06/01/2025

Ma 513 | Hands-on Machine Learning for Cybersecurity : Using Named Entity Recognition (NER) to Identify and Classify Critical Cybersecurity-Related Entities in Text

Alicia Heddadj, Aliyah Parfait
Supervising professor : Atilla Kaan Alkan

ABSTRACT

This project explores the application of Named Entity Recognition (NER) to the domain of cybersecurity, focusing on identifying and classifying key domain-specific entities such as malware types, vulnerabilities and threat actors from technical text documents. While NER models have been successfully applied in general contexts using advanced architectures such as BERT and CRF-based systems, cybersecurity poses unique challenges due to the specialized terminology and rapidly evolving threat environment. As students with limited computational resources, we propose an approach aiming to achieve a balance between model efficiency and accuracy without delving too much into overly complex architectures. Through our exploration to understand fundamentals of careful data preprocessing, model selection, and evaluation, our goal was to demonstrate the viability of NER in supporting cybersecurity applications.

1. Introduction

1.1. Problem Definition

In today's hyper-connected world, cybersecurity has become a critical concern for both organizations and individuals. Protecting all digital infrastructures relies heavily on the timely and accurate identification of key entities, such as malware types, system vulnerabilities, and attack signatures. These tasks are essential for threat intelligence, vulnerability management, and even incident response. So automating the extraction and classification of such information could significantly improve both the speed and efficiency of cybersecurity operations.

Natural Language Processing (NLP), particularly Named Entity Recognition (NER), can provide a powerful solution for handling the vast amount of textual data generated in the domain. However, traditional NER systems can struggle to keep up with the constantly evolving terminology in the field of cybersecurity : the rapid emergence of new threats and attack techniques requires models that are not only precise but also highly adaptable.

This project aims to address this challenge by designing and evaluating a NER system to extract and classify cybersecurity-related entities from unstructured text. The system would use the IOB2 tagging convention, where each token is labeled as the beginning (B-), inside (I-), or outside (O) of a named entity. This structured approach could help ensure a precise identification of entities such as malware types, vulnerabilities, etc. enabling the automated analysis of complex cybersecurity data.

1.2. Project Motivation & Goals

As part of the Ma513 – Hand-on Machine Learning for Cybersecurity course at IPSA, this project was conducted with the aim of providing us students with hands-on experience in applying machine learning techniques to solve practical problems.

The data used in this project is derived from the SemEval-2018 Task 8: "Semantic Extraction from Cybersecurity Reports using Natural Language Processing (SecureNLP)". Unlike other general-purpose datasets, this data is specialized and reflects the constantly evolving language of cybersecurity, making it ideal for testing the adaptability of our NER systems.

For students and professionals without access to advanced computational infrastructure, the challenge also lies in designing a NER system that is effective yet computationally efficient. Considering this, our project seeks to analyze the strengths and weaknesses of existing NER methodologies for cybersecurity applications, and compare them to other classical machine learning models. We will also focus on evaluating the system's performance on our cybersecurity-specific dataset to highlight the potential contributions and limitations of the different approaches.

2.3. Literary Review

Thanks to the rapid shift to deep learning-based approaches, NER has evolved significantly over the last few years as a key method for extracting structured information from unstructured text. Initially reliant on rule-based and statistical models, NER systems were effective but limited in their ability to capture more complex relationships within data. With advancements in machine learning, and particularly deep learning, NER has become increasingly powerful and can now be used across a wide range of domains. In cybersecurity, NER can help automate threat detection and incident response as it can play a crucial role in extracting key entities from security reports.

In general, traditional methods such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF) have been effective but limited in their ability to handle the complexity and evolving language of cybersecurity. [1] More recently, transformer models like BERT (Bidirectional Encoder Representations from Transformers) have set new standards in NER by being able to understand and capture rich context and relationships, making them ideal in handling specialized language. [2] However, their high computational demands can make them impractical for real-time applications, especially for resource-constrained users. [3]

To address this issue of finding balance between efficiency and accuracy in NER for cybersecurity, we will try to compare fine tree classifiers with transformer models like BERT. Fine tree classifiers, such as decision trees or random forests, are usually computationally efficient interpretable, making them suitable for quick and resource-constrained environments. In contrast, BERT excels in understanding complex and domain-specific text but requires much more computational power. [4] By comparing these two approaches, we would like to evaluate the trade-offs between performance and expandability, providing insights into how to balance these factors in real-world cybersecurity applications.

