperparameter_ranges,
    base_tuning_job_name="tuner-star",
    objective_type="Minimize",
    objective_metric_name="median-AE",
    metric_definitions=[
        {"Name": "median-AE", "Regex": "AE-at-50th-percentile: ([0-9.]+).*$"}
    ],  # extract tracked metric from logs with regexp
    max_jobs=10,
    max_parallel_jobs=2,
)
```

```
import time
tic = time.clock()

Optimizer.fit({"train": trainpath, "test": testpath})

toc = time.clock()
print(toc - tic)
```

```
.....................................................................................................
..........!
0.6400000000000006
```

After achieving the best model hyperparameters, we train a sklearn random forest model on these hyperparameters. We hope that it gives us the best metric scores of the model.

Training the model based on best hyperparameters:

```
hyperparameters = {"n-estimators": int(best_estimator.hyperparameters()['n-estimators'].replace('"', '')), \
                   "min-samples-leaf":  int(best_estimator.hyperparameters()['min-samples-leaf'].replace('"', '')),
                  "features": best_estimator.hyperparameters()['features'].replace('"', ''), \
                   "target": best_estimator.hyperparameters()['target'].replace('"', ''), }
hyperparameters
```

```
{'n-estimators': 147,
 'min-samples-leaf': 2,
 'features': 'time reward age income membership_duration Male Female offer_reward offer_difficulty offer_duration w
eb mobile social bogo discount',
 'target': 'cumulative_transaction'}
```

Launching a training job on best hyperparameters

```
# We use the Estimator from the SageMaker Python SDK
from sagemaker.sklearn.estimator import SKLearn
import time

FRAMEWORK_VERSION = "0.23-1"

sklearn_estimator = SKLearn(
    entry_point="script.py",
    role=get_execution_role(),
    instance_count=1,
    instance_type="ml.m4.xlarge",#"ml.c5.xlarge",#"ml.m4.xlarge",
    framework_version=FRAMEWORK_VERSION,
    base_job_name="train-star",
    metric_definitions=[{"Name": "median-AE", "Regex": "AE-at-50th-percentile: ([0-9.]+).*$"}],
    hyperparameters=hyperparameters,
)

tic = time.clock()

# launch training job
sklearn_estimator.fit({"train": trainpath, "test": testpath}, wait=True)

toc = time.clock()
print(toc - tic)
```

```
2022-02-12 17:08:23 Uploading - Uploading generated training modelvalidating model
AE-at-10th-percentile: 0.15321801101389632
AE-at-50th-percentile: 14.643744489354185
AE-at-90th-percentile: 64.97190191394014
model persisted at /opt/ml/model/model.joblib
2
2022-02-12 17:08:12,102 sagemaker-containers INFO     Reporting training SUCCESS

2022-02-12 17:09:23 Completed - Training job completed
ProfilerReport-1644685401: NoIssuesFound
Training seconds: 165
Billable seconds: 165
0.990000000000002
```

Then we should deploy the model to an endpoint. Here you can see an implemented code for this job:

```
predictor = model.deploy(instance_type="ml.c5.large", initial_instance_count=1)
```

```
2022-02-12 17:09:23 Starting - Preparing the instances for training
2022-02-12 17:09:23 Downloading - Downloading input data
2022-02-12 17:09:23 Training - Training image download completed. Training in progress.
2022-02-12 17:09:23 Uploading - Uploading generated training model
2022-02-12 17:09:23 Completed - Training job completed
Model artifact persisted at s3://sagemaker-us-east-1-201557610166/train-star-2022-02-12-17-03-21-256/output/model.t
ar.gz
------!
```

By invoking the endpoint we can see the prediction results:

```
response = predictor.predict(test[feature_names])
test_target = test['cumulative_transaction']
test_target = test_target.to_numpy()

print("response \t\t target")
for i in range(100):
    print(response[i] , "\t",test_target[i])
```

