Integrated Approach of RFM, Clustering, CLTV & Machine Learning Algorithms for Forecasting: an example with Python coding

Sarit Maitra Follow
Oct 3, 2019 · 9 min read ★

C**LTV** is a customer relationship management (CRM) issue with an enterprise approach to understanding and influencing customer behavior through meaningful communication to improve customer acquisition, customer retention, customer loyalty, and customer profitability. The whole idea is that, business wants to predict the average amount of $$ customers will spend on the business over the entire life of relationship.

Although statistical methods can be very powerful, but these methods make several stringent assumptions on the types of data and their distribution, and typically can only handle a limited number of variables. Regressionbased methods are usually based on a fixed-form equation, and assume a single best solution, which means that we can compare only a few alternative solutions manually. Further, when the models are applied to real data, the key assumptions of the methods are often violated. Here, I will show ***Machine Learning*** (ML) methods by integrating the ***CLTV*** and customer transaction variables with the ***RFM*** variables to forecast consumer purchases.

I will use two approaches here —

> *1st approach-* ***RFM (Recency, Frequency, and Monetary)*** *marketing analysis method is used in order to segmentation of customers and*
>
> *2nd approach using* ***Customer Lifetime Value (CLTV)*** *will train a* ***ML*** *algorithm for* ***prediction****. I will use 3 months of data to calculate* ***RFM*** *and use it for predicting next 6 months.*

RFM is a scoring model attempt to predict customers' behavior in the future and implicitly linked to ***CLTV***. One key limitation of ***RFM*** models is that they are scoring models and do not explicitly provide a $ number for customer value. A simple equation to derive ***CLTV*** for a customer

- pt= price paid by a consumer at time t,
- ct = direct cost of servicing the customer at time t,
- i = discount rate or cost of capital for the firm,
- rt = probability of customer repeat buying or being "alive" at time t,
- AC = acquisition cost, and
- T = time horizon for estimating ***CLTV***.

Data Mining

Let's load and see the data.

```
df = pd.read_excel("Online Retail.xlsx")
```

```
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

We have all the necessary information that we need:

- Customer ID
- Unit Price
- Quantity
- Invoice Date

With all these features, we can build the equation for Monetary value= *Active Customer Count * Order Count * Average Revenue per Order*

```
df['InvoiceDate'] = pd.to_datetime(df['InvoiceDate']) #convert the type of Invoice Date Field from string to datetime.

df['InvoiceYearMonth'] = df['InvoiceDate'].map(lambda date: 100*date.year + date.month) #create YearMonth field

df['Monetary'] = df['UnitPrice'] * df['Quantity'] #calculate Monetary for each row and create a new data frame with YearMonth — Monetary columns

monetary = df.groupby(['InvoiceYearMonth'])['Monetary'].sum().reset_index()
```

Before we dive into ***RFM*** score, we can do some analysis to know more about customer behavior such as Monthly Active Customers/ Monthly Order Count/Average Revenue per Order /New Customer Ratio/ Monthly Customer Retention Rate etc. Interested may visit ***here*** to know about the such analysis. So, I will start with segmentation.

Customer Segmentation

Let’s assume some common segments-

- Low Value- Customers who are less active than others, not very frequent buyer/visitor and generates very low — zero — maybe negative revenue.

- Mid Value- Customers who are fairly frequent and generates moderate revenue.
- High Value- Customers with High Revenue, Frequency and low Inactivity; business always want to retain these customers.

We shall calculate ***RFM*** Value and apply unsupervised ML to identify different clusters for each by applyting ***K-means*** clustering to assign a ***recency*** score. Number of clusters generally defined by business, we need to ***K-means*** algorithm. However, ***Elbow Method*** of ***K-means*** helps us to know the optimal cluster number.

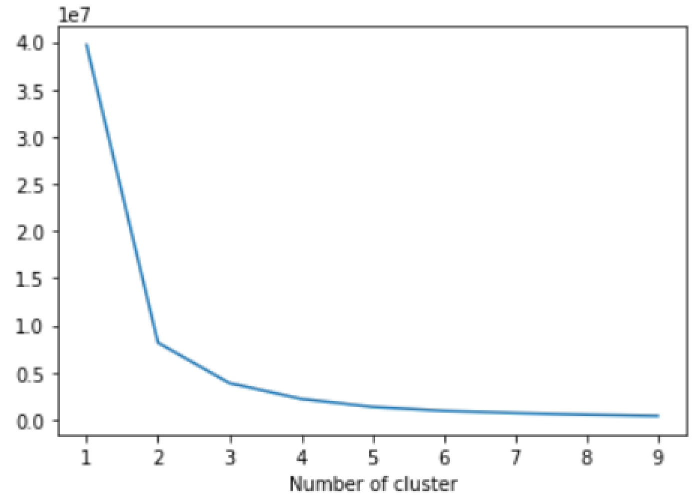
Recency

To calculate ***recency***, we need to find out most recent purchase date of each customer and see how many days they are inactive for. After having no. of inactive days for each customer, we will apply ***K-means*** clustering to assign customers a ***recency*** score.

