MICROSOFT : CLASSIFYING CYBERSECURITY INCIDENTS WITH MACHINE LEARNING

- Pooja Spandana

1. Project background and description

This project aims to enhance the efficiency of Security Operation Centers (SOCs) by developing a machine learning model capable of accurately predicting the triage grade of cybersecurity incidents. Using the comprehensive GUIDE dataset, the model categorizes incidents as true positive (TP), benign positive (BP), or false positive (FP) based on historical evidence and customer responses. The ultimate goal is to support guided response systems in providing SOC analysts with precise, context-rich recommendations, thereby improving the overall security posture of enterprise environments.

2. Project scope

The solution developed in this project can be implemented in various business scenarios, particularly in the field of cybersecurity. Some potential applications include:

- **Security Operation Centers (SOCs):** Automating the triage process by accurately classifying cybersecurity incidents, thereby allowing SOC analysts to prioritize their efforts and respond to critical threats more efficiently.
- **Incident Response Automation:** Enabling guided response systems to automatically suggest appropriate actions for different types of incidents, leading to quicker mitigation of potential threats.
- **Threat Intelligence:** Enhancing threat detection capabilities by incorporating historical evidence and customer responses into the triage process, which can lead to more accurate identification of true and false positives.
- **Enterprise Security Management:** Improving the overall security posture of enterprise environments by reducing the number of false positives and ensuring that true threats are addressed promptly.

3. Skills take away From This Project

- Data Preprocessing and Feature Engineering
- Machine Learning Classification Techniques
- Model Evaluation Metrics (Macro-F1 Score, Precision, Recall)
- Cybersecurity Concepts and Frameworks (MITRE ATT&CK)
- Handling Imbalanced Datasets
- Model Benchmarking and Optimization

4. Approach

(i) DATA CLEANING AND PREPROCESSING

Data Exploration and Understanding:

Basic Exploration of the train dataset gave away the following:

- *Dataset shape:* (4758418, 45)
- *Data type of columns in the dataset:* int64(30), object(14), float64(1)
- *Target variable Analysis*:

IncidentGrade	Count	Percentage
BenignPositive	2054774	43.417050
TruePositive	1662087	35.119636
FalsePositive	1015782	21.463313

- *Missing Data:* [ResourceType - 99.92%, ActionGranular - 99.40%, ActionGrouped - 99.40%, ThreatFamily - 99.21%, EmailClusterId - 98.98%, AntispamDirection - 98.13%, Roles - 97.70%, SuspicionLevel - 84.84%, LastVerdict - 76.54%, MitreTechniques - 57.43%, IncidentGrade - 0.54%]

Data Cleaning:

- Dropped columns with >50% missing values
- Dropped rows with null values in target variable
- Converted Timestamp to datetime
- Extracted time-related features
- Removed 155243 duplicate rows
- Shape after cleaning: (4577400, 39)

Exploratory Data Analysis (EDA):

- Used visualizations and statistical summaries to identify patterns & correlations.
- Based on the information gained upon EDA dropped some features from the dataset that are highly correlated, low impact features & some pure identity features like Ids –

  [“AccountSid", "AccountUpn", "Sha256", "FolderPath", "RegistryValueName", "OSFamily", "City", "CountryCode", "ApplicationId", "IncidentId", "FileName", "OAuthApplicationId", "ResourceIdName", "RegistryKey", "Id", "AccountObjectId", "AlertId", "NetworkMessageId", "RegistryValueData”]
- Ran Chi-Square (χ^2) test of independence & ANOVA tests to identify the relationship of categorical & numerical features wrt the target variable (IncidentGrade)

Feature Engineering:

- **Category Grouping:** Keeps the top 10 most frequent values in Category, grouped rest as "Other".
- **EntityType Grouping:** Same approach for EntityType—top 10 retained, rest grouped.
- **Impacted Flag**: New binary feature IsImpacted created from EvidenceRole indicating whether the entity is "Impacted".
- **AlertTitle Grouping**: Top 10 most frequent AlertTitle values retained; others replaced with 11
- **Time Features**: **IsWeekend:** 1 if DayOfWeek is Saturday/Sunday.