2. Dataset Analysis & Preparation

2.1. Data Exploration

Before diving into model training and evaluation, we need to analyze the structure and distribution of the dataset we are working with. This will help us understand its characteristics and inform us on the necessary preprocessing steps to make.

Data Format

The training, validation and testing datasets are provided in the jsonlines format, where each line contains a JSON object representing a sample. For the first two datasets, each sample consists of a unique ID number and tokens, which are individual words or punctuation marks, along with their corresponding NER tags. The data follows the IOB2 tagging convention, where tokens are labeled based on their role within a named entity ('B-Entity' for the beginning of an entity, 'I-Entity' for inside an entity, 'O' for outside any named entity, and similar tags for actions and modifiers). The testing dataset is given in a similar format, but doesn't present any NER tags as we will try to predict them using our models.

Data Visualization

Visualizing the data can also help us understand it better as we can represent how tags are distributed across the different datasets. Firstly, the graph and table we obtain (cf. figure 1) demonstrate the training and validation datasets present a similar distribution of the NER tags. This consistency between the two is a positive indicator, as it suggests that the validation set will be a reliable benchmark for evaluating the model's performance.

We can also see that our datasets contain seven distinct tags, representing entities, actions and modifiers, as well as the 'O' tag, which denotes tokens outside any of these. One notable observation we can make is in fact the overwhelming prevalence of the 'O' tag, which accounts for over 80% of the labels in both the training and validation sets. This means that most tokens are not within zones of interest, creating a significant class imbalance. If this issue stays unaddressed, the model may overly predict the 'O' tag to minimize errors, thus failing to capture less frequent but much more meaningful labels, such as 'B-Entity' or 'I-Entity'.

To mitigate this problem, we will need to apply effective preprocessing techniques to balance the data. For example, we can try removing 'O' tokens that are surrounded by 'O' tags to help the model focus on the zones of interest. Moreover, we can use the F1-score or the precision as our evaluation metrics, rather than rely on accuracy alone, as they better account the performance of the models in identifying underrepresented classes.

Tag	O	B-Entity	I-Entity	B-Action	I-Action	B-Modifier	I-Modifier
Training Data (%)	82.23	3.64	10.84	1.71	0.44	1.05	0.09
Validation Data (%)	81.67	3.63	11.44	1.64	0.43	1.10	0.09

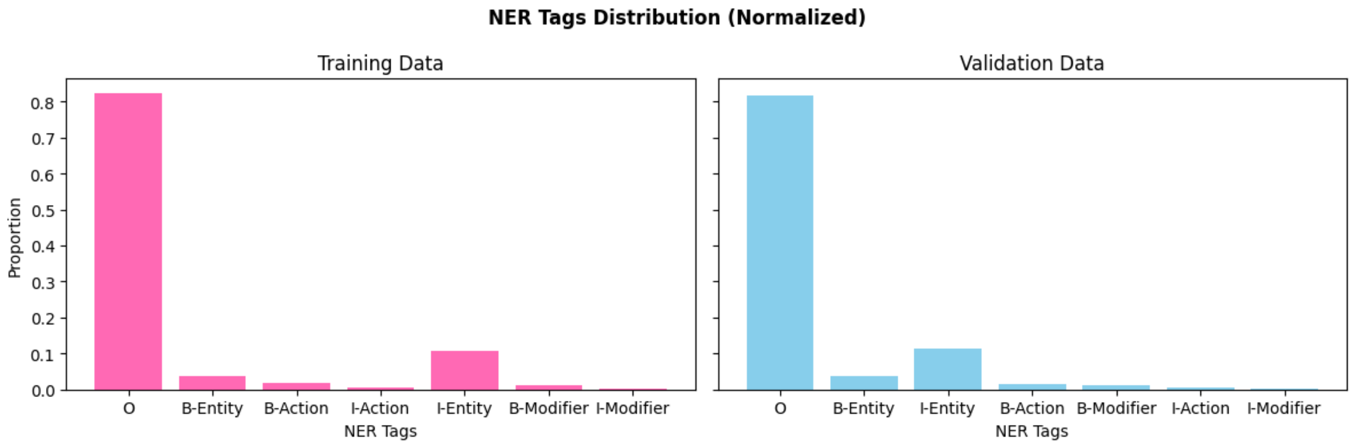


Figure 1. Tags distribution across training & validation datasets

2.2. Preprocessing

In order to prepare the data before developing and training our models, several preprocessing steps were applied to ensure that it is in an optimal format and to address issues inherent to the dataset which were discussed in the previous part such as class imbalance.

Filtering 'O' tags

To prevent the model from overly predicting 'O' to minimize errors, and failing to recognize the more meaningful entities, a filtering function was applied to remove 'O' tags when they were surrounded by other 'O' tags. The reason behind this approach is to focus the model's attention on more meaningful tokens (involved in entities). The filtering function processes each token in a sequence of tokens by checking if it is an 'O' and if both its preceding and following tokens are also 'O'. If so, the token is removed. This ensures that only those 'O' tokens that are potentially irrelevant to entity identification are discarded.

This function was designed to be applied to the training dataset only, the idea being that we would try training our models on both filtered and unfiltered data to confirm that this step is necessary in balancing out the classes distribution.

Extracting Features

Our approach to extracting the data features was to stay simple at first to see how effective and interpretable it could be for classical machine learning algorithms. Instead of employing complex feature representations like word embeddings (Word2Vec, GloVe), we aimed to do something that was functional before anything else. This decision was guided by our initial goal of exploring classical algorithms and their effectiveness on NER tasks before considering more advanced approaches. The extracted features include a combination of token-specific and contextual characteristics. For individual tokens, we considered attributes such as length, which provides insights into whether a token might represent specific entities like long names or complex words. We also included binary features such as the presence of an uppercase or a fully capitalization, which could be more useful for identifying proper nouns, acronyms, or titles. Additionally, the presence of digits and special characters in a token was captured. These features could mainly help the model identify entities like dates, numerical values, or specialized terms, which often contain unique patterns.

To further enhance the model's ability to understand a token within its surrounding sequence, we extracted contextual features from neighboring tokens. For each token, we included features from a configurable context window spanning previous and next tokens. These contextual features include the same attributes as for the target token (length, uppercase, digits, special characters), providing the model with a broader understanding of the sequence structure. This context should be crucial for distinguishing ambiguous tokens and understanding relationships between tokens in multi-word entities.

The choice to rely on simple features was deliberate. Initially, our focus was on exploring classical machine learning algorithms, such as decision trees and boosted ensembles, which excel with structured and interpretable features. These models do not necessarily benefit from high-dimensional inputs, like those produced by word embeddings, and they can perform well with explicitly defined numerical features. Moreover, using basic features allowed us to avoid the computational overload of generating embeddings, making the approach lightweight and accessible for initial experimentation.

3. Model Development

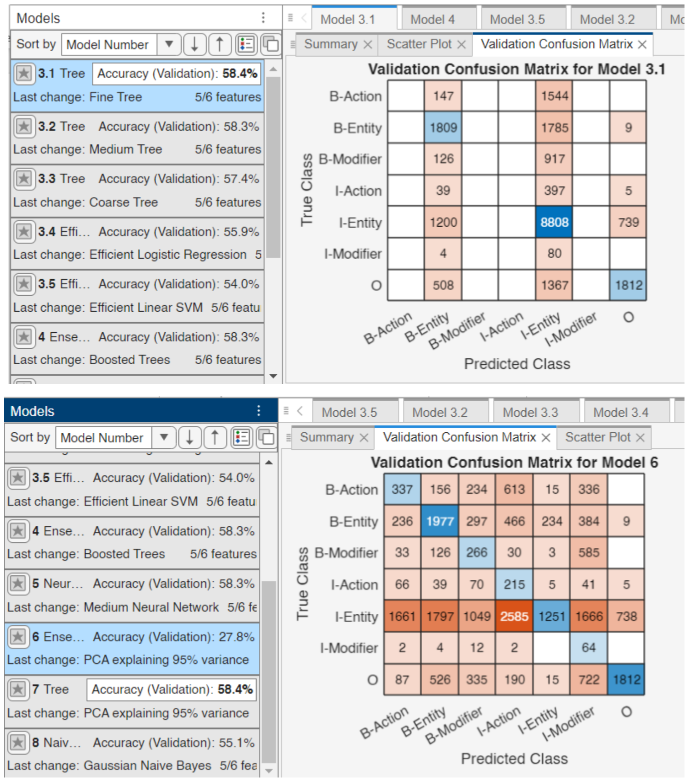
3.1. Model Selection

As mentioned in the previous part, we made the exploration of basic algorithms our first priority. This approach would allow us to build a strong foundational understanding before considering more complex models, such as transformer-based architectures like BERT, which require significantly more resources and data preprocessing.

MATLAB's Classification Learner

After preprocessing our data, we initially decided to explore various classical machine learning models using MATLAB's Classification Learner plug-in. By exporting our features and labels as .csv files, this tool can provide a convenient way to experiment rapidly with a wide range of models, such as linear regression, decision trees, and KNN, with minimal setup. It also allowed us to quickly test different feature subsets and cross-validation configurations, offering a rapid means to gain initial insights into model performance. The primary goal of this step was to identify a promising model that could later be implemented and fine-tuned in Python for more evaluation, alongside a transformer-based approach.

This approach seemed promising initially, as it provided metrics such as accuracy and confusion matrices almost instantly. However, we soon realized that the accuracy metrics provided during training were not truly reflective of the models' prediction performance. For instance, while the confusion matrix highlighted the models' ability to predict the majority class ('O' tags) effectively, they struggled significantly with other, more meaningful entity tags. This discrepancy underscored the need to evaluate models beyond the metrics provided during training.



3.1 Tree Accuracy (Validation): **58.4%**
Last change: Fine Tree 5/6 features

3.2 Tree Accuracy (Validation): 58.3%
Last change: Medium Tree 5/6 features

3.3 Tree Accuracy (Validation): 57.4%
Last change: Coarse Tree 5/6 features

3.4 Effi… Accuracy (Validation): 55.9%
Last change: Efficient Logistic Regression 5

3.5 Effi… Accuracy (Validation): 54.0%
Last change: Efficient Linear SVM 5/6 featu

4 Ense… Accuracy (Validation): 58.3%
Last change: Boosted Trees 5/6 features

Validation Confusion Matrix for Model 3.1

	B-Action	B-Entity	B-Modifier	I-Action	I-Entity	I-Modifier	O
B-Action		147			1544		
B-Entity		1809			1785		9
B-Modifier		126			917		
I-Action		39			397		5
I-Entity		1200			8808		739
I-Modifier		4			80		
O		508			1367		1812

3.5 Effi… Accuracy (Validation): 54.0%
Last change: Efficient Linear SVM 5/6 featu

4 Ense… Accuracy (Validation): 58.3%
Last change: Boosted Trees 5/6 features

5 Neur… Accuracy (Validation): 58.3%
Last change: Medium Neural Network 5/6 fe

6 Ense… Accuracy (Validation): 27.8%
Last change: PCA explaining 95% variance

7 Tree Accuracy (Validation): **58.4%**
Last change: PCA explaining 95% variance

8 Naiv… Accuracy (Validation): 55.1%
Last change: Gaussian Naive Bayes 5/6 fea

Validation Confusion Matrix for Model 6

	B-Action	B-Entity	B-Modifier	I-Action	I-Entity	I-Modifier	O
B-Action	337	156	234	613	15	336	
B-Entity	236	1977	297	466	234	384	9
B-Modifier	33	126	266	30	3	585	
I-Action	66	39	70	215	5	41	5
I-Entity	1661	1797	1049	2585	1251	1666	738
I-Modifier	2	4	12	2		64	
O	87	526	335	190	15	722	1812

Figure 2. Classification learner results

Despite these limitations, the experiment was still valuable. It oriented us towards exploring tree-based classifiers in our Python implementation, as they were the only models in the MATLAB experiments that showed some capacity to predict the more important classes in the confusion matrix. This early-stage exploration served as a useful first step in narrowing down our focus, allowing us to prioritize tree-based models for further experimentation in Python. So while the Classification Learner tool did not lead to a direct solution, we thought it interesting to explore as it provided a rapid way to identify promising directions, making it a practical approach for quickly gauging model performance in the initial stages of our project.

Exploring Tree Classifiers

Our exploration of tree-based classifiers began with the simple Decision Tree Classifier, specifically a fine tree, as an initial step to establish two key experimental parameters: the optimal context size and the necessity of filtering out the 'O' tags. By using a fine tree, we were able to quickly test different configurations and observe how the model performed in predicting meaningful NER tags compared to the dominant 'O' class. This step gave us a baseline understanding and oriented our experiments in subsequent stages.

Building on the insights gained from the fine tree, we advanced to the Random Forest Classifier, an ensemble method that combines the predictions of multiple decision trees to reduce overfitting and improve robustness. The randomness introduced during the tree-building process made Random Forest particularly effective in exploring how the features interacted across different samples, and its inherent feature importance metrics helped us refine our feature set a bit further.

Finally, inspired by how the ensemble tree gave us the best results on Matlab, we ended up chosing the LGBMClassifier (LightGBM) as our primary model due to its efficiency and performance advantages over the previous models. LightGBM, which is a gradient boosting framework, builds decision trees iteratively, with each tree attempting to correct the errors of the previous one. This process seemed to make it interesting for our NER task, which involved high-dimensional feature sets and imbalanced data. LightGBM's ability to handle categorical and numerical data effectively, while offering fine control over hyperparameters, allowed us to fine-tune the model for optimal performance.

By progressing through these tree-based classifiers, we wanted to establish a natural progression from simpler models to more sophisticated ones, each step building on the insights of the last.

Basic BERT model

All while exploring tree-based classifiers, we also turned our attention to developing a BERT-based model. As we mentioned in our literary review, models such as BERT have been establishing themselves as important tools for state-of-the-art NLP tasks due to their deeper contextual understanding of language.

We decided to explore BERT's capabilities for NER hoping that its richer language modeling could provide a more nuanced understanding of the entities, actions, and modifiers within our dataset. The primary advantage of using BERT over traditional machine learning methods would lie in its ability to capture long-range dependencies and contextual relationships between words, something that simpler models like decision trees or Random Forest struggle to do.

However, one of the significant challenges we knew we would have to face with BERT was the computational resources it demands. We knew that training such a model requires a significant amount of GPU power. Although we didn't have access to high-end resources, having old laptops and working on Google Colab, we still thought it interesting to try experimenting with different parameters and fine-tune the model to see how it could perform in less resourceful environments.

3.2. Experimental Settings

Tree classifiers

In the exploration of tree-based classifiers, we had the opportunity to fine-tune several hyperparameters to optimize their performance for our NER task.

For the Decision Tree Classifier, the main idea was to help us quickly determine the ideal context size, as well as confirm that our filtering of the ‘O’ tags was useful. We experimented with some of the key parameters, particularly the maximum depth of the tree, before trying the same tree for different context sizes and with or without filters.

Our goal when we moved to the Random Forest classifier was to confirm the clues we observed using the Decision Tree, experimenting not as much with the parameters but with the different conditions.

Once we got to develop the LGBM classifier, we focused a lot more on finding the parameters to optimize the model’s performance. The hyperparameter tuning expanded from the maximum depth of the trees, to the number of estimators (number of trees), which could directly impact the model's accuracy and robustness, with more trees generally improving performance, though coming with the cost of computational efficiency. Additionally, we could also try experimenting with the learning rate, since a smaller one would necessitate more trees to maintain performance, while a higher rate could cause the model to converge too quickly, risking overfitting.

Each of the classifiers we used provided unique strengths, helping us fine-tune the right combination of parameters to find the most effective model configuration for our data, though the process was time-consuming due to the extensive computational resources required for experimentation.

BERT

The process of integrating BERT involved fine-tuning the pre-trained bert-base-cased model specifically for our task so we started by configuring the model to recognize our seven distinct labels. While BERT’s default tokenizer handles complex tokenization well, we faced an additional challenge when it came to aligning these tokenized inputs with our labels. For each token in the input sequence, we ensured that the labels were aligned correctly by considering the possibility of token splitting, which is common in BERT's tokenization process, but could hinder the prediction performance. This required a careful adjustment of the tokenization pipeline to avoid mismatches between tokens and their corresponding entity labels.

To enhance the model’s performance, we tried integrating feature extraction techniques alongside the tokenization. While BERT is robust in capturing context, we wanted to explore if our additional custom features might offer complementary insights.

Moreover, training the BERT model effectively came with a lot of difficulties : the computational requirements for BERT were quite important (even for a basic model), and our limited access to GPU resources made the training process extremely slow and inefficient. Each experiment, whether adjusting the learning rate or changing the number of epochs, required significant time and computational power, which made it hard to perform extensive tuning.

4. Results Analysis

Our initial results from using the fine tree and random forest classifiers were somewhat disappointing, as they exhibited limited predictive performance. However, these early experiments were invaluable for refining our approach, allowing us to focus on key factors such as the context size and the presence of filtering for the 'O' tags. This initial phase of experimentation helped narrow our focus, guiding us toward more promising configurations. As shown in Figure 3 (results table), the findings highlighted that the model’s performance improves when the classes are better balanced, a benefit achieved through filtering. Additionally, the experiments

Fine Tree			
	Precision	Recall	F1 Score
No context ; No filtering	0.0	0.0	0.0
Context = 5 ; No filtering	0.030	0.024	0.027
Context = 8 ; No filtering	0.025	0.020	0.022
No context ; Filtering	0.027	0.103	0.042
Context = 5 ; Filtering	0.040	0.23	0.068
Context = 8 ; Filtering	0.035	0.21	0.060
Random Forest			
	Precision	Recall	F1 Score
No context ; Filtering	0.023	0.27	0.042
Context = 5 ; Filtering	0.057	0.34	0.098
Context = 8 ; Filtering	0.054	0.23	0.088

Figure 3. Result Table for Trees

underscored the importance of selecting an appropriate context size: while neglecting context severely reduced prediction quality, using an excessively large context did not yield significant improvements either. Ultimately, after testing various configurations, we decided that filtering the 'O' tags and using a context size of 5 would be the most effective setup for future developments. This iterative process allowed us to quickly identify these key parameters, saving us from investing time in more complex models at the outset.

Light Gradient-Boosting Machine			
	Precision	Recall	F1 Score
Context = 5 ; Filtering	0.066	0.25	0.104
Increasing depth (15 → 30)	0.060	0.24	0.096
Decreasing depth (5 → 30)	0.061	0.21	0.095
Modifying learning rate	0.042	0.08	0.055
100 → 250 estimators	0.065	0.27	0.105

Figure 4. Result Table for LGBM

The LGBM model provided a noticeable improvement in performance compared to the previous classifiers. However, while the results were better than those of the fine tree and random forest models, they were still relatively modest. Modifying key parameters such as depth and number of estimators produced some improvements, but results plateaued after a certain threshold. Increasing the number of estimators beyond 250 also started to show diminishing returns, which only emphasize the need for careful parameter tuning to avoid overfitting.

The most consistent improvements came from adjusting the number of estimators, which had a positive impact, but further increases required balancing to prevent overfitting. Although it was a bit time-consuming, this trial-and-error process proved effective in moving towards a more robust model, especially since LGBM remains an easier model to train than BERT.

Finally, our trials with BERT, despite its potential for better contextual understanding, proved disappointing in this particular setup. The BERT model, even with basic tokenization and custom feature extraction, struggled to produce meaningful results. The initial tokenization showed no predictions at all, and even after integrating custom feature extraction, the performance remained low. These disappointing results could be attributed to several factors, including the complexity of aligning tokenized inputs with labels and the resource-intensive nature of training BERT, which proved challenging given our limited computational resources.

Despite these setbacks, we recognize that the full potential of BERT could be realized with improved tokenization strategies, additional feature engineering, and more computational power, making it a valuable avenue for future research.

BERT			
	Precision	**Recall**	**F1 Score**
Tokenization	0.0	0.0	0.0
Tokenization + Custom feature extraction	0.039	0.015	0.021

Figure 5. Result Table for BERT

5. Discussion & Conclusion

This project set out to explore the application of NER to cybersecurity, focusing on extracting key entities from text using both classical and advanced machine learning methods. The ultimate goal was to compare the computationally efficient yet simple tree-based classifiers with the more complex but resource-intensive BERT model. By balancing efficiency, accuracy, and adaptability, we aimed to demonstrate the practical applicability of NER in cybersecurity contexts.

Achievements and Insights

Despite the constraints of limited computational resources and a specialized dataset, we successfully implemented and compared multiple approaches. Tree-based classifiers provided a practical entry point to the task, helping us experiment with parameters like context size and filtering of 'O' tags. These early steps enabled us to identify optimal configurations, such as the necessity of filtering and the importance of context size, which proved crucial for improving performance. The LGBM classifier emerged as the most promising tree-based model, demonstrating consistent improvements over simpler decision trees and random forests. However, its performance rapidly plateaued despite hyperparameter tuning, indicating limitations in its ability to capture the complexity of cybersecurity-specific text.

On the other hand, BERT offered the potential for deeper contextual understanding but we really struggled to deliver meaningful results under our constrained setup. Aligning tokenized inputs with labels posed a significant challenge, and the model's resource requirements limited our ability to conduct extensive experimentation.

Through this project, we gained a deeper understanding of the balance between simplicity and complexity, as well as efficiency and accuracy. While classical models allowed us to experiment rapidly, BERT underscored the importance of computational power for leveraging state-of-the-art NLP techniques.

Lessons Learned

Balancing the classes in our dataset proved to be a crucial step in addressing the overwhelming dominance of the 'O' tags, which often overshadowed the more meaningful entity predictions. Filtering these tags significantly improved the model's ability to focus on relevant entities, demonstrating the importance of handling class imbalance in NER tasks. Additionally, determining an optimal context size played a key role in enhancing performance; a moderate context size effectively captured necessary contextual information without burdening the model with irrelevant data.

In terms of model selection, tree-based classifiers emerged as efficient and interpretable solutions, providing a practical foundation for our experiments. However, the promise of deep learning became evident with BERT, which showcased its potential to capture nuanced relationships within the text, albeit at the cost of significant computational resources.

Future Improvements

The future improvements in our approach could include improving and enhancing the tokenization processes for BERT by refining token-label alignment, maybe through advanced techniques like word-piece tokenization combined with better feature integration. This could address some of the challenges we encountered in adapting the model to our dataset. Additionally, exploring hybrid models that merge the efficiency of tree-based classifiers with the contextual depth of transformers could offer a balanced solution, leveraging the strengths of both methodologies.

Expanding feature extracting efforts to incorporate domain-specific patterns or syntactic information may also boost model effectiveness, providing additional context for predictions. Increasing the size and diversity of the dataset with more labeled samples could also enhance generalization, particularly for the underrepresented tags. Finally, addressing resource constraints could involve experimenting with lighter transformer models, such as DistilBERT which we didn’t get to explore, to strike a better balance between computational efficiency and performance, making the approach more accessible to users with limited resources.

Conclusion

This study illustrates the challenges and opportunities of applying NER to cybersecurity. While tree-based models provide a reliable baseline, the contextual richness of transformers like BERT holds promise for capturing complex relationships in domain-specific text. However, resource constraints remain a significant barrier to fully leveraging these advanced techniques. By refining tokenization strategies, expanding datasets, and exploring hybrid approaches, future research can unlock the potential of NER to support critical cybersecurity tasks. Although we did not obtain major results, this project the groundwork for further exploration into balancing efficiency and accuracy in NER applications.

6. References

[1] **"SemEval-2018 Task 8: Semantic Extraction from Cybersecurity Reports using Natural Language Processing (SecureNLP)"**
Mengnan Du, Siddharth Mudgal, Xia Hu, Weiwei Sun, Jizhong Han, and Jing Gao, 2018

[2] **"Named Entity Recognition in Cybersecurity Literature using BERT Models"**
Q. Chen, Z. Liu, X. Liu, Y. Wang, and J. Sun, 2020

[3] **"Exploring Lighter Transformers for Named Entity Recognition: A Resource-Efficient Perspective"**

Mohamed Amine Ferrag, Mthandazo Ndhlovu, Norbert Tihanyi, Lucas C. Cordeiro, Merouane Debbah, Thierry Lestable, and Narinderjit Singh Thandi, 2023.

[4] **"Named Entity Recognition: A Complete Guide"**
Docsumo Team, 2023