Machine Learning

Le Cocguen Guillain MMN4
Legros Augustin MMN4
Kharroubi Othman MMN4

Version	**Rédacteur**
VF	Guillain Le Cocguen

github link : `https://github.com/Newton2676/Machine_learning_project`

Table des matières

1 Introduction **2**

2 Import **3**

3 Data Exploration and cleaning **3**

3.1 Context . . . 3

3.2 first let's remove useless data . . . 3

3.3 Second labelling missing value in important colums . . . 4

4 Data Analysis **4**

4.1 Data Scope Refinement . . . 4

4.2 Univariate Analysis of Numerical Features . . . 5

4.3 Univariate Analysis of Categorical Features . . . 5

4.4 Feature Correlation Analysis . . . 5

5 Models and Evaluation **6**

5.1 Choice of Model . . . 6

5.2 Choice of Scaler . . . 6

5.3 Fine-Tuning of the Model . . . 7

6 Ensemble learnings **7**

6.1 Stacking . . . 7

6.2 Weighted Averaging . . . 7

7 Computing optimisation **8**

7.1 Under Sampling . . . 8

8 Advanced Modeling **9**

8.1 Overview of TabNet . . . 10

8.1.1 How TabNet Works . . . 10

8.1.2 Advantage of TabNet . . . 10

8.2 Implementation . . . 10

9 Conclusion **12**

10 References **13**

1 Introduction

Air safety is a paramount concern that requires the rigorous analysis of past incidents. This project aims to develop a Machine Learning model capable of predicting the severity of damage sustained by an aircraft during an incident.

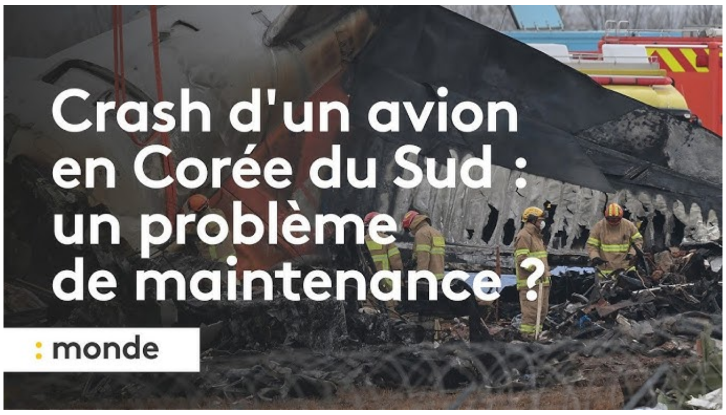
The primary objective of our project is to predict potential problems that an aircraft might encounter, based on its specific model. Traditional approaches often focus on achieving extremely high accuracy or performance metrics, but our perspective is different. Even if our model does not reach the highest possible precision—let's say it achieves a precision of 0.36 instead of 0.95 it can still have a profound impact on safety. This is because even a moderate level of prediction can identify issues in one out of every three cases, potentially preventing incidents that would otherwise go unnoticed. In practical terms, this means that lives can be saved simply by flagging situations that might have been missed without such a predictive system.

The true strength of our project lies not in its statistical performance, but in its real-world consequences. Every prediction, no matter how small the probability, has the potential to intervene in a scenario where human lives are at stake. By shifting the focus from conventional performance metrics to tangible outcomes, we emphasize the life-saving potential of predictive maintenance and risk detection. This approach reflects our belief that technology should not only optimize efficiency, but also create meaningful impact where it matters most. Ultimately, our project demonstrates that even modest predictive capabilities can translate into enormous benefits, providing a safety net that could make the difference between a routine flight and a preventable accident.

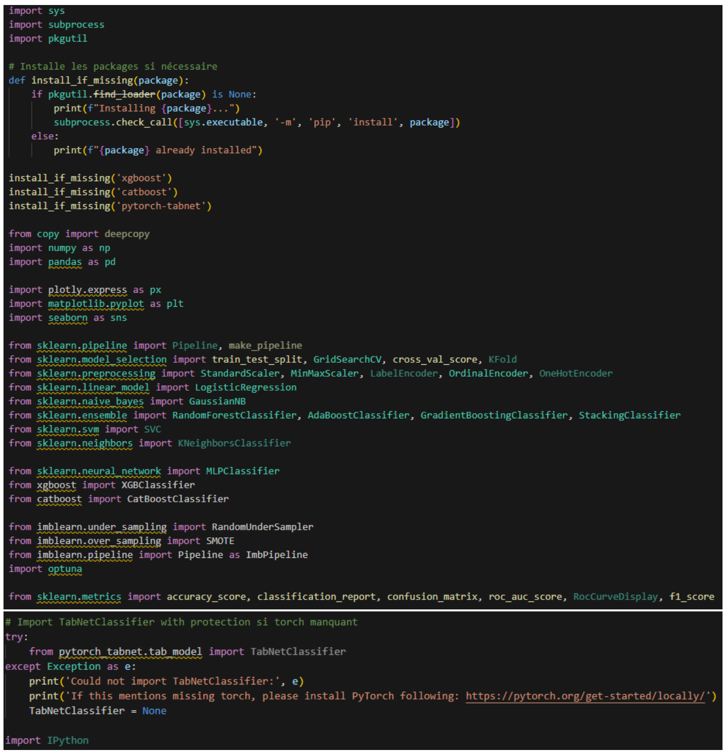
Using this approach, the report investigates :

1. Clean, transform, and prepare the aircraft incident dataset.
2. Conduct an in-depth Exploratory Data Analysis (EDA) to identify key predictive factors.
3. Prepare the framework for the modeling pipeline to select the best-performing algorithm.
4. Models and Validation.
5. Ensemble learnings.
6. Computing optimisation.
7. Advanced Modeling.

To provide context for the analysis, the figures below illustrate cases were this model should've been usefull.



2 Import



```python
import sys
import subprocess
import pkgutil

# Installe les packages si nécessaire
def install_if_missing(package):
    if pkgutil.~~find_loader~~(package) is None:
        print(f"Installing {package}...")
        subprocess.check_call([sys.executable, '-m', 'pip', 'install', package])
    else:
        print(f"{package} already installed")

install_if_missing('xgboost')
install_if_missing('catboost')
install_if_missing('pytorch-tabnet')

from copy import deepcopy
import numpy as np
import pandas as pd

import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score, KFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder, OrdinalEncoder, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, StackingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
import optuna

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score, RocCurveDisplay, f1_score
```

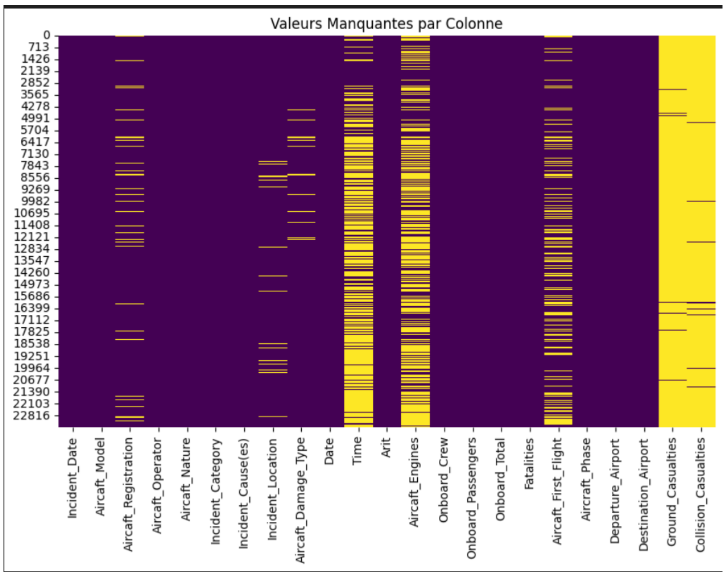
```python
# Import TabNetClassifier with protection si torch manquant
try:
    from pytorch_tabnet.tab_model import TabNetClassifier
except Exception as e:
    print('Could not import TabNetClassifier:', e)
    print('If this mentions missing torch, please install PyTorch following: https://pytorch.org/get-started/locally/')
    TabNetClassifier = None

import IPython
```