```
response                target
234.91247238419186      674.5600000000002
69.771200356333         8.59
34.19025275348234       8.48
44.34408811143504       13.22
0.0817891156462585      0.0
121.91559410430843      119.67000000000002
121.14870051830256      157.3
```

And finally, we can see metric scores of our model using these codes:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

mean_squared_error = mean_squared_error(test_target, response)
mean_absolute_error = mean_absolute_error(test_target, response)
r2_score = r2_score(test_target, response)
print(f"R2 score: {r2_score}, mean_squared_error: {mean_squared_error}, mean_absolute_error {mean_absolute_error}")
```

```
R2 score: 0.4722200999770133, mean_squared_error: 3815.057537021681, mean_absolute_error 28.686389836046114
```

As the final step, I used an AutoGluon-Tabular predictor that presents some good metric scores on my dataset. Using this Auto-ML package has a lot of benefits. “AutoGluon-Tabular can save time by automating time-consuming manual steps—handling missing data, manual feature transformations, data splitting, model selection, algorithm selection, hyperparameter selection, and tuning, ensembling multiple models, and repeating the process when data changes. Unlike existing AutoML frameworks that primarily focus on model/hyperparameter selection, AutoGluon-Tabular succeeds by ensembling multiple models and stacking them in multiple layers.” [6]. Here are some screenshots of implementing this predictor:

```
from autogluon.tabular import TabularPredictor

#Training with Autogluon Tabular
save_path = 'transaction_models'
predictor = TabularPredictor('cumulative_transaction', problem_type='regression',
                                        path=save_path, learner_kwargs={'ignored_columns':['person', 'offers','Oth
```

Then we can easily what the leaderboard of the auto-trained models:

```
predictor.leaderboard(extra_info=True, silent=True)
```

	model	score_val	pred_time_val	fit_time	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order	num_features	.
0	WeightedEnsemble_L3	-42.368650	104.664558	1053.279999	0.002498	0.907678	3	True	9	3	.
1	LightGBM_BAG_L2	-42.425539	82.439590	750.768911	2.914795	55.807490	2	True	7	19	.

And even we can see the best model in detail:

```
predictor.leaderboard(extra_info=True, silent=True)
```

ag_args_fit	features	child_hyperparameters	child_hyperparameters_fit	child_ag_args_fit	ancestors
{'max_memory_usage_ratio': 1.0, 'max_time_limit_ratio': 1.0, 'max_time_limit': None, 'min_time_limit': 0, 'ignored_type_group_special': None, 'ignored_type_group_raw': None, 'get_features_kwargs': None, 'get_features_kwargs_extra': None, 'drop_unique': False}	[LightGBMXT_BAG_L2, CatBoost_BAG_L2, LightGBM_BAG_L2]	{'ensemble_size': 100}	{'ensemble_size': 93}	{'max_memory_usage_ratio': 1.0, 'max_time_limit_ratio': 1.0, 'max_time_limit': None, 'min_time_limit': 0, 'ignored_type_group_special': None, 'ignored_type_group_raw': None, 'get_features_kwargs': None, 'get_features_kwargs_extra': None, 'drop_unique': False}	[KNeighborsUnif_BAG_L1, LightGBM_BAG_L1, KNeighborsDist_BAG_L1, LightGBMXT_BAG_L2, LightGBM_BAG_L2, CatBoost_BAG_L2, LightGBMXT_BAG_L1]

Autogluon predictions and its model evaluation implementations:

Getting predictions from TabularPredictor

```
response = predictor.predict(test)
test_target = test['cumulative_transaction']
```

Evaluate the Autogluon predictor

```
predictor.evaluate(test)
```

```
Evaluation: root_mean_squared_error on test data: -40.85419444299454
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric valu
e.
Evaluations on test data:
{
    "root_mean_squared_error": -40.85419444299454,
    "mean_squared_error": -1669.0652035860114,
    "mean_absolute_error": -18.183393678980757,
    "r2": 0.7690994021106778,
    "pearsonr": 0.8772191133365619,
    "median_absolute_error": -8.33363681793214
}
```

Refinement

I ran the AutoGluon tabular predictor several times using different features on the dataset and also different runtimes. When we define the 'time' columns as one of the ignored feature columns, we will get predictions with this R2, Pearson correlation, and error scores:

```
[154] predictor.evaluate(test)
```

```
Evaluation: root_mean_squared_error on test data: -49.1362145631647
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "root_mean_squared_error": -49.1362145631647,
    "mean_squared_error": -2414.367581597353,
    "mean_absolute_error": -21.4461307991239,
    "r2": 0.6659933255347521,
    "pearsonr": 0.8161694177709358,
    "median_absolute_error": -9.223929290771483
}
```

