home_credit_pycaret

October 25, 2024

[1]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
from pycaret.classification import *
from imblearn.under_sampling import RandomUnderSampler

# Display all columns and rows
pd.set_option('display.max_columns', None)
```

[2]:
```
# Load application data
train_data = pd.read_csv('/home/ignatiusvmk/Downloads/home-credit-default-risk/
↪application_train.csv')
test_data = pd.read_csv('/home/ignatiusvmk/Downloads/home-credit-default-risk/
↪application_test.csv')

# Checking the target variable distribution
train_data['TARGET'].value_counts(normalize=True)
```

[2]:
```
TARGET
0    0.919271
1    0.080729
Name: proportion, dtype: float64
```

[3]:
```
null_percentage =  train_data.isnull().mean()*100
# Display the DataFrame before cleaning
print(f"Train Data original shape: {train_data.shape}")
```

```
Train Data original shape: (307511, 122)
```

[4]:
```
null_percentage =  test_data.isnull().mean()*100

# Display the DataFrame before cleaning
print(f"Test Data original shape: {test_data.shape}")
```

```
Test Data original shape: (48744, 121)
```

[5]:
```
test_null = null_percentage >= 35
```

[6]:

```
#columns count with  >= 35% null values
print(f"Total columns with >= 35% null values: {test_null[test_null].count()}")
```

```
Total columns with >= 35% null values: 49
```

[7]:

```
# Load bureau data
bureau = pd.read_csv('/home/ignatiusvmk/Downloads/home-credit-default-risk/
 ↪bureau.csv')

# Load bureau balance data
bureau_balance = pd.read_csv('/home/ignatiusvmk/Downloads/
 ↪home-credit-default-risk/bureau_balance.csv')

# Load POS_CASH_balance data
pos_cash_balance = pd.read_csv('/home/ignatiusvmk/Downloads/
 ↪home-credit-default-risk/POS_CASH_balance.csv')

# Load credit card balance data
credit_card_balance = pd.read_csv('/home/ignatiusvmk/Downloads/
 ↪home-credit-default-risk/credit_card_balance.csv')

# Load installments payments data
installments_payments = pd.read_csv('/home/ignatiusvmk/Downloads/
 ↪home-credit-default-risk/installments_payments.csv')
```

[8]:

```
# Sum and mean of credit amount
bureau_agg = bureau.groupby('SK_ID_CURR').agg({
    'AMT_CREDIT_SUM': ['sum', 'median'],
    'CREDIT_DAY_OVERDUE': ['max'],
    'DAYS_CREDIT': ['mean'],
}).reset_index()

# Rename the columns
bureau_agg.columns = ['SK_ID_CURR', 'CREDIT_SUM_TOTAL', 'CREDIT_SUM_MEAN',␣
 ↪'CREDIT_OVERDUE_MAX', 'CREDIT_DURATION_MEAN']
bureau_agg.head()
```

[8]:

```
   SK_ID_CURR  CREDIT_SUM_TOTAL  CREDIT_SUM_MEAN  CREDIT_OVERDUE_MAX  \
0      100001       1453365.000        168345.00                   0
1      100002        865055.565         54130.50                   0
2      100003       1017400.500         92576.25                   0
3      100004        189037.800         94518.90                   0
4      100005        657126.000         58500.00                   0

   CREDIT_DURATION_MEAN
0           -735.000000
1           -874.000000
```

```
2          -1400.750000
3           -867.000000
4           -190.666667
```

[9]:
```
# Merge with train data
train_data = train_data.merge(bureau_agg, on='SK_ID_CURR', how='left')
test_data = test_data.merge(bureau_agg, on='SK_ID_CURR', how='left')
```

[10]:
```
# Replace categorical values with numeric ones
bureau_balance['STATUS'] = bureau_balance['STATUS'].replace(['C', '0'], 0).
 ↪replace(['1', '2', '3', '4', '5'], 1)

# Convert the STATUS column to numeric to handle any unexpected non-numeric␣
 ↪values
bureau_balance['STATUS'] = pd.to_numeric(bureau_balance['STATUS'],␣
 ↪errors='coerce')

# Group by SK_ID_BUREAU and aggregate status as sum (missed payments) and count␣
 ↪(total months)
bureau_balance_agg = bureau_balance.groupby('SK_ID_BUREAU').agg({
    'STATUS': ['sum', 'count']
}).reset_index()

