Machine Learning Task Documentation

Predictive Incident Management
Submitted by: Eingeline Nichole F. Roque - BIA Intern

This document outlines the end-to-end process of building a machine learning model to predict whether an incident will remain **open** or be **closed**, based on information available at the time the incident is created.

Prerequisites:

- Completed Nexus Academy Python Class
- Jupyter Lab/ Google Colab

Introduction and Project Goal

Goal: To develop a robust classification model that can predict the status ('state') of an incident (binary: 0 for Open, 1 for Closed) at the moment it is logged. This proactive prediction can help prioritize resources, alert teams to potentially "sticky" incidents, and improve service delivery.

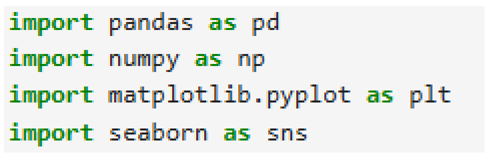
Machine Learning Lifecycle: The process follows a standard machine learning workflow:

1. Data Collection
2. Handling Class Imbalance
3. Data Cleaning & Preprocessing
4. Feature Engineering
5. Data Splitting
6. Model Selection
7. Hyperparameter Tuning
8. Model Evaluation
9. Model Deployment

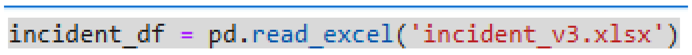
---

1. Data Collection

The first step is to gather and load your data in your ipython notebook. In my case, the dataset is already provided ('incident_v3.xlsx'). Import the necessary python libraries then load your dataset into the DataFrame using Pandas.



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

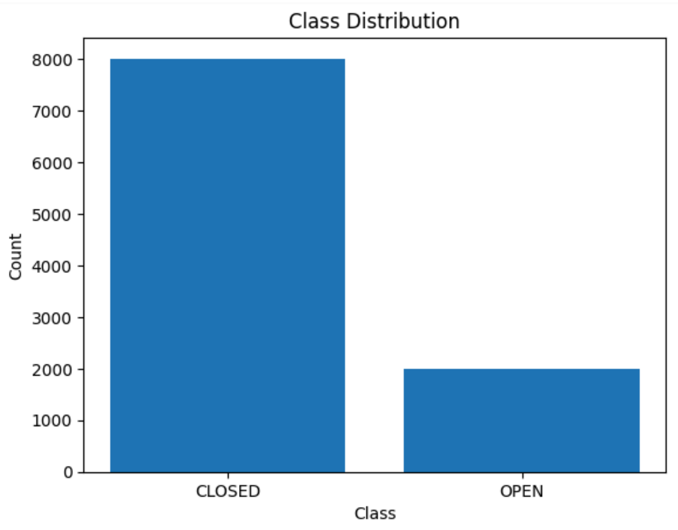


```python
incident_df = pd.read_excel('incident_v3.xlsx')
```

2. Data Resampling and Cleaning

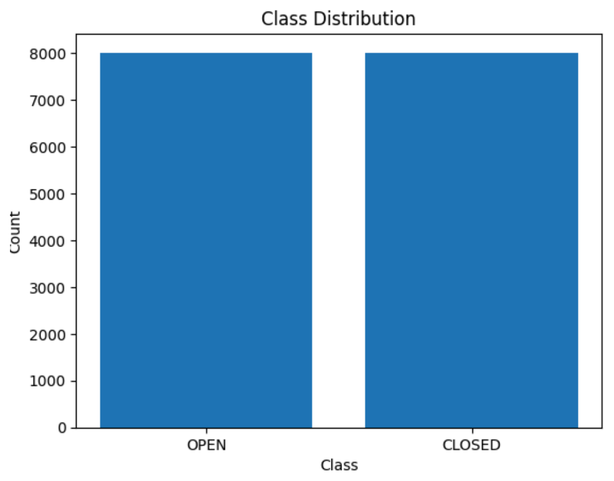
2.1 Data Resampling

Before processing the dataset, it is important to see the structure of it first. I noticed that there is an imbalance between the closed incidents (majority class) and the open incidents (minority class). This would cause a high bias result on the evaluation metrics if it was overlooked.



So to address this issue, we applied a resampling technique (RandomOverSampler) to address this issue. Its goal is to create random samples of the minority class to equal the number of samples in the majority class, giving equal understanding of their state.

```
# RandomOverSampler - a class to over-sample the minority class(es) by picking samples at random with replacement
ros = RandomOverSampler(sampling_strategy={target_state_open: target_size_closed}, random_state=42)
```



After RandomOverSampler

As you can see, we now have solved the imbalance issue of our dataset, and can now proceed with cleaning and preprocessing it.