Running the predictor several more times on other conditions, we can see that this model has the least R2 and Pearson correlation score and the most errors. This is sensible because we are predicting the cumulative transaction amounts during one month. So by passing the days we have an increase in transaction amounts or at least we are fixed on the last amount. I've run the predictor again (both first and second runtimes are 10 minutes), by adding the time as a feature for the data and here you can see the evaluation results:

```
predictor.evaluate(test)
```

```
Evaluation: root_mean_squared_error on test data: -46.86148063698137
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "root_mean_squared_error": -46.86148063698137,
    "mean_squared_error": -2195.9983674901614,
    "mean_absolute_error": -19.622714647028204,
    "r2": 0.6962027996701187,
    "pearsonr": 0.8346209643999175,
    "median_absolute_error": -8.842765007019043
}
```

	importance	stddev	p_value	n	p99_high	p99_low
income	37.866177	8.615488	0.008411	3	87.233877	-11.501522
time	34.289185	9.248793	0.011702	3	87.285797	-18.707427
membership_duration	32.137404	5.636153	0.005049	3	64.433188	-0.158381
age	27.884696	12.141632	0.028888	3	97.457594	-41.688203
offer_difficulty	9.734494	3.322671	0.018355	3	28.773768	-9.304779
offer_reward	7.502998	3.639348	0.035130	3	28.356868	-13.350872
offer_duration	5.870241	1.218331	0.007028	3	12.851411	-1.110929
reward	1.637861	0.382719	0.008859	3	3.830881	-0.555160
Male	0.426934	0.413678	0.107878	3	2.797358	-1.943489
offers	0.369504	0.753443	0.242550	3	4.686819	-3.947811
discount	0.123173	0.143444	0.137657	3	0.945122	-0.698775
Female	0.086867	0.186321	0.252070	3	1.154508	-0.980774
web	0.038662	0.158156	0.356592	3	0.944911	-0.867587
offer_viewed	0.032467	0.111621	0.332209	3	0.672068	-0.607134
mobile	0.014719	0.056344	0.347637	3	0.337574	-0.308136
Other	0.013667	0.043375	0.319990	3	0.262209	-0.234876
email	0.000000	0.000000	0.500000	3	0.000000	0.000000
offer_completed	0.000000	0.000000	0.500000	3	0.000000	0.000000
informational	-0.002904	0.041685	0.542502	3	0.235958	-0.241766
bogo	-0.005618	0.041760	0.581282	3	0.233671	-0.244906

We can observe that both the R2 and Pearson scores increased in this case and the reason is clear. Because time has a good correlation with our target (cumulative transactions), the Pearson score would be higher in amount, and because we have features that have a better correlation with our prediction goal, we would have a higher R2 score and also a lower amount of errors.

I trained this auto-ML model again and this time for a double runtime duration (20 minutes), and also ignored some columns that had the least importance. We could see that the Pearson and R2 scores are increased and both of the mean squared and absolute errors decreased.

Evaluate the Autogluon predictor

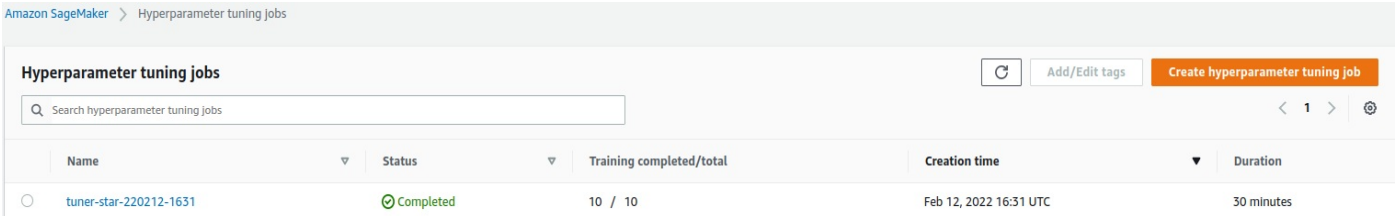
```
predictor.evaluate(test)
```

```
Evaluation: root_mean_squared_error on test data: -40.85419444299454
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric valu
e.
Evaluations on test data:
{
    "root_mean_squared_error": -40.85419444299454,
    "mean_squared_error": -1669.0652035860114,
    "mean_absolute_error": -18.183393678980757,
    "r2": 0.7690994021106778,
    "pearsonr": 0.8772191133365619,
    "median_absolute_error": -8.33363681793214
}
```

