Credit Card Fraud: Default Models

0.- Previous Tasks

In [1]:
```
# 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
import os
from IPython.display import display, HTML  # Importar HTML para aplicar formato
```

In [2]:
```
# Define general path:
path_general = r'C:\TFM'
path_default = os.path.join(path_general,'00_Default')
```

In [3]:
```
# 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
```

In [4]:
```
# 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 [5]:
```
# 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[5]:
```
((284807,), (284807, 29))
```

In [6]:
```
# 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[6]:
```
((227845, 29), (56962, 29), (227845,), (56962,))
```

In [7]:
```
# 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 [8]:
```
# 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 [9]:
```
# 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"]])
```

Data Transformation

Original Dataset

Smote

In [10]:
```
# 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 [11]:
```
# 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 [12]:
```
# - 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]:
```
# 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]:
```
# 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})
```

Model Loading Preparation: Libraries and Functions

In [15]:
```
# 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 [16]:
```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
```

In [17]:
```
import pandas as pd
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, f1_score, fbeta_score, confusion_matrix
```

Functions

In [18]:
```
def model_cv(distributions, model_type, save_directory_complete_model=None):
    '''
    This function performs cross-validation and allows training the following algorithms:
        - 'regression_logistic' - 'xgboost' - 'adaboost' - 'catboost'
        - 'decision_tree' - 'random_forest' - 'mlp' - 'knn'

    It makes and validates predictions and obtains the most common evaluation metrics.

    Parameters:
        - data_list: A list containing the name of the dataset, followed by the train dataset variables [name, X_train, y_train, X_val, y_val].
        - param_values: Primary parameter of the selected algorithm. If not provided, default values are used.
        - model_type: See the list of allowed algorithms; it returns an error if not allowed.

    Returns:
        - Returns a dataframe with the most frequent evaluation measures of an algorithm:
            - roc_auc - accuracy - precision - recall - f1_score - f2_score - confusion_matrix
    '''
    resultados_totales = []
    for distribution in distributions:
        # Unpack the data list
        name = distribution[0]
        X_train, y_train, X_val, y_val = distribution[1:]

        # Create lists to store the results
        roc_auc_scores = []
        accuracy_scores = []
        precision_scores = []
        recall_scores = []
        f1_scores = []
        f2_scores = []
        confusion_matrices = []

        # Create the model according to the specified type
        #------------/ GRUPO 1 /----------------------------
        if model_type == 'regression_logistic':
            model = LogisticRegression()
        #------------/ GRUPO 2 /----------------------------
        elif model_type == 'xgboost':
            model = XGBClassifier()
        elif model_type == 'adaboost':
            model = AdaBoostClassifier(DecisionTreeClassifier())
        elif model_type == 'catboost':
            model = CatBoostClassifier(verbose=0)

        #------------/ GRUPO 3 /-----------------------------
        elif model_type == 'decision_tree':
            model = DecisionTreeClassifier()
        elif model_type == 'random_forest':
            model = RandomForestClassifier()
        #------------/ GRUPO 4 /------------------------------
        elif model_type == 'mlp':
            model = MLPClassifier()
        #------------/ GRUPO 5 /------------------------------
        elif model_type == 'knn':
            model = KNeighborsClassifier()
        else:
            raise ValueError('Invalid model_type parameter.')

        # Train the model with the training set
        model.fit(X_train, y_train)

        # Get the predictions for the validation set
        y_pred = model.predict(X_val)

        # Calculate the 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)

        # Add the results to the lists
        roc_auc_scores.append(roc_auc)
        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f1_scores.append(f1)
        f2_scores.append(f2)
        confusion_matrices.append(confusion)

        # Create the DataFrame with the results
        results_df = pd.DataFrame({
            'Model': [model_type],
            'Description': [name],
            'Parameter': 'Default',
            'ROC-AUC': roc_auc_scores,
            'Accuracy': accuracy_scores,
            'Precision': precision_scores,
            'Recall': recall_scores,
            'F1 Score': f1_scores,
            'F2 Score': f2_scores,
            'Confusion Matrix': confusion_matrices
        })
        resultados_totales.append(results_df)
    df_resultados_final = pd.concat(resultados_totales, ignore_index=True)

    # Save current work directory by default
    if save_directory_complete_model is None:
        save_directory_complete_model = os.path.join(os.getcwd(), 'default')

    # Create path of file CSV using model name
    save_path = os.path.join(save_directory_complete_model, f"{model_type}_default.csv")

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

    # Save dataframe as CSV file
    df_resultados_final.to_csv(save_path, index=False)
    return (df_resultados_final)
```

In [19]:
```
def show_models(model_list):
    """
    This function shows models in screen in pretty_paint format

    Parameters:
        - data_list: A list containing the name of models.
    """
    for model in model_list:
        display(HTML(f"<h2 style='text-align: center;font-size:30px;'> Modelo {model}:</h2>"))

        # Call function y save result
        my_model = model_cv(distributions, model, path_default)

        # Show DF in screen
        display(my_model)
        print("\n\n\n")
```

Create distributions

In [20]:
```
# 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]

# Total Distributions
distributions = [OR_origin, OR_smote, OR_adasyn, PT_origin, PT_smote, PT_adasyn]
```

In [21]:
```
model_list = ['regression_logistic', 'adaboost', 'xgboost', 'catboost', 'decision_tree', 'random_forest', 'mlp' , 'knn']
```

In [22]:
```
show_models(model_list)
```

Modelo regression_logistic:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	regression_logistic	OR origin	Default	0.826416	0.999175	0.831169	0.653061	0.731429	0.682303	[[56851, 13], [34, 64]]
1	regression_logistic	OR smote	Default	0.946029	0.973596	0.056747	0.918367	0.106888	0.227503	[[55368, 1496], [8, 90]]
2	regression_logistic	OR adasyn	Default	0.922704	0.916857	0.018880	0.928571	0.037007	0.087299	[[52135, 4729], [7, 91]]
3	regression_logistic	PT origin	Default	0.836603	0.999175	0.814815	0.673469	0.737430	0.697674	[[56849, 15], [32, 66]]
4	regression_logistic	PT smote	Default	0.944807	0.971156	0.052174	0.918367	0.098738	0.212565	[[55229, 1635], [8, 90]]
5	regression_logistic	PT adasyn	Default	0.920330	0.912117	0.017878	0.928571	0.035081	0.082999	[[51865, 4999], [7, 91]]

Modelo adaboost:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	adaboost	OR origin	Default	0.877349	0.999175	0.762887	0.755102	0.758974	0.756646	[[56841, 23], [24, 74]]
1	adaboost	OR smote	Default	0.886550	0.997209	0.356808	0.775510	0.488746	0.628099	[[56727, 137], [22, 76]]
2	adaboost	OR adasyn	Default	0.871596	0.997858	0.429412	0.744898	0.544776	0.649466	[[56767, 97], [25, 73]]
3	adaboost	PT origin	Default	0.872220	0.999105	0.737374	0.744898	0.741117	0.743381	[[56838, 26], [25, 73]]
4	adaboost	PT smote	Default	0.891793	0.997507	0.388889	0.785714	0.520270	0.652542	[[56743, 121], [21, 77]]
5	adaboost	PT adasyn	Default	0.881677	0.997648	0.403226	0.765306	0.528169	0.648789	[[56753, 111], [23, 75]]

Modelo xgboost:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	xgboost	OR origin	Default	0.908102	0.999561	0.919540	0.816327	0.864865	0.835073	[[56857, 7], [18, 80]]
1	xgboost	OR smote	Default	0.943579	0.999210	0.719008	0.887755	0.794521	0.847953	[[56830, 34], [11, 87]]
2	xgboost	OR adasyn	Default	0.923188	0.999175	0.721739	0.846939	0.779343	0.818540	[[56832, 32], [15, 83]]
3	xgboost	PT origin	Default	0.908102	0.999561	0.919540	0.816327	0.864865	0.835073	[[56857, 7], [18, 80]]
4	xgboost	PT smote	Default	0.928290	0.999192	0.724138	0.857143	0.785047	0.826772	[[56832, 32], [14, 84]]
5	xgboost	PT adasyn	Default	0.933331	0.999087	0.685484	0.867347	0.765766	0.823643	[[56825, 39], [13, 85]]

Modelo catboost:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	catboost	OR origin	Default	0.903017	0.999579	0.940476	0.806122	0.868132	0.829832	[[56859, 5], [19, 79]]
1	catboost	OR smote	Default	0.933093	0.998613	0.562914	0.867347	0.682731	0.782689	[[56798, 66], [13, 85]]
2	catboost	OR adasyn	Default	0.922924	0.998648	0.572414	0.846939	0.683128	0.772812	[[56802, 62], [15, 83]]
3	catboost	PT origin	Default	0.903017	0.999579	0.940476	0.806122	0.868132	0.829832	[[56859, 5], [19, 79]]
4	catboost	PT smote	Default	0.912896	0.998964	0.658537	0.826531	0.733032	0.786408	[[56822, 42], [17, 81]]
5	catboost	PT adasyn	Default	0.917734	0.998455	0.532468	0.836735	0.650794	0.750916	[[56792, 72], [16, 82]]

Modelo decision_tree:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	decision_tree	OR origin	Default	0.872203	0.999070	0.722772	0.744898	0.733668	0.740365	[[56836, 28], [25, 73]]
1	decision_tree	OR smote	Default	0.881413	0.997121	0.347222	0.765306	0.477707	0.616776	[[56723, 141], [23, 75]]
2	decision_tree	OR adasyn	Default	0.850915	0.997244	0.350254	0.704082	0.467797	0.585739	[[56736, 128], [29, 69]]
3	decision_tree	PT origin	Default	0.882468	0.999228	0.781250	0.765306	0.773196	0.768443	[[56843, 21], [23, 75]]
4	decision_tree	PT smote	Default	0.907099	0.997560	0.398010	0.816327	0.535117	0.674536	[[56743, 121], [18, 80]]
5	decision_tree	PT adasyn	Default	0.871367	0.997402	0.372449	0.744898	0.496599	0.620748	[[56741, 123], [25, 73]]

Modelo random_forest:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	random_forest	OR origin	Default	0.903017	0.999579	0.940476	0.806122	0.868132	0.829832	[[56859, 5], [19, 79]]
1	random_forest	OR smote	Default	0.913160	0.999491	0.870968	0.826531	0.848168	0.835052	[[56852, 12], [17, 81]]
2	random_forest	OR adasyn	Default	0.902965	0.999473	0.877778	0.806122	0.840426	0.819502	[[56853, 11], [19, 79]]
3	random_forest	PT origin	Default	0.913221	0.999614	0.941860	0.826531	0.880435	0.847280	[[56859, 5], [17, 81]]
4	random_forest	PT smote	Default	0.913151	0.999473	0.861702	0.826531	0.843750	0.833333	[[56851, 13], [17, 81]]
5	random_forest	PT adasyn	Default	0.902947	0.999438	0.858696	0.806122	0.831579	0.816116	[[56851, 13], [19, 79]]

Modelo mlp:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	mlp	OR origin	Default	0.923241	0.999280	0.761468	0.846939	0.801932	0.828343	[[56838, 26], [15, 83]]
1	mlp	OR smote	Default	0.892637	0.999192	0.754902	0.785714	0.770000	0.779352	[[56839, 25], [21, 77]]
2	mlp	OR adasyn	Default	0.897520	0.998771	0.609375	0.795918	0.690265	0.750000	[[56814, 50], [20, 78]]
3	mlp	PT origin	Default	0.892760	0.999438	0.875000	0.785714	0.827957	0.802083	[[56853, 11], [21, 77]]
4	mlp	PT smote	Default	0.887553	0.999210	0.767677	0.775510	0.771574	0.773931	[[56841, 23], [22, 76]]
5	mlp	PT adasyn	Default	0.902894	0.999333	0.806122	0.806122	0.806122	0.806122	[[56845, 19], [19, 79]]

Modelo knn:

	Model	Description	Parameter	ROC-AUC	Accuracy	Precision	Recall	F1 Score	F2 Score	Confusion Matrix
0	knn	OR origin	Default	0.892787	0.999491	0.905882	0.785714	0.841530	0.807128	[[56856, 8], [21, 77]]
1	knn	OR smote	Default	0.937764	0.997770	0.427861	0.877551	0.575251	0.725126	[[56749, 115], [12, 86]]
2	knn	OR adasyn	Default	0.937764	0.997770	0.427861	0.877551	0.575251	0.725126	[[56749, 115], [12, 86]]
3	knn	PT origin	Default	0.887685	0.999473	0.904762	0.775510	0.835165	0.798319	[[56856, 8], [22, 76]]
4	knn	PT smote	Default	0.942919	0.997893	0.443878	0.887755	0.591837	0.739796	[[56755, 109], [11, 87]]
5	knn	PT adasyn	Default	0.942910	0.997876	0.441624	0.887755	0.589831	0.738540	[[56754, 110], [11, 87]]

In [ ]: