UNIVERSITAT POLITÈCNICA DE CATALUNYA

APA - Aprendizaje Automático

Microsoft Security Incident Prediction

Muhammad Yasin Khokhar

Momin Miah Begum

January Q1 2024-25

Index

1 Overview and Objectives

Nowdays, organizations today face an increasing number of cybersecurity threats, which result in a high volume of incidents that require analysis and management. This growing complexity poses a significant challenge to ensuring timely and effective responses. To address this issue, Security Operations Centers (SOCs) need advanced tools designed to help them respond the incident [6] efficiently and manage them effectively.

Relying entirely on automated systems can be challenging, as they require extremely high accuracy to avoid mistakes that could affect critical assets. To overcome this, guided response (GR) systems provide a more practical solution. GR systems assist SOC analysts by offering context-rich, data-driven recommendations rather than fully automating decisions. These systems present analysts with informed suggestions based on real-time data and historical context, helping them make better decisions and manage incidents more effectively.

This work focuses on developing and benchmarking machine learning models using Microsoft's GUIDE dataset, the largest publicly available collection of real-world cybersecurity incidents. GUIDE contains more than 13 million evidence pieces, 1.6 million alerts, and 1 million annotated incidents from 6,100 organizations, providing a unique opportunity to predict incident triage outcomes—***true positive (TP)***, ***benign positive (BP)***, and ***false positive (FP)***.

By leveraging the GUIDE_train dataset, which features a rich and complex structure of evidence, alerts, and incidents, we aim to create models that accurately classify triage incidents into TP, BP, and FP. The primary goal is to employ machine learning techniques to support Security Operations Centers (SOCs) in incident management. Model performance will be evaluated using standard metrics such as *macro-F1 score*, precision, and *recall*, with a focus on delivering actionable insights for guided response systems.

We expect that the models trained on this dataset will perform well and provide valuable assistance to SOCs in streamlining their procedures. These models have the potential to become an integral part of the incident response pipeline, offering actionable insights and enhancing the efficiency of guided response systems. After evaluating various models, **Random Forest** was chosen as the final model due to its strong performance and computational efficiency. It achieved 0.84 precision, 0.794 recall, and 0.809 F1-score, effectively handling class imbalances. Non-linear models, like Random Forest, outperformed linear models, making them better suited for this dataset.

The structure of this document is as follows: we introduce the dataset and its features, outline the data exploration and preprocessing methods, describe the machine learning models and evaluation metrics, present the results, and analyze the interpretability and implications of the models. Finally, we discuss limitations, successes, and future research directions, highlighting the significance of this work in advancing cybersecurity response capabilities.

2 Description and Previous Studies

As we mentioned earlier, the dataset is a subset of the original dataset available on Kaggle [5], which is part of the broader GUIDE collection. This dataset, hosted by Microsoft Security AI Research, contains detailed telemetry data from over 6,100 organizations, including security alerts, incident reports, and evidence. It provides a valuable resource for training and evaluating machine learning models in real-world cybersecurity scenarios, supporting efforts to improve incident response and guided remediation. We can consider this dataset as "new" since it was published alongside the paper titled AI-Driven Guided Response for Security Operation Centers with Microsoft Copilot for Security [4].

Additionally, we have not found any prior studies specifically evaluating models on this dataset. Although there are some Kaggle notebooks available, most of them focus on exploratory analysis of the features, and only a few apply models that we do not address in this report. Initially, when we selected the dataset, there was no model application within it. Now, as mentioned, there are a few attempts, but with different models from those discussed in our report. Therefore, our work contributes the first models evaluations for this dataset and provides valuable insights into its potential use for incident response and triage tasks.

3 Data Exploration and Preprocessing

3.1 Initial exploration

Exploring the dataset, we have 46 features. Let's explore their characteristics. First let's explore the **numerical features**.

1. **Id**: Unique ID for each OrgId-IncidentId pair.
2. **OrgId**: Organization identifier.
3. **IncidentId**: Organizationally unique incident identifier.
4. **Alert**: Unique identifier for an alert.
5. **Timestamp**: Time the alert was created.
6. **DetectorId**: Unique ID for the alert generating detector.
7. **AlertTitle**: Title of the alert.
8. **DeviceId**: Unique identifier for the device.
9. **Sha256**: SHA-256 hash of the file.
10. **IpAddress**: IP address involved.
11. **Url**: URL involved.
12. **AccountSid**: On-premises account identifier.
13. **AccountUpn**: Email account identifier.

14. **AccountObjectId**: Entra ID account identifier.
15. **AccountName**: Name of the on-premises account.
16. **DeviceName**: Name of the device.
17. **NetworkMessageId**: Org-level identifier for email message.
18. **EmailClusterId**: Unique identifier for the email cluster. This attribute contains 99% null values.
19. **RegistryKey**: Registry key involved.
20. **RegistryValueName**: Name of the registry value.
21. **RegistryValueData**: Data of the registry value.
22. **ApplicationId**: Unique identifier for the application.
23. **ApplicationName**: Name of the application.
24. **OAuthApplicationId**: OAuth application identifier.
25. **FileName**: Name of the file.
26. **FolderPath**: Path of the file folder.
27. **ResourceIdName**: Name of the Azure resource.
28. **OSFamily**: Family of the operating system.
29. **OSVersion**: Version of the operating system.
30. **CountryCode**: Country code evidence appears in.
31. **State**: State of evidence appears in.
32. **City**: City of evidence appears in.

Categorical features.

1. **Category**: Category of the alert.
2. **MitreTechniques**: MITRE ATT&CK techniques involved in the alert. This attribute contains some null values.
3. **IncidentGrade**: SOC grade assigned to the incident.
4. **ActionGrouped**: SOC alert remediation action (high level). This attribute contains 100% null values.
5. **ActionGranular**: SOC alert remediation action (fine-grain). This attribute contains 100% null values.
6. **EntityType**: Type of entity involved in the alert.
7. **EvidenceRole**: Role of the evidence in the investigation.
8. **ThreatFamily**: Malware family associated with a file. This attribute contains 99% null values.

9. **ResourceType**: Type of Azure resource. This attribute contains 100% null values.
10. **Roles**: Additional metadata on evidence role in the alert. This attribute contains 97% null values.
11. **AntispamDirection**: Direction of the antispam filter. This attribute contains 98% null values.
12. **SuspicionLevel**: Level of suspicion. This attribute contains 84% null values.
13. **LastVerdict**: Final verdict of threat analysis. This attribute contains 76% null values.
14. **Usage**: Public or Private.

Now, let's explore our main feature, which we need to predict: **IncidentGrade**. This feature has three possible values: True Positive, Benign Positive, and False Positive [7].

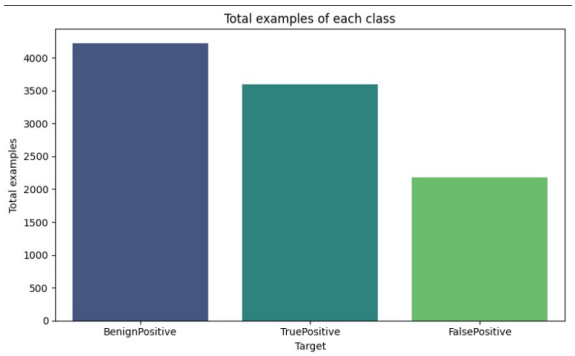

Figure 1: Total count from each class

We observe that the least frequent incident type is False Positive, which refers to cases where an alert incorrectly indicates a threat. On the other hand, the most frequent type is Benign Positive, which represents alerts that are mistakenly classified as threats but are actually harmless. In between, we find True Positive incidents, which are correctly identified threats.

We also have have variable called Timestamp which can help us to see trend of incidents.

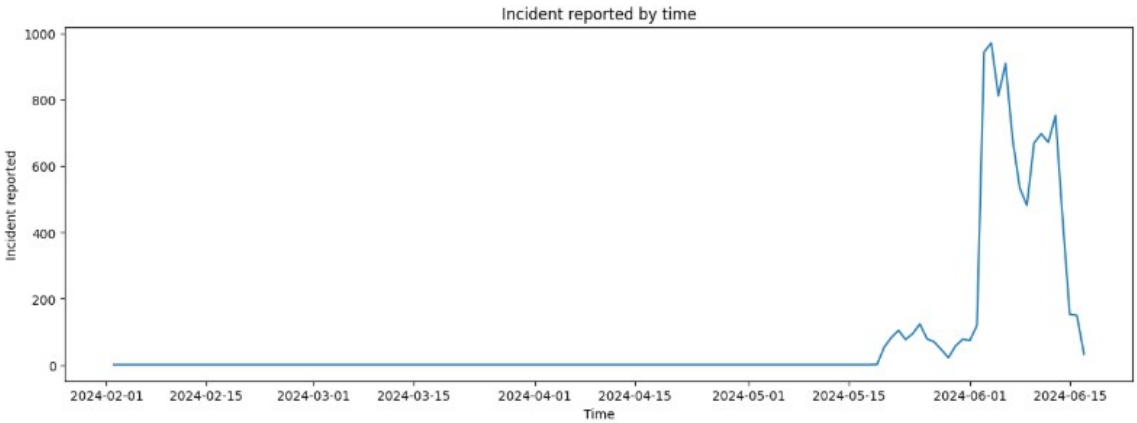

Figure 2: Total incidents reported by time.

From the chart we can observe that the data was gathered only from a certain period, which is relatively short and does not help us determine if there might be any patterns over time in the possible incidents.

Some other features are explored in the attached notebook; refer to it to explore them. Due to page limitations, we will not include them in the report, although they are discussed and addressed in the notebook.

3.2 Preprocessing

In preprocessing [1][2], we removed irrelevant features based on the context of the problem, excluded features with a high number of missing values, transformed categorical variables into numerical ones, treated outliers (if present), applied normalization, and performed PCA and t-SNE.

3.2.1 Attributes with High Missing values

Addressing the issue of missing values, we have decided not to include them for further processing.

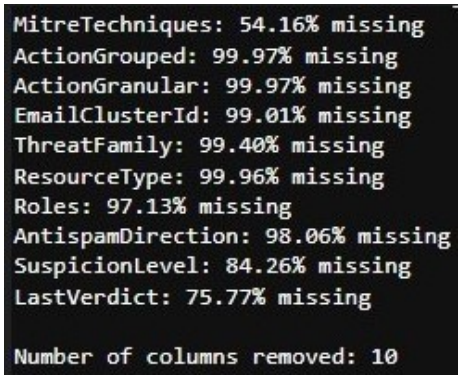

Figure 3: Removed features from the original dataset.

Addressing the issue of missing values, we have decided not to include them for further processing. The main reason is the substantial number of missing values, making imputation unfeasible. Variables with more than 50% missing values will be removed. On the other hand, we identified a relationship between two variables, **Last Verdict** and **Suspicion Level**, using *msno*. However, due to the high number of missing values in these variables, their usability remains limited.

3.3 Contextually Irrelevant Features Removal

1. OAuthApplicationId
2. DeviceId
3. AccountSid
4. Id
5. AlertId
6. Timestamp

After reviewing the descriptions of these features, we determined that they are not useful for the context of the problem. Firstly, some features, like Id, represent relationships between other features (e.g., OrgId and IncidentId), making them redun-

dant. Secondly, AlertId is nearly unique, as it has approximately the same number of unique values as rows, which limits its relevance. Features such as AccountSid, DeviceId, and OAuthApplicationId were removed because their importance in the context of the problem appears negligible. Specifically, OAuthApplicationId is the OAuth application identifier, DeviceId is a unique identifier for the device, and AccountSid is the on-premises account identifier. Additionally, we removed the Timestamp feature since our goal is not to predict incidents over time but rather to predict the type of incidents.

3.4 Transformation of Categorical Values

Initially, we applied one-hot encoding to transform categorical variables into numerical ones. However, applying this technique to categorical columns with many unique values resulted in a dramatic increase in the number of features, expanding the dataset from 29 columns to 60. This can cause several problems:

- **Increased Dimensionality:** The significant increase in the number of columns can lead to higher computational costs, longer training times, and make the model harder to interpret.

But in this case, the increase in the number of variables is not significant, but we would like to explore the other option. As an alternative, we opted for **Label Encoding**, which is more efficient as it reduces dimensionality while still converting categorical variables into numeric values, maintaining the number of variables as in the original dataset.

3.5 Normalization

For normalization, we used the `MinMaxScaler` because it scales the data to a fixed range, typically [0, 1], which helps to maintain the relative proportions between features. This method ensures that all features contribute equally to the model's performance, especially when working with algorithms sensitive to feature scaling.

3.6 Handling Outliers

In the visualization of the attributes, we observed that some numerical variables had a distribution where a few values were clustered around small values, while the majority of the data had much higher values. This could indicate the presence of outliers. Therefore, we decided to handle these outliers using the interquartile range (IQR) method.

