AdaBoost Hyperparameter

In [1]:
```python
# Generic Libraries
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  RobustScaler
import numpy as np
```

In [2]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [3]:
```python
# Metric Libraries
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score, f1_score,fbeta_score, confusion_matrix

# Grid
from sklearn.model_selection import GridSearchCV
```

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})
```

In [11]:
```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
```

Power Transformation

Original

In [12]:
```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 [13]:
```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 [14]:
```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})
```

In [15]:
```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]
```

Preparacion carga de modelos: librerias y funciones

In [16]:
```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 [17]:
```python
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
```

In [18]:
```python
def evaluate_adaboost(data_list, params_to_show=None, threshold=0.5, **ada_params):
    '''
    This function trains an AdaBoost model with a DecisionTree base estimator
    and evaluates it with a custom classification threshold.

    Parameters:
        - data_list: List containing [name, X_train, y_train, X_val, y_val].
        - params_to_show: Dictionary with parameters to display (optional).
        - threshold: The classification threshold (default = 0.5).
        - **ada_params: Additional AdaBoost parameters to be passed dynamically.

    Return:
        - A DataFrame with evaluation metrics (Accuracy, Precision, Recall, F1, F2, ROC-AUC, Confusion Matrix).
    '''
    # Diccionario de abreviaturas
    param_abbreviations = {
        'n_estimators': 'n_est',
        'learning_rate': 'lr',
        'threshold': 'th'
    }

    # Unpack the data list
    name = data_list[0]
    X_train, y_train, X_val, y_val = data_list[1:]

    # Define the AdaBoost model with a DecisionTree base estimator
    base_estimator = DecisionTreeClassifier(max_depth=ada_params.pop('max_depth', None), random_state=42)
    ada_model = AdaBoostClassifier(base_estimator=base_estimator, **ada_params)
    #ada_model = AdaBoostClassifier(DecisionTreeClassifier(random_state=42), **ada_params)

    # Train the model
    ada_model.fit(X_train, y_train)

    # Predict probabilities
    y_prob = ada_model.predict_proba(X_val)[:, 1]  # Probabilities for the positive class (fraud)

    # Adjust predictions based on the threshold
    y_pred = (y_prob > threshold).astype(int)

    # Calculate metrics
    cm = confusion_matrix(y_val, y_pred)
    roc_auc = roc_auc_score(y_val, y_prob)  # Use probabilities to calculate ROC-AUC
    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)

    # Create a string with the parameters to show
    if params_to_show is None:
        params_to_show = {'threshold': threshold}
        params_to_show.update(ada_params)

    # Create a version with abbreviations
    params_with_abbreviations = {
        param_abbreviations.get(key, key): value for key, value in params_to_show.items()
    }

    # Build the parameter string dynamically
    params_str = [f"{key}={value}" for key, value in params_with_abbreviations.items()]

    # Store the results in a DataFrame
    results_df = pd.DataFrame({
        'Model': ['Adaboost'],
        'Description': [data_list[0]],
        'Parameter': [params_str],  # Show abbreviated parameters here
        'ROC-AUC': [roc_auc],
        'Accuracy': [accuracy],
        'Precision': [precision],
        'Recall': [recall],
        'F1 Score': [f1],
        'F2 Score': [f2],
        'Confusion Matrix': [cm]
    })

    # Adjust for display
    pd.set_option('display.max_colwidth', None)

    return results_df
```

In [19]:
```python
# Parameters for AdaBoost
valores_learning_rate = [ 0.01, 0.05, 0.1, 0.5, 1]
valores_n_estimators = [10, 15, 20, 25, 30]
valores_max_depth = [3, 5, 7]

total_results = []

# Iterate over parameters to do combined testing
for learning_rate in valores_learning_rate:
    for n_estimators in valores_n_estimators:
        for max_depth in valores_max_depth:
            # Execute the function with different combinations of hyperparameters
            results = evaluate_adaboost(
                OR_smote,
                #thresold = 0.5,
                n_estimators=n_estimators,
                learning_rate=learning_rate,
                max_depth=max_depth
            )
            total_results.append(results)

# Combine all results into a single DataFrame for visualization
total_results_df = pd.concat(total_results, ignore_index=True)
```

In [20]:
```python
total_results_df_sorted = total_results_df.sort_values(by='F2 Score', ascending=False).reset_index(drop=True)
total_results_df_sorted
```

Out[20]:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	Adaboost	OR smote	[th=0.5, n_est=25, lr=1]	0.961835	0.999350	0.790476	0.846939	0.817734	0.835010	[[56842, 22], [15, 83]]
1	Adaboost	OR smote	[th=0.5, n_est=30, lr=1]	0.966495	0.999386	0.818182	0.826531	0.822335	0.824847	[[56846, 18], [17, 81]]
2	Adaboost	OR smote	[th=0.5, n_est=30, lr=0.5]	0.960542	0.999298	0.778846	0.826531	0.801980	0.816532	[[56841, 23], [17, 81]]
3	Adaboost	OR smote	[th=0.5, n_est=25, lr=0.5]	0.954105	0.999298	0.790000	0.806122	0.797980	0.802846	[[56843, 21], [19, 79]]
4	Adaboost	OR smote	[th=0.5, n_est=20, lr=1]	0.952125	0.999175	0.738318	0.806122	0.770732	0.791583	[[56836, 28], [19, 79]]
...	...	...	...	...	...	...	...	...	...	...
70	Adaboost	OR smote	[th=0.5, n_est=30, lr=0.01]	0.949811	0.975598	0.058743	0.877551	0.110115	0.231681	[[55486, 1378], [12, 86]]
71	Adaboost	OR smote	[th=0.5, n_est=15, lr=0.01]	0.955059	0.975545	0.058020	0.867347	0.108765	0.228864	[[55484, 1380], [13, 85]]
72	Adaboost	OR smote	[th=0.5, n_est=20, lr=0.01]	0.957507	0.975756	0.057891	0.857143	0.108457	0.227889	[[55497, 1367], [14, 84]]
73	Adaboost	OR smote	[th=0.5, n_est=25, lr=0.01]	0.949936	0.975580	0.057495	0.857143	0.107761	0.226659	[[55487, 1377], [14, 84]]
74	Adaboost	OR smote	[th=0.5, n_est=10, lr=0.01]	0.958325	0.969804	0.047433	0.867347	0.089947	0.194597	[[55157, 1707], [13, 85]]

75 rows × 10 columns

In [21]:
```python
adaboost_hyperparameters = total_results_df_sorted[total_results_df_sorted['F2 Score']>= .85].reset_index(drop=True)
```

In [22]:
```python
adaboost_hyperparameters
```

Out[22]:

Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix

In [23]:
```python
total_results_df
```

Out[23]:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	Adaboost	OR smote	[th=0.5, n_est=10, lr=0.01]	0.958325	0.969804	0.047433	0.867347	0.089947	0.194597	[[55157, 1707], [13, 85]]
1	Adaboost	OR smote	[th=0.5, n_est=10, lr=0.01]	0.940441	0.984340	0.089876	0.887755	0.163227	0.319853	[[55983, 881], [11, 87]]
2	Adaboost	OR smote	[th=0.5, n_est=10, lr=0.01]	0.935970	0.985780	0.094533	0.846939	0.170082	0.326772	[[56069, 795], [15, 83]]
3	Adaboost	OR smote	[th=0.5, n_est=15, lr=0.01]	0.955059	0.975545	0.058020	0.867347	0.108765	0.228864	[[55484, 1380], [13, 85]]
4	Adaboost	OR smote	[th=0.5, n_est=15, lr=0.01]	0.945361	0.984182	0.089048	0.887755	0.161860	0.317750	[[55974, 890], [11, 87]]
...	...	...	...	...	...	...	...	...	...	...
70	Adaboost	OR smote	[th=0.5, n_est=25, lr=1]	0.947812	0.998736	0.594203	0.836735	0.694915	0.773585	[[56808, 56], [16, 82]]
71	Adaboost	OR smote	[th=0.5, n_est=25, lr=1]	0.961835	0.999350	0.790476	0.846939	0.817734	0.835010	[[56842, 22], [15, 83]]
72	Adaboost	OR smote	[th=0.5, n_est=30, lr=1]	0.963473	0.993188	0.184783	0.867347	0.304659	0.498826	[[56489, 375], [13, 85]]
73	Adaboost	OR smote	[th=0.5, n_est=30, lr=1]	0.951707	0.998876	0.630769	0.836735	0.719298	0.785441	[[56816, 48], [16, 82]]
74	Adaboost	OR smote	[th=0.5, n_est=30, lr=1]	0.966495	0.999386	0.818182	0.826531	0.822335	0.824847	[[56846, 18], [17, 81]]

75 rows × 10 columns

In [ ]:
```python
adaboost_hyperparameters.to_csv(r'C:\TFM\06_hyperparameter\adaboost.csv', index=False)
```