IV. Results

Model Evaluation and Validation

I ran the tuner 10 times (about 30 minutes) to get the best hyperparameters for training a random forest model on my dataset. Here is a screenshot of all the completed tuning jobs:



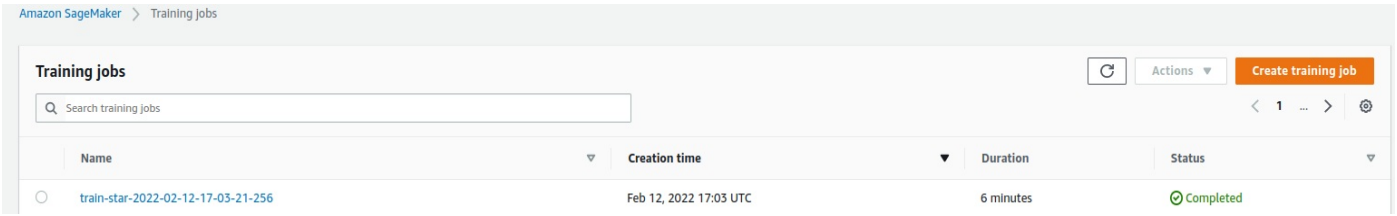
Getting the results of my tuner I found the best number of estimators to be 147 and the best minimum number of leaves to be 2. Here are the screenshots of the tuner's analytics and the best hyperparameters achieved:

	min-samples-leaf	n-estimators	TrainingJobName	TrainingJobStatus	FinalObjectiveValue	TrainingStartTime	TrainingEndTime	TrainingElapsedTimeSeconds
0	2.0	149.0	tuner-star-220212-1631-010-14b18466	Completed	14.619602	2022-02-12 16:57:49+00:00	2022-02-12 17:00:33+00:00	164.0
1	2.0	144.0	tuner-star-220212-1631-009-5f6264cf	Completed	14.598150	2022-02-12 16:56:01+00:00	2022-02-12 16:59:24+00:00	203.0
2	2.0	147.0	tuner-star-220212-1631-008-f423754e	Completed	14.565795	2022-02-12 16:51:41+00:00	2022-02-12 16:54:23+00:00	162.0
3	2.0	190.0	tuner-star-220212-1631-007-b1bdb91a	Completed	14.626628	2022-02-12 16:49:57+00:00	2022-02-12 16:52:59+00:00	182.0
4	2.0	174.0	tuner-star-220212-1631-006-16544424	Completed	14.662675	2022-02-12 16:45:37+00:00	2022-02-12 16:48:24+00:00	167.0