3.7 Dimensionality Reduction: PCA and t-SNE,

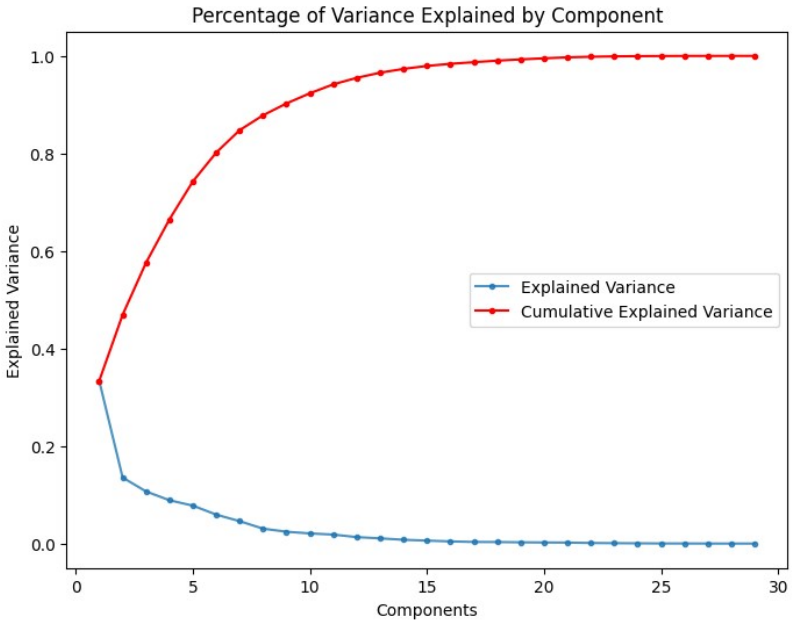

Figure 4: Explained Variance

From the previous chart, we can observe that a relatively high number of components is needed to explain approximately 90-95% of the variance in the data. With just 5 components, we are able to explain roughly 60% of the variance. This suggests that the data has high dimensionality, and reducing it further might lead to a loss of important information. To better understand the structure of the data, we can apply PCA for dimensionality reduction, which helps identify the most important features that capture the variability. Additionally we will apply t-SNE to visualize the data in lower dimensions, revealing underlying patterns or clusters that are not immediately apparent in the high-dimensional space.

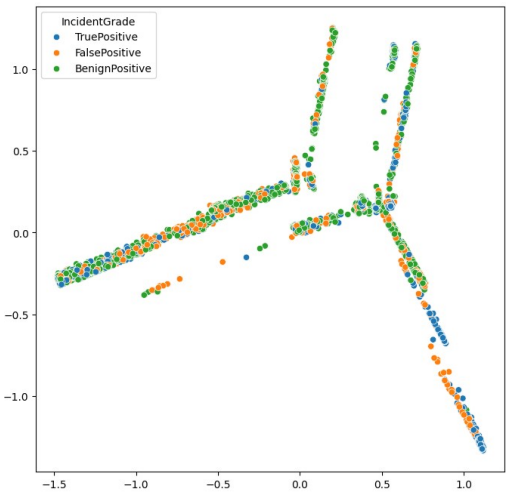

Figure 5: PCA 2D.

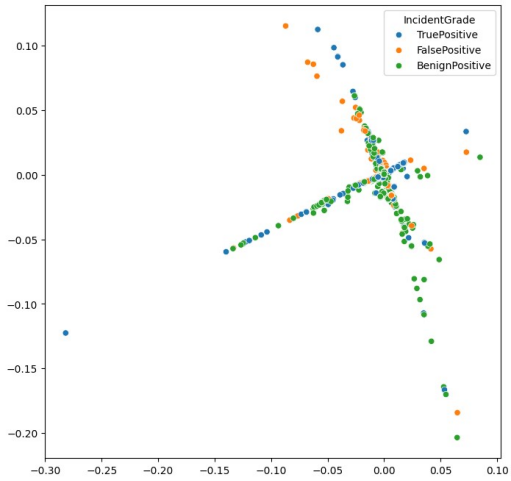

Figure 6: t-SNE 2D.

After applying both PCA and t-SNE, we observe significant noise in the classification of the incident grades. This suggests that the data is not linearly separable, which implies that linear models may struggle to capture the underlying patterns. Consequently, this provides a strong indication that non-linear models, might perform better on this dataset, as they are capable of capturing complex, non-linear relationships in the data.

4 Resampling

As is common practice when training models for problems, we split the data into two sets: the training data and the validation (or test) data. The training data represents 70% of the total, and the test data represents 30%. We stratified this split to avoid any class imbalance.

The theoretical reason behind this decision is that, while training the model on all data might make it perform better, we wouldn't be able to tell if it can generalize well to unseen data. This division helps ensure the model's ability to generalize [1].