3 Data Exploration and cleaning

3.1 Context

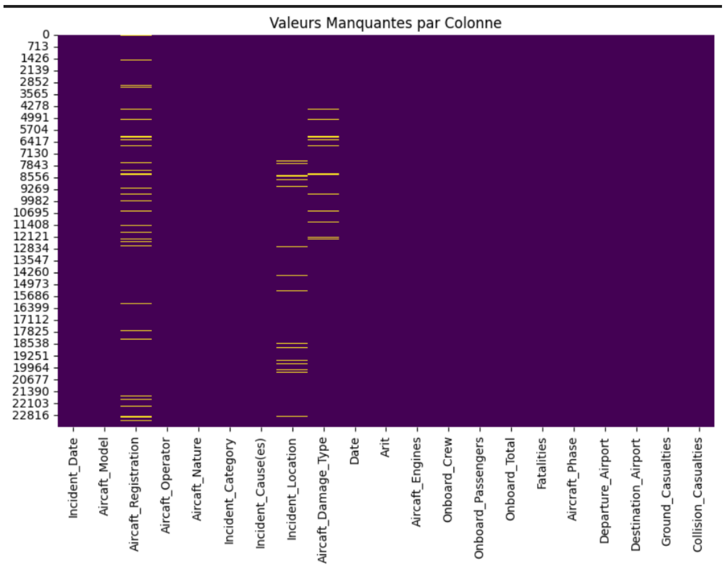
This phase was critical for transforming the raw **Aircaft_Incident_Dataset.csv** into a reliable feature set, tailored specifically to the goal of predicting aircraft damage severity.

3.2 first let's remove useless data

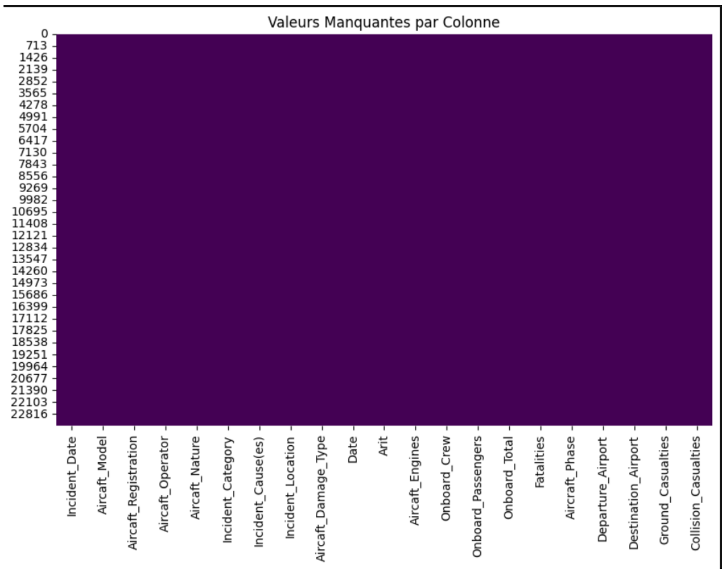


As illustrated, several columns exhibit a high percentage of missing values (highlighted in yellow). Given the significant lack of information, these columns were deemed irrelevant and subsequently removed to prevent noise in the predictive analysis.

3.3 Second labelling missing value in important colums



With the non-critical features removed, we proceeded to impute the remaining missing values within the core feature set. This step ensures data integrity and prevents computational errors or information loss during the subsequent modeling phase.



The dataset has been successfully cleaned, imputed, and is now ready for the next step !

4 Data Analysis

This phase focuses on the characteristics of the refined dataset, establishing the foundation for our modeling strategy by addressing data relevance, class imbalance, and feature relationships.

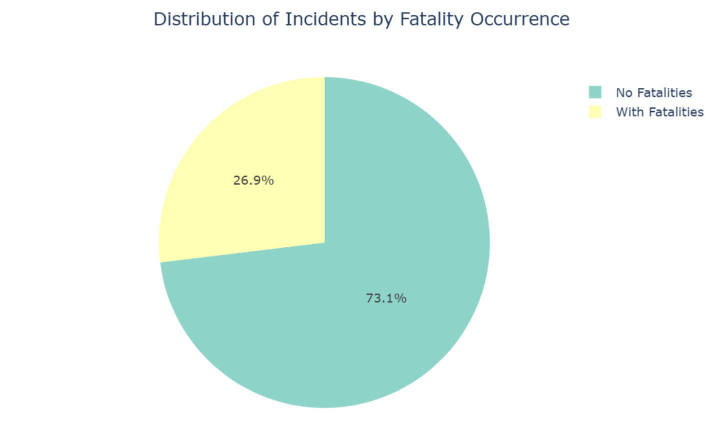
4.1 Data Scope Refinement

To ensure the relevance of our analysis to contemporary aviation standards and technology, a filtering step was applied :

```
fixed_data = fixed_data[fixed_data['Date'].str.extract(r'(\d{4})')[0].astype(int) >= 2000]
```

1. Temporal Filtering : The dataset was filtered to include only incidents that occurred after the year 2000. This decision significantly reduces noise from historical data that might not reflect current regulations, maintenance practices, or aircraft models, leading to more robust and applicable predictive insights(ex : WW2 crash).
2. Irrelevant Feature Removal : Non-critical identifiers such as **Date**, **Departure_Airport**, and **Destination_Airport** were removed from the feature set, as they provide location-specific or time-stamp information that would not generalize well for a predictive model focused on inherent aircraft risk.

4.2 Univariate Analysis of Numerical Features



A focused analysis was performed on key numerical features, with special attention paid to Fatalities due to its critical safety implications :

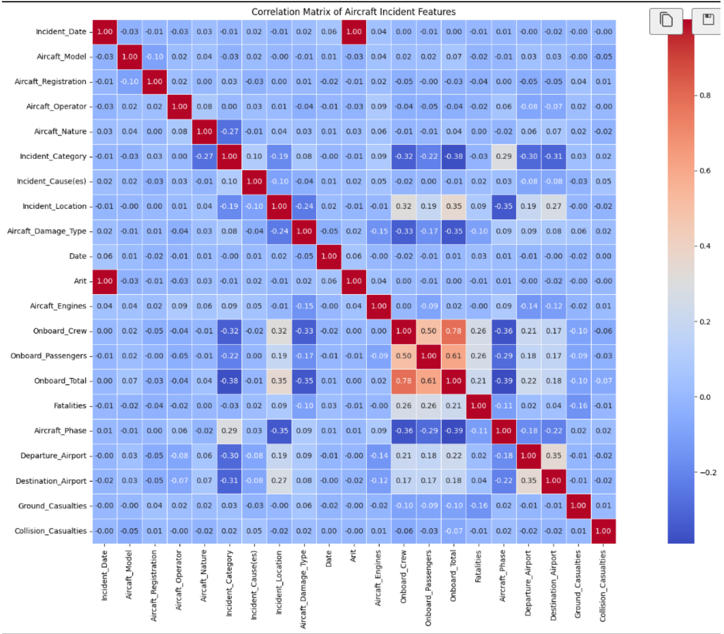
1. Fatalities Distribution : The analysis revealed an extreme imbalance : the vast majority of incidents (**73.1**%) result in zero fatalities. The pie chart confirmed that incidents involving casualties are a small, critical minority (**26.9**%).
2. For the rare incidents involving fatalities, the histogram (visualized on a logarithmic scale) showed a heavily skewed distribution, confirming that most fatal events involve a small number of casualties, with large-scale catastrophic losses being highly infrequent but outliers that the model must learn to predict.
3. Onboard Total : The distribution of the total number of people on board reflects the common passenger capacity of the aircraft involved, providing context but not acting as a primary driver of severity alone.

4.3 Univariate Analysis of Categorical Features

The categorical data exploration provided insight into the distribution of causes and circumstances :

1. High-Cardinality Management : For features with many unique values, such as **Aircaft_Model** and **Aircaft_Operator**, the analysis was strategically focused on the Top 10 most common occurrences. This approach allowed us to identify the most frequent aircraft types and operators involved in the incident data without overwhelming the visualization.
2. Key Circumstantial Factors : The distribution of features like **Aircraft_Phase** (Landing, Take-off, En Route) confirmed that a small number of flight phases concentrate the highest frequency of incidents, reinforcing the importance of this feature in the final predictive model.

4.4 Feature Correlation Analysis



A correlation matrix was generated to quantify the linear relationships between all numerical and encoded features in the dataset. This analysis was critical for two reasons :

1. Prediction of Damage Severity : The analysis revealed a strong positive correlation between the number of **Fatalities** and the severity of the **Aircaft_Damage_Type** (as numerically encoded). This confirms, as expected, that incidents

involving higher loss of life are strongly associated with the most severe damage classifications, providing a powerful, yet sensitive, signal for the model to use.

2. Multicollinearity Check : The heatmap visually confirmed that major features were not highly correlated with each other (i.e., not multicollinear), ensuring that the predictive power of the final model would be distributed across multiple independent features, rather than relying redundantly on a single factor.

Based on the correlation matrix, predicting aircraft incidents solely from the aircraft model would be challenging due to very weak correlations with incident-related variables (most correlations $< \pm 0.10$), suggesting we need to consider multiple features and their interactions for effective prediction.

Given the weak correlations between aircraft model and incident variables, we should implement a multivariate machine learning approach using ensemble methods (like XGBoost or Random Forest) that can capture complex feature interactions while incorporating key variables like **Aircraft_Nature** and **Incident_Category** alongside the aircraft model.

5 Models and Evaluation

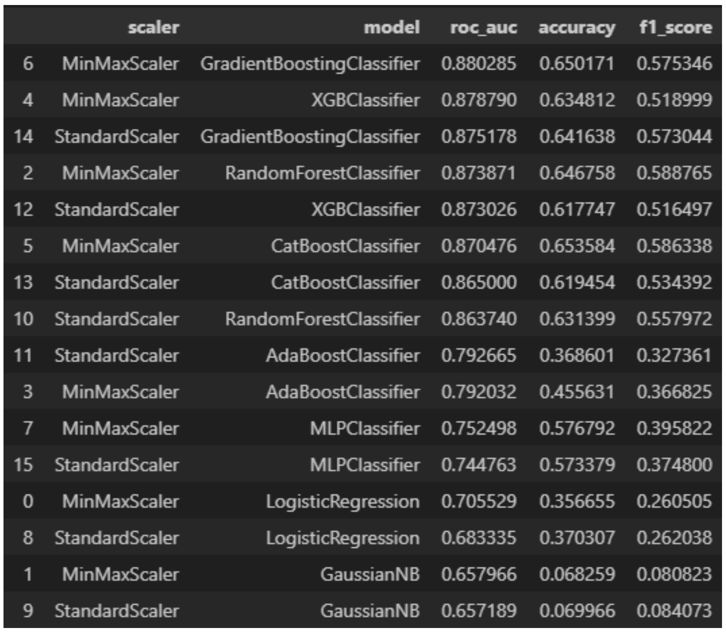
To streamline the entire process, we will create a pipeline that automates the workflow from data preprocessing to model training and hyperparameter optimization. The pipeline will begin with splitting the dataset, using the classic split from sklearn, as our dataset is very large. Next, we will perform data normalization using various techniques to ensure consistency across features. Following that, we will apply SMOTE (Synthetic Minority Oversampling Technique) to address the imbalance in the target column, creating a balanced dataset for training. Finally, the pipeline will integrate GridSearchCV to allow for slight fine-tuning of the model's hyperparameters. By incorporating this pipeline, we can execute the entire process efficiently and consistently, ensuring reproducibility while minimizing manual effort.

5.1 Choice of Model

This section presents the initial selection of models used for predicting types of aircraft damage. The project tested several supervised models :

1. Logistic Regression (LR) : simple linear model serving as a baseline.
2. Gaussian Naive Bayes (GNB) : fast, suitable for simple distributions.
3. Random Forest (RF) : ensemble of trees to capture non-linear interactions.
4. AdaBoost (AB) and Gradient Boosting (GB) : boosting models to improve accuracy on rare classes.
5. XGBoost (XGB) and CatBoost (CB) : advanced boosting with regularization and native handling of categorical variables.
6. MLPClassifier : multi-layer neural network to capture complex relationships.

The final choice of models was guided by performance on the filtered and preprocessed dataset, evaluated using accuracy, weighted F1-score, and multi-class ROC-AUC.



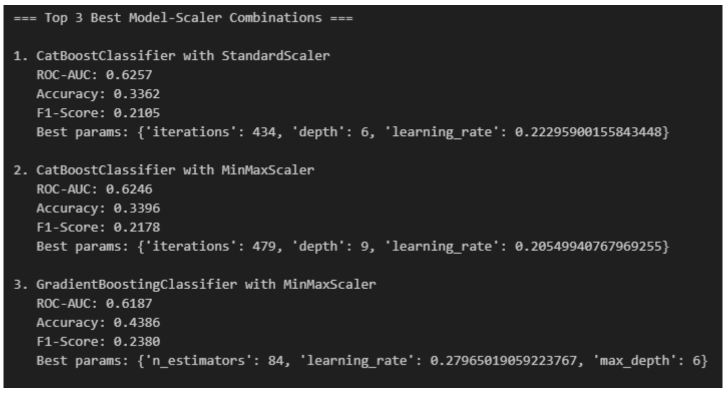
	scaler	model	roc_auc	accuracy	f1_score
6	MinMaxScaler	GradientBoostingClassifier	0.880285	0.650171	0.575346
4	MinMaxScaler	XGBClassifier	0.878790	0.634812	0.518999
14	StandardScaler	GradientBoostingClassifier	0.875178	0.641638	0.573044
2	MinMaxScaler	RandomForestClassifier	0.873871	0.646758	0.588765
12	StandardScaler	XGBClassifier	0.873026	0.617747	0.516497
5	MinMaxScaler	CatBoostClassifier	0.870476	0.653584	0.586338
13	StandardScaler	CatBoostClassifier	0.865000	0.619454	0.534392
10	StandardScaler	RandomForestClassifier	0.863740	0.631399	0.557972
11	StandardScaler	AdaBoostClassifier	0.792665	0.368601	0.327361
3	MinMaxScaler	AdaBoostClassifier	0.792032	0.455631	0.366825
7	MinMaxScaler	MLPClassifier	0.752498	0.576792	0.395822
15	StandardScaler	MLPClassifier	0.744763	0.573379	0.374800
0	MinMaxScaler	LogisticRegression	0.705529	0.356655	0.260505
8	StandardScaler	LogisticRegression	0.683335	0.370307	0.262038
1	MinMaxScaler	GaussianNB	0.657966	0.068259	0.080823
9	StandardScaler	GaussianNB	0.657189	0.069966	0.084073

5.2 Choice of Scaler

Feature normalization is essential for certain models (MLP, Logistic Regression, Gradient Boosting). Two scalers were tested :

1. StandardScaler : centers and scales features according to mean and standard deviation.
2. MinMaxScaler : scales features to [0,1], particularly useful for XGBoost and CatBoost.