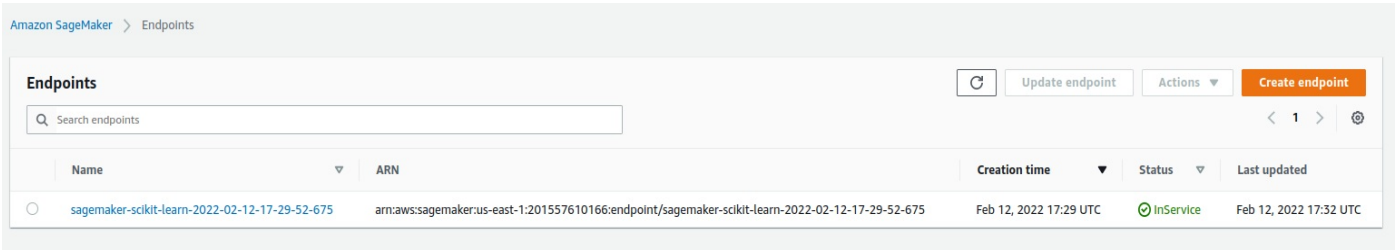
```
best_estimator.hyperparameters()
```

```
{'_tuning_objective_metric': '"median-AE"',
 'features': '"time reward age income membership_duration Male Female offer_reward offer_difficulty offer_duration
web mobile social bogo discount"',
 'min-samples-leaf': '2',
 'n-estimators': '147',
 'sagemaker_container_log_level': '20',
 'sagemaker_estimator_class_name': '"SKLearn"',
 'sagemaker_estimator_module': '"sagemaker.sklearn.estimator"',
 'sagemaker_job_name': '"train-star-2022-02-12-16-31-17-481"',
 'sagemaker_program': '"script.py"',
 'sagemaker_region': '"us-east-1"',
 'sagemaker_submit_directory': '"s3://sagemaker-us-east-1-201557610166/train-star-2022-02-12-16-31-17-481/source/s
ourcedir.tar.gz"',
 'target': '"cumulative_transaction"'}
```

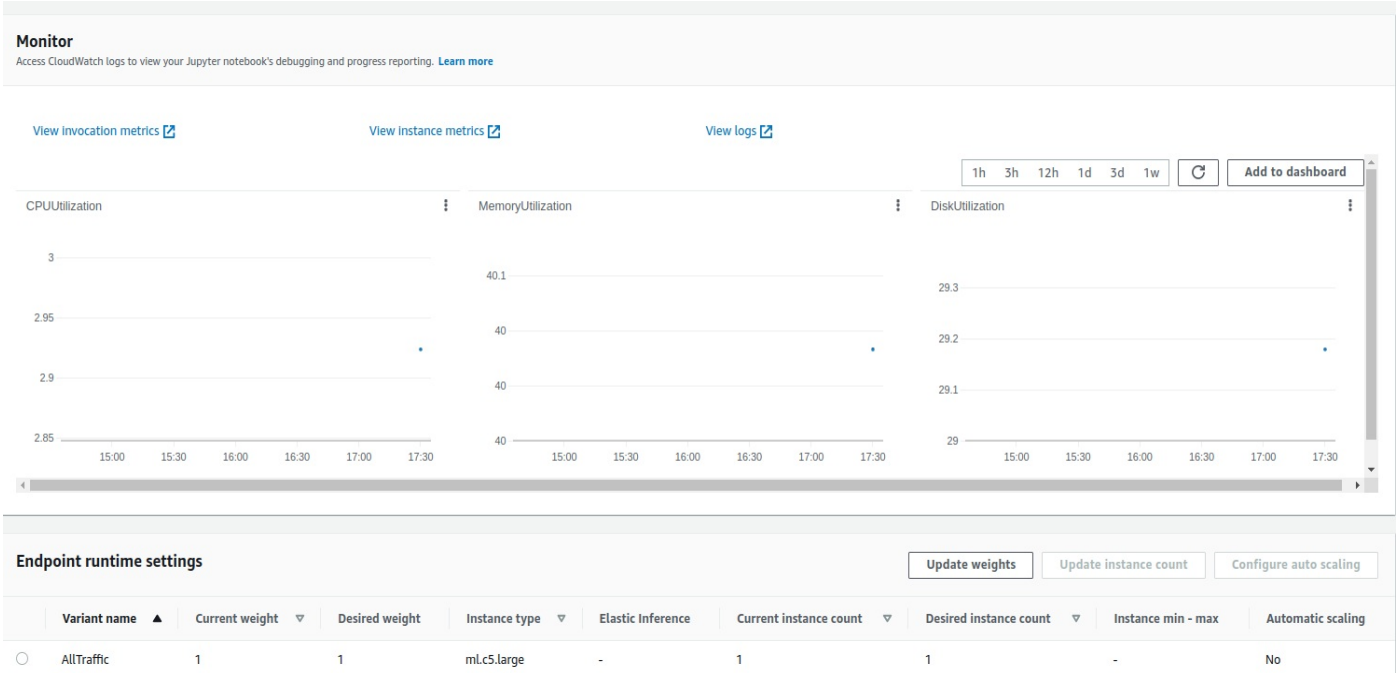
We've done a training job on these hyperparameters that successfully completed:



We deployed the model to an endpoint and here you can see an in-service endpoint that is one of the most important parts of creating a model:



We monitored the endpoint to observe its invocations:



This is not the only training job that we did. Before normalizing the train and test features, we received an R2 score of 40% on the test data. However, by normalizing the features we have reached the R2 score of more than 45%. By changing the features and removing unimportant features, the accuracy of 47% is the most R2 score that we gained:

```
print(f"R2 score: {r2_score}, mean_squared_error: {mean_squared_error}, mean_absolute_error {mean_absolute_error}")
```

```
R2 score: 0.4722200999770133, mean_squared_error: 3815.057537021681, mean_absolute_error 28.686389836046114
```

Because the R2 score of the Scikit_Learn Random Forest model is rather low, we trained the dataset using Autogluon Tabular models to check if we obtain better results. We know that autogluon.tabular contains RFModel (Random Forest model (scikit-learn)) as one of its models. So we hoped we will certainly improve the result and we did. After training the data with the same features using an AutoGluon Tabular predictor we obtained a good R2 score of about 77% which is obtained from a WeightedEnsemble model that trained in the stack level 3. This model also has the least errors compared to all previously trained models. “A weighted ensemble is an extension of a model averaging ensemble where the contribution of each member to the final prediction is weighted by the performance of the model” [7].

```
predictor.get_model_best()
```

```
'WeightedEnsemble_L3'
```

AutoGluon Tabular best model:

AutoGluon Tabular feature used:

```
print("Predictor features: ", predictor.features())
```

```
Predictor features:  ['time', 'reward', 'age', 'income', 'membership_duration', 'Male', 'Female', 'offer_reward',
'offer_difficulty', 'offer_duration', 'web', 'mobile', 'social', 'bogo', 'discount']
```

I changed the train and test data using random selection several times and the evaluation results are continuously showing that the model is predicting in almost the same scores and error ranges. I also have changed the runtime of the predictor that shows giving more time, a WeightedEnsemble_L3 model is performing better than the LightGBM_BAG_L2 in fewer runtimes. Also, all the R2 scores of the best models achieved by using an AutoGluon Tabular predictor, are greater than 65%.

Justification

We can consider the R2 score, mean-squared error, and absolute error for comparing any trained model on our dataset. The R2 score that has been obtained from every sklearn random forest model is not preceding 47%. We tried several strategies like normalizing all the columns, normalizing just the columns containing very big amounts, removing the features that are less important (using the feature importance achieved from the AutoGluon predictor), changing in the number of removed features, fine-tuning for more jobs, fine-tuning in a wider range of hyperparameters, etc, but unfortunately, the highest R2 score achieved is 47% and the predicted result errors are high. So I don’t hope that this would be a good model to be trained on this dataset.

We can run the AutoGluon Tabular predictor for more time and this may change the best model and also may make the R2 score of the best model higher, as this occurred in increasing the time from 10 minutes to 20 minutes.

V. Conclusion

Because the R2 score and also the mean-squared/absolute errors of the Weighted Ensemble model that is the best model gained by an auto_ML training are significantly better than the results of training with a sklearn random forest model, so developing a Weighted Ensemble model for predicting my target is looking sensible. You can see how to Develop a Weighted Average Ensemble for Deep Learning Neural Networks in this reference [8]. Although we can consider changing in hyperparameters of the random forest model that the number of estimators can be more than 200 and for example try it up to 1000 estimators.

The problem is a good regression problem and as we can see in using an auto-ML model, we can use pre-trained deep neural network models or even design one that can give us the prediction results of our goal feature with high accuracy.

