05. Capstone Project

Starbucks Offer Acceptance Forecast Model With AWS

1. Introduction

This article describes my capstone project developed in for the Machine Learning Engineer nanodegree program from Udacity. I developed this Starbucks offer acceptance forecast model in the context of the Machine Learning Engineer nanodegree program from Udacity, using a dataset that is also utilized in the Data Scientist nanodegree program but with a different apporach. My main goal was to apply and demonstrate my understanding of machine learning engineering concepts, using multiple AWS services such as S3, Lambda, Cloud Watch, IAM and, of curse, Sagemaker.

2. Project Overview

As a marketing strategy, Starbucks constantly sends offers for the users of the mobile app. The type of offer can vary significantly from a simply informational advertise to a 'buy one, get one" (BOGO) offer. Since not all customers receive different offers, in different quantities at different times, it's not elementary to indicate its effectiveness o indicate for each customers are prone to accept the offer. Our task focus on deploying a machine learning model in aws that indicates if a customer is prone to accept the offer given some data that is available.

The data provided for this project is composed by 3 json files. It's general meaning and fields are indicated above:

- portfolio.joson: Information about 10 different offer strategies.
  - id (string) — offer id
  - offer_type (string) — a type of offer ie BOGO, discount, informational
  - difficulty (int) — the minimum required to spend to complete an offer
  - reward (int) — the reward is given for completing an offer
  - duration (int) — time for the offer to be open, in days
  - channels (list of strings)
- profile.json: Demographic data for each customer.
  - age (int) — age of the customer
  - became_member_on (int) — the date when customer created an app account
  - gender (str) — gender of the customer ('M','F' and 'O' for any other)
  - id (str) — customer-id
  - income (float) — customer's income
- transcript.json: Event registries for offer interaction or transaction.
  - event (str) — record description (ie transaction, offer received, offer viewed, etc.)
  - person (str) — customer-id
  - time (int) — time in hours since the start of the test. The data begins at time t=0
  - value — (dictionary of strings) — either an offer id or transaction amount depending on the record

3. Data Preparation and Analysis

   3.1. 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

Changes:

- No integrity issues found;
- ‘channels’ and ‘offer_type’ were encoded for better compatibility with ML algorithims;
- Email information as droped because is present in all offers;
- Some columns were renamed for clarity.

```
# one-hot encoding channels column
portfolio = portfolio_.copy()
portfolio['channels'] = portfolio['channels'].apply(lambda x: ','.join(map(str, x)))
portfolio = portfolio.join(portfolio['channels'].str.get_dummies(','))
portfolio.drop('channels', axis=1, inplace=True)

# one-hot encoding offer_type column
portfolio = portfolio.join(pd.get_dummies(portfolio['offer_type']))
portfolio.drop('offer_type', axis=1, inplace=True)

#drop email column since it it contains no useful information
portfolio.drop('email', axis=1, inplace=True)

#rename id column to offer_id, reward to offer_reward and duration to offer_duration
portfolio.rename(columns={'id':'offer_id', 'reward':'offer_reward', 'duration':'offer_duration'}, inplace=True)
portfolio
```