```
#create a user dataframe to hold CustomerID and new segmentation scores
user = pd.DataFrame(df['CustomerID'].unique())
user.columns = ['CustomerID']

#get the max purchase date for each customer and create a dataframe with it
max_purchase = uk.groupby('CustomerID').InvoiceDate.max().reset_index()
max_purchase.columns = ['CustomerID','MaxPurchaseDate']

#we take the observation point as the max invoice date in the dataset
max_purchase['Recency'] = (max_purchase['MaxPurchaseDate'].max() - max_purchase['MaxPurchaseDate']).dt.days

#merge this dataframe to the new user dataframe
user = pd.merge(user, max_purchase[['CustomerID','Recency']], on='CustomerID')
user.head()
```

	CustomerID	Recency
0	17850.0	301
1	13047.0	31
2	13748.0	95
3	15100.0	329
4	15291.0	25

```
from sklearn.cluster import KMeans
sse={}
recency = user[['Recency']]

for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(recency)
    recency["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_

plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.show()
```



Here it looks we have 3 clusters. Based on business requirements, we can go with less or more clusters. Let us select 4 for this example:

```
#build 4 clusters for recency and add it to dataframe

kmeans = KMeans(n_clusters=4)
kmeans.fit(user[['Recency']])
user['RecencyCluster'] = kmeans.predict(user[['Recency']])

#function for ordering cluster numbers

def order_cluster(cluster_field_name, target_field_name,data,ascending):
    new_cluster_field_name = 'new_' + cluster_field_name
    data_new = data.groupby(cluster_field_name)[target_field_name].mean().reset_index()
    data_new = data_new.sort_values(by=target_field_name,ascending=ascending).reset_index(drop=True)
    data_new['index'] = data_new.index
    data_final = pd.merge(data,data_new[[cluster_field_name,'index']], on=cluster_field_name)
    data_final = data_final.drop([cluster_field_name],axis=1)
    data_final = data_final.rename(columns={"index":cluster_field_name})

    return data_final

user = order_cluster('RecencyCluster', 'Recency',user,False)
user.head()
```

	CustomerID	Recency	RecencyCluster
0	17850.0	301	0
1	15100.0	329	0
2	18074.0	373	0
3	16250.0	260	0
4	13747.0	373	0

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(user[['Recency']])
user['RecencyCluster'] = kmeans.predict(user[['Recency']])

#order the recency cluster
user = order_cluster('RecencyCluster', 'Recency',user,True)

#see details of each cluster
user.groupby('RecencyCluster')['Recency'].describe()
```

RecencyCluster	count	mean	std	min	25%	50%	75%	max
0	1950.0	17.488205	13.237058	0.0	6.00	16.0	28.0	47.0
1	952.0	77.567227	22.743569	48.0	59.00	72.0	93.0	130.0

