Subhadeep Chatterjee
@schatterjeecs

Starbucks Capstone Project

Udacity - Machine Learning Engineer Nanodegree

Project Report

I. Definition

Project Overview

The dataset contains simulated data that mimics customer behavior on the Starbucks rewards mobile app. Starbucks sends out an offer to users of the mobile app. An offer can be merely an advertisement for a drink (Informational) or an actual offer such as a discount or BOGO.

The challenge with this dataset was that there could be a lot of permutations and combinations based on the multiple features available in the dataset. Also, the data cleaning and preprocessing was one of the very crucial stages we need to take care of, while working with this dataset.

Our task is to combine the transactions, demographic and offer data to determine given a customer profile and transactions, how likely it will be for that customer to accept and complete the offer.

Every offer has a validity period before it expires. As an example, a BOGO offer might be valid for 7 or 5 days only depending on the mode of communication (channels) and difficulty. Similarly, Informational offers also have a validity period of 3 or 4 days even though these advertisements are merely providing information about a product. Discount offers might be valid for 7 or 10 days again based on the same factors.

Problem Statement

Starbucks wants to find a way to give each customer the right in-app special offer. That is, how likely will the customer accept the offer that is sent to them.

Our goal is to analyze historical data about the offers transacted by the customers and develop an algorithm that can improve the conversion rate of the offers and thereby increase the overall profit.
We will use the dataset which has been provided by Starbucks for this task.

Evaluation Metrics

I will mostly leverage the statistical metrics provided by sklearn library and they include the following:

- Precision/Recall - which will help determine the percentage of true positives.
- F1-score
- Accuracy

Accuracy = (TP + TN)/(TP + FP + FN + TN)

where,
*TP - True Positives (which means a result indicating correct positive class predictions)
*TN - True Negatives (which means a result indicating correct negative class predictions)
*FP - False Positives (which means a result indicating that a given condition is present while it is actually not)
*FN - False Negatives (which means a result indicating that a given condition is not present while it is actually present)

Mostly 80% of the data will be leveraged for training purposes and the rest will be reserved for testing. On top of that I will also cook up some random data to see if the model is able to classify them correctly.

II. Analysis

Data Exploration

The data is contained in three files:

- portfolio.json - containing offer ids and meta data about each offer (duration, type, etc.)
- profile.json - demographic data for each customer
- transcript.json - records for transactions, offers received, offers viewed, and offers completed

Here is the schema and explanation of each variable in the files:

portfolio.json

- id (string) - offer id
- offer_type (string) - type of offer ie BOGO, discount, informational
- difficulty (int) - minimum required spend to complete an offer
- reward (int) - reward given for completing an offer
- duration (int) - time for offer to be open, in days
- channels (list of strings)

```
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

profile.json

- age (int) - age of the customer
- became_member_on (int) - date when customer created an app account
- gender (str) - gender of the customer (note some entries contain 'O' for other rather than M or F)
- id (str) - customer id
- income (float) - customer's income

```
profile.head(3)
```

	gender	age	id	became_member_on	income
1	F	55	0610b486422d4921ae7d2bf64640c50b	20170715	112000.0
3	F	75	78afa995795e4d85b5d9ceeca43f5fef	20170509	100000.0
5	M	68	e2127556f4f64592b11af22de27a7932	20180426	70000.0

transcript.json

- 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 - (dict of strings) - either an offer id or transaction amount depending on the record