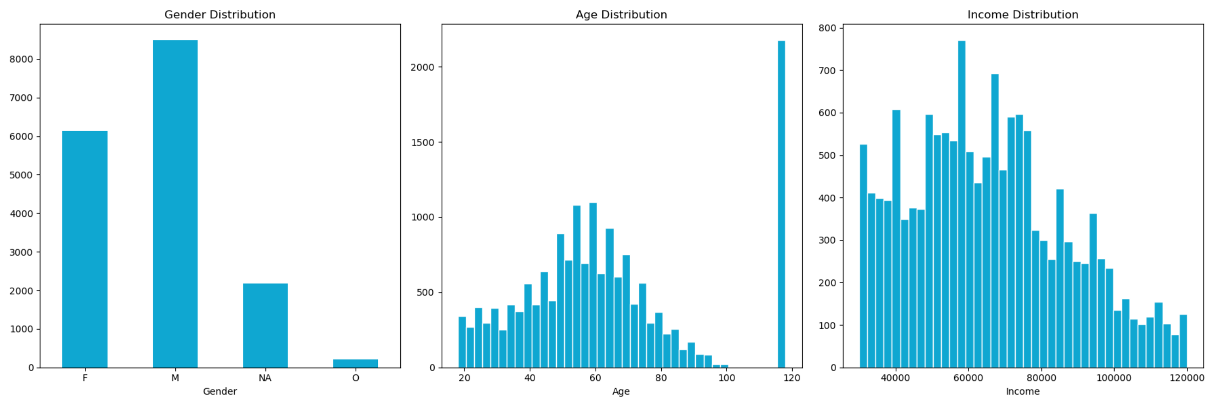
Final portfolio dataset:

	offer_reward	difficulty	offer_duration	offer_id	mobile	social	web	bogo	discount	informational
0	10	10	7	ae264e3637204a6fb9bb56bc8210ddfd	1	1	0	1	0	0
1	10	10	5	4d5c57ea9a6940dd891ad53e9dbe8da0	1	1	1	1	0	0
2	0	0	4	3f207df678b143eea3cee63160fa8bed	1	0	1	0	0	1
3	5	5	7	9b98b8c7a33c4b65b9aebfe6a799e6d9	1	0	1	1	0	0
4	5	20	10	0b1e1539f2cc45b7b9fa7c272da2e1d7	0	0	1	0	1	0
5	3	7	7	2298d6c36e964ae4a3e7e9706d1fb8c2	1	1	1	0	1	0
6	2	10	10	fafdcd668e3743c1bb461111dcafc2a4	1	1	1	0	1	0
7	0	0	3	5a8bc65990b245e5a138643cd4eb9837	1	1	0	0	0	1
8	5	5	5	f19421c1d4aa40978ebb69ca19b0e20d	1	1	1	1	0	0
9	2	10	7	2906b810c7d4411798c6938adc9daaa5	1	0	1	0	1	0

3.2. 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

Some profile data is missing both on income and gender information. Histograms were used to verify the presence of outliers.



Since the number of outliers in the ‘Age’ field and the registers with missing data were approximately 2 thousand, I suspected they were the same, what turned out to be the case.

Changes:

- Records with missing data/outliers identified;
- Some columns were renamed for clarity;
- ‘member days’ column calculated.
- Gender encoded.

```
profile = profile_.copy()
# Since the age outliers and missing age values are the same customers, we lets identify them

# identify the age outliers
profile['incomplete_data'] = profile['gender'].isnull()

#rename id column to customer_id, became_member_on to became_member_date and income to customer_income
profile.rename(columns={'id':'customer_id', 'became_member_on':'became_member_date', 'income':'customer_income'}, inplace=True)

# adjust became_member_date from string in format YYYYMMDD to datetime
profile['became_member_date'] = pd.to_datetime(profile['became_member_date'], format='%Y%m%d')

# create a new column with the number of days since the customer became a member
profile['memberdays'] = (datetime.datetime.today() - profile['became_member_date']).dt.days

# one hot encoding gender column
profile['gender_M'] = profile['gender'].apply(lambda x: 1 if x == 'M' else 0)
```

