Artificial Intelligence (CSE 422)

Classification of Customer Base based on Promotion Proclivity

24 / 08 / 2022

Submitted By:

Group#1

Md Nawaz-S-Salekeen Nayeem (17301082)

Eshrak Ahmed (19301003)

Sadia Alam (18301200)

Washik Al Mahmud (18101179)

Section: 11

Submitted to:

Sk. Atik Tajwar

Sumaiya Akter

Computer Science and Engineering

School of Data and Sciences

BRAC UNIVERSITY

Classification of Customer Base based on Promotion Proclivity

Introduction

One of the essential business requirements in a company is to determine the usefulness of a promotion that has been run towards its customer base. Mainly to determine which customers given the promotion will respond positively and which will not be affected at all. As running a promotion is costly it's very important to determine the pre-mentioned result as it will help the company to optimize their cost by running a promotion to certain types of customers who they seem will respond according to their desired metric. To make this happen a company might collect data about their customers, their buying habits, their history of response to previous promotions and so and so forth. Given all these necessities are met, a company may apply data science techniques to find the solution they are looking for.

In this project, we are trying to imitate just that by using a dataset that seems suitable for the problem with proper data cleaning, preprocessing, and model building.

Methodology

The way we have framed our solution is by using *Supervised Binary Classification.* Where the class set defined as $[0, 1]$ and encoded as `0:No Response`, `1: Response`; context being the promotion. Framing the problem this way allows us to use various tools across different domains from Business Intelligence, Machine Learning, Data Science, and Statistical Learning; as though these domains overlap in their practices.

The primary hurdle that needed to be overcome to get to the solution was getting suitable data. As any Data Science project lives or dies according to the quality of the data. The dataset we collected was from kaggle.com, one of the most reputable sources to practice Data Science and get data for learning purposes.

For data cleaning and preprocessing we have used **Pandas** and for model building and evaluation **Sickit-Learn** has been used. For visualization, during Exploratory Data Analysis (EDA) **Matplotlib** and **Seaborn** have been used. All these libraries are Python-based and they use **Numpy** for their vector computation, enabling us to avoid the dreaded python loop which is extremely slow.

Dataset Description

The shape of our dataset is $(2240, 29)$, meaning a sample size of 2240 and contains 29 features.

A brief description of features is given below:

People

- ID: Customer's unique identifier - *Categorical*
- Year_Birth: Customer's birth year - *Categorical*
- Education: Customer's education level - *Categorical(Ordinal)*
- Marital_Status: Customer's marital status - *Categorical*
- Income: Customer's yearly household income - *Quantitative*
- Kidhome: Number of children in customer's household - *Categorical*
- Teenhome: Number of teenagers in customer's household - *Categorical*
- Dt_Customer: Date of customer's enrollment with the company - *Quantitative*
- Recency: Number of days since customer's last purchase - *Quantitative*
- Complain: 1 if the customer complained in the last 2 years, 0 otherwise - *Categorical*

Products

- MntWines: Amount spent on wine in last 2 years - *Quantitative*
- MntFruits: Amount spent on fruits in last 2 years - *Quantitative*
- MntMeatProducts: Amount spent on meat in last 2 years - *Quantitative*
- MntFishProducts: Amount spent on fish in last 2 years - *Quantitative*
- MntSweetProducts: Amount spent on sweets in last 2 years - *Quantitative*
- MntGoldProds: Amount spent on gold in last 2 years - *Quantitative*

Promotion

- NumDealsPurchases: Number of purchases made with a discount - *Quantitative*
- AcceptedCmp1: 1 if customer accepted the offer in the 1st campaign, 0 otherwise - *Categorical*
- AcceptedCmp2: 1 if customer accepted the offer in the 2nd campaign, 0 otherwise - *Categorical*
- AcceptedCmp3: 1 if customer accepted the offer in the 3rd campaign, 0 otherwise - *Categorical*

- AcceptedCmp4: 1 if customer accepted the offer in the 4th campaign, 0 otherwise - *Categorical*
- AcceptedCmp5: 1 if customer accepted the offer in the 5th campaign, 0 otherwise - *Categorical*
- Response: 1 if the customer accepted the offer in the last campaign, 0 otherwise - *Categorical*