2	570.0	184.436842	31.856230	131.0	156.00	184.0	211.0	244.0
3	478.0	304.393305	41.183489	245.0	266.25	300.0	336.0	373.0

Likewise, we can do ***Frequency*** and ***Monetary*** and finally the Overall Score.

```
#calculate overall score and use mean() to see details
user['OverallScore'] = user['RecencyCluster'] + user['FrequencyCluster'] + user['MonetaryCluster']
user.groupby('OverallScore')['Recency','Frequency','Monetary'].mean()
```

OverallScore	Recency	Frequency	Monetary
0	19.223657	64.579878	1031.923869
1	62.194676	100.247920	1495.978587
2	144.224299	108.899866	2335.065048
3	284.768482	72.778210	1609.635136
4	83.900000	934.400000	74360.852000
5	76.250000	4516.250000	32955.605000

We divide these cluster in High/Mid/Low — 0 to 2- Low / 3 to 4- Value / 5+- High Value customers

```
user['Segment'] = 'Low-Value'
user.loc[user['OverallScore'] > 2,'Segment'] = 'Mid-Value'
user.loc[user['OverallScore'] > 4,'Segment'] = 'High-Value'
user.head()
```

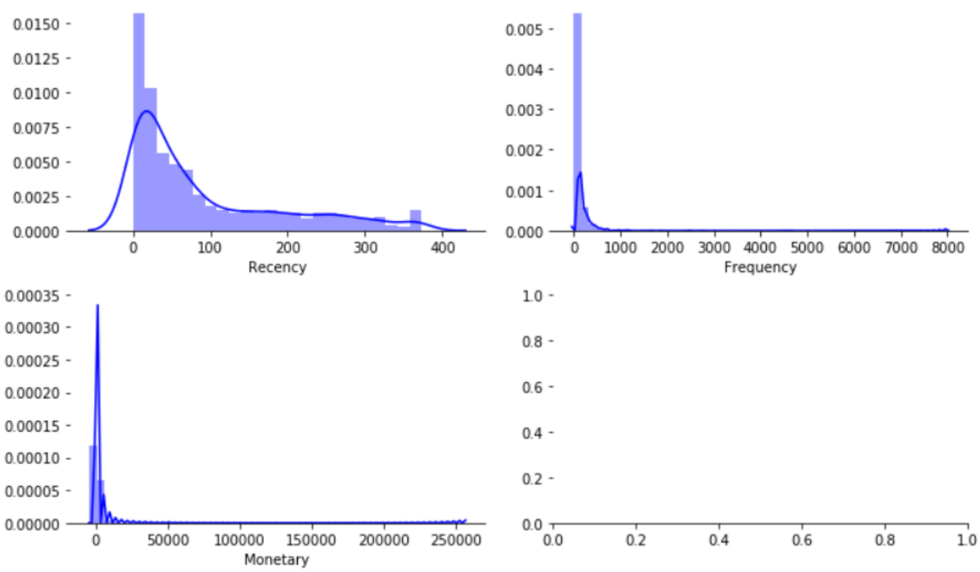
	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	RevenueCluster	OverallScore	Segment
0	17850.0	301	0	312	1	5288.63	1	2	Low-Value
1	14688.0	7	3	359	1	5107.38	1	5	High-Value
2	13767.0	1	3	399	1	16945.71	1	5	High-Value
3	15513.0	30	3	314	1	14520.08	1	5	High-Value
4	14849.0	21	3	392	1	7904.28	1	5	High-Value