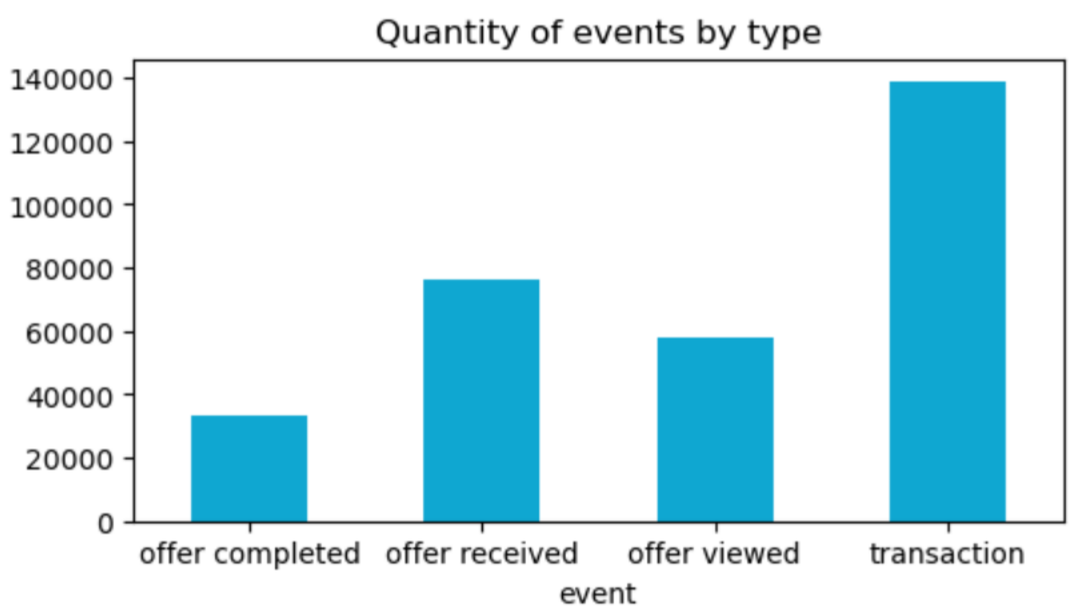
```
profile['gender_F'] = profile['gender'].apply(lambda x: 1 if x == 'F' else 0)
profile['gender_0'] = profile['gender'].apply(lambda x: 1 if x == '0' else 0)
```

	gender	age	customer_id	became_member_date	customer_income	incomplete_data	memberdays	gender_M	gender_F	gender_0
0	None	118	68be06ca386d4c31939f3a4f0e3dd783	2017-02-12	NaN	True	2293	0	0	0
1	F	55	0610b486422d4921ae7d2bf64640c50b	2017-07-15	112000.0	False	2140	0	1	0
2	None	118	38fe809add3b4fcf9315a9694bb96ff5	2018-07-12	NaN	True	1778	0	0	0
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	2017-05-09	100000.0	False	2207	0	1	0
4	None	118	a03223e636434f42ac4c3df47e8bac43	2017-08-04	NaN	True	2120	0	0	0

3.3 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

The main point of attention in this dataset is that the column values contain different information depending on whether the event is an offer or a transaction. There are 4 types of event not equali distibuted as shown in the follwing image.



Changes:

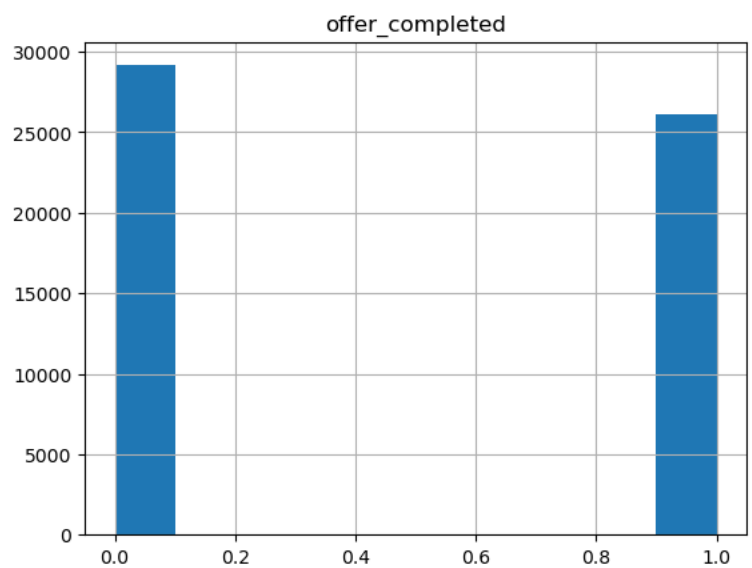
- The transcript dataset was modified by renaming the 'person' column to 'customer_id' for clarity.
- The 'value' column was split into 'offer_id' and 'amount' columns for better compatibility with ML algorithms.