Free-Form Visualization

Here we show two outputs of this project. We first show an output of our trained random forest model. We just need to invoke the endpoint that we created before and get the prediction results. Also for evaluating this model, we use sklearn functions. This is an output example on a random test data:

Invoke the endpoint

```
# the SKLearnPredictor does the serialization from pandas for us
response = predictor.predict(test[feature_names])
test_target = test['cumulative_transaction']
test_target = test_target.to_numpy()

print("response \t\t target")
for i in range(100):
    print(response[i] , "\t",test_target[i])
```

```
response                target
42.741547781017154      42.75
62.41656648850017       31.57
5.431426303854877       0.0
111.42925178166503      61.63
119.83790046971173      146.2
30.473342484612882      37.57
81.09920667314543       29.75
73.52277721088433       73.36
5.504794001727676       2.54
35.42429664723034       20.41
132.62019595616022      138.12
45.86012331281719       23.98
57.533254994061096      33.6
61.2219977324263        37.06
56.77866553287983       20.52
14.928407446672761      128.26
20.11640678652412       29.56
39.824623744735995      121.08
64.26413265306118       33.97
6.332217624867687       8.84
20.02664085951841       12.79
15.32111321671525       0.0
138.58601587301595      789.81
110.23857701652086      55.65
144.37812649821834      155.09
107.90099708454815      124.15
69.68707636864272       83.38
124.63938921282798      130.97
0.0     0.0
8.765819327214734       3.92
1.18198223733938        0.05
81.47381827016518       99.23
89.95767454918476       108.05
48.186443013713415      109.19
0.0     0.0
89.26236799481691       76.01
```

Evaluate the sklearn random forest model

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

mean_squared_error = mean_squared_error(test_target, response)
mean_absolute_error = mean_absolute_error(test_target, response)
r2_score = r2_score(test_target, response)
print(f"R2 score: {r2_score}, mean_squared_error: {mean_squared_error}, mean_absolute_error {mean_absolute_error}")
```

```
R2 score: 0.4523693254058069, mean_squared_error: 4236.3759983377595, mean_absolute_error 29.62099167956011
```

We also got the best model predicting a random train data using autogluon tabular. The autogluon tabular predictor uses some part of the train data reserved for validation inside fit. So we can easily watch the model leaderboard after finishing fitting the predictor. Here you can see a leaderboard and prediction and evaluation output of an autogluon tabular predictor that fitted on the train data for 20 minutes:

My Tabular predictor leaderboard

```
#The predictor leaderboard that is produced using the data previously reserved for validation inside fit, and can di
predictor.leaderboard(extra_info=True, silent=True)
```

	model	score_val	pred_time_val	fit_time	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order	num_features	.
0	WeightedEnsemble_L3	-42.368650	104.664558	1053.279999	0.002498	0.907678	3	True	9	3	.
1	LightGBM_BAG_L2	-42.425539	82.439590	750.768911	2.914795	55.807490	2	True	7	19	.
2	LightGBMXT_BAG_L2	-45.913178	101.696354	988.669068	22.171559	293.707647	2	True	6	19	.
3	WeightedEnsemble_L2	-50.032328	12.983774	113.341726	0.002828	1.338707	2	True	5	2	.

```
print("response:\n ",response)
print("target:\n",test_target)
```

```
response:
  53680    766.571960
60296     44.871216
93002     23.226473
51223     16.035902
63364      0.295056
            ...
38068     32.941368
96695     11.738913
51438     95.026451
85069    133.813339
88609     19.691917
Name: cumulative_transaction, Length: 41896, dtype: float32
target:
 53680    674.56
60296      8.59
93002      8.48
51223     13.22
63364      0.00
           ...
38068     34.04
96695      4.60
51438     44.72
85069     71.67
88609      9.95
Name: cumulative_transaction, Length: 41896, dtype: float64
```

Evaluate the Autogluon predictor

