"Propensify: Identifying Customer Response to Marketing Campaigns"

Greetings and welcome to this comprehensive notebook that walks you through the development, evaluation, and deployment of our groundbreaking 'Identifying Customer Response to Marketing Campaigns'. In the following sections, you will discover step-by-step instructions, insightful analyses, and hands-on demonstrations of our innovative approach to identifying customer response to marketing campaign.

Define IAM Role

```
# Define IAM role

import boto3
from sagemaker import get_execution_role

role = get_execution_role()
```

Dependencies and Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image
from IPython.display import display
from time import gmtime, strftime
import sys
import math
import json
import os
import sagemaker
```

Load Dataset:

Reads the both dataset(train (1), test(1)) from the specified S3 path and merge the both dataset into a Pandas DataFrame (df)

```
# Read dataset
dt = pd.read_csv('s3://propensify01/train (1).csv')
dte = pd.read_csv('s3://propensify01/test (1).csv')

# Add a column to each dataset indicating whether it's from train or
test
dt['dataset'] = 'train'
dte['dataset'] = 'test'

# Merge the datasets using concat
df = pd.concat([dt, dte], ignore_index=True)
```

Drop unnecessary columns

```
# Specify the columns to drop
columns_to_drop = ['profit', 'id', 'dataset']

# Drop the specified columns
df = df.drop(columns=columns_to_drop)

# Verify the DataFrame after dropping the columns
print(df.head())
```

EDA and data preparation

```
print(df.shape)
print(df.columns)

# Check for missing values in each column
df.isnull().sum()

# Check class distribution to determine balance or imbalance
df['responded'].value_counts()

# Generate summary statistics for numerical features in the dataset
df.describe()

# We have 'responsed' column contains 'yes' and 'no' values
# Mapping 'yes' to 1 and 'no' to 0
df['responded'] = df['responded'].map({'yes': 1, 'no': 0})

# Checking the first few rows to verify the changes
print(df.head())

# Balance the dataset by separating no_respo (Class 0) and yes_repo
(Class 1) transactions
no_respo = df[df.responded == 0]
yes_respo= df[df.responded == 1]

print(no_respo.shape)
```

```
print(yes_respo.shape)
```

Building a Sample Dataset with Balanced Distribution¶

```
no_respo_sample =  no_respo.sample(n = 928)
df = pd.concat([no_respo_sample, yes_respo],axis = 0)

df.shape

# Display the responded distribution in the sample dataset
df['responded'].value_counts()
```

Feature Engineering

```
df.columns

# Check the data types of each column
print(df.dtypes)

# Calculate the mean of each numeric feature grouped by the
'responded' column
numeric_columns = df.select_dtypes(include=['int', 'float']).columns
mean_by_responded = df.groupby('responded')[numeric_columns].mean()
print(mean_by_responded)

# Frequency tables for each categorical feature
for column in df.select_dtypes(include=['object']).columns:
    display(pd.crosstab(index=df[column], columns='% observations',
normalize='columns'))

# Histogram for each numeric features
display(df.describe())
%matplotlib inline
hist = df.hist(bins=30, sharey=True, figsize=(10, 10))
```

Correlation Analysis

```
# Drop non-numeric columns
numeric_df = df.select_dtypes(include=[np.number])

# Calculate correlations
correlation_matrix = numeric_df.corr()

# Display correlation matrix
display(correlation_matrix)

# Plot scatter matrix
pd.plotting.scatter_matrix(numeric_df, figsize=(12, 12))
plt.show()
```

Splitting Data into Features and Target Variables

Before training a machine learning model, it's necessary to separate the dataset into input features (independent variables) and the target variable (dependent variable). This allows us to train the model to predict the target variable based on the input features. The following code snippet demonstrates the process of splitting the dataset into input features (X) and the target variable (Y):

```
# Splitting the data into features and target variables
# X = Input features (independent features)
# Y = Dependent feature
X = df.drop('responded', axis=1)
Y = df['responded']

X.shape

Y.shape

train_data, validation_data, test_data =
np.split(model_data.sample(frac=1, random_sate=1729),
                                                    [int(0.7 *
len(model_data)). int(0.9 * len(model_data))])

print(train_data.shape)

print(train_data.shape)

print(validation_data.shape)
```

Save datasets to CSV files

We have split our dataset into three parts: training, testing, and validation sets. Now, we will save these datasets as CSV files.