# Rename the columns
bureau_balance_agg.columns = ['SK_ID_BUREAU', 'MISSED_PAYMENTS', 'TOTAL_MONTHS']

bureau_balance_agg.head()
```

[10]:

	SK_ID_BUREAU	MISSED_PAYMENTS	TOTAL_MONTHS
0	5001709	0.0	86
1	5001710	0.0	53
2	5001711	0.0	3
3	5001712	0.0	19
4	5001713	0.0	0

[11]:
```
# Count of active loans
pos_cash_agg = pos_cash_balance.groupby('SK_ID_CURR').agg({
    'MONTHS_BALANCE': 'count',   # Count of months with POS loans
    'SK_DPD': ['mean', 'sum'],   # Delay in payment (mean and total)
}).reset_index()

# Rename columns
pos_cash_agg.columns = ['SK_ID_CURR', 'POS_LOANS_COUNT', 'POS_DPD_MEAN',␣
 ↪'POS_DPD_TOTAL']

# Merge with train and test data
train_data = train_data.merge(pos_cash_agg, on='SK_ID_CURR', how='left')
```

```
test_data = test_data.merge(pos_cash_agg, on='SK_ID_CURR', how='left')
```

[12]:
```
# Average balance over time
credit_card_agg = credit_card_balance.groupby('SK_ID_CURR').agg({
    'AMT_BALANCE': ['mean', 'max'],
    'MONTHS_BALANCE': 'count'
}).reset_index()

# Rename columns
credit_card_agg.columns = ['SK_ID_CURR', 'CREDIT_BALANCE_MEAN',␣
 ↪'CREDIT_BALANCE_MAX', 'CREDIT_CARD_MONTHS']

# Merge with train and test data
train_data = train_data.merge(credit_card_agg, on='SK_ID_CURR', how='left')
test_data = test_data.merge(credit_card_agg, on='SK_ID_CURR', how='left')
```

[13]:
```
# Total and mean of payments
installments_agg = installments_payments.groupby('SK_ID_CURR').agg({
    'AMT_PAYMENT': ['sum', 'mean'],
    'DAYS_INSTALMENT': 'count'
}).reset_index()

# Rename columns
installments_agg.columns = ['SK_ID_CURR', 'PAYMENT_TOTAL', 'PAYMENT_MEAN',␣
 ↪'TOTAL_INSTALLMENTS']

# Merge with train and test data
train_data = train_data.merge(installments_agg, on='SK_ID_CURR', how='left')
test_data = test_data.merge(installments_agg, on='SK_ID_CURR', how='left')
```

[14]:
```
train_null_percentage =  train_data.isnull().mean()*100

train_null = train_null_percentage >= 35

#columns count with  >= 35% null values
print(f"Total columns with >= 35% null values: {train_null[train_null].
 ↪count()}")
```

```
Total columns with >= 35% null values: 52
```

[15]:
```
train_data.TARGET.value_counts()
```

[15]:
```
TARGET
0    282686
1     24825
Name: count, dtype: int64
```

[16]:
```python
X = train_data.drop(columns=['TARGET'])
y = train_data['TARGET']

# Initializing the RandomUnderSampler
rus = RandomUnderSampler(sampling_strategy='auto', random_state=42)

# Downsample the majority class
X_res, y_res = rus.fit_resample(X, y)

# Combine the resampled into a single DataFrame
train_data_resampled = pd.concat([X_res, y_res], axis=1)
```

[17]:
```python
train_data_resampled.TARGET.value_counts()
```

[17]:
```
TARGET
0    24825
1    24825
Name: count, dtype: int64
```

[18]:
```python
train_data_resampled.shape
```

[18]:
```
(49650, 135)
```

[19]:
```python
train_data_resampled = train_data_resampled.drop(columns=['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
                                                          'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
                                                          'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
                                                          'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13',
                                                          'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
                                                          'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21'
                                                          ])
```

- The `remove_outliers` function in PyCaret allows you to identify and remove outliers from the dataset before training the model.
- It can be achieved using `remove_outliers` parameter within `setup()`. The proportion of outliers are controlled through outliers_threshold parameter.

[20]:
```python
train = setup(data=train_data_resampled, target="TARGET", remove_outliers = True)
```

