Credit Card Fraud: Model With Principal Parameter.

Previous Tasks

Import Libraries

In [1]:
```python
# Generic Libraries
import warnings
warnings.filterwarnings('ignore')
import os
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  RobustScaler
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, HTML

from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score
from sklearn.metrics import  f1_score, fbeta_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
```

In [2]:
```python
# Define general path:
path_general = r'C:\TFM'
path_total = os.path.join(path_general,'01_total_models')
```

In [3]:
```python
# Model Libraries.

# Cross validation
from sklearn.model_selection import cross_val_score

#------------- / Regresion Logistica /--------------
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression

#--------------/ XGBoost /--------------------------
from xgboost import XGBClassifier
import xgboost as xgb

#-------------/ AdaBoost /--------------------------
from sklearn.ensemble import AdaBoostClassifier

#-------------/ CatBoost /--------------------------
from catboost import CatBoostClassifier

#------------/ Decission Tree /----------------------
from sklearn.tree import DecisionTreeClassifier

#------------/ Random Forest /-----------------------
from sklearn.ensemble import RandomForestClassifier

#-----------/   MLP /--------------------------------
from sklearn.neural_network import MLPClassifier

#------------/ KNN /----------------------------------
from sklearn.neighbors import KNeighborsClassifier

#------------/ Naive - Bayes /-------------------------
from sklearn.naive_bayes import GaussianNB
```

Load Dataset

In [4]:
```python
# Load dataset.
df = pd.read_csv('creditcard.csv')
df = df.drop("Time", axis = 1)

y= df["Class"]
X = df.drop("Class", axis = 1)
y.shape,X.shape
```

Out[4]:
```
((284807,), (284807, 29))
```

In [5]:
```python
# Separation of the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42,stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[5]:
```
((227845, 29), (56962, 29), (227845,), (56962,))
```

In [6]:
```python
# Check dataset composition

print(" Fraudulent Count for Full data :  ",np.sum(y))
print(" Fraudulent Count for Train data : ",np.sum(y_train))
print(" Fraudulent Count for Test data :   ",np.sum(y_test))
```

```
 Fraudulent Count for Full data :   492
 Fraudulent Count for Train data :  394
 Fraudulent Count for Test data :    98
```

In [7]:
```python
# Save the testing set for evaluation
X_test_saved = X_test.copy()
y_test_saved = y_test.copy()
print("Saved X_test & y_test")
```

```
Saved X_test & y_test
```

In [8]:
```python
# As PCA is already performed on the dataset from V1 to V28 features, we are scaling only Amount field
scaler = RobustScaler()

# Scaling the train data
X_train[["Amount"]] = scaler.fit_transform(X_train[["Amount"]])

# Transforming the test data
X_test[["Amount"]] = scaler.transform(X_test[["Amount"]])
```

1.- Transformaciones de datos.

Dataset Original

Smote

In [9]:
```python
# Import of specific libraries
from collections import Counter
from imblearn.over_sampling import SMOTE

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

print('Resampled dataset shape %s' % Counter(y_train_smote))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({0: 227451, 1: 227451})
```

Adasyn

In [10]:
```python
# Import of specific libraries
from imblearn.over_sampling import ADASYN

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
adasyn = ADASYN(random_state=42)
X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)

print('Resampled dataset shape %s' % Counter(y_train_adasyn))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({1: 227458, 0: 227451})
```

Power Transformation

Original

In [11]:
```python
# - Apply : preprocessing.PowerTransformer(copy=False) to fit & transform the train & test data

from sklearn import metrics
from sklearn import preprocessing


from sklearn.preprocessing import PowerTransformer

pt= preprocessing.PowerTransformer(method='yeo-johnson', copy=True)  # creates an instance of the PowerTransformer class.
pt.fit(X_train)

X_train_pt = pt.transform(X_train)
X_test_pt = pt.transform(X_test)

y_train_pt = y_train
y_test_pt = y_test
```

Smote

In [12]:
```python
# Import of specific libraries
from collections import Counter
from imblearn.over_sampling import SMOTE