Place

- NumWebPurchases: Number of purchases made through the company's website - *Quantitative*
- NumCatalogPurchases: Number of purchases made using a catalog - *Quantitative*
- NumStorePurchases: Number of purchases made directly in stores - *Quantitative*
- NumWebVisitsMonth: Number of visits to the company's website in the last month - *Quantitative*

One of the things we had to do is to define our *Input Features (X)* and *Target Features (y).* The way we have defined our target feature is by checking whether a sample accepted an offer in any of the promotion categories if so the value is 1 otherwise 0. As this will be consistent with our framing of binary classification. All other features except for ID have been used as *Input Features.*

Pre-Processing Techniques applied

Train-Test Split

To avoid data leakage which can plague our entire pipeline we have divided the raw dataset into two parts Train Set, and Test Set. Hyperparameter Tuning and Model Selection will be done with Train Set and final model evaluation will be done on Test Set.

- **Missing Value Handling:**

During EDA we noticed that only the feature ***Income*** had missing values of size 24 which is around 1% of our whole sample size. As this is only a small portion of the dataset, we just removed those samples from our dataset. This way we are not adding extra biases to our system by using any statistical method.

- **Outliers and Error Analysis:**

One thing to keep in mind is that outlier detection that does not make sense for categorical variables. So our focus was on numerical variables. We used a boxplot to detect outliers in

the dataset. There were three samples in ***Year_Birth*** features considered to be outliers. We removed those samples. In the ***Marital_Status*** column, we encountered values like [Alone, Absurd, YOLO]. For consistency, we encoded these values to Single. In the ***Income*** feature, there was only one outlier, we removed that.

Now it should be noted that whether a value is an outlier or not obviously a subjective choice and depends on the data analyst. Although to be even considered to be an outlier we had made sure it falls outside the 25th and 75th percentile of the values of the given feature.

- **Encoding Categorical Features**

The values in ***Education*** column were [Graduation,PhD,Master,2n Cycle,Basic]. These values needed to convert to numerics. For that, we used Ordinal Encoding as there is intrinsic order in the feature values. The order we had chosen was: PhD>2n Cycnle>Master>Graduation>Basic.

Marital_Status column values are encoded using One-Hot-Encoder as there is no intrinsic order in the domain.

The feature ***Dt_Customer*** was a DateTime feature and it was encoded as a string (object in Pandas DataFrame) in the raw data. To make this usable we first converted to a Datetime Feature. And we noticed that there's no intrinsic cycle in the domain thus eliminating the need for Sin-Cos transformation. What we did, we counted how many days a customer is with the company from today. And encoded that as an *Integer* in the feature. Code snippet is given below:

```
from datetime import datetime
dates = pd.to_datetime(data['Dt_Customer'].astype('str'))
elapsed_time = (datetime.now() - dates)
elapsed_days = elapsed_time.apply(lambda x: x.days)
data['Dt_Customer'] = elapsed_days
```

- **Feature Scaling**

There are basically two scaling techniques to apply depending on the scenario. Normalization or Standardization. We used Standardization as most of the features are normally distributed as observed in the EDA part of this project. In Sickit-Learn its implemented as Standard Scaler. Also, as not all features not need to be transformed, we used ColumnTransformer from Sickit-Learn to make our life easier.

The code snippet is given below for the transformation:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
```

```
cols = ['Year_Birth','Education', 'Income', 'Kidhome',
        'Teenhome', 'Dt_Customer', 'Recency', 'MntWines', 'MntFruits',
        'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts',
        'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
        'NumCatalogPurchases', 'NumStorePurchases',
        'NumWebVisitsMonth', 'Z_CostContact', 'Z_Revenue']