```
<pandas.io.formats.style.Styler at 0x7926d48771f0>
```

[21]:
```python
# compare baseline models
best_model = compare_models()
```

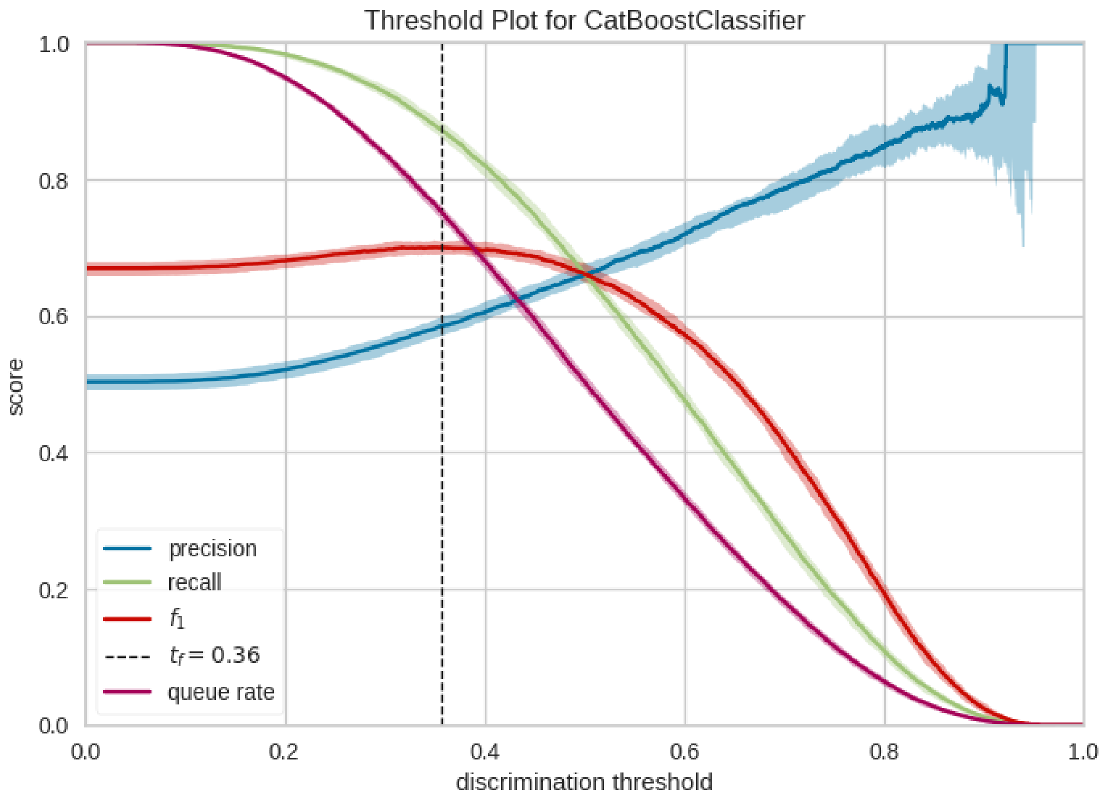
```
<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7926d4affc10>

<IPython.core.display.HTML object>
```

[22]:
```python
plot_model(best_model, plot='threshold')
```

