Customer Purchase Behavior Classification

Louis Bailey

Introduction

This dataset contains information on customer purchase behavior across various attributes, aiming to help data scientists and analysts understand the factors influencing purchase decisions. The dataset includes demographic information, purchasing habits, and other relevant features. The goal is to build a classifier to predict PurchaseStatus - Likelihood of the customer making a purchase (0: No, 1: Yes).

In [21]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset
from torch.optim import RMSprop
import optuna
import torch.optim as optim

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import classification_report
```

In [2]:
```
df = pd.read_csv('customer_purchase_data.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	Age	Gender	AnnualIncome	NumberOfPurchases	ProductCategory	TimeSpentOnWebsite	LoyaltyProgram	DiscountsAvailed	PurchaseStat
0	40	1	66120.26794	8	0	30.568601	0	5	
1	20	1	23579.77358	4	2	38.240097	0	5	
2	27	1	127821.30640	11	2	31.633212	1	0	
3	24	1	137798.62310	19	3	46.167059	0	4	
4	31	1	99300.96422	19	1	19.823592	0	0	

clean

In [4]:
```
print(f'Size: {df.shape}')
print(f'Missing: {df.isna().sum().sum()}')
print(f'Duplicates: {df.duplicated().sum()}')
```

```
Size: (1500, 9)
Missing: 0
Duplicates: 112
```

In [5]:
```
df = df.drop_duplicates()
```

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
for col in df.columns:
    print(col,df[col].unique(), '\n')
```

```
Age [40 20 27 24 31 66 39 64 43 70 54 19 51 18 57 59 46 22 62 67 48 52 63 36
 37 25 53 30 68 45 50 61 47 35 32 42 33 23 55 65 26 58 29 44 34 41 38 21
 56 60 28 49 69]

Gender [1 0]

AnnualIncome [ 66120.26794  23579.77358 127821.3064  ...  57363.24754 134021.7755
  52625.66597]

NumberOfPurchases [ 8  4 11 19 14 16 13 20  9 17  7  2  0 12  3 18 15  1  5 10  6]

ProductCategory [0 2 3 1 4]

TimeSpentOnWebsite [30.56860116 38.24009661 31.6332115  ... 12.20603321 37.3116338
 25.34801665]

LoyaltyProgram [0 1]

DiscountsAvailed [5 0 4 2 3 1]

PurchaseStatus [1 0]
```

In [8]:
```
df.dtypes
```

Out[8]:
```
Age                   int64
Gender                int64
AnnualIncome        float64
NumberOfPurchases     int64
ProductCategory       int64
TimeSpentOnWebsite  float64
LoyaltyProgram        int64
DiscountsAvailed      int64
PurchaseStatus        int64
dtype: object
```

feature transformations

In [9]:
```
categorical_columns     = ['Gender', 'ProductCategory', 'LoyaltyProgram', 'DiscountsAvailed']
df[categorical_columns] = df[categorical_columns].astype('category')
```

In [10]:
```
df = pd.get_dummies(df, columns=categorical_columns, drop_first=True).apply(pd.to_numeric)
```

In [11]:
```
df = df.astype({col: 'int64' for col in df.columns if df[col].dtype == 'bool'})
```

feature selection

In [12]:
```
X = df.drop('PurchaseStatus', axis=1)
y = df.PurchaseStatus
```

In [13]:
```
model = RandomForestClassifier()
model.fit(X, y)

feature_importance = model.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print(importance_df)
```

```
               Feature  Importance
3   TimeSpentOnWebsite    0.201087
0                  Age    0.172467
1         AnnualIncome    0.162716
2    NumberOfPurchases    0.142568
9     LoyaltyProgram_1    0.096307
13  DiscountsAvailed_4    0.036782
14  DiscountsAvailed_5    0.032032
12  DiscountsAvailed_3    0.030106
11  DiscountsAvailed_2    0.027728
10  DiscountsAvailed_1    0.027583
4             Gender_1    0.018885
7    ProductCategory_3    0.013791
5    ProductCategory_1    0.013002
6    ProductCategory_2    0.012769
8    ProductCategory_4    0.012176
```

> All the features will be kept.

---

data prep

In [89]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

In [90]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [91]:
```
X_train = np.array(X_train)
X_test  = np.array(X_test)
y_train = np.array(y_train)
y_test  = np.array(y_test)
```

In [92]:
```
#data to pytorch tensors
x_train = torch.from_numpy(X_train).float()
y_train = torch.from_numpy(y_train).float().view(-1,1)
x_test  = torch.from_numpy(X_test).float()
y_test  = torch.from_numpy(y_test).float().view(-1,1)
```

model

In [93]:
```
#random seed
torch.manual_seed(0)
np.random.seed(0)
```

In [95]:
```
#define neural net
class SimpleNeuralNet(nn.Module):
    def __init__(self):
        super(SimpleNeuralNet, self).__init__()
        self.sequential = torch.nn.Sequential(
            torch.nn.Linear(15,64),
            torch.nn.ReLU(),
            torch.nn.Linear(64,64),
            torch.nn.ReLU(),
            torch.nn.Linear(64,1),
            torch.nn.Sigmoid()
        )

    def forward(self, x):
        x = self.sequential(x)
        return x

#initialize neural net
```

```
network = SimpleNeuralNet()

#loss function
criterion = nn.BCELoss()
optimizer = torch.optim.RMSprop(network.parameters(), lr=0.01)

#data loader
train_data  = TensorDataset(x_train, y_train)
train_loader = DataLoader(train_data, batch_size=200, shuffle=True)

#train neural net
epochs = 12
for epoch in range(epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = network(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
    print('Epoch', epoch+1, '\tLoss', loss.item())

#evaluate neural net
with torch.no_grad():
    output = network(x_test)
    test_loss = criterion(output, y_test)
    test_accuracy = (output.round() == y_test).float().mean()
    print('Test Loss:', test_loss.item(), '\tTest Accuracy:', test_accuracy.item())
```

```
Epoch 1         Loss 0.435377299785614
Epoch 2         Loss 0.30158618092536926
Epoch 3         Loss 0.3116520345211029
Epoch 4         Loss 0.3089436888694763
Epoch 5         Loss 0.3502837121486664
Epoch 6         Loss 0.3185897469520569
Epoch 7         Loss 0.33283913135528564
Epoch 8         Loss 0.3078848123550415
Epoch 9         Loss 0.3366880714893341
Epoch 10        Loss 0.2039964348077774
Epoch 11        Loss 0.27225521206855774
Epoch 12        Loss 0.24404853582382202
Test Loss: 0.38377484679222107  Test Accuracy: 0.8741007447242737
```

classification report

In [96]:
```
with torch.no_grad():
    output = network(x_test)
    predicted = output.round()

y_pred = predicted.numpy()
y_true = y_test.numpy()

report = classification_report(y_true, y_pred, target_names=['Class 0', 'Class 1'])
print(report)
```

```
              precision    recall  f1-score   support

     Class 0       0.87      0.90      0.89       152
     Class 1       0.88      0.84      0.86       126

    accuracy                           0.87       278
   macro avg       0.87      0.87      0.87       278
weighted avg       0.87      0.87      0.87       278
```

conclusion

A neural net with a sigmoid activation function was built to predict whether a customer made a purchase. The performace metrics were good pretty good. The accuracy and f1 macro scores were both 87%. This is still a work in progress.