The descriptive statistics of the respective ***RFM*** is show below—

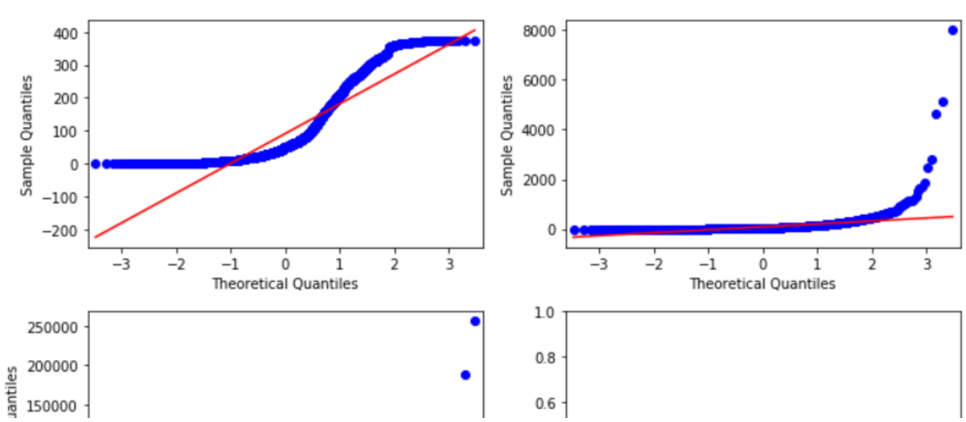
	Recency	Frequency	Monetary
Count	3950.00	3950.00	3950.00
Mean	90.77	91.61	1713.38
Std	100.23	220.55	6548.60
min	0.00	1.00	-4287.63
25%	16.00	17.00	282.25
50%	49.00	41.00	627.01
75%	142.00	101.00	1521.78
max	373.00	7983.00	256438.49
skewness	1.25	18.65	23.35
kurtosis	0.44	541.73	765.03
Jarque_bera	1059.30	48406538.83	96442753.15

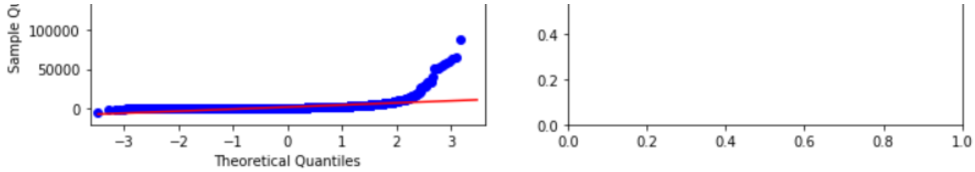
We see that even though the average is 90 day recency, median is 49. Negative Monetary value at min indicating return of items. The test statistic values and below distribution & QQ plots confirm that data set do not follow a normal distribution. Therefore, the use of nonparametric framework for making predictions is justified.

```python
# Set up the matplotlib figure
f, axes = plt.subplots(2, 2, figsize=(10, 6))
sns.despine(left=True)

# Plot a simple histogram with binsize determined automatically
sns.distplot(user.Recency, color="b", ax=axes[0, 0])
sns.distplot(user.Frequency, color="b", ax=axes[0, 1])
sns.distplot(user.Monetary, color="b",ax=axes[1, 0])

plt.tight_layout()
```




```python
from statsmodels.graphics.gofplots import qqplot

# Set up the matplotlib figure
f, axes = plt.subplots(2, 2, figsize=(10, 6))

# Plot a simple histogram with binsize determined automatically
qqplot(user.Recency, line= 'r', ax=axes[0, 0], label='Recency')
qqplot(user.Frequency, line= 'r',label='Frequency', ax=axes[0, 1])
qqplot(user.Monetary, line= 'r',label='Monetary',ax=axes[1, 0])

#plt.setp(axes, yticks=[])
plt.tight_layout()
```





Evidences from the statistical tests imply that data characterized by their nonparametric nature behavior. This justifies the deployment of advanced ML
and deep learning algorithms for predictive modeling exercise. However, I have not exercised deep learning algorithm here.