```
<IPython.core.display.HTML object>
```



0.0.1 *Receiver-Operatng Characteristic Curve*

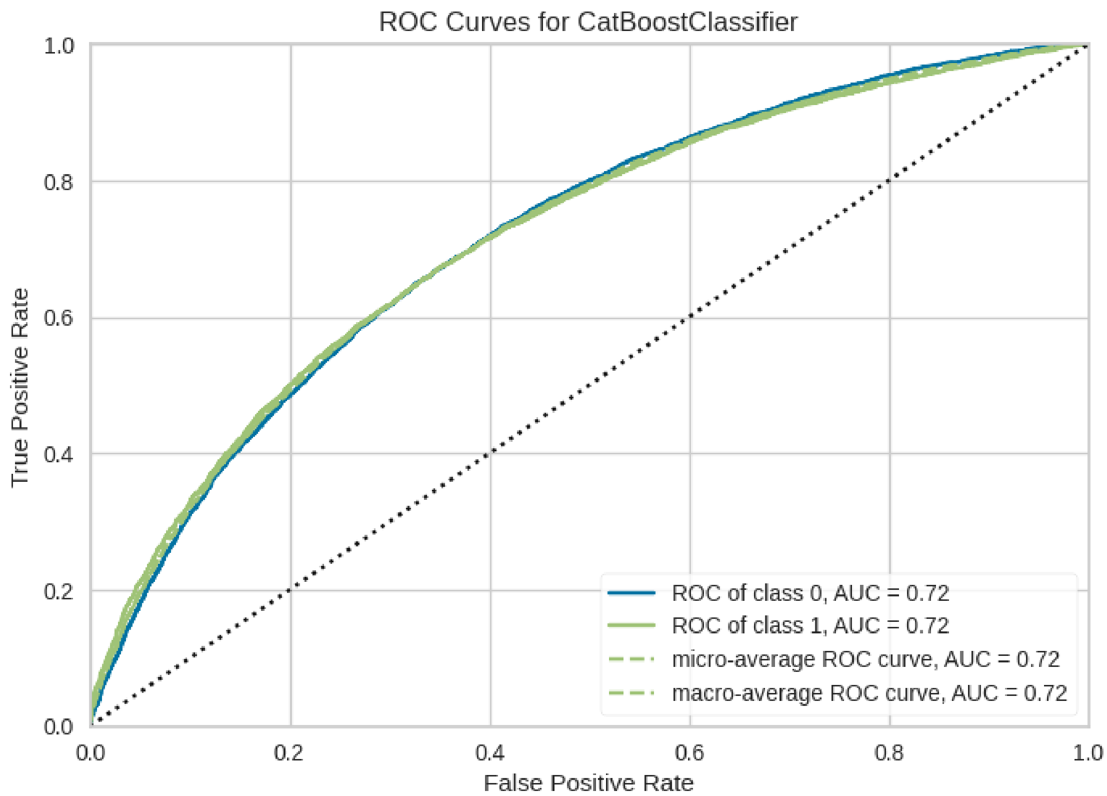
The ROC Curve is a visual reresentation of model perfomance across all thresholds

ROC can be quantified using AUC

---

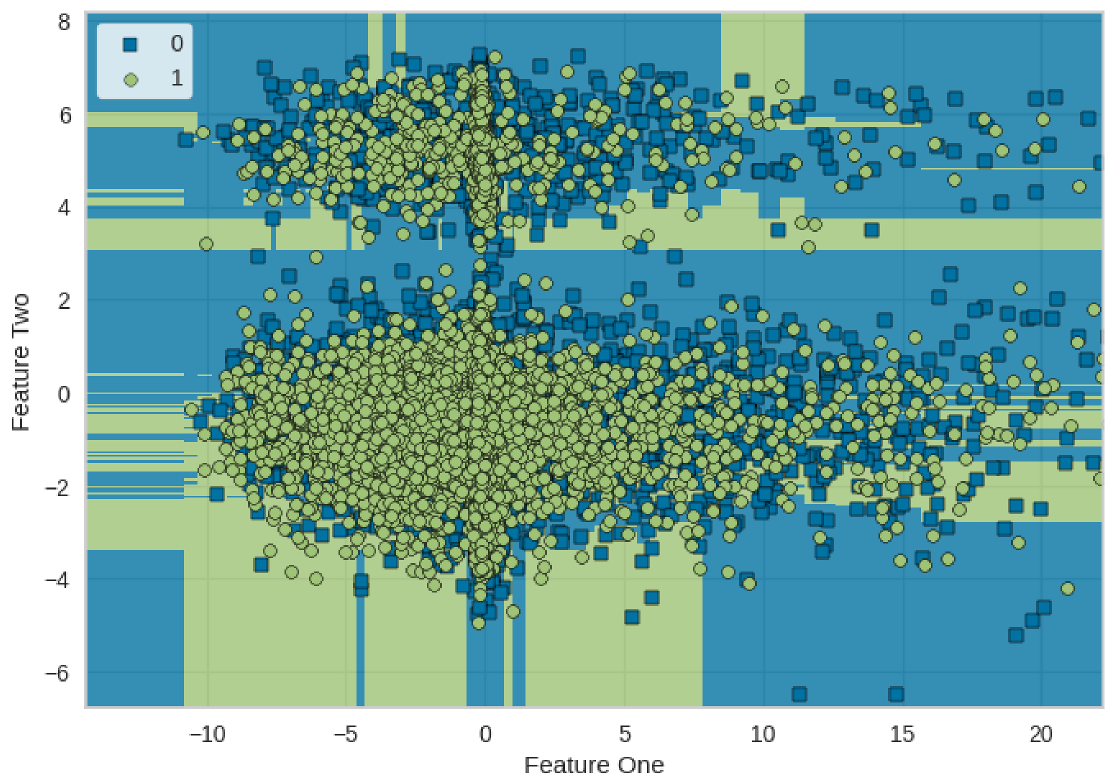
[23]:
```python
# Plot Area Under the Curve.
plot_model(best_model, plot='auc')
```