```
transcript.head(3)
```

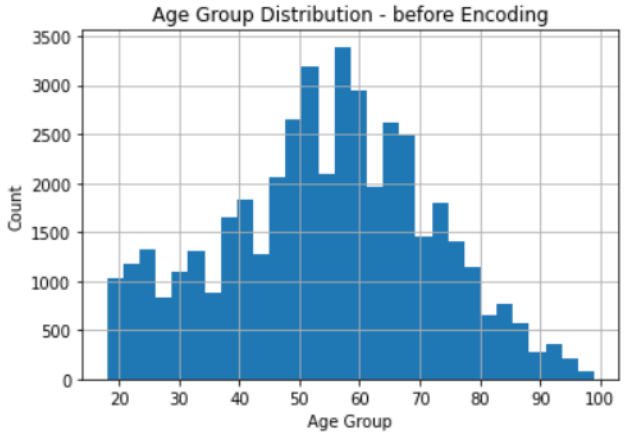
	person	event	value	time
0	78afa995795e4d85b5d9ceeca43f5fef	offer received	{'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}	0
1	a03223e636434f42ac4c3df47e8bac43	offer received	{'offer id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}	0
2	e2127556f4f64592b11af22de27a7932	offer received	{'offer id': '2906b810c7d4411798c6938adc9daaa5'}	0

Exploratory Visualization

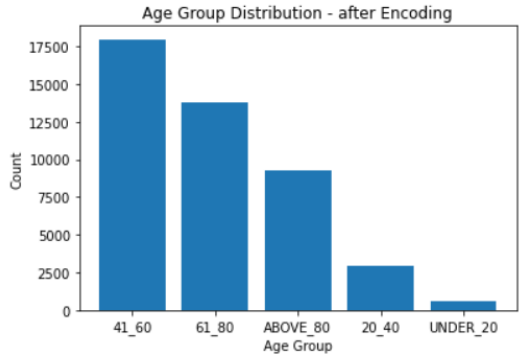
1. **Age Group:** I derived age group data by segregating each customer based on their ages.

   Observations:
   - Outliers are present.
   - Ages more than 100 years does not really make sense, because a customer in that age group is very unlikely to think about offers. So, only ages less than 100 years have been considered for our analysis.
   - Average customer age ranges between 49 to 63

```
df.age.hist(bins = 30)
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Age Group Distribution - before Encoding');
```




```
plt.bar(df['age_group'].unique(), df['age_group'].value_counts())
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.title('Age Group Distribution - after Encoding');
plt.show()
```

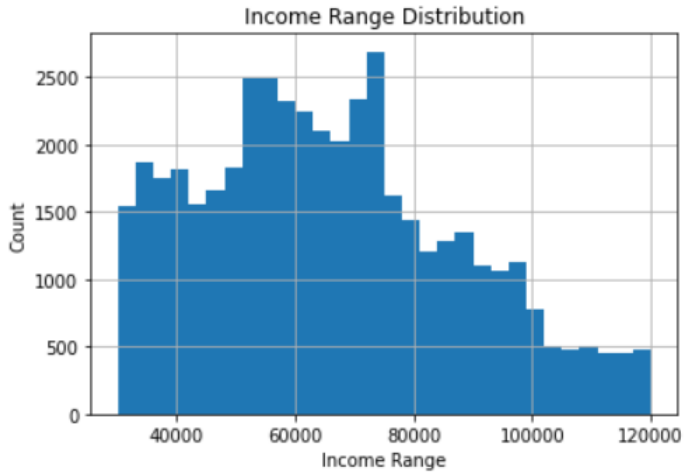


2. **Income Range:**

Observations:

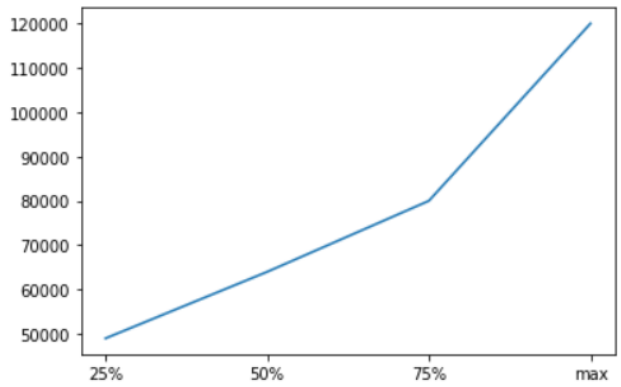
- Average income users fall from 62000 to 74000 .

```
df.income.hist(bins = 30);
plt.xlabel('Income Range')
plt.ylabel('Count')
plt.title('Income Range Distribution');
```