- The 'event' column was one-hot encoded using a for loop to create new columns for each unique event.

```
transcript = transcript_.copy()
#rename person column to customer_id
transcript.rename(columns={'person':'customer_id'}, inplace=True)

transcript['offer_id'] = transcript['value'].apply(lambda x: x.get('offer id')  if x.get('offer id') != None else x.get('offer_id'))
transcript['amount'] = transcript['value'].apply(lambda x: x.get('amount'))
# transcript.drop('value', axis=1, inplace=True)

# one-hot encoding event column with a for loop
for event in transcript['event'].drop_duplicates().reset_index(drop=True):
    try:
        transcript[event.split(' ')[1]] = transcript['event'].apply(lambda x: 1 if x == event else 0)
    except:
        transcript[event.split(' ')[0]] = transcript['event'].apply(lambda x: 1 if x == event else 0)
```

	customer_id	event	value	time	offer_id	amount	received	viewed	transaction	completed
12658	9fa9ae8f57894cc9a3b8a9bbe0fc1b2f	offer completed	{'offer_id': '2906b810c7d4411798c6938adc9daaa5...	0	2906b810c7d4411798c6938adc9daaa5	NaN	0	0	0	1
12672	fe97aa22dd3e48c8b143116a8403dd52	offer completed	{'offer_id': 'fafdcd668e3743c1bb461111dcafc2a4...	0	fafdcd668e3743c1bb461111dcafc2a4	NaN	0	0	0	1
12679	629fc02d56414d91bca360decdfa9288	offer completed	{'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9...	0	9b98b8c7a33c4b65b9aebfe6a799e6d9	NaN	0	0	0	1

It should be noted that the ratio between completed and not completed offers is similar, so was not nedded to take any effort related do ballancing the dataset.



3.4 Final dataset

After treating each data source separately, I grouped the transcripts events by customer and offer for it to make sense from a analysis point of view, since its crucial to know what happed to every offer after being received. In a model building perpective, the offer_completed id the target value for every offer received records.

I merged them into a single dataset. I then created three new columns to identify whether the offer was completed, received, or if a transaction was made. We filtered the dataset to only include records where the offer was received, and dropped any records with incomplete data. Finally, we dropped unnecessary columns to create our final dataset that can now be used as an input to train our desired model.

```
offers = transcript.copy().groupby(['customer_id', 'offer_id']).sum()[[ 'viewed',  'received', 'completed', 'transaction']].reset_index
offers['percent_completed'] = offers['completed'] / offers['received']

df_ = offers.merge(portfolio, on='offer_id', how='inner').merge(profile, on='customer_id', how='inner')

df_['offer_completed'] = df_['percent_completed'].apply(lambda x: 1 if x > 0.5 else 0)
df_['offer_recieved']  = df_['received'].apply(lambda x: 1 if x > 0 else 0)
df_['transaction_made'] = df_['transaction'].apply(lambda x: 1 if x > 0 else 0)

df_ =df_.query('offer_recieved == 1')

df_ = df_[df_['imcomplete_data'] == False]
df_.drop(['imcomplete_data', 'became_member_date','gender'], axis=1, inplace=True)
df_.drop(['gender_0'], axis=1, inplace=True)
df_.drop(['customer_id', 'offer_id'], axis=1, inplace=True)
df_.drop(['index','viewed','received','completed','transaction','percent_completed','transaction_made','offer_recieved','gender_F'], ax

df_.shape
```