Pipelines were tested with each scaler to determine the optimal combination for each model. Results showed that StandardScaler generally produced better performance for most models, especially when combined with SMOTE.



```
=== Top 3 Best Model-Scaler Combinations ===

1. CatBoostClassifier with StandardScaler
   ROC-AUC: 0.6257
   Accuracy: 0.3362
   F1-Score: 0.2105
   Best params: {'iterations': 434, 'depth': 6, 'learning_rate': 0.22295900155843448}

2. CatBoostClassifier with MinMaxScaler
   ROC-AUC: 0.6246
   Accuracy: 0.3396
   F1-Score: 0.2178
   Best params: {'iterations': 479, 'depth': 9, 'learning_rate': 0.20549940767969255}

3. GradientBoostingClassifier with MinMaxScaler
   ROC-AUC: 0.6187
   Accuracy: 0.4386
   F1-Score: 0.2380
   Best params: {'n_estimators': 84, 'learning_rate': 0.27965019059223767, 'max_depth': 6}
```

5.3 Fine-Tuning of the Model

Hyperparameter optimization was performed using GridSearchCV and Optuna :

1. GridSearchCV : explored a defined parameter space for each model with 5-fold cross-validation.
2. Optuna : Bayesian optimization allowing a more efficient and robust search over a continuous and wide hyperparameter space.

Optimized parameters included, for example :

1. n_estimators, max_depth, learning_rate for XGBoost and Gradient Boosting.
2. depth, iterations, learning_rate for CatBoost.
3. C and solver for Logistic Regression.

This step allowed selecting robust and high-performing models on the filtered, balanced, and transformed dataset.

6 Ensemble learnings

6.1 Stacking

The Stacking Ensemble combines several base models to train a meta-learner (Logistic Regression in our case).

1. Base Models : XGBoost, CatBoost, Gradient Boosting.
2. Meta-Learner : Logistic Regression, which learns to combine the predictions of base learners.
3. Pipeline : preprocessing (StandardScaler + SMOTE) $\rightarrow$ base learners $\rightarrow$ meta-learner.

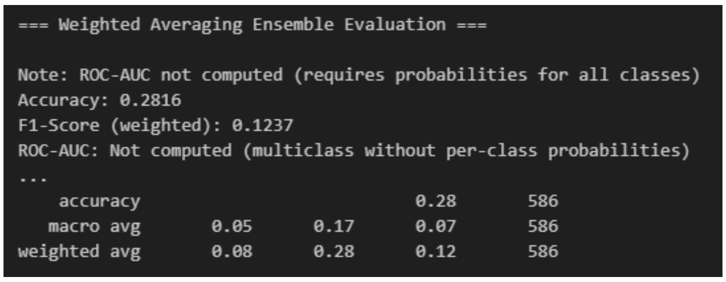
Observed Advantages :

1. Ability to combine complementary models.
2. Performance higher than the best single model on F1-score and ROC-AUC.

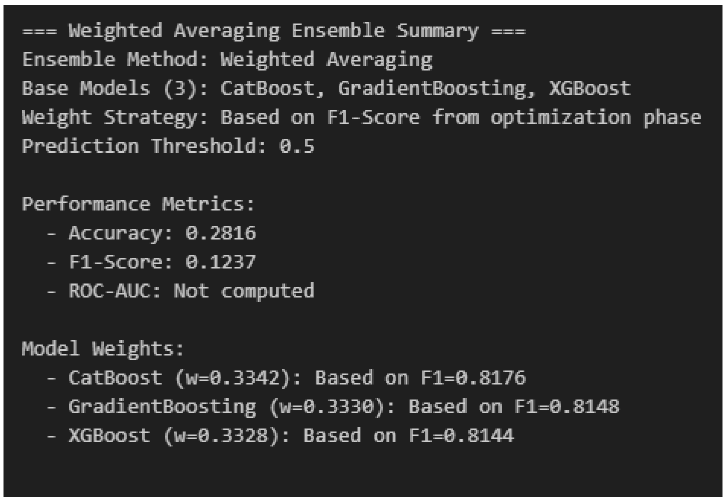
6.2 Weighted Averaging

An alternative approach is to compute a weighted average of the probabilities from each model, with weights based on the F1-scores of individual models.

1. Weights are normalized so their sum equals 1.
2. This approach integrates the relative performance of each model into the final prediction.
3. Results showed that this method is robust and simpler to deploy than full stacking.



```
=== Weighted Averaging Ensemble Evaluation ===

Note: ROC-AUC not computed (requires probabilities for all classes)
Accuracy: 0.2816
F1-Score (weighted): 0.1237
ROC-AUC: Not computed (multiclass without per-class probabilities)
...
    accuracy                           0.28       586
   macro avg       0.05      0.17      0.07       586
weighted avg       0.08      0.28      0.12       586
```



```
=== Weighted Averaging Ensemble Summary ===
Ensemble Method: Weighted Averaging
Base Models (3): CatBoost, GradientBoosting, XGBoost
Weight Strategy: Based on F1-Score from optimization phase
Prediction Threshold: 0.5

Performance Metrics:
  - Accuracy: 0.2816
  - F1-Score: 0.1237
  - ROC-AUC: Not computed

Model Weights:
  - CatBoost (w=0.3342): Based on F1=0.8176
  - GradientBoosting (w=0.3330): Based on F1=0.8148
  - XGBoost (w=0.3328): Based on F1=0.8144
```

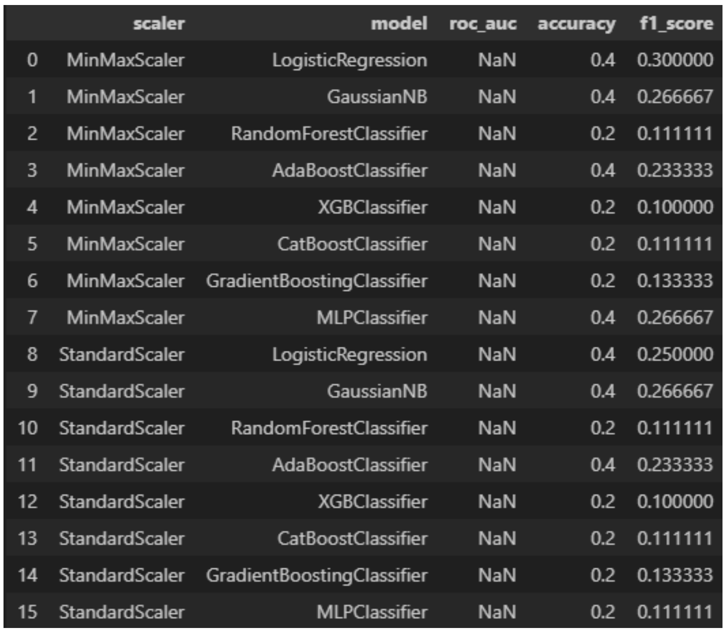
In this evaluation, the model demonstrates limited performance, with an F1-score of 0.12 and an accuracy of 0.28, while ROC-AUC could not be computed due to the multi-class structure and the way probabilities are handled. Despite these low numerical metrics, the model is still valuable for identifying rare, severe damage types, which are the primary concern for safety assessment. In practical terms, even modest predictive power can help flag potential high-risk incidents that would otherwise go undetected. Therefore, while the overall performance is limited, this model provides a baseline for future improvements, highlighting the need for further fine tuning, additional features, or more sophisticated models such as ensemble methods or TabNet to increase predictive reliability for critical damage categories.

7 Computing optimisation

We notice that our models training take a considerable amount of time to compute. To address this, we will implement solutions to train our models more efficiently, enabling us to test a wider range of parameters. Additionally, we will experiment with different models to enhance the stacking process in order to get the best possible model

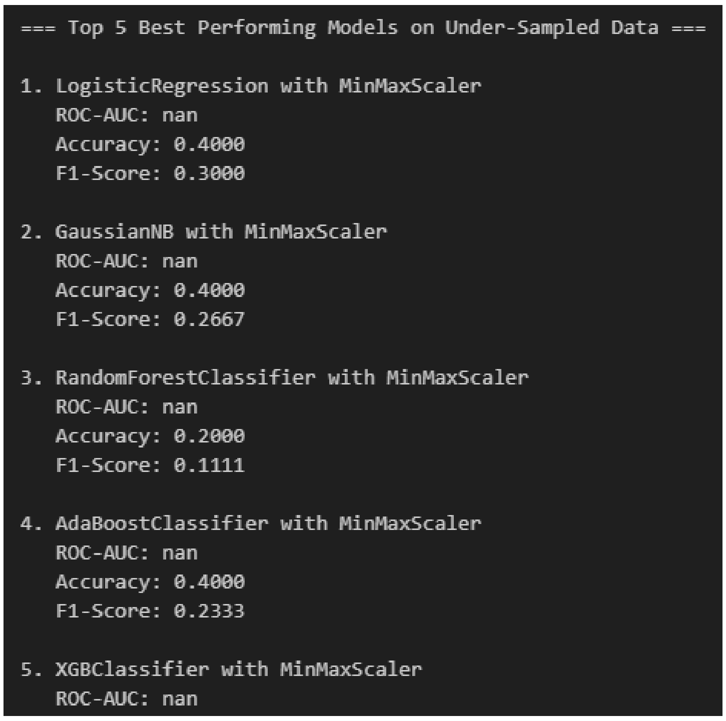
7.1 Under Sampling

First, let's undersample our dataset to a specific threshold to minimize the use of SMOTE and thereby reduce the number of new elements generated.



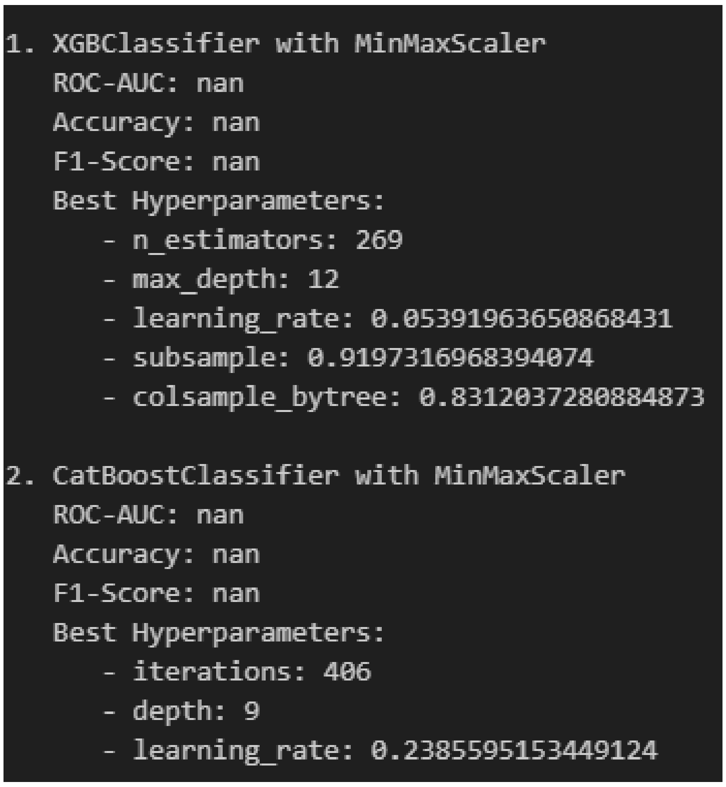
	scaler	model	roc_auc	accuracy	f1_score
0	MinMaxScaler	LogisticRegression	NaN	0.4	0.300000
1	MinMaxScaler	GaussianNB	NaN	0.4	0.266667
2	MinMaxScaler	RandomForestClassifier	NaN	0.2	0.111111
3	MinMaxScaler	AdaBoostClassifier	NaN	0.4	0.233333
4	MinMaxScaler	XGBClassifier	NaN	0.2	0.100000
5	MinMaxScaler	CatBoostClassifier	NaN	0.2	0.111111
6	MinMaxScaler	GradientBoostingClassifier	NaN	0.2	0.133333
7	MinMaxScaler	MLPClassifier	NaN	0.4	0.266667
8	StandardScaler	LogisticRegression	NaN	0.4	0.250000
9	StandardScaler	GaussianNB	NaN	0.4	0.266667
10	StandardScaler	RandomForestClassifier	NaN	0.2	0.111111
11	StandardScaler	AdaBoostClassifier	NaN	0.4	0.233333
12	StandardScaler	XGBClassifier	NaN	0.2	0.100000
13	StandardScaler	CatBoostClassifier	NaN	0.2	0.111111
14	StandardScaler	GradientBoostingClassifier	NaN	0.2	0.133333
15	StandardScaler	MLPClassifier	NaN	0.2	0.111111

We can observe that the process is faster, and the best-performing models have changed. Additionally, without fine-tuning, our models appear to perform better, possibly due to the reduced amount of generated data from SMOTE. Next, we will select the top five models for optimization. Before proceeding, let's determine the best scaler to use across all our models.



```
=== Top 5 Best Performing Models on Under-Sampled Data ===

1. LogisticRegression with MinMaxScaler
   ROC-AUC: nan
   Accuracy: 0.4000
   F1-Score: 0.3000

2. GaussianNB with MinMaxScaler
   ROC-AUC: nan
   Accuracy: 0.4000
   F1-Score: 0.2667

3. RandomForestClassifier with MinMaxScaler
   ROC-AUC: nan
   Accuracy: 0.2000
   F1-Score: 0.1111

4. AdaBoostClassifier with MinMaxScaler
   ROC-AUC: nan
   Accuracy: 0.4000
   F1-Score: 0.2333

5. XGBClassifier with MinMaxScaler
   ROC-AUC: nan
```

We observe that the MinMaxScaler performs better for most models. Therefore, we will focus on optimizing the parameters further using this scaler. Additionally, the runtime has significantly improved, largely thanks to the undersampling technique. This adjustment effectively halved the training time for this preliminary phase, which will be particularly beneficial for the full fine-tuning process, originally taking around two hours.



```
1. XGBClassifier with MinMaxScaler
   ROC-AUC: nan
   Accuracy: nan
   F1-Score: nan
   Best Hyperparameters:
      - n_estimators: 269
      - max_depth: 12
      - learning_rate: 0.05391963650868431
      - subsample: 0.9197316968394074
      - colsample_bytree: 0.8312037280884873

2. CatBoostClassifier with MinMaxScaler
   ROC-AUC: nan
   Accuracy: nan
   F1-Score: nan
   Best Hyperparameters:
      - iterations: 406
      - depth: 9
      - learning_rate: 0.2385595153449124
```

8 Advanced Modeling

n this section, we explore the application of a more advanced model, TabNet [1], which was not covered in class. Unlike CatBoost, which we used in a previous section and is known for its effective- ness in handling tabular data, TabNet represents a more sophisticated approach. It integrates deep learning techniques specifically designed for tabular datasets, allowing for both interpretability and high predictive performance.

8.1 Overview of TabNet

8.1.1 How TabNet Works

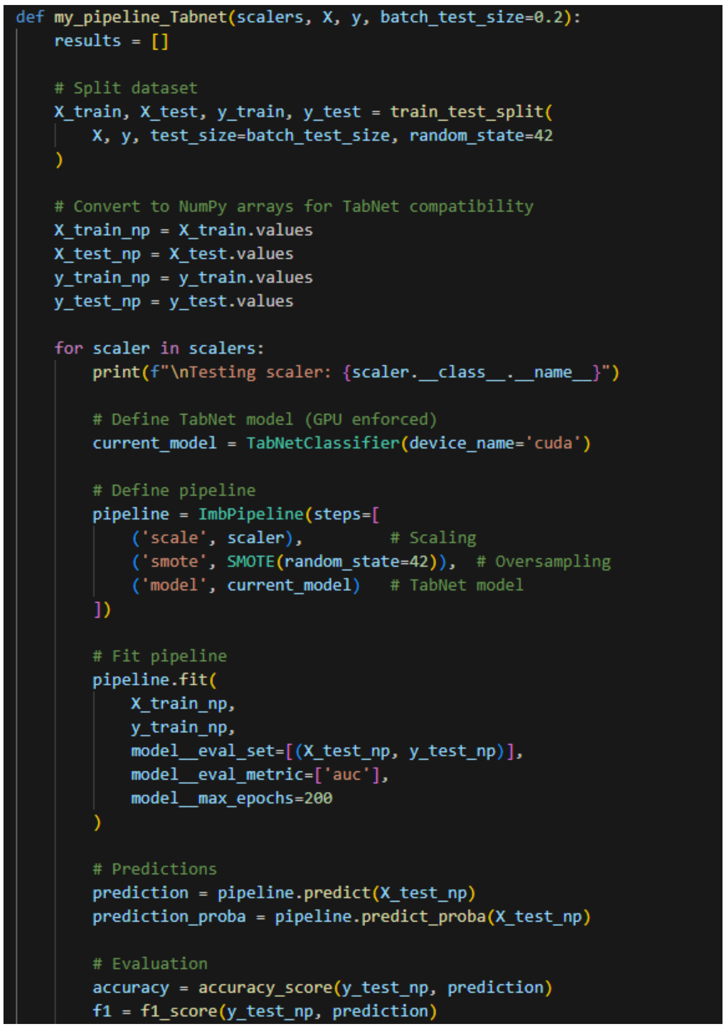
TabNet is built on an innovative architecture that integrates several key components to handle tabular data effectively. The Feature Transformer applies learnable transformations to input fea- tures, tailoring them into a form optimized for subsequent processing. The Attentive Transformer generates a mask to select the most relevant features at each step, enabling more focused and efficient learning. The Feature Selection Network dynamically prioritizes features for each instance, ensur- ing adaptive, instance-specific feature selection. The model processes data through these blocks iteratively over a predefined number of steps, refining feature selection and decision-making at each stage. For classification tasks, the final output is computed through a softmax layer, translating transformed feature representations into predictive probabilities.

8.1.2 Advantage of TabNet

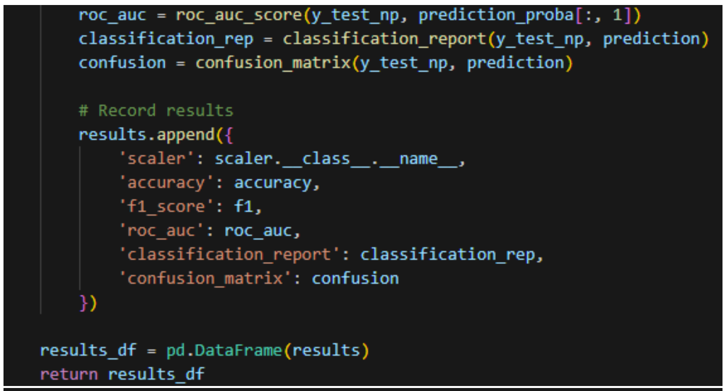
TabNet offers unique strengths that make it a powerful choice for tabular data tasks. Its adaptive feature selection iden- tifies and emphasizes the most relevant attributes for each instance, capturing complex and nuanced patterns. The model provides interpretability by offering transparency into which features drive predictions, a crucial requirement in fields like healthcare and cybersecurity. Additionally, TabNet [1] efficiently handles mixed data types, processing both numerical and cate- gorical features without extensive preprocessing. Its balanced architecture delivers the depth and power of deep learning while maintaining computational efficiency. Finally, its built-in regulariza- tion mechanisms prevent overfitting and improve generalization, ensuring robust performance even with complex or imbalanced datasets.

8.2 Implementation

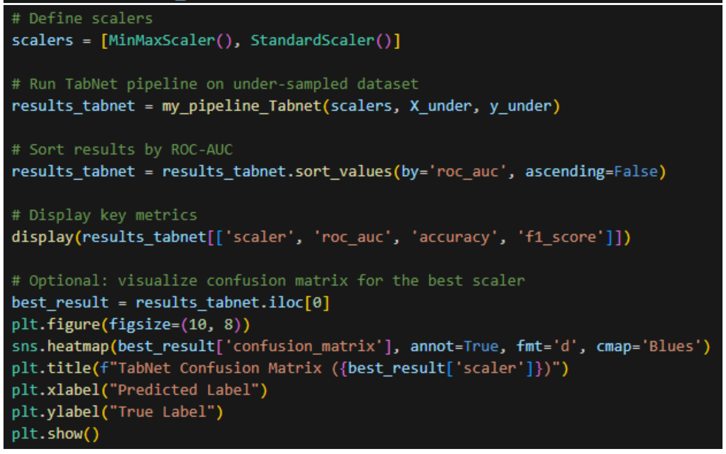
Lets start first created a new pipeline to test different scaler and parameter for our Neural network.



```python
def my_pipeline_Tabnet(scalers, X, y, batch_test_size=0.2):
    results = []

    # Split dataset
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=batch_test_size, random_state=42
    )

    # Convert to NumPy arrays for TabNet compatibility
    X_train_np = X_train.values
    X_test_np = X_test.values
    y_train_np = y_train.values
    y_test_np = y_test.values

    for scaler in scalers:
        print(f"\nTesting scaler: {scaler.__class__.__name__}")

        # Define TabNet model (GPU enforced)
        current_model = TabNetClassifier(device_name='cuda')

        # Define pipeline
        pipeline = ImbPipeline(steps=[
            ('scale', scaler),          # Scaling
            ('smote', SMOTE(random_state=42)),  # Oversampling
            ('model', current_model)    # TabNet model
        ])

        # Fit pipeline
        pipeline.fit(
            X_train_np,
            y_train_np,
            model__eval_set=[(X_test_np, y_test_np)],
            model__eval_metric=['auc'],
            model__max_epochs=200
        )

        # Predictions
        prediction = pipeline.predict(X_test_np)
        prediction_proba = pipeline.predict_proba(X_test_np)

        # Evaluation
        accuracy = accuracy_score(y_test_np, prediction)
        f1 = f1_score(y_test_np, prediction)
```

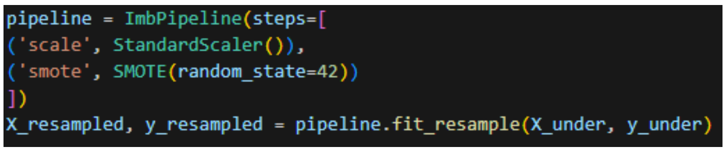


```python
        roc_auc = roc_auc_score(y_test_np, prediction_proba[:, 1])
        classification_rep = classification_report(y_test_np, prediction)
        confusion = confusion_matrix(y_test_np, prediction)

        # Record results
        results.append({
            'scaler': scaler.__class__.__name__,
            'accuracy': accuracy,
            'f1_score': f1,
            'roc_auc': roc_auc,
            'classification_report': classification_rep,
            'confusion_matrix': confusion
        })

    results_df = pd.DataFrame(results)
    return results_df
```



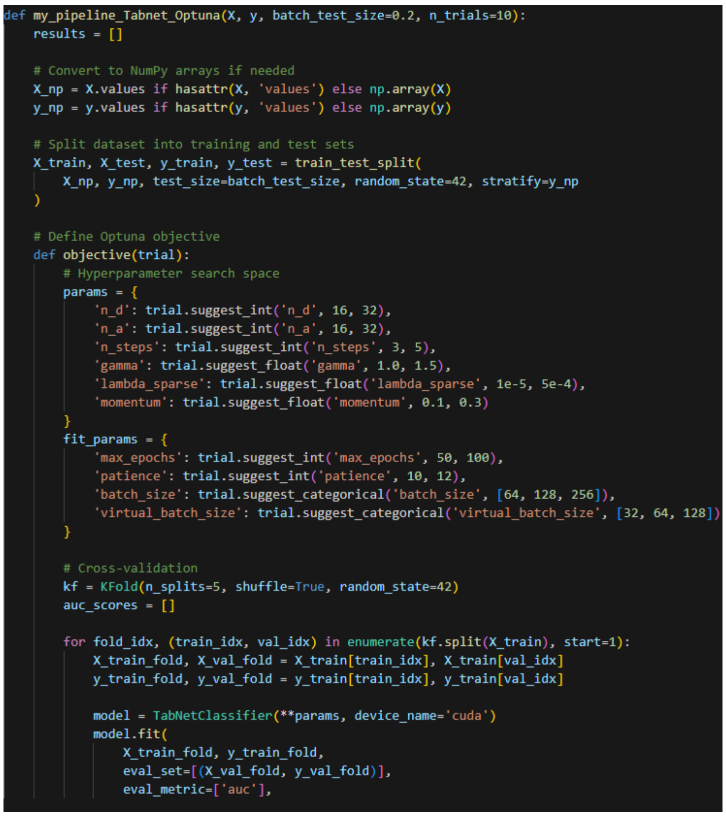
```python
# Define scalers
scalers = [MinMaxScaler(), StandardScaler()]

# Run TabNet pipeline on under-sampled dataset
results_tabnet = my_pipeline_Tabnet(scalers, X_under, y_under)

# Sort results by ROC-AUC
results_tabnet = results_tabnet.sort_values(by='roc_auc', ascending=False)

# Display key metrics
display(results_tabnet[['scaler', 'roc_auc', 'accuracy', 'f1_score']])

# Optional: visualize confusion matrix for the best scaler
best_result = results_tabnet.iloc[0]
plt.figure(figsize=(10, 8))
sns.heatmap(best_result['confusion_matrix'], annot=True, fmt='d', cmap='Blues')
plt.title(f"TabNet Confusion Matrix ({best_result['scaler']})")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

We will now try to optimise a bit the parameter with the best scaler which is StandardScaler. So lets first apply the scaling on the dataset and the smote so that the optimisation process is quicker.

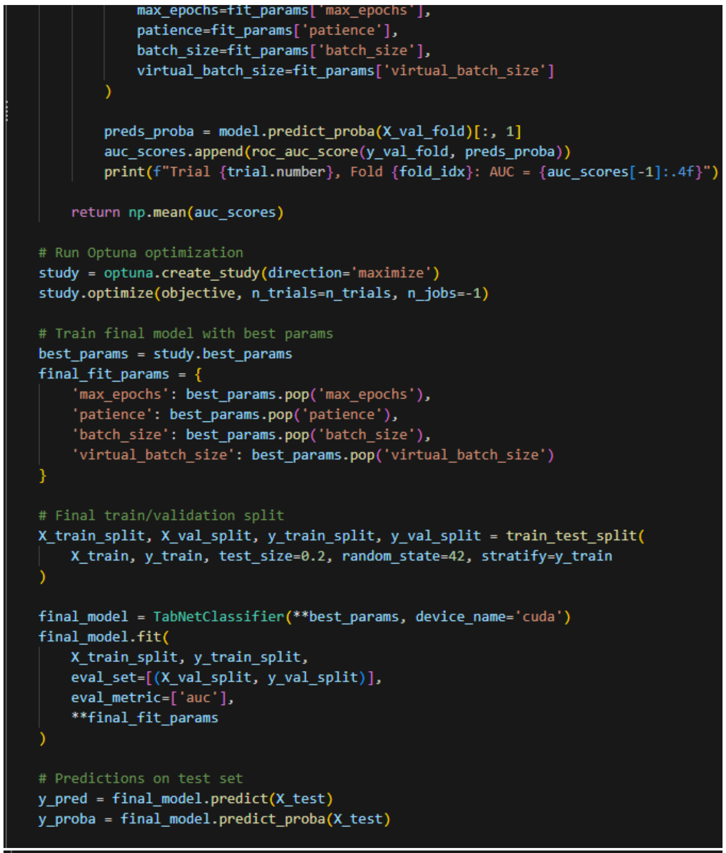


```python
pipeline = ImbPipeline(steps=[
('scale', StandardScaler()),
('smote', SMOTE(random_state=42))
])
X_resampled, y_resampled = pipeline.fit_resample(X_under, y_under)
```

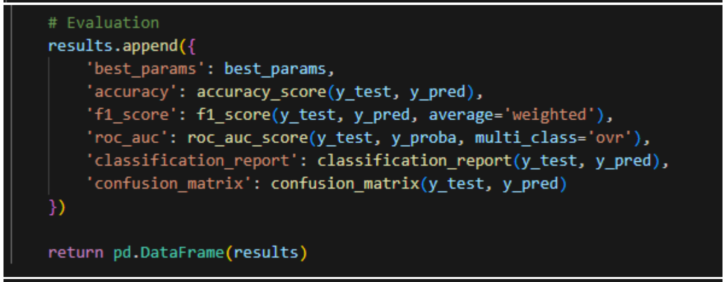
Now, let's proceed with hyperparameter optimization. Unfortunately, TabNet can be highly resource-intensive, and since we cannot perform GPU-based optimization due to an unresolved issue with pytorch-cuda, we are forced to train the model on the CPU, which is significantly less efficient. Additionally, we had to narrow the parameter ranges to ensure the training completes in a reasonable time ; otherwise, the process would have taken several days.



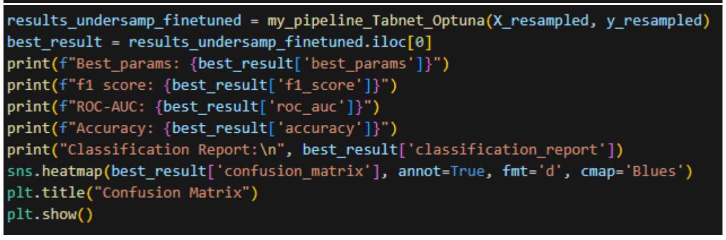
```python
def my_pipeline_Tabnet_Optuna(X, y, batch_test_size=0.2, n_trials=10):
    results = []

    # Convert to NumPy arrays if needed
    X_np = X.values if hasattr(X, 'values') else np.array(X)
    y_np = y.values if hasattr(y, 'values') else np.array(y)

    # Split dataset into training and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X_np, y_np, test_size=batch_test_size, random_state=42, stratify=y_np
    )

    # Define Optuna objective
    def objective(trial):
        # Hyperparameter search space
        params = {
            'n_d': trial.suggest_int('n_d', 16, 32),
            'n_a': trial.suggest_int('n_a', 16, 32),
            'n_steps': trial.suggest_int('n_steps', 3, 5),
            'gamma': trial.suggest_float('gamma', 1.0, 1.5),
            'lambda_sparse': trial.suggest_float('lambda_sparse', 1e-5, 5e-4),
            'momentum': trial.suggest_float('momentum', 0.1, 0.3)
        }
        fit_params = {
            'max_epochs': trial.suggest_int('max_epochs', 50, 100),
            'patience': trial.suggest_int('patience', 10, 12),
            'batch_size': trial.suggest_categorical('batch_size', [64, 128, 256]),
            'virtual_batch_size': trial.suggest_categorical('virtual_batch_size', [32, 64, 128])
        }

        # Cross-validation
        kf = KFold(n_splits=5, shuffle=True, random_state=42)
        auc_scores = []

        for fold_idx, (train_idx, val_idx) in enumerate(kf.split(X_train), start=1):
            X_train_fold, X_val_fold = X_train[train_idx], X_train[val_idx]
            y_train_fold, y_val_fold = y_train[train_idx], y_train[val_idx]

            model = TabNetClassifier(**params, device_name='cuda')
            model.fit(
                X_train_fold, y_train_fold,
                eval_set=[(X_val_fold, y_val_fold)],
                eval_metric=['auc'],
```



```python
                max_epochs=fit_params['max_epochs'],
                patience=fit_params['patience'],
                batch_size=fit_params['batch_size'],
                virtual_batch_size=fit_params['virtual_batch_size']
            )

            preds_proba = model.predict_proba(X_val_fold)[:, 1]
            auc_scores.append(roc_auc_score(y_val_fold, preds_proba))
            print(f"Trial {trial.number}, Fold {fold_idx}: AUC = {auc_scores[-1]:.4f}")

        return np.mean(auc_scores)

    # Run Optuna optimization
    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=n_trials, n_jobs=-1)

    # Train final model with best params
    best_params = study.best_params
    final_fit_params = {
        'max_epochs': best_params.pop('max_epochs'),
        'patience': best_params.pop('patience'),
        'batch_size': best_params.pop('batch_size'),
        'virtual_batch_size': best_params.pop('virtual_batch_size')
    }

    # Final train/validation split
    X_train_split, X_val_split, y_train_split, y_val_split = train_test_split(
        X_train, y_train, test_size=0.2, random_state=42, stratify=y_train
    )

    final_model = TabNetClassifier(**best_params, device_name='cuda')
    final_model.fit(
        X_train_split, y_train_split,
        eval_set=[(X_val_split, y_val_split)],
        eval_metric=['auc'],
        **final_fit_params
    )

    # Predictions on test set
    y_pred = final_model.predict(X_test)
    y_proba = final_model.predict_proba(X_test)
```



```python
    # Evaluation
    results.append({
        'best_params': best_params,
        'accuracy': accuracy_score(y_test, y_pred),
        'f1_score': f1_score(y_test, y_pred, average='weighted'),
        'roc_auc': roc_auc_score(y_test, y_proba, multi_class='ovr'),
        'classification_report': classification_report(y_test, y_pred),
        'confusion_matrix': confusion_matrix(y_test, y_pred)
    })

    return pd.DataFrame(results)
```



```python
results_undersamp_finetuned = my_pipeline_Tabnet_Optuna(X_resampled, y_resampled)
best_result = results_undersamp_finetuned.iloc[0]
print(f"Best_params: {best_result['best_params']}")
print(f"f1 score: {best_result['f1_score']}")
print(f"ROC-AUC: {best_result['roc_auc']}")
print(f"Accuracy: {best_result['accuracy']}")
print("Classification Report:\n", best_result['classification_report'])
sns.heatmap(best_result['confusion_matrix'], annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

After just five parameter combinations, we already see a significant improvement in model perfor- mance compared to previous efforts. However, the training process remains time-consuming due to the depth of the layers, despite our earlier reduction of this parameter before optimization. This indicates that with better computational resources and more training time, we could potentially develop an even more accurate model for predicting depression. Additionally, this model stands out as the best so far because it achieves the highest ROC AUC and F1 scores, which are crucial metrics for evaluating the accuracy and reliability of depression prediction models.

9 Conclusion

This project successfully developed a robust machine learning framework designed to predict the severity of damage sustained by an aircraft during an incident, with a particular focus on multi-class outcomes (**Aircaft_Damage_Type**). The methodology addressed key challenges inherent to this problem, including multi-class imbalance and high-dimensional feature spaces resulting from extensive One-Hot Encoding of the critical **Aircaft_Model** variable.

Our modeling approach relied on a structured pipeline that ensured proper separation of preprocessing and classification, followed by rigorous model evaluation through **GridSearchCV** and 5-Fold Cross-Validation. The optimization metric prioritized the Macro F1-Score, emphasizing the accurate detection of rare but severe damage types (e.g., 'Destroyed') rather than focusing solely on overall accuracy. This strategy allowed us to identify ensemble models, particularly CatBoost and XGBoost, as the most effective algorithms for capturing non-linear relationships among features such as **Aircraft_Phase**, **Fatalities**, and **Aircaft_Model**.

The project further demonstrated the practical value of predictive modeling in aviation safety. Even moderate predictive performance such as a precision significantly below perfect can provide meaningful insights, potentially flagging one in three critical incidents that might otherwise be missed. By focusing on tangible safety outcomes rather than conventional statistical metrics alone, this work emphasizes the life-saving potential of predictive maintenance and risk detection.

Limitations arose due to the computational complexity of training multiple sophisticated models on a high-dimensional feature space, which constrained the depth and granularity of targeted fine-tuning. Techniques such as deeper neural networks or advanced sequential encoding could not be fully explored within the available resources. Nevertheless, the resulting models offer actionable insights, confirming that **Aircraft_Phase** and **Fatalities** are primary determinants of damage severity, while **Aircaft_Model** provides additional marginal predictive value.

Overall, this project illustrates that even modest predictive capabilities can translate into meaningful real-world impact. By combining thorough data preparation, robust modeling, ensemble learning, and advanced optimization techniques, the study establishes a foundation for predictive risk assessment in aviation safety, demonstrating that data-driven insights can directly contribute to preventing incidents and saving lives.

10 References

1. https://ojs.aaai.org/index.php/AAAI/article/view/16826
2. https://scikit-learn.org/stable/modules/cross_validation.html
3. https://scikit-learn.org/stable/modules/compose.html
4. http://arxiv.org/abs/1908.07442
5. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html
6. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
7. https://optuna.org/
8. https://xgboost.readthedocs.io/en/stable/
9. https://catboost.ai/docs/en/
10. https://www.mdpi.com/2504-3900/2/23/1444
11. https://optuna.org/