```
predictor.evaluate(test)
```

```
Evaluation: root_mean_squared_error on test data: -40.85419444299454
        Note: Scores are always higher_is_better. This metric score can be multiplied by -1 to get the metric valu
e.
Evaluations on test data:
{
    "root_mean_squared_error": -40.85419444299454,
    "mean_squared_error": -1669.0652035860114,
    "mean_absolute_error": -18.183393678980757,
    "r2": 0.7690994021106778,
    "pearsonr": 0.8772191133365619,
    "median_absolute_error": -8.33363681793214
}
```

Looking at the outputs we can obviously consider that the Weighted Ensemble model is predicting our data at best, by an R2 score of about 77% and minimum errors between all other models.

As we can see in the feature importance table obtained by the autogluon tabular predictor, the customer's income is the most important feature to have a higher cumulative transaction amount among customers who are receiving offers. The next important features are age, time and duration of the membership of the customer. Offer difficulty is the next important feature that makes sense, because how much less money the offer needs to be completed, the customers will complete it faster.

	importance	stddev	p_value	n	p99_high	p99_low
income	47.165292	7.170958	0.003809	3	88.255674	6.074910
age	37.275802	9.836164	0.011216	3	93.638111	-19.086508
time	34.273346	6.786459	0.006409	3	73.160507	-4.613814
membership_duration	33.096836	4.776246	0.003435	3	60.465258	5.728414
offer_difficulty	6.951953	1.607391	0.008679	3	16.162484	-2.258577
offer_reward	3.767608	0.957463	0.010428	3	9.253979	-1.718764
offer_duration	3.376406	0.957187	0.012879	3	8.861195	-2.108382
Male	3.344184	2.300363	0.064053	3	16.525521	-9.837153
reward	1.312406	0.448571	0.018402	3	3.882767	-1.257955
Female	1.013403	1.332393	0.159194	3	8.648163	-6.621358
social	0.454156	0.196503	0.028555	3	1.580140	-0.671828
discount	0.047031	0.254238	0.389518	3	1.503842	-1.409780
bogo	0.006032	0.137368	0.473147	3	0.793167	-0.781102
mobile	-0.013164	0.081005	0.597604	3	0.451004	-0.477333
web	-0.140804	1.404616	0.560929	3	7.907802	-8.189411

Reflection

The hardest and the most time-consuming part of this project was editing and modifying the raw input tables to obtain a suitable source table. After passing this challenge, we should design estimators to train and fine tune a Scikit-Learn random forest model. After training the model on the best hyperparameters obtained from the fine-tuning job, we should deploy the model to an endpoint. Having an endpoint we can simply get the prediction result of the model on a test data. Then we should evaluate the model prediction and check how well our model is predicting. For me, another challenge occurred here when I was facing a low R2 score model (less than 30 percent). So I normalized the data and did a lot of preprocessing on my main source table and I could see major progress in my model metric scores. Then I tried an AutoGluon Tabular predictor to fit on the data and fortunately, there was a high R2 score model that can predict my test data. I also tried changing the parameters of this predictor and watched if there is any progress. I used different features for my train dataset and I fitted the predictor for various amounts of time. At last, I could get the prediction with an R2 score of 77% on 20 minutes fitting, which is quite fine as an early-stage result.

Improvement

I think it's a good idea to develop a weighted ensemble model to predict my test data with an R2 score of above 90%. Also by having a neat table of data, I consider designing various classification and regression problems on the same dataset that can answer other aspects of the effects of advertisement on sales. For example, we can create a binary classification model to predict if a customer receives a special kind of offer, would they complete the offer or not, or will they view the offer or not (predicting a class label). In the case of predicting a class probability, we can ask what are the probabilities of a specified customer to complete any of 10 classes of offers. There are many problems that we can design and answer on this dataset. Solving each of them needs a lot of effort, but at last your hard work always pays off.

References

[1] Gharibshah, Zhabiz & Zhu, Xingquan. (2021). User Response Prediction in Online Advertising.

[2] https://www.investopedia.com/terms/r/r-squared.asp#:~:text=In%20other%20fields%2C%20the%20standards,would%20show%20a%20low%20correlation.

[3] https://towardsdatascience.com/avoid-r-squared-to-judge-regression-model-performance-5c2bc53c8e2e

[4] https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=14537dd96f63

[5] https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-python-sdk/scikit_learn_randomforest/Sklearn_on_SageMaker_end2end.html

[6] https://aws.amazon.com/marketplace/pp/prodview-n4zf5pmjt7ism

[7] https://machinelearningmastery.com/weighted-average-ensemble-for-deep-learning-neural-networks/#:~:text=A%20weighted%20ensemble%20is%20an,the%20performance%20of%20the%20model.

[8] https://machinelearningmastery.com/weighted-average-ensemble-for-deep-learning-neural-networks/