```
# Save datasets to CSV files
train_data.to_csv('train.csv', index=False)
test_data.to_csv('test.csv', index=False)
validation_data.to_csv('validation.csv', index=False)

# Upload datasets to S3
s3 = boto3.client('s3')
bucket_name = 'propensify01'
s3.upload_file('train.csv', 'propensify01',  'dataset/train.csv')
s3.upload_file('test.csv', 'propensify01', 'dataset/test.csv')
s3.upload_file('validation.csv', 'propensify01',
'dataset/validation.csv')
```

Training Model

Retrieve XGBoost Container Image URI

To use the XGBoost algorithm in Amazon SageMaker, we need to retrieve the container image URI. We can do this using the image_uris.retrieve function provided by SageMaker.

```
from sagemaker import image_uris
container = image_uris.retrieve('xgboost', region='us-east-1',
version='latest')
```

SageMaker XGBoost Model Training

To train an XGBoost model using Amazon SageMaker, we first need to set up the SageMaker session, define the estimator, set hyperparameters, define data channels for training, and then train the model.

```
import sagemaker
from sagemaker import get_execution_role
from sagemaker.amazon.amazon_estimator import get_image_uri

# Create a SageMaker session
sess = sagemaker.Session()

# Get the ECR container URI for XGBoost algorithm
container = get_image_uri(sess.boto_region_name, 'xgboost',
repo_version="latest")

# Define the estimator
xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    instance_count=1,
                                    instance_type='ml.m4.xlarge',

output_path='s3://{}/{}/{}'.format('propensify01', 'output', 'xgboost-
model'),
                                    sagemaker_session=sess)

# Set hyperparameters
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        num_round=100)

# Define data channels for training
train_channel =
```

```
sagemaker.session.s3_input('s3://{}/{}/train.csv'.format('propensify01
', 'dataset'), content_type='text/csv')
validation_channel =
sagemaker.session.s3_input('s3://{}/{}/validation.csv'.format('propens
ify01', 'dataset'), content_type='text/csv')

# Train the model
xgb.fit({'train': train_channel, 'validation': validation_channel})
```

Deploying model

To deploy the trained XGBoost model using Amazon SageMaker, we can use the deploy method of the estimator. This will create an endpoint for real-time inference.

```
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type = 'ml.m4.xlarge'
```

Setting Serializer for XGBoost Predictor

To ensure that the input data sent to the deployed XGBoost model endpoint is properly formatted, we can set the serializer to CSVSerializer provided by SageMaker.

```
from sagemaker.serializers import CSVSerializer
xgb_predictor.serializer = CSVSerializer()
```

To make predictions using the deployed XGBoost model, you can use the following `predict` function:

```
def predict (df, rows = 186):
    split_arry = np.array_split(df, int(df.shape[0] / float(rows) +
1))
    predictions = ''
    for array in split in split_array:
        prediction = ','.json([predictions,
xgb_predictor.predict(array).decode('utf-8')])

    return np.fromstring(predictions[1:], sep=',')

predictions = predict(test_data.drop(['no_respo', 'yes_respo'],
axis=1).to_numpy())


pd.crosstab(index=test_data['yes_respo'],
column=np.round(predictions),
rownames=['actuals'],colnames=['predictions'])
```

To delete the deployed endpoint associated with the XGBoost model, you can use the `delete_endpoint` method as follows:

```
xgb_predictor.delete_endpoint()
```

Confusion Matrix and ROC AUC Score

The following code calculates the confusion matrix and ROC AUC score to evaluate the performance of the model:

```
from sklearn.metrics import confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

# Make predictions
predictions = predict(test_data.drop(['no_respo', 'yes_respo'],
axis=1).to_numpy())

# Generate confusion matrix
cm = confusion_matrix(test_data['yes_respo'], np.round(predictions))

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Calculate ROC AUC score
roc_auc = roc_auc_score(test_data['yes_respo'], predictions)
print("ROC AUC Score:", roc_auc)
```

Classification report

```
from sklearn.metrics import classification_report

# Calculate precision, recall, f1-score, and support
report = classification_report(test_data['yes_respo'],
np.round(predictions))

print("Classification Report:")
print(report)
```

Thank You

Riddhi Sharma