# Initial situation
print('Original dataset shape %s' % Counter(y_train_pt))

# Calculate OverSampling model
smote = SMOTE(random_state=42)
X_train_smote_pt, y_train_smote_pt = smote.fit_resample(X_train_pt, y_train_pt)

print('Resampled dataset shape %s' % Counter(y_train_smote_pt))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({0: 227451, 1: 227451})
```

Adasyn

In [13]:
```python
# Import of specific libraries
from imblearn.over_sampling import ADASYN

# Initial situation
print('Original dataset shape %s' % Counter(y_train))

# Calculate OverSampling model
adasyn = ADASYN(random_state=42)
X_train_adasyn_pt, y_train_adasyn_pt = adasyn.fit_resample(X_train_pt, y_train_pt)

print('Resampled dataset shape %s' % Counter(y_train_adasyn_pt))
```

```
Original dataset shape Counter({0: 227451, 1: 394})
Resampled dataset shape Counter({1: 227459, 0: 227451})
```

Load Model: Libraries and Functions.

In [14]:
```python
# LOAD OF MODELS.
# perfom cross validation on the X_train & y_train
from sklearn.model_selection import StratifiedKFold

# Initialize StratifiedKFold cross-validator
# perform cross validation
skf = StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
#  Shuffle is False because we need a constant best model when we use GridSearchCV
```

In [15]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
```

Create dataset_list

In [16]:
```python
# Original distribution
OR_origin = ['OR origin',X_train, y_train, X_test, y_test]
OR_smote =['OR smote',X_train_smote, y_train_smote, X_test, y_test]
OR_adasyn = ['OR adasyn', X_train_adasyn, y_train_adasyn, X_test, y_test]

# Power Transformation
PT_origin = ['PT origin',X_train_pt, y_train_pt, X_test_pt, y_test_pt]
PT_smote = ['PT smote',X_train_smote_pt, y_train_smote_pt, X_test_pt, y_test_pt ]
PT_adasyn = ['PT adasyn', X_train_adasyn_pt, y_train_adasyn_pt, X_test_pt, y_test_pt]
```

Create models

In [17]:
```python
model_list = ['regression_logistic', 'adaboost', 'xgboost', 'catboost', 'decision_tree', 'random_forest', 'mlp', 'knn']
parameters = [
    [0.1, 0.5, 1, 1.5, 2, 2.5, 3],       # For 'regression_logistic'
    [5, 7, 9],                           # For 'adaboost'
    [0.001, 0.01, 0.1, 0.5, 1, 3],       # For 'xgboost'
    [100, 200, 300, 400, 500, 600],       # For 'catboost'
    [1, 2, 3, 4, 5],                     # For 'decision_tree'
    [100, 200, 400, 600, 800, 1000, 1200],  # For 'random_forest'
    [(50,), (100,), (120,), (150,)],      # For 'mlp'
    [3, 5, 7]  #knn
    ]
```

In [19]:
```python
distributions =[OR_origin, OR_smote, OR_adasyn, PT_origin, PT_smote, PT_adasyn]
complete_model = zip(model_list, parameters)
complete_model_list = list(complete_model )
complete_model_list
```