2.2 Data Cleaning

Data Leakage Prevention:

- closed_at: We should NOT fill NaN values in closed_at. The presence of a NaN in closed_at is precisely what defines an 'Open' incident. Filling it would directly leak the target information.
- close_code: This column is also a direct indicator of closure. It is only populated when an incident is closed. Therefore, close_code MUST BE DROPPED from the features to prevent data leakage.

2.3 Conversion of ‘state’ or the dependent variable to binary

```
df_balanced['is_closed'] = (df_balanced['closed_at'].notnull()).astype(int)
```

3. Feature Engineering

This involves creating new features from existing ones to provide more predictive power to the model.

Libraries used:

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
```

3.1 Numerical Features

These features are derived solely from opened_at to ensure they are available at the time of prediction (incident creation) and prevent data leakage.

```
numerical_features = ['priority_encoded',
                      'severity_encoded',
                      'day_of_week', 'is_weekend',
                      'day_of_month', 'week_of_year',
                      'quarter_of_year',
                     'incident_age_at_snapshot_seconds',
                     'priority_severity_interaction']
```

3.2 Categorical Features

Two algorithms were used: OrdinalEncoder (for columns such as priority and severity to retain their weight), and OneHotEncoder for other categorical variables that are multicollinear.

```
categorical_cols = [
    'category',
    'subcategory',
    'assignment_group',
    'assigned_to',
    'cmdb_ci',
    'location',
]
```

The ColumnTransformer is used to apply different preprocessing steps to different columns (e.g., one-hot encoding for nominal categoricals, ordinal encoding for ordered categoricals, scaling for numericals).

```
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(handle_unknown='ignore',
sparse_output=True), categorical_cols),
        ('numerical_scaler', MinMaxScaler(), numerical_features)
    ],
    remainder='drop'
)
```

After creating the needed features, drop all the columns that are irrelevant and already extracted.

Columns_to_drop = [`'opened_at', 'closed_at', 'state', 'close_code', 'priority', 'severity'`]

4. Data Splitting and Training

```
from sklearn.model_selection import train_test_split
from collections import Counter
```

We followed the usual ratio of data splitting (80:20).

```
# X is the independent variable (predictors) while Y is the dependent
variable

X = df_balanced.drop(['is_closed'], axis=1, errors='ignore')
Y = df_balanced['is_closed']
```

Data Splitting

```
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42, stratify=Y
)
```

To perform the preprocessor:

```
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

INITIAL MODEL TRAINING

```
# xgboost classifier
xgb_clf = XGBClassifier(objective='binary:logistic',
use_label_encoder=False, eval_metric='logloss', random_state=42,
class_weight='balanced')

# train
xgb_clf.fit(X_train_processed, Y_train)
```

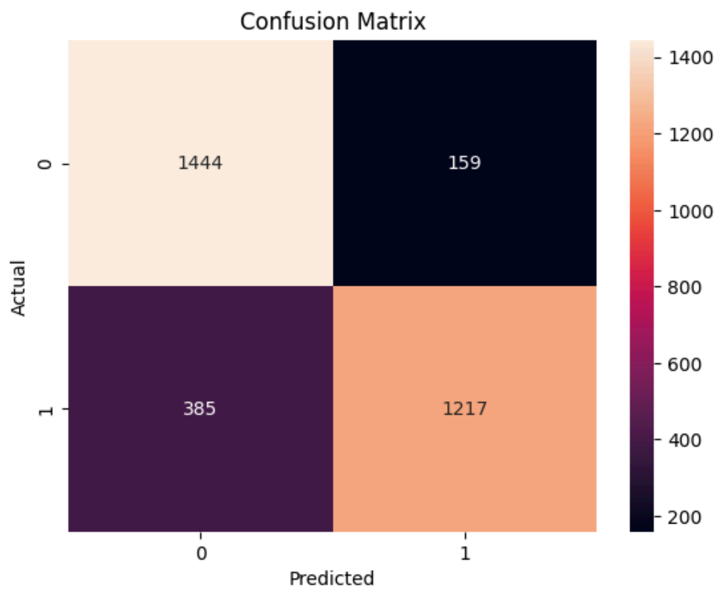
We added a parameter of class_weight so that the model will handle the majority class and minority class equally, avoiding bias tuning.