	offer_reward	difficulty	offer_duration	mobile	social	web	bogo	discount	informational	age	customer_income	memberdays	gender_M	offer_completed
0	5	5	7	1	0	1	1	0	0	52	71000.0	2868	1	0
1	0	0	3	1	1	0	0	0	1	52	71000.0	2868	1	0
2	2	10	7	1	0	1	0	1	0	52	71000.0	2868	1	1
3	5	5	7	1	0	1	1	0	0	32	37000.0	1944	1	0
4	10	10	7	1	1	0	1	0	0	32	37000.0	1944	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
63282	5	20	10	0	0	1	0	1	0	59	61000.0	3226	1	0
63283	5	20	10	0	0	1	0	1	0	56	35000.0	1848	0	0
63284	5	20	10	0	0	1	0	1	0	25	73000.0	2605	1	1
63286	5	20	10	0	0	1	0	1	0	64	72000.0	1820	1	0
63287	5	20	10	0	0	1	0	1	0	45	63000.0	2180	0	0

4. First model development

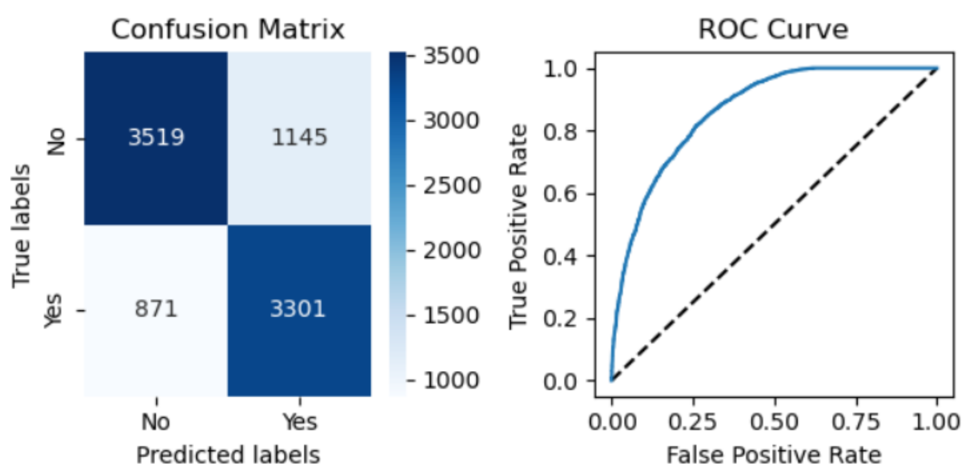
Since the data was already prepared and the fields of interest were defined, the process of testing a classification model was very straightforward. The model chosen was an XGBClassifier.

```
# creating a function to train and evaluate the model
def train_evaluate_model(model, X_train, y_train, X_val, y_val):
    # training the model
    model.fit(X_train, y_train)

    # predicting the model
    y_pred = model.predict(X_val)

    # evaluating the model
    accuracy = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred)
    recall = recall_score(y_val, y_pred)
    f1 = f1_score(y_val, y_pred)
    roc_auc = roc_auc_score(y_val, y_pred)

    # printing the metrics
    print('Accuracy: {:.2f}%'.format(accuracy*100))
    print('Precision: {:.2f}%'.format(precision*100))
    print('Recall: {:.2f}%'.format(recall*100))
    print('F1 Score: {:.2f}%'.format(f1*100))
    print('ROC AUC: {:.2f}%'.format(roc_auc*100))
```

```
    # plotting the confusion matrix and ROC AUC curve side by side
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(6, 3))

    # Confusion Matrix
    cm = confusion_matrix(y_val, y_pred)
    sns.heatmap(cm, annot=True, ax=ax1, cmap='Blues', fmt='g')
    ax1.set_xlabel('Predicted labels')
    ax1.set_ylabel('True labels')
    ax1.set_title('Confusion Matrix')
    ax1.xaxis.set_ticklabels(['No', 'Yes'])
    ax1.yaxis.set_ticklabels(['No', 'Yes'])

    # ROC Curve
    y_pred_proba = model.predict_proba(X_val)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_val, y_pred_proba)
    ax2.plot([0, 1], [0, 1], 'k--')
    ax2.plot(fpr, tpr)
    ax2.set_xlabel('False Positive Rate')
    ax2.set_ylabel('True Positive Rate')
    ax2.set_title('ROC Curve')

    # Adjust the spacing between plots
    plt.tight_layout()

    # Show the plots
    plt.show()

# training and evaluating the model
xgb = XGBClassifier(random_state=42)
train_evaluate_model(xgb, X_train, y_train, X_val, y_val)
```