We can start taking actions with this segmentation. The strategies are simple for all three classes:

- Improve retention of High Value customer
- Improve retention and increase frequency of Mid Value customer
- Increase Frequency of Low Value customer

Customer Lifetime Value (CLTV)

CLTV is quite simple here. First we will select a time window anything from 3, 6, 12, or 24 months. We can have compute the ***CLTV*** for each customer in that specific time window with an equation: ***Total Gross Revenue -Total Cost***. This equation based on historical data and gives us the historical value. If we see some customers having very high negative lifetime value historically then probably we are too late to take an action. Let's use ML algorithm to predict.

CLTV Prediction

So, let's follow the steps-

- Define an appropriate time frame for ***CLTV*** calculation
- Identify the features we are going to use to predict future and create them
- Calculate ***CLTV*** for training the ML model
- Build and run the ML model

- Check if the model is useful

We already have obtained the ***RFM*** scores for each customer ID. To implement it correctly, let's split our dataset. I will take 3 months of data, calculate ***RFM*** and use it for predicting next 6 months.

> *#create 3m and 6m dataframes*
> *m3 = DF_uk[(DF_uk.InvoiceDate < date(2011,6,1)) & (DF_uk.InvoiceDate >= date(2011,3,1))].reset_index(drop=True)*
> *m6 = DF_uk[(DF_uk.InvoiceDate >= date(2011,6,1)) & (DF_uk.InvoiceDate < date(2011,12,1))].reset_index(drop=True)*

Now, the similar process of clustering, computing ***RFM*** and overall scoring of each data frame and finally merging the 3 months and 6 months data frames to see correlations between ***CLTV*** 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)
```

```
import seaborn as sns
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

corr= DF_merge.corr(method='pearson') # correlation (pearson)
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

	CustomerID	Recency	RecencyCluster	Frequency	FrequencyCluster	Revenue	RevenueCluster	OverallScore	Monetary	MonetaryCluster	m6_Monetary
CustomerID	1	-0.0032	0.0077	-0.033	-0.0095	-0.007	-0.02	2.4e-06	-0.007	-0.02	0.031
Recency	-0.0032	1	-0.97	-0.24	-0.23	-0.23	-0.19	-0.9	-0.23	-0.19	-0.14
RecencyCluster	0.0077	-0.97	1	0.23	0.22	0.23	0.19	0.93	0.23	0.19	0.14
Frequency	-0.033	-0.24	0.23	1	0.77	0.38	0.38	0.47	0.38	0.38	0.22
FrequencyCluster	-0.0095	-0.23	0.22	0.77	1	0.35	0.34	0.52	0.35	0.34	0.18
Revenue	-0.007	-0.23	0.23	0.38	0.35	1	0.84	0.45	1	0.84	0.82
RevenueCluster	-0.02	-0.19	0.19	0.38	0.34	0.84	1	0.45	0.84	1	0.65
OverallScore	2.4e-06	-0.9	0.93	0.47	0.52	0.45	0.45	1	0.45	0.45	0.29
Monetary	-0.007	-0.23	0.23	0.38	0.35	1	0.84	0.45	1	0.84	0.82
MonetaryCluster	-0.02	-0.19	0.19	0.38	0.34	0.84	1	0.45	0.84	1	0.65
m6_Monetary	0.031	-0.14	0.14	0.22	0.18	0.82	0.65	0.29	0.82	0.65	1

Here, by applying K-means clustering, we can identify the existing ***CLTV*** groups and build segments on top of it. Considering business part of this analysis, we need to treat customers differently based on their predicted ***CLTV***. For this example, we will apply clustering and have 3 segments (number of segments really depends on your business dynamics and goals):
- Low ***CLTV***
- Mid ***CLTV***
- High ***CLTV***