```
plt.plot(df.income.describe()[['25%', '50%', '75%', 'max']])
```

```
[<matplotlib.lines.Line2D at 0x7f819c9fde50>]
```

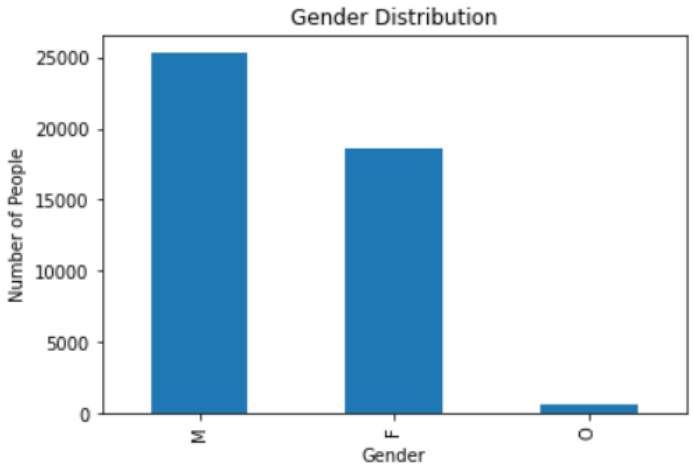


3. **Gender:**

Observations:

- Male proportions are more than 50% of the total users.

```
gend = df.gender.value_counts()
gend.plot(kind='bar')
plt.ylabel('Number of People')
plt.xlabel('Gender')
plt.title('Gender Distribution');
```



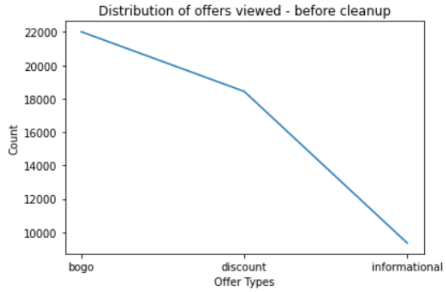
4. **Offers classification:**

Observations:

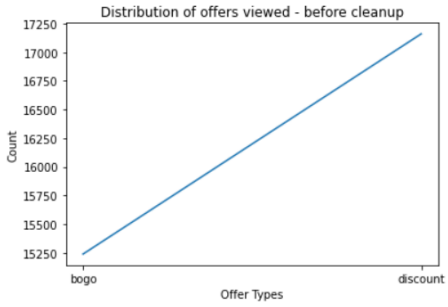
- Majority of the offers received are of type BOGO.
- Majority of the offers completed are of type DISCOUNT
- Interestingly, the distribution of offers completed is Linear across all the given types
- Here, “offers viewed” won’t really make an impact on completion of the offer. Reason - offer received and offer viewed followed similar pattern and that even though majority of the users viewed BOGO offers but BOGO has been the least offer which has been completed. Hence, we will concentrate mainly on two event types - “Offer Received” and “Offer Completed”

```
plt.plot(df_offer_final[df_offer_final['event']=='offer received'].groupby(['offer_type']).count()['person_id'])
plt.ylabel('Count')
plt.xlabel('Offer Types')
plt.title('Distribution of offers received - before cleanup');
```


```
plt.plot(df_offer_final[df_offer_final['event']=='offer viewed'].groupby(['offer_type']).count()['person_id'])
plt.ylabel('Count')
plt.xlabel('Offer Types')
plt.title('Distribution of offers viewed - before cleanup');
```

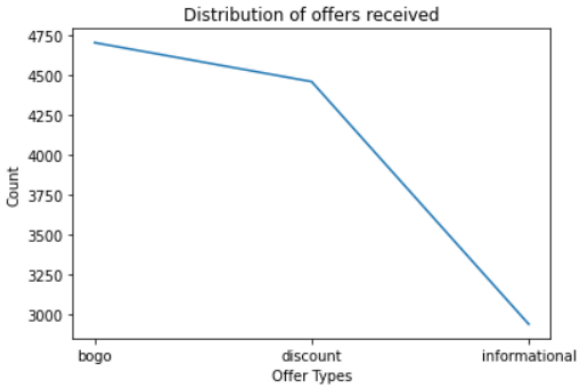



```
plt.plot(df_offer_final[df_offer_final['event']=='offer completed'].groupby(['offer_type']).count()['person_id'])
plt.ylabel('Count')
plt.xlabel('Offer Types')
plt.title('Distribution of offers viewed - before cleanup');
```

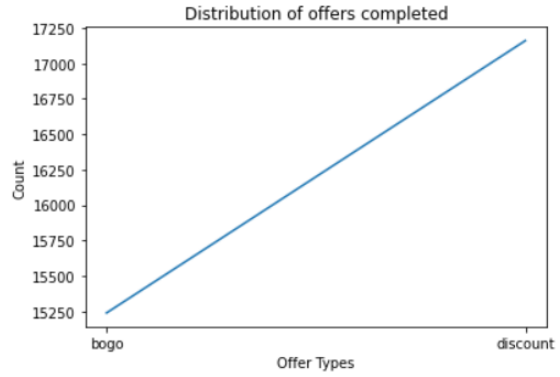


After Cleanup

```
plt.plot(df[df['event']=='offer received'].groupby(['offer_type']).count()['person_id'])
plt.ylabel('Count')
plt.xlabel('Offer Types')
plt.title('Distribution of offers received');
```




```
plt.plot(df[df['event']=='offer completed'].groupby(['offer_type']).count()['person_id'])
plt.ylabel('Count')
plt.xlabel('Offer Types')
plt.title('Distribution of offers completed');
```



Algorithms and Techniques

I believe the fact that we should use simple machine learning algorithms before leveraging any Deep Learning techniques. Because ML algorithms are less expensive, if we can achieve the good results and near equivalent results corresponding to the Deep learning techniques, then we should go with the ML model instead of the DPL models.

Having said that, as part of this project work, I have tried out several ML algorithms and the one which stood out with great results was XGBoost Algorithm.

XGBoost is a decision-tree based ensemble Machine Learning algorithm that uses a gradient boosting framework. This algorithm applies the principle of boosting weak learners using gradient descent architecture. On top of that it also improves upon the base GBM framework through system optimization and algorithmic enhancements.

Some System optimization techniques include Parallelization, Tree Pruning and Hardware Optimization.

Besides choosing the right algorithm, it is very important to choose the right configuration of the algorithm for a given dataset by hyper-parameter tuning. While tuning the parameters we should make sure not to overfit or underfit the model.

Benchmark Model

Model evaluation will mainly include comparing the confusion matrix, accuracy scores after trying out several Machine learning models. Principal Component Analysis will also be applied on the dataset to reduce dimensionality, noise removal. Although PCA is mostly used on unsupervised algorithms, still this gives good results on some aspects while dealing with supervised algorithms is what I have observed. After exploring the datasets I felt that for the current goal of this project, ML algorithms should be good enough to bring out the best results and hence, we won't try any deep learning algorithms as such.

III. Methodology

Data Preprocessing

Techniques such as Data grouping or classification, Encoding, Dataset split up, Feature Scaling, Standardization and Normalization of the data are important in this preprocessing phase.

1. **Data classification:**
    a. Customer ages will be grouped under derived categories - [‘UNDER_20’, ‘20_40’, '41_60', '61_80', 'ABOVE_80']
    b. Membership age will be derived from present date till the customers’ date of joining

2. **One-Hot Encoding** :
    a. Channels, offer_type, age_group, gender and event columns will be encoded

3. **MinMaxScaler**:
    a. Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1.
    b. Standardization is another scaling technique where values are centered around the mean with a unit standard deviation.
    c. Features will be scaled using sklearn preprocessing MinMaxScaler library.

4. **Train_Test_Split**:
    a. Dataset will be split up for training and testing
    b. 20% of data will be used for testing and the rest for training the model.

Features: `['income', 'time', 'difficulty', 'duration', 'reward', 'membership_age',`
`'email', 'mobile', 'social', 'web', 'bogo', 'discount', 'informational',`
`'20_40', '41_60', '61_80', 'ABOVE_80', 'UNDER_20', 'F', 'M', 'O']`
Label： `['event']`

Implementation

I have mostly leveraged the below mentioned models from sklearn library.

1. **MultinomialNB**
2. **DecisionTreeClassifier**
3. **KNeighborsClassifier**
4. **SVM**
5. **Kernel SVM**
6. **RandomForrestClassifier**
7. **XGBoost**

After finding the optimal model, the model was saved in pickle format. Once a model is saved it can be reused by invoking it with any new dataset containing similar features and with similar purpose.
Transformation functionality: I have written a transformation function so that we can use the same saved model to perform prediction on the newly fed dataset.

Refinement

It has been an iterative learning process.
Initially the model was underfit, that is, High Bias. Parameter tuning was required, and in fact the data cleaning was required to remove the noise from the data. For example, initially I went ahead with the exact age values and did not consider the become_member_on feature, which led to poor accuracy, because these features really play a vital role. Specially, for a newly registered customer it is observed that the customer is more interested in offers than the old customers. Also, age group played a vital role in the predictions.

Few ways to overcome underfitting -

1. Select the correct features (the features which really creating an impact on the model)
2. Reduce the data variance within a model

IV. Results

Model Evaluation and Validation

Below ML models have been applied on the given dataset, their evaluation metrics and validations have been captured.

1. **MultinomialNB**

```
Confusion Matrix:
[[ 632 1781]
 [  23 6465]]
Accuracy: 0.7973261431299854
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.26      0.41      2413
           1       0.78      1.00      0.88      6488

    accuracy                           0.80      8901
   macro avg       0.87      0.63      0.64      8901
weighted avg       0.83      0.80      0.75      8901
```

2. **DecisionTreeClassifier**

```
Confusion Matrix:
[[2128  285]
 [ 251 6237]]
Accuracy: 0.9397820469610156
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      2413
           1       0.96      0.96      0.96      6488

    accuracy                           0.94      8901
   macro avg       0.93      0.92      0.92      8901
weighted avg       0.94      0.94      0.94      8901
```

3. KNeighborsClassifier

```
Confusion Matrix:
[[1646  767]
 [ 731 5757]]
Accuracy: 0.831704302887316
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.68      0.69      2413
           1       0.88      0.89      0.88      6488

    accuracy                           0.83      8901
   macro avg       0.79      0.78      0.79      8901
weighted avg       0.83      0.83      0.83      8901
```

4. SVM

```
Confusion Matrix:
[[1091 1322]
 [ 201 6287]]
Accuracy: 0.8288956297045276
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.45      0.59      2413
           1       0.83      0.97      0.89      6488

    accuracy                           0.83      8901
   macro avg       0.84      0.71      0.74      8901
weighted avg       0.83      0.83      0.81      8901
```

5. Kernel SVM

```
Confusion Matrix:
[[1121 1292]
 [ 246 6242]]
Accuracy: 0.8272104257948545
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.46      0.59      2413
           1       0.83      0.96      0.89      6488

    accuracy                           0.83      8901
   macro avg       0.82      0.71      0.74      8901
weighted avg       0.83      0.83      0.81      8901
```

6. RandomForrestClassifier

```
Confusion Matrix:
[[1894  519]
 [ 518 5970]]
Accuracy: 0.8834962363779351
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.78      0.79      2413
           1       0.92      0.92      0.92      6488

    accuracy                           0.88      8901
   macro avg       0.85      0.85      0.85      8901
weighted avg       0.88      0.88      0.88      8901
```

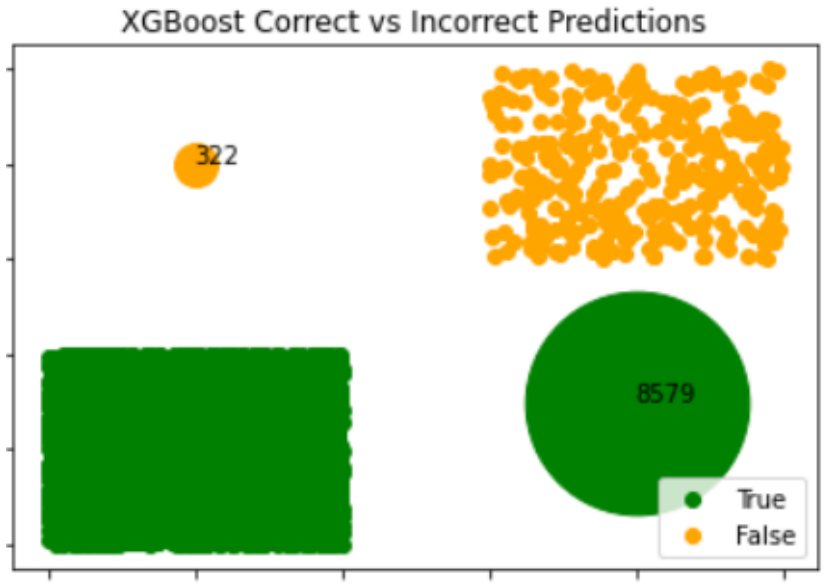
7. XGBoost

```
Confusion Matrix:
[[2272  141]
 [ 181 6307]]
Accuracy: 0.9638242894056848
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93      2413
           1       0.98      0.97      0.98      6488

    accuracy                           0.96      8901
   macro avg       0.95      0.96      0.95      8901
weighted avg       0.96      0.96      0.96      8901
```

XGBoost Model accuracy is ~96% which is great.



Justification

Mocked up some data to validate how the model predicts.

```
prof = [{"gender": "M", "age": 32, "id": "0b486422d4921ae7d2",
                            "became_member_on": "20210627", "income": 20982322},
                            {"gender": "F", "age": 31, "id": "0b4864225645ae7d2",
                            "became_member_on": "20210627", "income": 342324}]
new_profile = pd.DataFrame(prof)
trans = [{"person": "0b486422d4921ae7d2", "event": "offer received", "time": 0,
                                "value": {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}},
         {"person": "0b486422d4921ae7d2", "event": "offer viewed", "time": 10,
                                "value": {'offer id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}},
         {"person": "0b486422d4921ae7d2", "event": "offer received", "time": 200,
                                "value": {'offer_id': '0b1e1539f2cc45b7b9fa7c272da2e1d7'}},
         {"person": "0b4864225645ae7d2", "event": "offer received", "time": 0,
                                "value": {'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}},
         {"person": "0b4864225645ae7d2", "event": "offer completed", "time": 23,
                                "value": {'offer_id': '9b98b8c7a33c4b65b9aebfe6a799e6d9'}}]
new_transaction = pd.DataFrame.from_dict(trans)
```

```
X_new, y_new = transform_data(profile=new_profile, transcript=new_transaction, portfolio=portfolio)
```

```
Index(['income', 'time', 'difficulty', 'duration', 'reward', 'membership_age',
       'email', 'mobile', 'web', 'social', 'bogo', 'discount', 'informational',
       '20_40', 'UNDER_20', '41_60', '61_80', 'ABOVE_80', 'F', 'M', 'O'],
      dtype='object')
X:
[[0.    0.115 0.    0.    0.    0.    0.    1.    0.    0.    1.    0.
  0.    0.    0.    0.    0.    0.    1.    0.    0.   ]
 [1.    0.    0.    0.    0.    0.    0.    1.    0.    0.    1.    0.
  0.    0.    0.    0.    0.    0.    0.    1.    0.   ]
 [1.    1.    1.    1.    0.    0.    0.    0.    0.    0.    0.    1.
  0.    0.    0.    0.    0.    0.    0.    1.    0.   ]]
y:
[1 0 0]
```

```
y_pred_new = model.predict(X_new)
y_pred_new
```

```
array([0, 0, 1])
```

Predictions did not give good outcomes. Model could only predict one of the outcomes correctly out of 3.

V. Conclusion

Reflection

We ended up encoding the training data with 21 features, which eventually gave us the most optimal results when tested on the Test data.

Although the trained model gave ~96% accuracy on the test data, it's accuracy was very poor (~33%) when some random data was fed in for prediction on the same trained model.

Probably causes of this result -

1. The mocked up data might be missing some important parameters if the assumptions taken on whether the customer will accept the offer or not were incorrect.
2. More data might need to be fed into the model for training it to cover various scenarios.

```
cm = confusion_matrix(y_new, y_pred_new)
print(f"Confusion Matrix:\n{cm}")
accuracy = accuracy_score(y_new, y_pred_new)
print(f"Accuracy: {accuracy}")
print(f"Classification Report:\n{classification_report(y_new, y_pred_new)}")
```

```
Confusion Matrix:
[[1 1]
 [1 0]]
Accuracy: 0.3333333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50         2
           1       0.00      0.00      0.00         1

    accuracy                           0.33         3
   macro avg       0.25      0.25      0.25         3
weighted avg       0.33      0.33      0.33         3
```

Improvement

The result we got when tried on the new dataset (~33%) is ambiguous as I have mentioned above factors which might have caused this issue. Still it is worth giving it a try by using some of the Deep learning models and finding the outcomes of those results.
Hence, as a future scope of work we can use Tensorflow Deep learning networks, create full fledged APIs (one for Model training and creating the model) and the other for real-time predictions.

References

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/modules/svm.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://xgboost.readthedocs.io/en/latest/

Acknowledgement

I thank Udacity & Starbucks for facilitating such a great project. It is a great learning experience.

Thank You