Additionally, during the search for the best model with optimal hyperparameters, we use cross-validation. Cross-validation is a technique where the data is split into multiple subsets. The model is trained on some of these subsets and tested on others to ensure that the model performs well on different parts of the data. This helps to avoid overfitting and gives a better estimate of the model's performance.

5 Results with Linear/Quadratic Methods

In this section, we detail the performance of the linear/quadratic models[1][3] and discuss the parameter configurations that yielded the best outcomes.

5.1 Naive Bayes

Since the data didn't show clear Gaussian characteristics, we decided to test three Naïve Bayes models: Multinomial (for categorical features), Gaussian (for continuous features), and Bernoulli (for binary features). Despite assuming total feature independence, which isn't true, we wanted to observe the model's performance.

BernoulliNB achieved the best performance with an accuracy of 0.57.

GaussianNB performed poorly, classifying most instances as 'BenignPositive'. In comparison, BernoulliNB performed better at identifying 'FalsePositive', while MultinomialNB was more balanced but still struggled with 'FalsePositive'.

Conclusion: BernoulliNB outperformed the others in recognizing 'FalsePositive', while **GaussianNB** had the worst overall performance.

5.2 Logistic Regression

Logistic Regression is a simple model that assumes a linear relationship between input features and the target. We applied L2 regularization with a C value of 1.99 to balance overfitting and underfitting.

Key features include `originID`, `incidentID`, and `detectorID`.

The performance metrics are as follows: - **Accuracy**: 0.56 - **Precision/Recall**: High precision for `TruePositive` (0.86), low recall for `FalsePositive` (0.03) - **F1-Score**: 0.54 for `TruePositive`

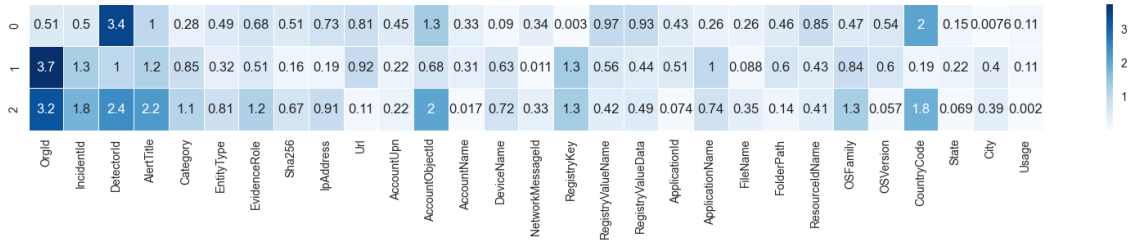

Figure 7: Features Importance Logistic Regression Model

Attributes like `originID`, `incidentID`, `detectorID`, and `alertTitle` seem most important. While `originID` represents the organization and may impact context, the other IDs lack clear relationships. Their importance might stem from correlations with outcomes, requiring further investigation.

5.3 KNN

K-nearest Neighbours (`KNN`) is a simple algorithm that classifies a test point based on the nearest neighbors. It was selected for its ease of implementation and good performance on smaller datasets. The best models used 15 or more neighbors, weighted distance, and the `L1` metric.

Params	Mean Test Score	Rank Test Score	
30	{'metric': 'l1', 'n_neighbors': 17, 'weights': 'distance'}	0.720	1
28	{'metric': 'l1', 'n_neighbors': 15, 'weights': 'distance'}	0.720	2
32	{'metric': 'l1', 'n_neighbors': 19, 'weights': 'distance'}	0.719	3
26	{'metric': 'l1', 'n_neighbors': 11, 'weights': 'distance'}	0.718	4
34	{'metric': 'l1', 'n_neighbors': 21, 'weights': 'distance'}	0.716	5

Table 1: Summary of Parameters and Test Scores

The model achieved an accuracy of 0.69. For precision/recall, `BenignPositive` had high precision (0.63) and recall (0.83), `FalsePositive` had a precision of 0.64 and recall of 0.40, and `TruePositive` had a precision of 0.78 and recall of 0.67.

Important features included `IncidentId` and `EntityType`, while `ResourceIdName` and `City` were less important.

5.4 Comparison of Linear/Quadratic Methods

`KNN` outperforms all models with the highest test accuracy (0.69), precision (0.685), and recall (0.643), leading to the best F1 score (0.653), making it the best option for this dataset.

`Logistic Regression` shows good precision (0.720) but lower recall (0.470) and F1 score (0.420), indicating a trade-off between precision and recall.