  IsBusinessHour: 1 if the hour falls between 9 AM and 6 PM.
- **Geographic Flag**: **IsMajorState** is 1 if State equals 1445.
- **OS Version Flag**: **IsOSVersion66** is 1 if OSVersion equals 66.
- **Cleanup**: Drops original columns used to create the engineered ones and renames grouped columns back to their original names for consistency.

- **After Feature Engineering the dataset contains:**
    - **Categorical features: 3** - ['IncidentGrade', 'Category', 'EntityType']
    - **Numerical features: 19** - ['OrgId', 'DetectorId', 'DeviceId', 'IpAddress', 'Url', 'AccountName', 'DeviceName', 'ApplicationName', 'Hour', 'Day', 'DayOfWeek', 'Month', 'Year', 'IsImpacted', 'AlertTitle', 'IsWeekend', 'IsBusinessHour', 'IsMajorState', 'IsOSVersion66']
- **Encoded ['Category', 'EntityType'] using LabelEncoder()**

Followed the same techniques for test dataset to maintain consistency.

Stored the final cleaned train & test datasets as Train_DS_Cleaned.csv & Test_DS_Cleaned.csv

(ii) MODEL BUILDING AND EVALUATION

Data Sampling:

Sampled 5% of the data for faster prototyping by using Stratified sampling to preserve class ratios

Data Splitting:

- Feature Matrix (X) & Target variable (y)
- Train-test split using stratification: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
- Encoded the target variable using LabelEncoder(), scaled features that are in different ranges using StandardScaler() after train_test_split to avoid data leakage & ensuring test data is unseen to the model.

Model Building and Training:

- **Baseline Models:**
    - Linear Regression
    - Decision Tree
    - Random Forest
    - K-Nearest Neighbors
    - Gradient Boosting
    - XGBoost

- These models are trained on the imbalanced dataset with all default parameters & following are the results.

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1-Score (Macro)	F1-Score (Weighted)
XGBoost	0.9037	0.9062	0.8903	0.8971	0.9032
Random Forest	0.8890	0.8912	0.8753	0.8821	0.8885
K-Nearest Neighbors	0.7904	0.7810	0.7752	0.7779	0.7899
Gradient Boosting	0.7874	0.8304	0.7450	0.7664	0.7820
Decision Tree	0.6825	0.6996	0.7011	0.6808	0.6910
Logistic Regression	0.4821	0.4622	0.4542	0.4551	0.4776

Best Model: XGBoost
Best F1-Score (Macro): 0.8971

- The top three models were selected to treat class imbalance using RandomUnderSampler().
- **Using RandomUnderSampler() is a practical and an effective approach as:**
    - The dataset is large (so discarding some majority samples is acceptable), prevents memory overload or very long training times
    - Balances the dataset quickly without introducing synthetic noise as compared to SMOTE
    - Boosts recall and F1-score for underrepresented classes — critical in security/incident classification tasks, where missing a minority class (e.g., True Positive incidents) can have serious consequences.

- **Top 3 Baseline Models with RUS:**
    - XGBoost
    - Random Forest
    - K-Nearest Neighbors
- These models are trained on balanced dataset with all default parameters & following are the results.

```
              Model  Accuracy  Precision (Macro)  Recall (Macro)  F1-Score (Macro)  F1-Score (Weighted)
            XGBoost    0.8952             0.8836          0.8985            0.8887               0.8965
      Random Forest    0.8763             0.8664          0.8746            0.8697               0.8769
K-Nearest Neighbors    0.7578             0.7455          0.7611            0.7493               0.7606

Best Model: XGBoost
Best F1-Score (Macro): 0.8887
```

- K-Nearest Neighbors model does not show promising results so hyperparameter tuned only XGBoost & Random Forest models with RUS.
- **Hyperparameter Tuned Models with RUS:**
    - XGBoost
    - Random Forest
- These models are hyperparameter tuned on resampled data using RUS, and uses cross-validation (StratifiedKFold cross-validation) to ensure the model's performance is consistent across different subsets of the data. Following are the results.

```
        Model  Accuracy  Precision (Macro)  Recall (Macro)  F1-Score (Macro)  F1-Score (Weighted)
Random Forest    0.9381             0.9295          0.9390            0.9337               0.9384
      XGBoost    0.9188             0.9087          0.9207            0.9136               0.9194

Best Model: Random Forest
Best F1-Score (Macro): 0.9337
```

Model Evaluation:

Compare the best model with the baseline models.

```
             Model  Accuracy  Precision (Macro)  Recall (Macro)  F1-Score (Macro)  F1-Score (Weighted)
       Baseline RF    0.8890             0.8912          0.8753            0.8821               0.8885
 Baseline RF + RUS    0.8763             0.8664          0.8746            0.8697               0.8769
 RUS + HP Tuned RF    0.9381             0.9295          0.9390            0.9337               0.9384

 Best Model: RUS + HP Tuned RF
 Best F1-Score (Macro): 0.9337
```

```
-------------------------------------------------------
Hyperparameter Tuning Random Forest with RUS
-------------------------------------------------------
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best Params: {'n_estimators': 200, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': None, 'max_depth': None, 'bootstrap': True}

Confusion Matrix:
[[27721  1144   744]
 [  458 14087   373]
 [  648   885 22601]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.94      0.95     29609
           1       0.87      0.94      0.91     14918
           2       0.95      0.94      0.94     24134

    accuracy                           0.94     68661
   macro avg       0.93      0.94      0.93     68661
weighted avg       0.94      0.94      0.94     68661
```

- RUS alone helps improve minority class prediction, but tuning is essential to regain and boost overall performance. This can clearly be seen in the Tuned Random Forest with RUS as it provides a robust, fair, and highly accurate solution for imbalanced multiclass classification.

(iii) FINALLY TRAIN THE BEST MODEL ON THE ENTIRE TRAIN DATASET & EVALUATE ON THE TEST DATASET

RESULTS ON TRAIN DATA:

```
Accuracy score:
0.9789181631493861

Confusion Matrix:
[[385958   5631   3200]
 [  1806 195171   1923]
 [  2299   4441 315051]]

Classification Report:
                precision    recall  f1-score   support

BenignPositive       0.99      0.98      0.98    394789
 FalsePositive       0.95      0.98      0.97    198900
  TruePositive       0.98      0.98      0.98    321791

      accuracy                           0.98    915480
     macro avg       0.97      0.98      0.98    915480
  weighted avg       0.98      0.98      0.98    915480
```

RESULTS ON TEST DATA:

```
Accuracy score:
0.9446977141990391

Confusion Matrix:
[[1556707   48736   25499]
 [  38038  797927   32932]
 [  25496   46233 1351127]]

Classification Report:
                precision    recall  f1-score   support

BenignPositive       0.96      0.95      0.96   1630942
 FalsePositive       0.89      0.92      0.91    868897
  TruePositive       0.96      0.95      0.95   1422856

      accuracy                           0.94   3922695
     macro avg       0.94      0.94      0.94   3922695
  weighted avg       0.95      0.94      0.94   3922695
```

5. Conclusion

This project demonstrated the application of machine learning in classifying cybersecurity incidents based on historical data. The **Random Forest** model provided the best performance, offering a balanced trade-off between accuracy and fairness across all classes. Further refinements such as hyperparameter tuning and feature engineering could lead to even better results, making this approach a valuable tool for SOCs in prioritizing and addressing cybersecurity threats more effectively.