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

```
#remove outliers
DF_merge = DF_merge[DF_merge['m6_Monetary'] < DF_merge['m6_Monetary'].quantile(0.99)]

#creating 3 clusters
kmeans = KMeans(n_clusters=3)
```

```
kmeans.fit(DF_merge[['m6_Monetary']])
DF_merge['LTVCluster'] = kmeans.predict(DF_merge[['m6_Monetary']])

#order cluster number based on LTV
DF_merge = order_cluster('LTVCluster', 'm6_Monetary',DF_merge,True)

#creatinga new cluster dataframe
DF_cluster = DF_merge.copy()

#see details of the clusters
DF_cluster.groupby('LTVCluster')['m6_Monetary'].describe()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:6: 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: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

LTVCluster	count	mean	std	min	25%	50%	75%	max
0	1394.0	396.137189	419.891843	-609.40	0.000	294.220	682.4300	1429.87
1	371.0	2492.794933	937.341566	1445.31	1731.980	2162.930	3041.9550	5287.39
2	56.0	8222.565893	2983.572030	5396.44	6151.435	6986.545	9607.3225	16756.31

2 is the best with average 8.2k ***CLTV*** whereas 0 is the worst with 396. There are few more step before training the ML 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, ***CLTV*** clusters.
- We will split our feature set and label (***CLTV***) as X and y. We use X to predict y.
- Will create Training and Test dataset. Training set will be used for building the ML model.

We will apply our model to Test set to see its real performance.

```
from sklearn.model_selection import KFold, cross_val_score,
train_test_split
#convert categorical columns to numerical
DF_class = pd.get_dummies(DF_cluster)

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

#create X and y, X will be feature set and y is the label – LTV
X = DF_class.drop(['LTVCluster','m6_Monetary'],axis=1)
y = DF_class['LTVCluster']

#split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.05, random_state=42)
```

```
#corr_matrix = DF_class.corr()
corr_matrix['LTVCluster'].sort_values(ascending=False)
```

```
LTVCluster             1.000000
m6_Monetary            0.845933
Monetary               0.600491
Revenue                0.600491
MonetaryCluster        0.467191
RevenueCluster         0.467191
OverallScore           0.373114
FrequencyCluster       0.366366
Frequency              0.359601
Segment_High-Value     0.352387
RecencyCluster         0.236899
Segment_Mid-Value      0.168473
CustomerID            -0.028401
Recency               -0.237249
Segment_Low-Value     -0.266008
Name: LTVCluster, dtype: float64
```

We see that 3 months Revenue, Frequency and ***RFM*** scores will be helpful for our ML models. With the training and test sets we can build our model.

Machine Learning Algorithm comparison

```
xgb = xgb.XGBClassifier()
logreg2= LogisticRegressionCV()
knn = KNeighborsClassifier()
svcl = SVC()
adb = AdaBoostClassifier()
dtclf = DecisionTreeClassifier()
rfclf = RandomForestClassifier()

# prepare configuration for cross validation test harness
seed = 42
# prepare models
models = []
models.append(('LR', LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=5, multi_class='auto')))
models.append(('XGB', XGBClassifier()))
models.append(('KNN', KNeighborsClassifier(5)))
models.append(('DT', DecisionTreeClassifier(max_depth=5)))
models.append(('RF', RandomForestClassifier(n_estimators=100)))
models.append(('ADA', AdaBoostClassifier()))
models.append(('SVC', SVC(gamma='scale')))

# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=5, random_state=seed)
    cv_results = model_selection.cross_val_score(model,X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.799916 (0.031641)
XGB: 0.780834 (0.035113)
KNN: 0.790068 (0.015387)
DT: 0.779658 (0.015667)
RF: 0.774459 (0.024823)
ADA: 0.779097 (0.033471)
SVC: 0.781399 (0.020855)
```

Predictive model based on ML algorithms are kind of black box models which can be opened by using

Sensitivity & Specificity analysis.

> *FP = confusion_matrix.sum(axis=0) — np.diag(confusion_matrix)*
>
> *FN = confusion_matrix.sum(axis=1) — np.diag(confusion_matrix)*
>
> *TP = np.diag(confusion_matrix)*
>
> *TN = confusion_matrix.values.sum() — (FP + FN + TP)*
>
> *TPR = TP/(TP+FN) # Sensitivity, hit rate, recall, or true positive rate*
>
> *TNR = TN/(TN+FP) # Specificity or true negative rate*
>
> *PPV = TP/(TP+FP) # Precision or positive predictive value*
>
> *NPV = TN/(TN+FN) # Negative predictive value*
>
> *FPR = FP/(FP+TN) # Fall out or false positive rate*
>
> *FNR = FN/(TP+FN) # False negative rate*
>
> *FDR = FP/(TP+FP)# False discovery rate*
>
> *ACC = (TP+TN)/(TP+FP+FN+TN) # Overall accuracy*

XGB model