`MNB` and `GNB` perform poorly, with `MNB` having the lowest metrics (precision: 0.375, recall: 0.438, F1: 0.380), and `GNB` struggling across all metrics, especially precision (0.141).

In conclusion, `KNN` is the best-performing model, while `MNB` and `GNB` are unsuitable for the task.

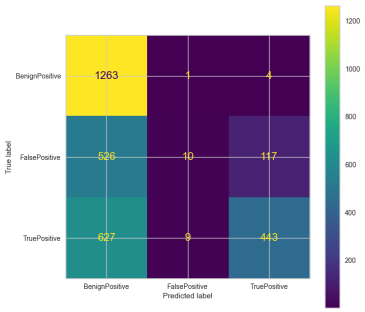

Figure 8: Confusion Matrix BernulliMB

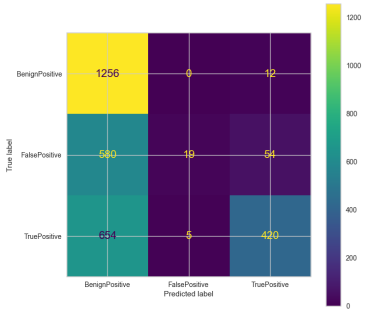

Figure 9: Confusion Matrix Logistic Regression

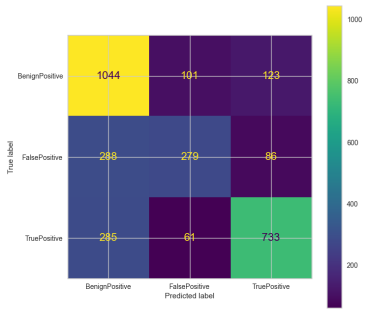

Figure 10: Confusion Matrix KNN

6 Results with Nonlinear Methods

Below, we present the results of three nonlinear models[1][3] along with their best hyperparameters.

6.1 Random Forest

Random Forest builds multiple decision trees and combines their outputs for prediction. It is suitable for this classification problem due to its ability to capture complex relationships and reduce overfitting.

The explored hyperparameters include: `criterion`: Set to `entropy` in the top configurations, indicating a preference for information gain. `min_samples_leaf`: Set to 1 in the best-performing configurations. `max_depth`: Not specified, allowing the model to grow trees without depth restrictions. `n_estimators`: Set to 200 in the top configurations.

For feature importance: Most important features: `IncidentId`, `OrgId`, `DetectorId`. Less important features: `ApplicationId`, `Url`, `State`. Features with zero importance: `ResourceIdName`, `RegistryValueData`, `RegistryKey`, `RegistryValueName`.

The ROC curve score is approximately 0.95 for each class (BenignPositive, FalsePositive, TruePositive).

6.2 SVM with RBF Kernel

SVM with RBF kernel is a powerful method for classifying non-linear data. It transforms the data into a higher-dimensional space where separating the classes becomes easier by finding a decision boundary.

The best-performing hyperparameter combination is `C = 660.69`, `gamma = scale`, and `probability = true`, with `gamma = scale` common across top configurations.

For feature importance, SVM emphasizes features that were not as prominent in other models, such as `EntityType`, `AccountName`, and `CountryCode`. On the other hand, features like `IncidentId` and `OrgId`, which were more important in models like Random Forest, are less significant here.

The ROC curve values for SVM show relatively lower performance compared to Random Forest: FalsePositive: 0.66, TruePositive: 0.82, BenignPositive: 0.74

6.3 Gradient Boosting

Gradient Boosting is a method that builds models iteratively, where each step focuses on correcting the errors made by the previous one. It combines multiple decision trees to create a more accurate model.

After exploring the hyperparameters, two top-performing combinations were identified, differing only in `n_estimators` (50 vs. 75). Both use the following hyperparameters: `Criterion (friedman_mse)`: Optimizes tree splits for better performance. `Learning Rate (0.5)`: Balances the impact of each tree, controlling how much each step corrects errors. `Loss (log_loss)`: Minimizes classification errors. `Max Depth (None)`: Allows trees to grow without depth restrictions. `Min Samples Leaf (10)`: Ensures each leaf node has at least 10 samples, avoiding overfitting. `n_estimators (50/75)`: Number of trees in the ensemble.

The Gradient Boosting model achieves an overall accuracy of 81`OrgId`, `IncidentId`, `DetectorId`, `AlertTitle`,

6.4 Conclusion: Non-Linear Models

Among the non-linear models, **SVM** is not an ideal option due to its poor performance when classifying classes with fewer instances, as observed with the `FalsePositive` class. Additionally, the feature selection process in SVM differs significantly from that of more robust models like **Random Forest** and **Gradient Boosting**, suggesting that it is not fully utilizing the data.

On the other hand, **Gradient Boosting** demonstrated strong performance, particularly in classifying the less frequent classes more effectively, which is valuable for this problem. However, its computational cost is relatively high, which may be a limiting factor in large-scale scenarios.

Finally, **Random Forest** stands out as the most balanced option. It provides solid performance with relatively low computational cost, making it the most suitable choice for this problem.

The ROC curve highlights how well models distinguish between classes by plotting the True Positive Rate against the False Positive Rate. Both **Random Forest** and **Gradient Boosting** achieve excellent ROC scores around **0.95** for all classes, reflecting strong discriminatory power. In contrast, **SVM** underperforms significantly, with lower scores, particularly for the `FalsePositive` class (0.66). This confirms that SVM struggles with minority classes, unlike Random Forest and Gradient Boosting, which provide more balanced performance. Overall, Random Forest demonstrates superior and consistent class distinction.

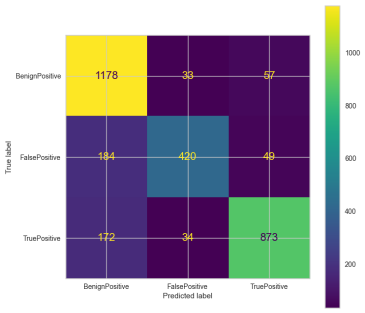

Figure 11: Confusion Matrix Random Forest

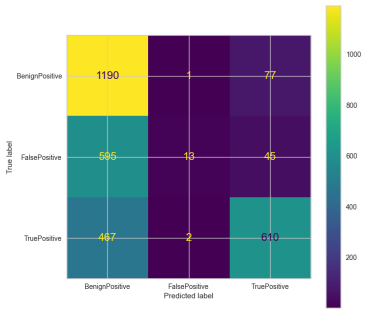

Figure 12: Confusion Matrix SVM RBF Kernel

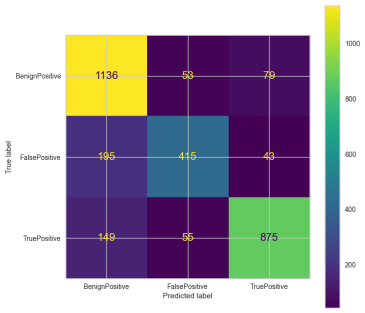

Figure 13: Confusion Matrix Gradient Boosting

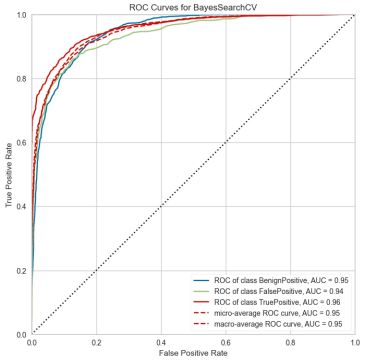

Figure 14: ROC Random Forest

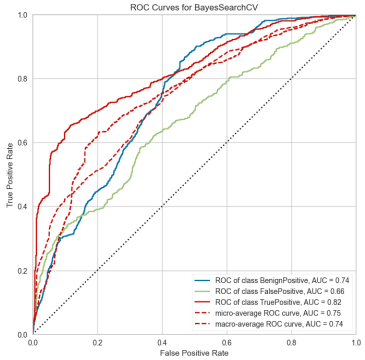

Figure 15: ROC SVM RBF Kernel

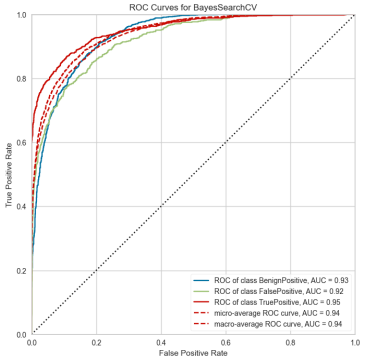

Figure 16: ROC Gradient Boosting

7 Model Comparison

Among all the models, `KNN` performed the best with the highest accuracy (0.69), precision (0.685), and recall (0.643), achieving the best F1 score (0.653). `Logistic Regression` had decent precision (0.720) but low recall (0.470) and F1 score (0.420), while `MNB` and `GNB` performed poorly across all metrics.