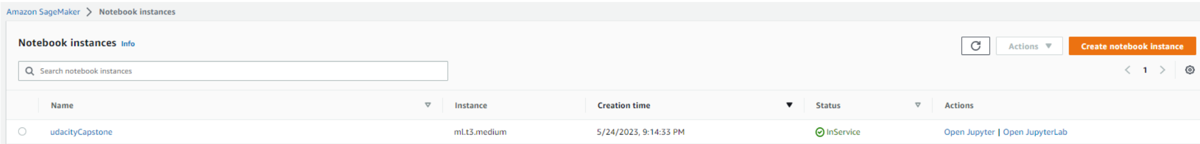
With an accuracy and ROC AUC of 77%, this type of modeling is considered satisfactory. The next step is to prepare the environment for deploying the solution with AWS.

```
Accuracy: 77.18%
Precision: 74.25%
Recall: 79.12%
F1 Score: 76.61%
ROC AUC: 77.29%
```



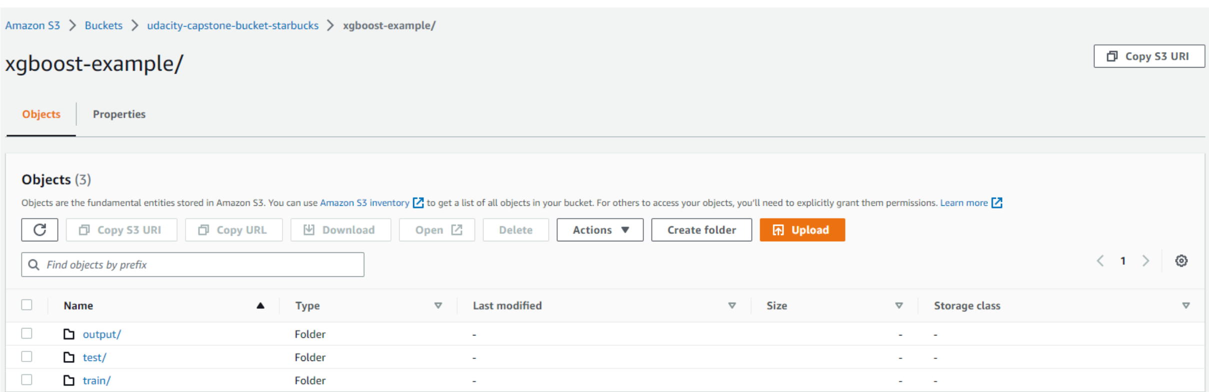
5. AWS Model training.

The first step in aws was to executed the code that was first run locally in the notebook instance created. An instance type ml.t3.medium was chosen.



After that, train and test data were saved in the instance in order to be sent to the S3 bucket created to this project.

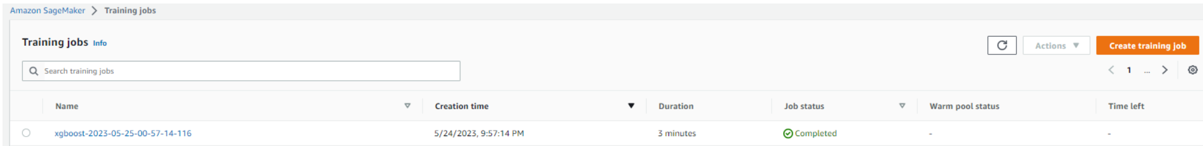
```
train_data = pd.concat([y_train,X_train], axis=1)
train_data.to_csv('train_data.csv', index=False,header=False)

test_data = pd.concat([y_test,X_test], axis=1)
test_data.to_csv('test_data.csv', index=False,header=False)

prefix = 'xgboost-example'
input_train = sagemaker_session.upload_data(path='train_data.csv', bucket=bucket, key_prefix=prefix + '/train')
input_test = sagemaker_session.upload_data(path='test_data.csv', bucket=bucket, key_prefix=prefix + '/test')
```