Using the evaluation metrics scikit-learn provides, we got the following scores for the initial model:

```
Accuracy: 0.8303
Precision: 0.8844
Recall: 0.7597
F1-score: 0.8173

Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.90      0.84      1603
           1       0.88      0.76      0.82      1602
```



This shows that the model is performing exceptionally well in both classes, but since this is just an initial model, we can apply hyperparameter tuning to further enhance its scores, and to know which are the best parameters the model can apply to it.

5. Hyperparameter Tuning

I used Optuna as it is a great tool for automated searching of optimal hyperparameters that is efficient for large datasets and provides faster results. Another thing, I applied SMOTETomek to handle imbalance on data (if it is still existing) and StratifiedKFold for cross-validation.

After creating a study with optuna, running on 10 trials, it gave me this result for the best parameters.

```
Best trial:
  Value: 0.860459612819683
  Params:
    n_estimators: 914
    learning_rate: 0.03397655562503694
    max_depth: 10
    subsample: 0.7543553240517082
    colsample_bytree: 0.8766268643227735
    gamma: 0.3387458397928187
    min_child_weight: 3
    reg_alpha: 0.9421360427886788
    reg_lambda: 0.23504617439657882
```

After being satisfied with the parameters, I proceed then with the final training of the model, applying these parameters to it.

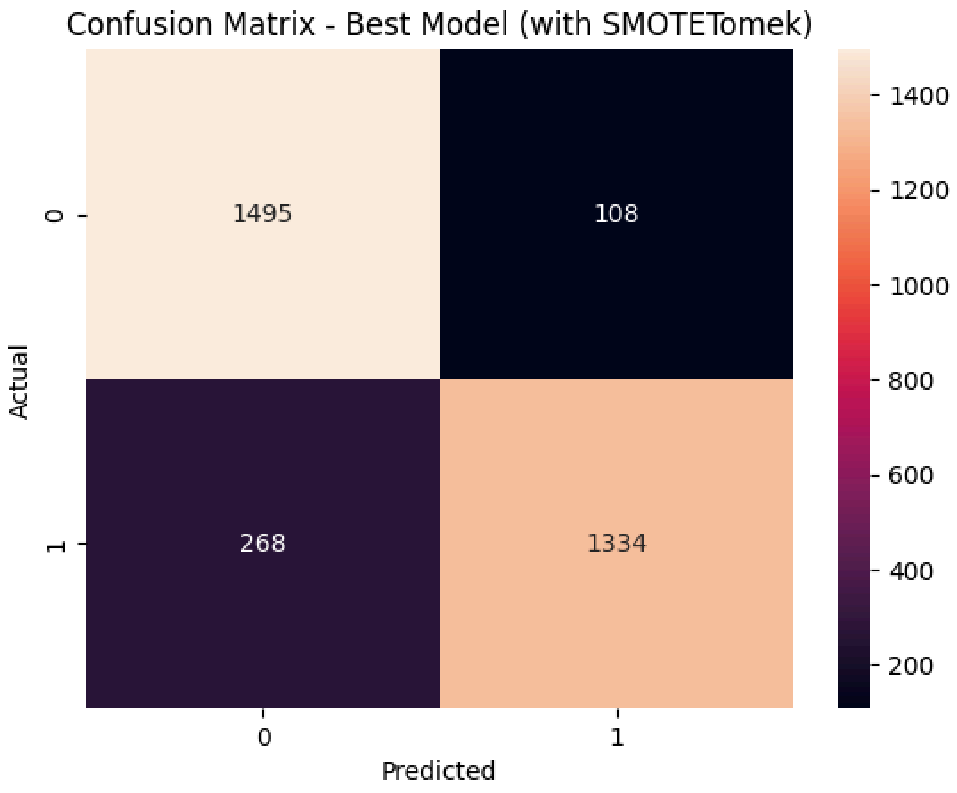
6. Final Model evaluation

My final predictive model presented me with outstanding results.

```
ccuracy: 0.8827
Precision: 0.9251
Recall: 0.8327
F1-score: 0.8765

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89      1603
           1       0.93      0.83      0.88      1602
```



Interpretation of the Result:
This result of the evaluation metrics indicates that XGBoost Classifier, combined with SMOTETomek for handling the class imbalance and effective hyperparameter tuning, is performing exceptionally well for this prediction task.

- Carefully curated and solved the imbalance ratio of the dataset.
- The model is now highly capable of identifying 'Open' incidents (high recall) while also being reasonably precise in its 'Open' predictions.
- It maintains strong performance on the 'Closed' incidents as well.

This model should be highly valuable and actionable for prediction incident management systems, allowing for much more reliable and proactive identification of incidents that are likely to remain open.