For non-linear models, **SVM** struggled with the `FalsePositive` class and had poor feature selection, making it less effective. **Gradient Boosting** performed well but is computationally expensive, while **Random Forest** provided the best balance of performance and efficiency, making it the most suitable model overall.

It seems like `KNN` is the best among linear models, and **Random Forest** is the optimal choice for this dataset.

Model	Cross-val acc	Test acc	Precision (M)	Recall (M)	F1 (M)
Linear Models					
KNN	0.720	0.685	0.685	0.643	0.653
Logistic Regression	0.607	0.565	0.720	0.470	0.420
Non-Linear Models					
Random Forest	0.831	0.824	0.841	0.794	0.809
Gradient Boosting	0.866	0.809	0.813	0.781	0.792
RBF SVM binary	0.673	0.604	0.725	0.508	0.463

Table 2: Model Comparison: Linear and Non-Linear Models

The table compares models based on cross-validation accuracy and test performance. **Gradient Boosting** (0.866) and **Random Forest** (0.831) have strong cross-validation accuracy. **KNN** (0.720) also performs well. This aligns with our expectation that non-linear models would perform better due to the data's complexity.

8 Final Model Selection

After evaluating all models, **Random Forest** is selected as the final model. It offers a good balance between performance and computational efficiency, handling class imbalances effectively and providing solid results.

9 Conclusion

During the project, we faced challenges with missing values in the dataset, which required careful handling. Due to the dataset's large size, resampling was applied, and a smaller subset of 10,000 rows was taken to ensure model training. This raised some concerns. First, we noticed that, despite the small subset size, the target variable **IncidentGrade** remained unaffected by preprocessing. However, when reviewing the Kaggle dashboard, we observed that a small number of **IncidentGrade** values were null or missing. Due to the randomness and small subset size, we did not encounter this issue, which is why we did not apply any handling to this variable during preprocessing. By the time we realized this, it was quite late, and we re-ran everything with this in mind. The difference in the model performances was negligible.

We believe 10,000 individuals may not be the ideal sample size, as it was chosen somewhat arbitrarily. To improve, we could try progressively larger data chunks

and analyze their impact on performance. While more data generally improves performance, we expect a point where additional data yields minimal gains, potentially indicating the optimal sample size. We explored various models, and **SVM** struggled with classifying less frequent classes, making it unsuitable for this dataset. In contrast, **Random Forest** showed strong performance, handling class imbalances effectively. Going forward, we should explore advanced non-linear models like **XGBoost** and consider model combinations, such as **Stacking Regression**, to further enhance performance.

In conclusion, throughout this project, we have learned how to handle an unknown dataset, process it, and train models effectively. We discovered that preprocessing is far more important than simply executing the model training code. A deep understanding of the dataset is crucial to prepare the best data possible for the models. Additionally, we have gained valuable experience using machine learning libraries by consulting their documentation and applying them to our analysis. After completing this report, we are motivated to continue applying machine learning algorithms, aiming to improve the sample size and experiment with different algorithms on the current dataset. We are now familiar with the process and excited to continue refining our analysis in this dataset.

10 References

[1] Javier Bejar. *APA*. 2024/25. URL: https://sites.google.com/upc.edu/aprenentatge-automatic/p%C3%A1gina-principal/teor%C3%ADa?authuser=0.

[2] Scikit-learn developers. *Preprocessing Data*. Accessed: 2025-01-12. 2024. URL: https://scikit-learn.org/stable/modules/preprocessing.html.

[3] Scikit-learn developers. *Supervised Learning*. Accessed: 2025-01-12. 2024. URL: https://scikit-learn.org/stable/supervised_learning.html.

[4] Scott Freitas et al. "AI-Driven Guided Response for Security Operation Centers with Microsoft Copilot for Security". In: *arXiv preprint arXiv:2407.09017* (2024).

[5] Scott Freitas et al. *Microsoft Security Incident Prediction*. 2024. DOI: 10.34740/KAGGLE/DSV/8929038. URL: https://www.kaggle.com/dsv/8929038.

[6] Jim Holdsworth and Matthew Kosinski. *What is Incident Response? Minimalist abstract pattern of reflective tiles on a modern architecture*. Security. AI and the Future of Incident Response. Aug. 2024.

[7] Microsoft. *Understanding security alerts*. Accessed: 2025-01-12. 2024. URL: https://learn.microsoft.com/en-us/defender-for-identity/understanding-security-alerts.