# Init the Standard Scaler
scaler = StandardScaler()

# Init the Column Transformer
ct = ColumnTransformer([
    ('standarizer', scaler, cols)
], remainder='drop'
scaled_features = pd.DataFrame(ct.fit_transform(data),columns=cols)
```

Models applied

Before we discuss our model it is to be noted that, one of the primary concerns while building a model is keeping in mind the evaluation process and hyperparameter tuning. For that reason, the Cross Validation needed to be used as our dataset was rather small and we could not divide it into three sets. We use StratifiedKFold from sklearn to as our cross validator to keep the class proportion intact.

- **Decision Trees:**

In a nutshell, A decision tree is an acyclic graph that can be used to make decisions. In each branching node of the graph, a specific feature j of the feature vector is examined. If the value of the feature is below a specific threshold, then the left branch is followed; otherwise, the right branch is followed. As the leaf node is reached, the decision is made about the class to which the example belongs.

We used DecisonTreeClassifier from sklearn with the default parameter. Which are given below:

```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 42,
 'splitter': 'best'}
```

- **Support Vector Machine:**

The Support Vector Machine (SVM) is a linear classifier that can be viewed as an extension of the Perceptron developed by Rosenblatt in 1958. The Perceptron guaranteed that you find **a** hyperplane if it exists. The SVM finds the **maximum margin** separating the hyperplane.

From sklearn we used SVC(SupportVectorClassifier) and the hyperparameters we have used are given below:

```
{'C': 1.0,
 'break_ties': False,
 'cache_size': 200,
 'class_weight': None,
 'coef0': 0.0,
 'decision_function_shape': 'ovr',
 'degree': 3,
 'gamma': 'scale',
 'kernel': 'linear',
 'max_iter': -1,
 'probability': False,
 'random_state': 42,
 'shrinking': True,
 'tol': 0.001,
 'verbose': False}
```

- **Random Forest:**

A widely used and effective machine learning algorithm based on the idea of bagging is random forest. Bagging consists of creating many “copies” of the training data (each copy is slightly different from another) and then applying the weak learner (In our case it was Decision Trees) to each copy to obtain multiple weak models and then combine them.

From sklearn we used RandomForestClassifier and after hyperparameter tuning, with RandomizedSearchCV the values are:

```
{'bootstrap': False,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 90,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 2,
 'min_samples_split': 5,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 200,
```

```
 'n_jobs': -1,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

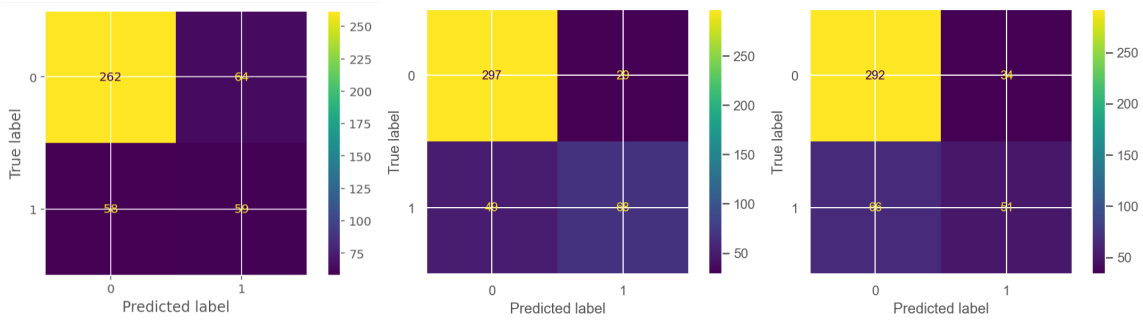
Results

- **Model Evaluation Scores:**

It is to be noted that, all scores have been calculated on the *TEST SET.*

Model	ROC-AUC	Precision	Recall	F1-Score
Random Forest	0.74	0.78	0.75	0.76
SVM	0.66	0.71	0.67	0.68
Decision Tree	0.65	0.65	0.65	0.65

- **Confusion Matrix:**



Confusion Matrix Plot for DT, RFC, SVC (from left to right)

References

1. The Hundred-Page Machine Learning Book, Book by Andriy Burkov
2. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking, Book by Tom Fawcett
3. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, Book by Wes McKinney
4. https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/
5. https://scikit-learn.org/stable/
6. https://pandas.pydata.org/