Customer_Lifetime_Value

April 2, 2023

[1]:
```
from __future__ import division
import pandas as pd
from datetime import datetime, timedelta, date
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
import xgboost as xgb
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import KFold, cross_val_score, train_test_split

import chart_studio.plotly as py
import plotly.offline as pyoff
import plotly.graph_objs as go

#initiate plotly
pyoff.init_notebook_mode()
```

[2]:
```
df = pd.read_excel("data/Online_Retail.xlsx", sheet_name='Online Retail')
df.head(10)
```

[2]:
```
  InvoiceNo StockCode                          Description  Quantity  \
0    536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                  WHITE METAL LANTERN         6
2    536365    84406B       CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E       RED WOOLLY HOTTIE WHITE HEART.         6
5    536365     22752         SET 7 BABUSHKA NESTING BOXES         2
6    536365     21730    GLASS STAR FROSTED T-LIGHT HOLDER         6
7    536366     22633               HAND WARMER UNION JACK         6
8    536366     22632            HAND WARMER RED POLKA DOT         6
9    536367     84879        ASSORTED COLOUR BIRD ORNAMENT        32

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
```

```
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
5 2010-12-01 08:26:00       7.65     17850.0  United Kingdom
6 2010-12-01 08:26:00       4.25     17850.0  United Kingdom
7 2010-12-01 08:28:00       1.85     17850.0  United Kingdom
8 2010-12-01 08:28:00       1.85     17850.0  United Kingdom
9 2010-12-01 08:34:00       1.69     13047.0  United Kingdom
```

[3]:
```
#convert string date into datetime
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate'])

#using only data from the UK
df_uk = df.query("Country == 'United Kingdom' ").reset_index(drop=True)
df_uk.head(10)
```

[3]:
```
  InvoiceNo StockCode                         Description  Quantity  \
0    536365    85123A  WHITE HANGING HEART T-LIGHT HOLDER         6
1    536365     71053                 WHITE METAL LANTERN         6
2    536365    84406B      CREAM CUPID HEARTS COAT HANGER         8
3    536365    84029G KNITTED UNION FLAG HOT WATER BOTTLE         6
4    536365    84029E      RED WOOLLY HOTTIE WHITE HEART.         6
5    536365     22752        SET 7 BABUSHKA NESTING BOXES         2
6    536365     21730   GLASS STAR FROSTED T-LIGHT HOLDER         6
7    536366     22633              HAND WARMER UNION JACK         6
8    536366     22632           HAND WARMER RED POLKA DOT         6
9    536367     84879       ASSORTED COLOUR BIRD ORNAMENT        32

          InvoiceDate  UnitPrice  CustomerID         Country
0 2010-12-01 08:26:00       2.55     17850.0  United Kingdom
1 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
2 2010-12-01 08:26:00       2.75     17850.0  United Kingdom
3 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
4 2010-12-01 08:26:00       3.39     17850.0  United Kingdom
5 2010-12-01 08:26:00       7.65     17850.0  United Kingdom
6 2010-12-01 08:26:00       4.25     17850.0  United Kingdom
7 2010-12-01 08:28:00       1.85     17850.0  United Kingdom
8 2010-12-01 08:28:00       1.85     17850.0  United Kingdom
9 2010-12-01 08:34:00       1.69     13047.0  United Kingdom
```

[4]:
```
# Divide the dataset in 2
# 3 months to calculate RFM (Recency, Frequency, and Monetary)
# and 6 month for predicting

#df_3m = df_uk[(df_uk.InvoiceDate < date(2011,6,1)) & (df_uk.InvoiceDate >=
↪date (2011,3,1))].reset_index(drop=True)
```

```
#df_6m = df_uk[(df_uk.InvoiceDate >= date(2011,6,1)) & (df_uk.InvoiceDate < 
↪date (2011,12,1))].reset_index(drop=True)

df_3m = df_uk[(df_uk.InvoiceDate < pd.to_datetime('2011-06-01')) & (df_uk.
↪InvoiceDate >= pd.to_datetime('2011-03-01'))].reset_index(drop=True)
df_6m = df_uk[(df_uk.InvoiceDate >= pd.to_datetime('2011-06-01')) & (df_uk.
↪InvoiceDate < pd.to_datetime('2011-12-01'))].reset_index(drop=True)



#create df_user for assigning clustering
#Grouping together similar data points into clusters or groups. The goal is to
↪identify patterns & r/ships within a dataset.

df_user = pd.DataFrame(df_3m['CustomerID'].unique())
df_user.columns = ['CustomerID']
```