Having done this steps successfully, the estimator was built, had it's hyperparameters set and then trained. The training job can be observed above.

```
container = get_image_uri(boto3.Session().region_name, 'xgboost')

xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    instance_count=1,
                                    instance_type='ml.m4.xlarge',
                                    output_path=f's3://{bucket}/{prefix}/output',
                                    sagemaker_session=sagemaker_session)

xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        objective='binary:logistic',
                        num_round=100)

train_data = sagemaker.inputs.TrainingInput(s3_data=input_train, content_type='csv')
test_data = sagemaker.inputs.TrainingInput(s3_data=input_test, content_type='csv')

data_channels = {'train': train_data, 'validation': test_data}
```

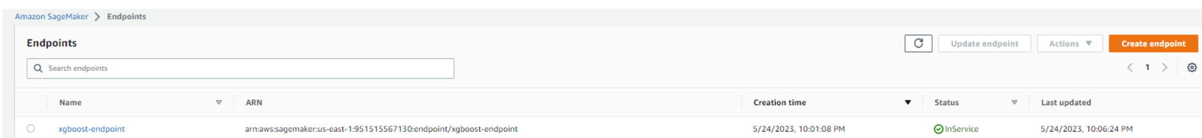
```
xgb.fit(inputs=data_channels)
```



6. Deployment and inference.

Once the training job is completed, on the same notebook I deployed the model with an instance 'ml.m4.xlarge', what is even to much processing power for a simple model such as he one created.

```
model_name = 'xgboost-model'
model = xgb.create_model(name=model_name)

endpoint_name = 'xgboost-endpoint'
predictor = model.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge', endpoint_name=endpoint_name)
```



Finaly, with some hardcoded data the endpoint was tested. This code could be transfered to a Lamnda function for more acessible usage.

```
import json
# Hardcoded data for prediction
hardcoded_data = {
"offer_reward":30.0,
"difficulty":30.0,
"offer_duration":30.0,
"mobile":1.0,
"social":1.0,
"web":1.0,
"bogo":1.0,
"discount":10.0,
"informational":1.0,
"age":58.0,
"customer_income":550000.0,
"memberdays":32981.0,
"gender_M":1.0
}

# Convert the hardcoded data to a list of values
input_data = [list(hardcoded_data.values())]
# Create a low-level SageMaker client
sagemaker_client = boto3.client('sagemaker-runtime')
# Specify the endpoint name
endpoint_name = 'xgboost-endpoint'
# Convert the input data to CSV format
input_data_csv = ','.join([str(val) for val in input_data[0]])
# Make a request to the endpoint
response = sagemaker_client.invoke_endpoint(
    EndpointName=endpoint_name,
```

```
    ContentType='text/csv',
    Body=input_data_csv
)

# Parse the response
prediction = response['Body'].read().decode()
prediction = json.loads(prediction)

if prediction > 0.5:
    print('The offer will likely be completed. (probability: {:.2f}%)'.format(prediction*100))
else:
    print('The offer will not likely be completed. (probability: {:.2f}%)'.format(prediction*100))
```

```
The offer will likely be completed. (probability: 82.38%)
```

7. Conclusion:

By leveraging the power of AWS services, I successfully implemented a Starbucks offer acceptance forecast model. Through the seamless integration of AWS tools and machine learning techniques, achieving 77% accuracy in predicting customer offer acceptance. The scalability and flexibility offered by AWS allowed us to efficiently deploy the model, ensuring its effectiveness in a dynamic business environment. This project achieved it’s objective.