```
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
import xgboost as xgb
from datetime import datetime, timedelta,date

#XGBoost Multiclassification Model
xgb_model = xgb.XGBClassifier(max_depth=5, learning_rate=0.1,objective= 'multi:softprob',n_jobs=-1).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.2f}'
       .format(xgb_model.score(X_train, y_train)))
print('*'*60)

print('Accuracy of XGB classifier on test set: {:.2f}'
       .format(xgb_model.score(X_test[X_train.columns], y_test)))
print('*'*60)

y_pred = xgb_model.predict(X_test)

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

```
Accuracy of XGB classifier on training set: 0.91
************************************************************
Accuracy of XGB classifier on test set: 0.78
************************************************************
              precision    recall  f1-score   support

           0       0.85      0.90      0.87        69
           1       0.56      0.43      0.49        21
```

```
           2       0.33      0.50      0.40         2

    accuracy                           0.78        92
   macro avg       0.58      0.61      0.59        92
weighted avg       0.77      0.78      0.77        92
```

We have a multi classification model with 3 groups (clusters). Accuracy shows 78% on the test set. Our True positives are on the diagonal axis and are the largest numbers here. The False Negatives are the sum of the other values along the rows. The False Positives are the sum of the other values down the columns. Precision and recall are acceptable for 0. For cluster 0 which is Low ***CLTV***, if model identifies customer belongs to cluster 0, 85% chance that it will be correct(precision).The classifier successfully identifies 90% of actual cluster 0 customers (recall). We need to improve the model for other clusters. The classifier barely detect 43% of Mid ***CLTV*** customers.

Let's experiment changing the depth and OneVsRestClassifier —

```python
from sklearn.multiclass import OneVsRestClassifier
from xgboost import XGBClassifier
from sklearn.preprocessing import MultiLabelBinarizer

clf = OneVsRestClassifier(XGBClassifier(n_jobs=-1, max_depth=4, learning_rate=0.1))
clf.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                            colsample_bylevel=1,
                                            colsample_bynode=1,
                                            colsample_bytree=1, gamma=0,
                                            learning_rate=0.1, max_delta_step=0,
                                            max_depth=4, min_child_weight=1,
                                            missing=None, n_estimators=100,
                                            n_jobs=-1, nthread=None,
                                            objective='binary:logistic',
                                            random_state=0, reg_alpha=0,
                                            reg_lambda=1, scale_pos_weight=1,
                                            seed=None, silent=None, subsample=1,
                                            verbosity=1),
                    n_jobs=None)
```

```python
print('Accuracy of XGB classifier on training set: {:.2f}'
        .format(clf.score(X_train, y_train)))
print('*'*60)

print('Accuracy of XGB classifier on test set: {:.2f}'
        .format(clf.score(X_test[X_train.columns], y_test)))
print('*'*60)
pred = clf.predict(X_test)
print(classification_report(y_test, pred))
```

```
Accuracy of XGB classifier on training set: 0.87
************************************************************
Accuracy of XGB classifier on test set: 0.80
************************************************************
              precision    recall  f1-score   support

           0       0.86      0.91      0.89        69
           1       0.59      0.48      0.53        21
           2       0.50      0.50      0.50         2
```