[5]:
```
#order cluster
def order_cluster(cluster_field_name, target_field_name, df, ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    df_new = df.groupby(cluster_field_name)[target_field_name].mean().
↪reset_index()
    df_new = df_new.sort_values(by=target_field_name, ascending=ascending).
↪reset_index(drop=True)
    df_new['index'] = df_new.index
    df_final = pd.merge(df, df_new[[cluster_field_name, 'index']],
↪on=cluster_field_name)
    df_final = df_final.drop([cluster_field_name], axis = 1)
    df_final = df_final.rename(columns={"index": cluster_field_name})
    return  df_final
```

[6]:
```
# Calculate the recency score -- ref: 3 months to calculate RFM (Recency,
↪Frequency, and Monetary)

df_max_purchase = df_3m.groupby('CustomerID').InvoiceDate.max().reset_index()
df_max_purchase.columns = ['CustomerID', 'MaxPurchaseDate']
df_max_purchase['Recency'] = (df_max_purchase['MaxPurchaseDate'].max() -
↪df_max_purchase['MaxPurchaseDate']).dt.days
df_user = pd.merge(df_user, df_max_purchase[['CustomerID', 'Recency']], on=
↪'CustomerID')
df_user.head()
```

[6]:

	CustomerID	Recency
0	14620.0	12
1	14740.0	4
2	13880.0	25

```
3     16462.0       91
4     17068.0       11
```

[7]:
```python
kmeans = KMeans(n_clusters=4)
kmeans.fit(df_user[['Recency']])
df_user['RecencyCluster'] = kmeans.predict(df_user[['Recency']])

df_user = order_cluster('RecencyCluster', 'Recency', df_user, False)
```

[8]:
```python
# Calculate the Frequency score -- ref: 3 months to calculate RFM (Recency,␣
↪Frequency, and Monetary)

df_frequency = df_3m.groupby('CustomerID').InvoiceDate.count().reset_index()
df_frequency.columns = ['CustomerID', 'Frequency']
df_user = pd.merge(df_user, df_frequency, on='CustomerID')

kmeans = KMeans(n_clusters=4)
kmeans.fit(df_user[['Frequency']])
df_user['FrequencyCluster'] = kmeans.predict(df_user[['Frequency']])

df_user = order_cluster('FrequencyCluster', 'Frequency', df_user, True)
```

[9]:
```python
# Calculate the Revenue Score

df_3m['Revenue'] = df_3m['UnitPrice'] * df_3m['Quantity']
df_revenue = df_3m.groupby('CustomerID').Revenue.sum().reset_index()
df_user = pd.merge(df_user, df_revenue, on='CustomerID')

kmeans = KMeans(n_clusters=4)
kmeans.fit(df_user[['Revenue']])
df_user['RevenueCluster'] = kmeans.predict(df_user[['Revenue']])
df_user = order_cluster('RevenueCluster', 'Revenue',df_user,True)
```

[10]:
```python
#overall scoring
df_user['OverallScore'] = df_user['RecencyCluster'] +␣
↪df_user['FrequencyCluster'] + df_user['RevenueCluster']
df_user['Segment'] = 'Low-Value'
df_user.loc[df_user['OverallScore']>2,'Segment'] = 'Mid-Value'
df_user.loc[df_user['OverallScore']>4,'Segment'] = 'High-Value'
```

[11]:
```python
df_user.head()
```

[11]:
```
   CustomerID  Recency  RecencyCluster  Frequency  FrequencyCluster  Revenue  \
0     14620.0       12               3         30                 0   393.28
1     15194.0        6               3         64                 0  1439.02
2     18044.0        5               3         57                 0   808.96
3     18075.0       12               3         35                 0   638.12
```

```
4     15241.0        0                3         64                  0   947.55

   RevenueCluster  OverallScore    Segment
0               0             3  Mid-Value
1               0             3  Mid-Value
2               0             3  Mid-Value
3               0             3  Mid-Value
4               0             3  Mid-Value
```

[12]:
```
#calculate revenue and create a new dataframe for it
df_6m['Revenue'] = df_6m['UnitPrice'] * df_6m['Quantity']
df_user_6m = df_6m.groupby('CustomerID')['Revenue'].sum().reset_index()
df_user_6m.columns = ['CustomerID','m6_Revenue']


#plot CLV histogram
plot_data = [
    go.Histogram(
        x=df_user_6m.query('m6_Revenue < 10000')['m6_Revenue']
    )
]

plot_layout = go.Layout(
        title='6m Revenue'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

[15]:
```
#Merge our 3 months and 6 months dataframes to see correlations between LTV and␣
↪the feature set we have.

df_merge = pd.merge(df_user, df_user_6m, on='CustomerID', how='left')
df_merge = df_merge.fillna(0)

df_graph = df_merge.query("m6_Revenue < 30000")

plot_data = [
    go.Scatter(
        x=df_graph.query("Segment == 'Low-Value'")['OverallScore'],
        y=df_graph.query("Segment == 'Low-Value'")['m6_Revenue'],
        mode='markers',
        name='Low',
        marker= dict(size= 7,
            line= dict(width=1),
            color= 'blue',
            opacity= 0.8
           )
```

```
    ),
        go.Scatter(
        x=df_graph.query("Segment == 'Mid-Value'")['OverallScore'],
        y=df_graph.query("Segment == 'Mid-Value'")['m6_Revenue'],
        mode='markers',
        name='Mid',
        marker= dict(size= 9,
            line= dict(width=1),
            color= 'green',
            opacity= 0.5
           )
    ),
        go.Scatter(
        x=df_graph.query("Segment == 'High-Value'")['OverallScore'],
        y=df_graph.query("Segment == 'High-Value'")['m6_Revenue'],
        mode='markers',
        name='High',
        marker= dict(size= 11,
            line= dict(width=1),
            color= 'red',
            opacity= 0.9
           )
    ),
]

plot_layout = go.Layout(
        yaxis= {'title': "6m CLV"},
        xaxis= {'title': "RFM Score"},
        title='CLV'
    )
fig = go.Figure(data=plot_data, layout=plot_layout)
pyoff.iplot(fig)
```

1 Notes

Positive correlation is quite visible here, High RFM score means high CLV. Before building the machine learning model, we need to identify what is the type of this machine learning problem. CLV itself is a regression problem. A machine learning model can predict the $ value of the CLV. But here, we want CLV segments. Because it makes it more actionable and easy to communicate with other people. By applying K-means clustering, we can identify our existing CLV groups and build segments on top of it.

Considering business part of this analysis, we need to treat customers differently based on their predicted CLV. For this example, we will apply clustering and have 3 segments (number of segments really depends on your business dynamics and goals):

- Low LTV

- Mid LTV

- High LTV

We are going to apply K-means clustering to decide segments and observe their characteristics:

[17]:
```
#remove outliers
df_merge = df_merge[df_merge['m6_Revenue'] < df_merge['m6_Revenue'].quantile(0.
↪99)]


#creating 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(df_merge[['m6_Revenue']])
df_merge['CLVCluster'] = kmeans.predict(df_merge[['m6_Revenue']])

#order cluster number based on clv
df_merge = order_cluster('CLVCluster', 'm6_Revenue',df_merge,True)

#creatinga new cluster dataframe
df_cluster = df_merge.copy()

#see details of the clusters
df_cluster.groupby('CLVCluster')['m6_Revenue'].describe()
```

```
C:\Users\nakhu\AppData\Local\Temp/ipykernel_8972/1839946941.py:8:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[17]:
```
             count         mean          std      min      25%      50%  \
CLVCluster
0           1261.0   306.174291   330.055061  -609.40     0.00   227.86
1            430.0  1837.498279   580.771221  1075.37  1352.04  1700.34
2            111.0  4841.077297  1250.578406  3355.21  3826.99  4361.35

                  75%      max
CLVCluster
0            550.7000  1072.00
```

```
1           2169.1825  3321.55
2           5801.4050  7945.35
```

2 is the best with average 4.8 k clv whereas 0 is the worst with 306.

Before training the machine learning model:

Need to do some feature engineering. We should convert categorical columns to numerical columns.

We will check the correlation of features against our label, clv clusters.

We will split our feature set and label (CLV) as X and y. We use X to predict y.

Will create Training and Test dataset. Training set will be used for building the machine learning model. We will apply our model to Test set to see its real performance.

[18]:
```
#convert categorical columns to numerical
df_class = pd.get_dummies(df_cluster) #  converts categorical columns to 0-1␣
↪notations.

#calculate and show correlations
corr_matrix = df_class.corr()
corr_matrix['CLVCluster'].sort_values(ascending=False)

#create X and y, X will be feature set and y is the label - LTV
X = df_class.drop(['CLVCluster','m6_Revenue'],axis=1)
y = df_class['CLVCluster']

#split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05,␣
↪random_state=56)
```

[19]:
```
df_class.head()
```

[19]:
```
   CustomerID  Recency  RecencyCluster  Frequency  FrequencyCluster  Revenue  \
0     14620.0       12               3         30                 0   393.28
1     18044.0        5               3         57                 0   808.96
2     15241.0        0               3         64                 0   947.55
3     15660.0        4               3         34                 0   484.62
4     14560.0        3               3         12                 0   562.28

   RevenueCluster  OverallScore  m6_Revenue  CLVCluster  Segment_High-Value  \
0               0             3        0.00           0                   0
1               0             3      991.54           0                   0
2               0             3      791.04           0                   0
3               0             3      858.09           0                   0
```

```
4             0           3     911.33          0                   0

   Segment_Low-Value  Segment_Mid-Value
0                  0                  1
1                  0                  1
2                  0                  1
3                  0                  1
4                  0                  1
```

[20]:
```
corr_matrix
```

[20]:
```
                    CustomerID   Recency  RecencyCluster  Frequency  \
CustomerID            1.000000 -0.005530        0.009652  -0.036052
Recency              -0.005530  1.000000       -0.965079  -0.248706
RecencyCluster        0.009652 -0.965079        1.000000   0.242727
Frequency            -0.036052 -0.248706        0.242727   1.000000
FrequencyCluster      0.003827 -0.208537        0.200143   0.787864
Revenue              -0.061375 -0.331619        0.332736   0.564723
RevenueCluster       -0.048287 -0.148502        0.146150   0.342915
OverallScore          0.003114 -0.918922        0.946907   0.467373
m6_Revenue           -0.025928 -0.259100        0.255672   0.411463
CLVCluster           -0.026530 -0.245550        0.238553   0.401080
Segment_High-Value   -0.045500 -0.134313        0.139477   0.481066
Segment_Low-Value     0.001618  0.728333       -0.805695  -0.326680
Segment_Mid-Value     0.009640 -0.702177        0.779016   0.210677

                    FrequencyCluster   Revenue  RevenueCluster  OverallScore  \
CustomerID                  0.003827 -0.061375       -0.048287      0.003114
Recency                    -0.208537 -0.331619       -0.148502     -0.918922
RecencyCluster              0.200143  0.332736        0.146150      0.946907
Frequency                   0.787864  0.564723        0.342915      0.467373
FrequencyCluster            1.000000  0.510191        0.289106      0.479634
Revenue                     0.510191  1.000000        0.693835      0.518147
RevenueCluster              0.289106  0.693835        1.000000      0.336293
OverallScore                0.479634  0.518147        0.336293      1.000000
m6_Revenue                  0.394955  0.655272        0.450156      0.388016
CLVCluster                  0.387231  0.583894        0.357561      0.358820
Segment_High-Value          0.404155  0.506286        0.631405      0.312696
Segment_Low-Value          -0.342267 -0.382937       -0.207582     -0.822990
Segment_Mid-Value           0.245473  0.261235        0.053163      0.753560

                    m6_Revenue  CLVCluster  Segment_High-Value  \
CustomerID           -0.025928   -0.026530           -0.045500
Recency              -0.259100   -0.245550           -0.134313
RecencyCluster        0.255672    0.238553            0.139477
Frequency             0.411463    0.401080            0.481066
FrequencyCluster      0.394955    0.387231            0.404155
```

```
Revenue               0.655272   0.583894           0.506286
RevenueCluster        0.450156   0.357561           0.631405
OverallScore          0.388016   0.358820           0.312696
m6_Revenue            1.000000   0.900331           0.351043
CLVCluster            0.900331   1.000000           0.294638
Segment_High-Value    0.351043   0.294638           1.000000
Segment_Low-Value    -0.271140  -0.251233          -0.161976
Segment_Mid-Value     0.186811   0.180685          -0.084220

                    Segment_Low-Value  Segment_Mid-Value
CustomerID                   0.001618           0.009640
Recency                      0.728333          -0.702177
RecencyCluster              -0.805695           0.779016
Frequency                   -0.326680           0.210677
FrequencyCluster            -0.342267           0.245473
Revenue                     -0.382937           0.261235
RevenueCluster              -0.207582           0.053163
OverallScore                -0.822990           0.753560
m6_Revenue                  -0.271140           0.186811
CLVCluster                  -0.251233           0.180685
Segment_High-Value          -0.161976          -0.084220
Segment_Low-Value            1.000000          -0.969647
Segment_Mid-Value           -0.969647           1.000000
```

[21]:

```
# Lines related to correlation make us have the data below
corr_matrix['CLVCluster'].sort_values(ascending=False)
```

[21]:

```
CLVCluster            1.000000
m6_Revenue            0.900331
Revenue               0.583894
Frequency             0.401080
FrequencyCluster      0.387231
OverallScore          0.358820
RevenueCluster        0.357561
Segment_High-Value    0.294638
RecencyCluster        0.238553
Segment_Mid-Value     0.180685
CustomerID           -0.026530
Recency              -0.245550
Segment_Low-Value    -0.251233
Name: CLVCluster, dtype: float64
```

[22]:

```
# We have the training and test sets we can build our model.
#XGBoost Multiclassification Model
clv_xgb_model = xgb.XGBClassifier(max_depth=5, learning_rate=0.1,objective=␣
↪'multi:softprob',n_jobs=-1).fit(X_train, y_train)
```

```python
print('Accuracy of XGB classifier on training set: {:.2f}'
      .format(clv_xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.2f}'
      .format(clv_xgb_model.score(X_test[X_train.columns], y_test)))

y_pred = clv_xgb_model.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
Accuracy of XGB classifier on training set: 0.89
Accuracy of XGB classifier on test set: 0.86
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        70
           1       0.70      0.47      0.56        15
           2       0.67      0.67      0.67         6

    accuracy                           0.86        91
   macro avg       0.75      0.70      0.72        91
weighted avg       0.85      0.86      0.85        91
```

Accuracy shows 86% on the test set ! THIS IS REALLY GOOD

2 Accuracy Report

This report describes the performance of an XGB (eXtreme Gradient Boosting) classifier on a classification task. The report shows the following information:

The classifier has an accuracy of 0.89 on the training set, meaning it correctly classified 89% of the training data.

The classifier has an accuracy of 0.86 on the test set, meaning it correctly classified 86% of the test data. This is a good indication that the model generalizes well to unseen data.

The report also provides a detailed breakdown of the model's performance for each class (0, 1, and 2) in the form of precision, recall, and F1-score.

2.0.1 Class 0:

- Precision: 0.89 - Out of all the samples predicted as class 0, 89% were actually class 0.

- Recall: 0.96 - Out of all the actual class 0 samples, 96% were predicted correctly as class 0.

- F1-score: 0.92 - A harmonic mean of precision and recall, providing a single score that balances both. In this case, it is quite high, suggesting a good model performance for class 0.

2.0.2 Class 1:

- Precision: 0.70 - Out of all the samples predicted as class 1, 70% were actually class 1.

- Recall: 0.47 - Out of all the actual class 1 samples, 47% were predicted correctly as class 1.

- F1-score: 0.56 - A lower F1-score compared to class 0, indicating a weaker model performance for class 1.

2.0.3 Class 2:

- Precision: 0.67 - Out of all the samples predicted as class 2, 67% were actually class 2.

- Recall: 0.67 - Out of all the actual class 2 samples, 67% were predicted correctly as class 2.

- F1-score: 0.67 - A moderate F1-score, suggesting that the model performance for class 2 is average.

2.0.4 Additionally, the report provides macro and weighted averages:

Macro average:

A simple average of the individual metric scores across classes, without considering the number of samples in each class. In this case, the macro average for precision, recall, and F1-score are 0.75, 0.70, and 0.72, respectively.

Weighted average:

An average of the individual metric scores across classes, weighted by the number of samples in each class. In this case, the weighted average for precision, recall, and F1-score are 0.85, 0.86, and 0.85, respectively.

The XGB classifier performs well for class 0, but its performance could be improved for classes 1 and 2.

[ ]:
```

```