```
<IPython.core.display.HTML object>
```



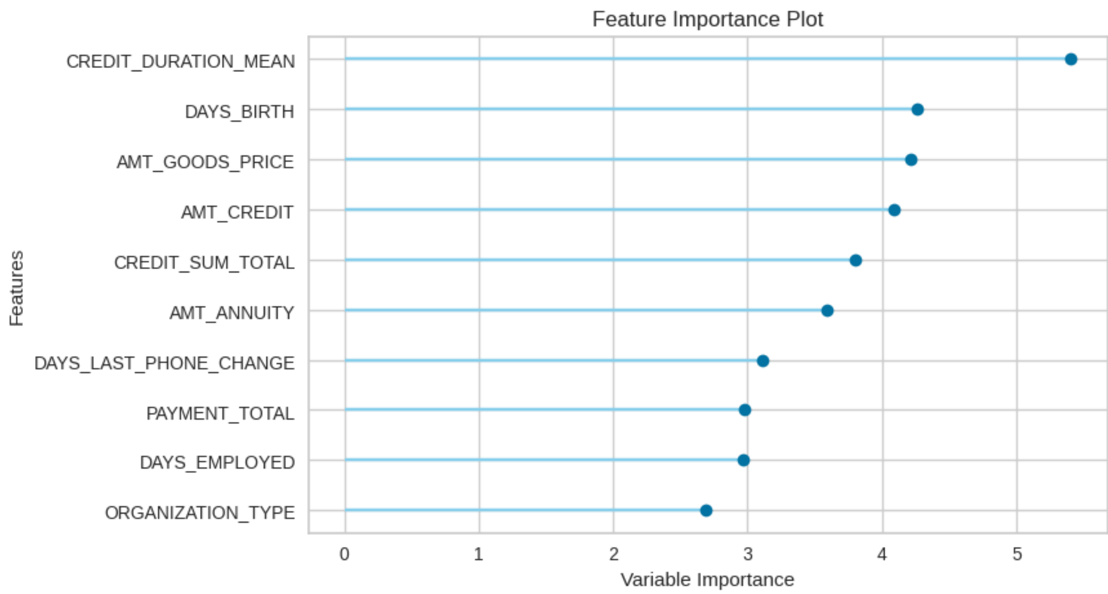
[24]:
```
# Plot
plot_model(best_model, plot='boundary')
```

```
<IPython.core.display.HTML object>
```



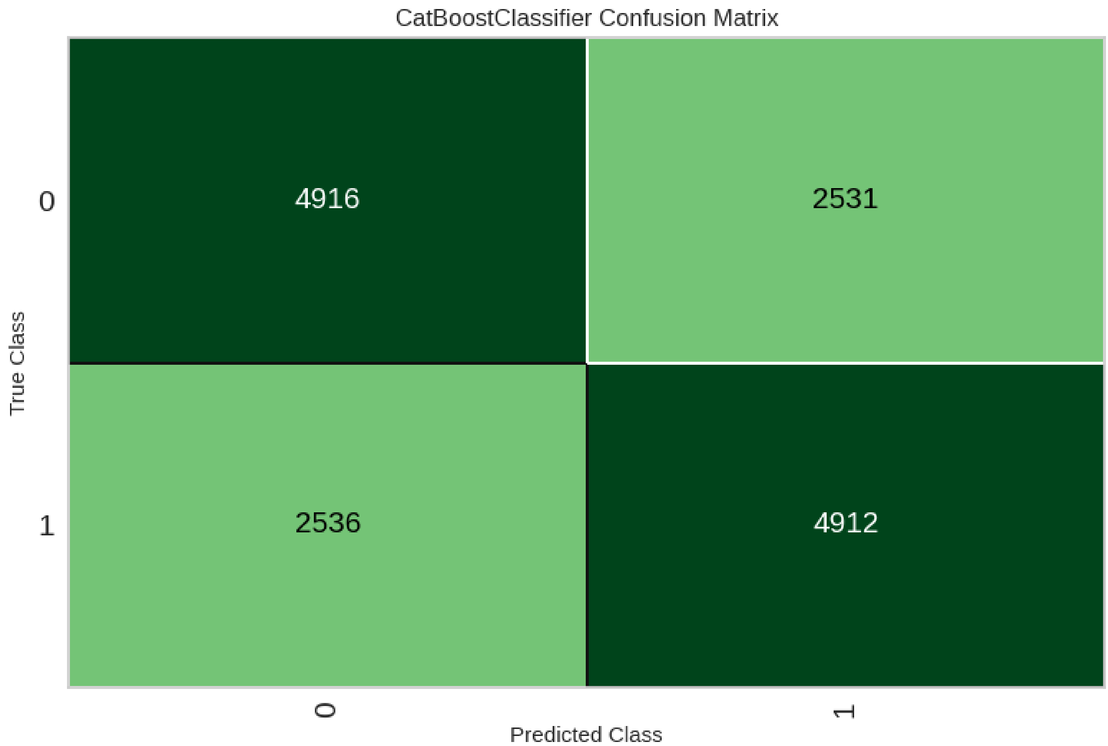
[25]:
```
# Plot feature importance
plot_model(best_model, plot='feature')
```

<IPython.core.display.HTML object>



[26]:
```
# Plot Confusion Matrix
plot_model(best_model, plot='confusion_matrix')
```

<IPython.core.display.HTML object>



[27]:
```
# Retrieve feature importance as a DataFrame
importance = pull()
importance
```

[27]:

	Model	Accuracy	AUC	Recall	Prec.	\
catboost	CatBoost Classifier	0.6574	0.7154	0.6564	0.6578	
gbc	Gradient Boosting Classifier	0.6491	0.7066	0.6529	0.6481	
ada	Ada Boost Classifier	0.6479	0.7006	0.6495	0.6475	
lda	Linear Discriminant Analysis	0.6405	0.6895	0.6510	0.6376	
ridge	Ridge Classifier	0.6396	0.6902	0.6489	0.6371	
xgboost	Extreme Gradient Boosting	0.6396	0.6906	0.6392	0.6398	
rf	Random Forest Classifier	0.6338	0.6820	0.6128	0.6397	
et	Extra Trees Classifier	0.6257	0.6686	0.6127	0.6291	
lr	Logistic Regression	0.5964	0.6314	0.6030	0.5951	
dt	Decision Tree Classifier	0.5483	0.5483	0.5465	0.5485	

```
nb                            Naive Bayes    0.5433  0.6120  0.2482  0.6191
knn                  K Neighbors Classifier    0.5383  0.5513  0.5415  0.5381
qda       Quadratic Discriminant Analysis    0.5364  0.6079  0.8413  0.5276
svm                     SVM - Linear Kernel    0.5217  0.5351  0.5486  0.5464
dummy                      Dummy Classifier    0.5000  0.5000  1.0000  0.5000

             F1   Kappa     MCC  TT (Sec)
catboost  0.6570  0.3147  0.3148    16.379
gbc       0.6504  0.2982  0.2983    11.585
ada       0.6484  0.2958  0.2958     2.842
lda       0.6442  0.2809  0.2810     1.278
ridge     0.6429  0.2793  0.2794     0.749
xgboost   0.6394  0.2792  0.2792     1.732
rf        0.6259  0.2675  0.2678     4.745
et        0.6207  0.2514  0.2515     4.230
lr        0.5990  0.1928  0.1929     8.725
dt        0.5475  0.0965  0.0965     1.527
nb        0.3282  0.0865  0.1135     0.750
knn       0.5397  0.0767  0.0767     1.735
qda       0.6411  0.0727  0.0952     1.303
svm       0.4857  0.0435  0.0633     0.952
dummy     0.6667  0.0000  0.0000     0.709
```

[28]:
```
predictions = predict_model(best_model, data=test_data)
```

```
<IPython.core.display.HTML object>
```

0.1 ## Submission Format

For each `SK_ID_CURR` in the test set, you must predict a probability for the `TARGET` variable.

- The file should contain a header and have the following format:

`SK_ID_CURR,TARGET` 1. `100001,0.1` 2. `100005,0.9` 3. `100013,0.2`

[29]:
```
selected_preds = predictions [['SK_ID_CURR','prediction_label',␣
↪'prediction_score']]
```

[30]:
```
selected_preds
```

[30]:

	SK_ID_CURR	prediction_label	prediction_score
0	100001	0	0.5991
1	100005	1	0.6540
2	100013	0	0.6863
3	100028	0	0.6542
4	100038	1	0.5932
…	…	…	…
48739	456221	1	0.5208

48740	456222	1	0.6180
48741	456223	0	0.6294
48742	456224	0	0.7794
48743	456250	1	0.7098

[48744 rows x 3 columns]