accuracy			0.80	92
macro avg	0.65	0.63	0.64	92
weighted avg	0.79	0.80	0.80	92

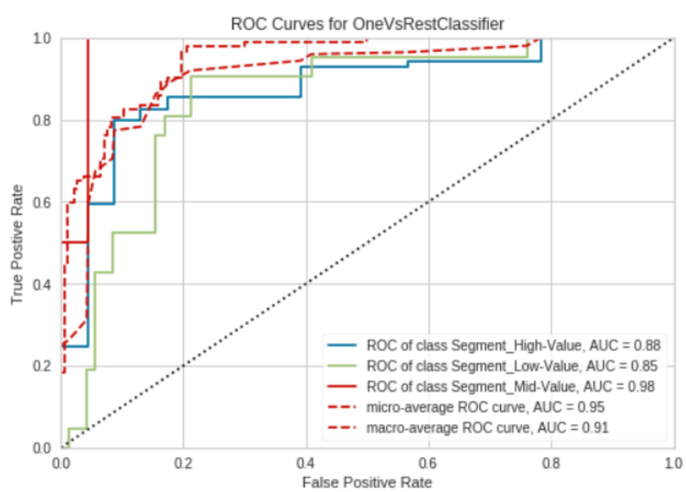
Some improvement can be seen here. However, there are still rooms for improvement e.g.

- Adding more features and improve feature engineering
- Try ANN /DNN

ROCAUC

By default with multi-class ROCAUC visualizations, a curve for each class is plotted, in addition to the micro- and macro-average curves for each class. This enables the user to inspect the tradeoff between sensitivity and specificity on a per-class basis.

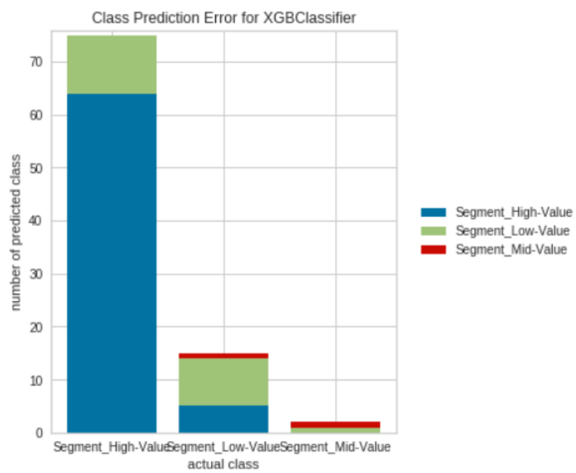
```
from yellowbrick.classifier import ROCAUC
visualizer = ROCAUC(clf, classes=["Segment_High-Value", "Segment_Low-Value","Segment_Mid-Value"])

visualizer.fit(X_train, y_train)        # Fit the training data to the visualizer
visualizer.score(X_test, y_test)        # Evaluate the model on the test data
visualizer.poof()
visualizer
```



Class Prediction Error

```
from yellowbrick.classifier import ClassPredictionError
# Instantiate the classification model and visualizer
visualizer = ClassPredictionError(
    XGBClassifier(random_state=42), classes=["Segment_High-Value",  "Segment_Low-Value","Segment_Mid-Value"]
)

# Fit the training data to the visualizer
visualizer.fit(X_train, y_train)
```

```
# Evaluate the model on the test data
visualizer.score(X_test, y_test)

# Draw visualization
visualizer.poof()
```



Understanding prediction errors and determining how to fix them is critical to building effective predictive systems. If you are interested, I will recommend to read this ***article*** to know more about prediction errors.

Summary

In ML models parameters are tuned/estimated based on the data and the parameters control how the algorithms learn from the data (without making any assumptions about the data, and downstream of the data generation). XGB is a tree based algorithm and hence can be considered nonparametric. The tree depth used here is a parameter of the algorithm, but it is not inherently derived from the data, but rather an input parameter.

I can be reached *here*.

Machine Learning Customer Service Data Science Analytics