Out[19]:
```
[('regression_logistic', [0.1, 0.5, 1]), ('knn', [3, 5])]
```

In [22]:
```python
def gen_models(complete_model_list, distributions, save_directory_complete_model=None):

    # test if directory is None and add path
    if save_directory_complete_model is None:
        save_directory_complete_model = os.path.join(os.getcwd(), 'total')

    #Create folder if not exits
    os.makedirs(save_directory_complete_model, exist_ok=True)


    # 1.- Iterate model
    for model_name, param_values in complete_model_list:
        #print(f"Processing model: {model_name}")

        resultados_totales = []

        # 2.- Iterate over distributions:
        for distribution in distributions:
            try:
                # Unpack distribution
                name = distribution[0]  # Nombre de la distribución
                X_train, y_train, X_val, y_val = distribution[1:]

                #print(f"  Distribution: {name}")  # Log distribución

                # Verify if data is valid:
                if X_train is None or y_train is None or X_val is None or y_val is None:
                    print(f"    Skipping due to missing data in {name}")
                    continue

                # 3.- Iterate over params:
                for param in param_values:
                    #print(f"    Training {model_name} with parameter {param}")  # Log parámetro

                    # Inicializate model and parameters
                    model_instance = None
                    if model_name == 'regression_logistic':
                        model_instance = LogisticRegression(C=param)
                        parameter_name ='C='
                    elif model_name == 'adaboost':
                        #model_instance = AdaBoostClassifier(DecisionTreeClassifier(max_depth=param))
                        model_instance = AdaBoostClassifier(DecisionTreeClassifier(iterations=param))
                        #parameter_name = 'max_depth='
                        parameter_name = 'iterations='
                    elif model_name == 'xgboost':
                        model_instance = XGBClassifier(learning_rate=param)
                        parameter_name  = 'learning_rate='
                    elif model_name == 'catboost':
                        model_instance = CatBoostClassifier(iterations=param, verbose=0)
                        parameter_name  = 'iterations='
                    elif model_name == 'decision_tree':
                        model_instance = DecisionTreeClassifier(max_depth=param)
                        parameter_name  = 'max_depth='
                    elif model_name == 'random_forest':
                        model_instance = RandomForestClassifier(n_estimators=param)
                        parameter_name  = 'n_estimators'
                    elif model_name == 'mlp':
                        model_instance = MLPClassifier(hidden_layer_sizes=param)
                        parameter_name  = 'hidden_layer_sizes'
                    elif model_name == 'knn':
                        model_instance = KNeighborsClassifier(n_neighbors=param)
                        parameter_name  = 'n_neighbors'
                    else:
                        print(f"    Invalid model name: {model_name}")
                        continue


                    # Train model
                    model_instance.fit(X_train, y_train)

                    # Get predictions
                    y_pred = model_instance.predict(X_val)

                    # Calculate metrics
                    roc_auc = roc_auc_score(y_val, y_pred)
                    accuracy = accuracy_score(y_val, y_pred)
                    precision = precision_score(y_val, y_pred)
                    recall = recall_score(y_val, y_pred)
                    f1 = f1_score(y_val, y_pred)
                    f2 = fbeta_score(y_val, y_pred, beta=2)
                    confusion = confusion_matrix(y_val, y_pred)

                    # Save results in DataFrame:
                    results_df = pd.DataFrame({
                        'Model': [model_name],
                        'Description': [name],
                        'Parameter':[parameter_name + str(param)],
                        'ROC-AUC': [roc_auc],
                        'Accuracy': [accuracy],
                        'Precision': [precision],
                        'Recall': [recall],
                        'F1 Score': [f1],
                        'F2 Score': [f2],
                        'Confusion Matrix': [confusion],
                    })

                    resultados_totales.append(results_df)

            except Exception as e:
                print(f"  Error processing distribution {name}: {str(e)}")
                continue

        # Save results:
        if resultados_totales:
            df_resultados_final = pd.concat(resultados_totales, ignore_index=True)
            save_path = os.path.join(save_directory_complete_model, f"{model_name}_total.csv")
            df_resultados_final.to_csv(save_path, index=False)
            display(HTML(f"<h2 style='text-align: center;font-size:60px;'> Modelo: {model_name}</h2>"))
            display(df_resultados_final)
            print(f"\n\n\nResults for {model_name} saved to {save_path}")
        else:
            print(f"No results generated for {model_name}")
```

In [7]:

Modelo: random_forest

Out[7]:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	random_forest	OR origin	n_estimators=100	0.908119	0.999596	0.941176	0.816327	0.874317	0.838574	[[56859 5], [18 80]]
1	random_forest	OR origin	n_estimators=200	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859 5], [17 81]]
2	random_forest	OR origin	n_estimators=300	0.908128	0.999614	0.952381	0.816327	0.879121	0.840336	[[56860 4], [18 80]]
3	random_forest	OR origin	n_estimators=400	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859 5], [17 81]]
4	random_forest	OR origin	n_estimators=500	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859 5], [17 81]]
5	random_forest	OR origin	n_estimators=600	0.908119	0.999596	0.941176	0.816327	0.874317	0.838574	[[56859 5], [18 80]]
6	random_forest	OR smote	n_estimators=100	0.908058	0.999473	0.869565	0.816327	0.842105	0.826446	[[56852 12], [18 80]]
7	random_forest	OR smote	n_estimators=200	0.913151	0.999473	0.861702	0.826531	0.843750	0.833333	[[56851 13], [17 81]]
8	random_forest	OR smote	n_estimators=300	0.908067	0.999491	0.879121	0.816327	0.846561	0.828157	[[56853 11], [18 80]]
9	random_forest	OR smote	n_estimators=400	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853 11], [19 79]]
10	random_forest	OR smote	n_estimators=500	0.908067	0.999491	0.879121	0.816327	0.846561	0.828157	[[56853 11], [18 80]]
11	random_forest	OR smote	n_estimators=600	0.902956	0.999456	0.868132	0.806122	0.835979	0.817805	[[56852 12], [19 79]]
12	random_forest	OR adasyn	n_estimators=100	0.908058	0.999473	0.869565	0.816327	0.842105	0.826446	[[56852 12], [18 80]]
13	random_forest	OR adasyn	n_estimators=200	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853 11], [19 79]]
14	random_forest	OR adasyn	n_estimators=300	0.902938	0.999421	0.849462	0.806122	0.827225	0.814433	[[56850 14], [19 79]]
15	random_forest	OR adasyn	n_estimators=400	0.902947	0.999438	0.858696	0.806122	0.831579	0.816116	[[56851 13], [19 79]]
16	random_forest	OR adasyn	n_estimators=500	0.902956	0.999456	0.868132	0.806122	0.835979	0.817805	[[56852 12], [19 79]]
17	random_forest	OR adasyn	n_estimators=600	0.902947	0.999438	0.858696	0.806122	0.831579	0.816116	[[56851 13], [19 79]]
18	random_forest	PT origin	n_estimators=100	0.913230	0.999631	0.952941	0.826531	0.885246	0.849057	[[56860 4], [17 81]]
19	random_forest	PT origin	n_estimators=200	0.908119	0.999596	0.941176	0.816327	0.874317	0.838574	[[56859 5], [18 80]]
20	random_forest	PT origin	n_estimators=300	0.918323	0.999631	0.942529	0.836735	0.886486	0.855950	[[56859 5], [16 82]]
21	random_forest	PT origin	n_estimators=400	0.908119	0.999596	0.941176	0.816327	0.874317	0.838574	[[56859 5], [18 80]]
22	random_forest	PT origin	n_estimators=500	0.918323	0.999631	0.942529	0.836735	0.886486	0.855950	[[56859 5], [16 82]]
23	random_forest	PT origin	n_estimators=600	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859 5], [17 81]]
24	random_forest	PT smote	n_estimators=100	0.913169	0.999508	0.880435	0.826531	0.852632	0.836777	[[56853 11], [17 81]]
25	random_forest	PT smote	n_estimators=200	0.908058	0.999473	0.869565	0.816327	0.842105	0.826446	[[56852 12], [18 80]]
26	random_forest	PT smote	n_estimators=300	0.908058	0.999473	0.869565	0.816327	0.842105	0.826446	[[56852 12], [18 80]]
27	random_forest	PT smote	n_estimators=400	0.908084	0.999526	0.898876	0.816327	0.855615	0.831601	[[56855 9], [18 80]]
28	random_forest	PT smote	n_estimators=500	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853 11], [19 79]]
29	random_forest	PT smote	n_estimators=600	0.913160	0.999491	0.870968	0.826531	0.848168	0.835052	[[56852 12], [17 81]]
30	random_forest	PT adasyn	n_estimators=100	0.902973	0.999491	0.887640	0.806122	0.844920	0.821206	[[56854 10], [19 79]]
31	random_forest	PT adasyn	n_estimators=200	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853 11], [19 79]]
32	random_forest	PT adasyn	n_estimators=300	0.902956	0.999456	0.868132	0.806122	0.835979	0.817805	[[56852 12], [19 79]]
33	random_forest	PT adasyn	n_estimators=400	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853 11], [19 79]]
34	random_forest	PT adasyn	n_estimators=500	0.902947	0.999438	0.858696	0.806122	0.831579	0.816116	[[56851 13], [19 79]]
35	random_forest	PT adasyn	n_estimators=600	0.902956	0.999456	0.868132	0.806122	0.835979	0.817805	[[56852 12], [19 79]]

In [10]:

Modelo: knn

Out[10]:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	knn	OR origin	n_neighbors=3	0.908093	0.999544	0.909091	0.816327	0.860215	0.833333	[[56856 8], [18 80]]
1	knn	OR origin	n_neighbors=5	0.892787	0.999491	0.905882	0.785714	0.841530	0.807128	[[56856 8], [21 77]]
2	knn	OR origin	n_neighbors=7	0.877489	0.999456	0.913580	0.755102	0.826816	0.782241	[[56857 7], [24 74]]
3	knn	OR smote	n_neighbors=3	0.927930	0.998473	0.535032	0.857143	0.658824	0.765027	[[56791 73], [14 84]]
4	knn	OR smote	n_neighbors=5	0.937764	0.997770	0.427861	0.877551	0.575251	0.725126	[[56749 115], [12 86]]
5	knn	OR smote	n_neighbors=7	0.947520	0.996910	0.346457	0.897959	0.500000	0.681115	[[56698 166], [10 88]]
6	knn	OR adasyn	n_neighbors=3	0.927930	0.998473	0.535032	0.857143	0.658824	0.765027	[[56791 73], [14 84]]
7	knn	OR adasyn	n_neighbors=5	0.937764	0.997770	0.427861	0.877551	0.575251	0.725126	[[56749 115], [12 86]]
8	knn	OR adasyn	n_neighbors=7	0.947502	0.996875	0.343750	0.897959	0.497175	0.679012	[[56696 168], [10 88]]
9	knn	PT origin	n_neighbors=3	0.918288	0.999561	0.901099	0.836735	0.867725	0.848861	[[56855 9], [16 82]]
10	knn	PT origin	n_neighbors=5	0.887685	0.999473	0.904762	0.775510	0.835165	0.798319	[[56856 8], [22 76]]
11	knn	PT origin	n_neighbors=7	0.872352	0.999368	0.869048	0.744898	0.802198	0.766807	[[56853 11], [25 73]]
12	knn	PT smote	n_neighbors=3	0.933032	0.998490	0.537975	0.867347	0.664062	0.772727	[[56791 73], [13 85]]
13	knn	PT smote	n_neighbors=5	0.942919	0.997893	0.443878	0.887755	0.591837	0.739796	[[56755 109], [11 87]]
14	knn	PT smote	n_neighborsv7	0.947687	0.997244	0.374468	0.897959	0.528529	0.701754	[[56717 147], [10 88]]
15	knn	PT adasyn	n_neighbors=3	0.933032	0.998490	0.537975	0.867347	0.664062	0.772727	[[56791 73], [13 85]]
16	knn	PT adasyn	n_neighbors=5	0.942910	0.997876	0.441624	0.887755	0.589831	0.738540	[[56754 110], [11 87]]
17	knn	PT adasyn	n_neighbors=7	0.947687	0.997244	0.374468	0.897959	0.528529	0.701754	[[56717 147], [10 88]]

In [12]:

Modelo: mlp

Out[12]:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	mlp	OR origin	hidden_layer_sizes=(50,)	0.908058	0.999473	0.869565	0.816327	0.842105	0.826446	[[56852 12],[18 80]]
1	mlp	OR origin	hidden_layer_sizes=(100,)	0.918218	0.999421	0.828283	0.836735	0.832487	0.835031	[[56847 17],[16 82]]
2	mlp	OR origin	hidden_layer_sizes=(120,)	0.913160	0.999491	0.870968	0.826531	0.848168	0.835052	[[56852 12],[17 81]]
3	mlp	OR origin	hidden_layer_sizes=(150,)	0.928343	0.999298	0.763636	0.857143	0.807692	0.836653	[[56838 26],[14 84]]
4	mlp	OR smote	hidden_layer_sizes=(50,)	0.907864	0.999087	0.701754	0.816327	0.754717	0.790514	[[56830 34],[18 80]]
5	mlp	OR smote	hidden_layer_sizes=(100,)	0.918130	0.999245	0.752294	0.836735	0.792271	0.818363	[[56837 27],[16 82]]
6	mlp	OR smote	hidden_layer_sizes=(120,)	0.918104	0.999192	0.732143	0.836735	0.780952	0.813492	[[56834 30],[16 82]]
7	mlp	OR smote	hidden_layer_sizes=(150,)	0.897783	0.999298	0.795918	0.795918	0.795918	0.795918	[[56844 20],[20 78]]
8	mlp	OR adasyn	hidden_layer_sizes=(50,)	0.913037	0.999245	0.757009	0.826531	0.790244	0.811623	[[56838 26],[17 81]]
9	mlp	OR adasyn	hidden_layer_sizes=(100,)	0.902771	0.999087	0.705357	0.806122	0.752381	0.783730	[[56831 33],[19 79]]
10	mlp	OR adasyn	hidden_layer_sizes=(120,)	0.892558	0.999034	0.693694	0.785714	0.736842	0.765408	[[56830 34],[21 77]]
11	mlp	OR adasyn	hidden_layer_sizes=(150,)	0.923258	0.999315	0.775701	0.846939	0.809756	0.831663	[[56840 24],[15 83]]
12	mlp	PT origin	hidden_layer_sizes=(50,)	0.913151	0.999473	0.861702	0.826531	0.843750	0.833333	[[56851 13],[17 81]]
13	mlp	PT origin	hidden_layer_sizes=(100,)	0.892787	0.999491	0.905882	0.785714	0.841530	0.807128	[[56856 8],[21 77]]
14	mlp	PT origin	hidden_layer_sizes=(120,)	0.928316	0.999245	0.743363	0.857143	0.796209	0.831683	[[56835 29],[14 84]]
15	mlp	PT origin	hidden_layer_sizes=(150,)	0.913151	0.999473	0.861702	0.826531	0.843750	0.833333	[[56851 13],[17 81]]
16	mlp	PT smote	hidden_layer_sizes=(50,)	0.897607	0.998947	0.661017	0.795918	0.722222	0.764706	[[56824 40],[20 78]]
17	mlp	PT smote	hidden_layer_sizes=(100,)	0.887588	0.999280	0.800000	0.775510	0.787565	0.780287	[[56845 19],[22 76]]
18	mlp	PT smote	hidden_layer_sizes=(120,)	0.897731	0.999192	0.750000	0.795918	0.772277	0.786290	[[56838 26],[20 78]]
19	mlp	PT smote	hidden_layer_sizes=(150,)	0.902833	0.999210	0.752381	0.806122	0.778325	0.794769	[[56838 26],[19 79]]
20	mlp	PT adasyn	hidden_layer_sizes=(50,)	0.907847	0.999052	0.689655	0.816327	0.747664	0.787402	[[56828 36],[18 80]]
21	mlp	PT adasyn	hidden_layer_sizes=(100,)	0.882433	0.999157	0.750000	0.765306	0.757576	0.762195	[[56839 25],[23 75]]
22	mlp	PT adasyn	hidden_layer_sizes=(120,)	0.897695	0.999122	0.722222	0.795918	0.757282	0.780000	[[56834 30],[20 78]]
23	mlp	PT adasyn	hidden_layer_sizes=(150,)	0.892655	0.999228	0.770000	0.785714	0.777778	0.782520	[[56841 23],[21 77]]

In [ ]: