جامعة خليفة
Khalifa University

COSC 435: Introduction to AI/ML for Cybersecurity Project

By: Amr Tamer - 100059484
Ahmed Khalid Aljallaf - 100053634
Elyas Saeed – 100061210
Humaid Alhammadi – 100059158

Fall 2024

Table of Contents

Abstract

This project explored various Machine Learning and Deep Learning models when tackling phishing website detection by utilizing the URL, title of the website and its metadata, if available. The models included Logistic regression, KNN, XGBoost as the traditional Machine Learning algorithms as well as more complex Deep Learning models such as Encoder-Only Transformers, LSTMs and MLPs. The models were trained and later evaluated on both the original dataset and an augmented version that purposefully tested the models' robustness at detecting slightly manipulated inputs, that could be used to bypass the models. This comprehensive report details the dataset preparation, model training, model evaluation as well as adversarial evaluation on the models. It also includes a conclusion and a roadmap to further enhance this project.

Introduction

Phishing attacks are a persistent cybersecurity threat, often performed by producing deceptive websites with URLs that may, upon first look, seem legitimate. This project attempts at resolving this problem by training models on pre-processed datasets, leveraging machine learning and deep learning methodologies. The models are subsequentially evaluated by various metrics, with augmented adversarial examples, that were produced by three different transformations, also evaluated. This pipeline will provide a comprehensive analysis on the current ability of these models at detecting phishing attacks.

Data Preparation

The dataset preparation process was pivotal in ensuring the effectiveness of the models. This section outlines the steps followed, including cleaning, tokenization, metadata augmentation, and splitting of the dataset.

Dataset Overview

The dataset included URLs, the website's accompanying title and some metadata information about the website. The dataset preparation aimed at producing a structured, stratified dataset, with the textual data properly tokenized. Tokenization would allow the models to have to process less data, significantly decreasing memory and computational requirements, while improving sub-word understanding.

Cleaning and Metadata Augmentation

To enhance the dataset, we completed the following:

1. **Cleaning and Metadata Augmentation**

- Metadata Normalization: The numerical metadata columns were normalized to ensure consistent scaling across features. These normalized columns were then packed into tensors for efficient processing during model training.
- Stratification Label: A new column named, stratify_label, was added to facilitate stratified splitting, ensuring the train, validation, and test sets maintained similar class distributions. This will allow the model to experience enough examples of both class types consistently across all training runs.
- Upon research of the dataset, the team learnt that this dataset is well cleaned. Due to the comprehensive amount of data and well-structured state of the dataset, additional feature engineering was not performed. This decision reduced processing complexity while maintaining focus on leveraging the existing metadata effectively.

2. **Parallelized Processing**

- As the dataset was quite large, parallel computation was utilized throughout the dataset preparation process, with multiple CPU cores working simultaneously. This significantly reduced preparation time.

3. **Tokenization**

- Raw text inputs, including URLs and titles, were tokenized using a custom built tokenizer. This process transformed text data into sequences of token indices.
    - Example transformation:
        - Original Input: <URL>https://example.com <TITLE>Example Title
        - Tokenized Output: https://example.com Example Title
- Where <URL> and <TITLE> are additional special tokens added to the tokenizer that signal to each section.
- A maximum sequence length of 256 tokens was enforced during tokenization, truncating inputs if necessary. This was done after analyzing the dataset and observing that < 1% of the dataset would be affected, deeming it to be reasonable.
- The tokenizer was trained on the dataset itself, which is usually not ideal practice because it will overfit the tokens on the dataset on not general data. However, the tokenization algorithm was incapable of parallelization and could take a very long time to complete. Given more time, the team would have elected to produce a tokenization vocabulary produced from general URL examples.

4. **Stratified Splitting**

- The dataset was split into training, validation, and test sets while preserving class distributions:
    - Train-Validation Split Ratio: 80-20

    - Validation-Test Split Ratio: 50-50
- The stratified splitting ensured a balanced representation of classes across the splits, critical for robust evaluation. This preparation formed this split:
    - Training Set: 188636 examples
    - Validation Set: 23579examples
    - Test Set: 23580 examples

5. Dataset Format

The processed dataset included the following elements:

- Tokenized Inputs: Sequences of token indices, ready for direct use in PyTorch models.
- Attention Masks: Binary masks indicating valid tokens (1) and padding (0) within each sequence, ensuring models ignored padded regions during processing.
- Metadata Tensors: Normalized and packed numerical features representing auxiliary information to complement the textual inputs.
- Labels: Target labels for supervised learning tasks, indicating the classification outcomes.

These elements were systematically structured and saved in a format compatible with PyTorch, facilitating seamless integration into the modeling pipeline.

6. Decision Against Additional Feature Engineering

- While feature engineering can enhance model performance, it was deemed unnecessary for this project. The metadata already provided sufficient numerical features in the dataset, which were normalized and incorporated directly. Introducing additional engineered features was considered redundant and would have added complexity without clear benefits. It was decided by the team that we would not perform any data engineering or augmentation until we observe model performance.
- This structured and efficient preparation ensured the dataset was ready for model training, testing, and adversarial evaluations.

Methodology

This section describes the step-by-step approach used in the project, covering data preparation, model architecture, training pipeline, and evaluation.

Dataset Preparation

Data Cleaning and Normalization

- The raw dataset consisted of URLs, titles, and additional numerical metadata.
- Numerical metadata was normalized to ensure consistent feature scaling, while text inputs (URLs and titles) were tokenized using a custom tokenizer developed for this project.

- Attention masks were generated to distinguish valid tokens from padding, ensuring the models processed meaningful parts of the input.

Byte-Pair Encoding

- A Byte Pair Encoding (BPE) tokenizer was built from scratch to tokenize the URLs and titles. The tokenizer split strings into sub-word units, balancing vocabulary size and granularity. A vocabulary of 10000 tokens was generated, with special tokens added to improve model understanding.
- The tokenized inputs were packed into a format ready for PyTorch, with sequences truncated or padded to a maximum length.

Metadata Handling

- Metadata features were pre-processed by normalization and packed into tensors to complement the tokenized textual inputs.

Label Encoding

- Target labels for supervised learning were processed and encoded, allowing the models to perform binary classification effectively.

Model Architectures

A choice was made between multiple architectures, these are:

Logistic Regression

- Purpose: A lightweight and interpretable baseline model for binary classification, used as a benchmark for comparing more complex architectures.
- Input: Tokenized Inputs, using an embedded layer and Metadata Features, normalized and concatenated with tokenized methods
- Architecture: A single linear layer for classification, with input features flattened and passed to the linear layer.
- Output: A single node output representing logits for binary classification, with the use of sigmoid activation for probability computation
- Key Features:
  - Lightweight and interpretable.
  - Loss Function: Binary Cross-Entropy with logits.
  - Serves as a baseline for more advanced models.

Multilayer Perceptron (MLP)

- Purpose: A dense neural network for classification, with greater capacity to learn non-linear relationships than logistic regression.
- Input: Tokenized Inputs, using an embedded layer and Metadata Features, normalized and concatenated with tokenized methods
- Architecture: Configurable hidden layers with GELU activation, dropout for regularization, and fully connected layers, with a final linear layer mapping features to output classes.
- Output: Multi-class probabilities computed using softmax.
- Key Features:
    - Configurable depth and number of units per layer.
    - Loss Function: Cross-Entropy Loss.
    - Versatile and well-suited for tabular data combined with tokenized inputs.

XGBoost

- Purpose: A tree-based gradient boosting model, ideal for structured data.
- Input: Flattened tokenized inputs being treated as numerical features directly combined with metadata token outputs.
- Architecture: Gradient-boosted decision trees with hyperparameters like max depth, number of estimators, learning rate, and more, handling non-linear feature interactions effectively.
- Output: Probabilities for each class via the XGBoost classifier.
- Key Features:
    - Effective for structured, tabular data.
    - Loss Function: Binary Log Loss (via XGBoost's API).
    - Supports early stopping during training.

k-Nearest Neighbors (k-NN)

- Purpose: A simple, non-parametric, distance-based classifier.
- Input: Flattened tokenized inputs being treated as numerical features directly combined with metadata token outputs..
- Architecture: Distance metrics configurable (minkowski, cosine, etc.) and weighted strategies (uniform, distance) allow tuning the influence of neighbors depending on the dataset.
- Output: Class probabilities based on neighbors' labels.
- Key Features:
    - No training phase, making it suitable for quick experimentation.

    - High memory usage and slower inference for large datasets.

Transformer

- Purpose: A powerful sequence model that leverages attention mechanisms to understand complex patterns in tokenized inputs.
- Input: Tokenized inputs augmented and embedded with positional encoding to represent sequence order, which is then processed via a metadata layer with sequence embeddings.
- Architecture: Stacked transformer blocks, each consisting of multi-head self-attention layers and feed-forward layers with dropout and normalization, with a global pooling layer to aggregate sequence-level information.
- Output: Class probabilities using softmax for multi-class classification.
- Key Features:
    - Scalable to long sequences.
    - Configurable hyperparameters like the number of blocks, heads, and embedding size.
    - Loss Function: Cross-Entropy Loss.

LSTM

- Purpose: A recurrent neural network (RNN) for sequence modeling, capable of learning order-dependent relationships in tokenized inputs.
- Input: Tokenized inputs augmented and embedded with into sequences represent sequence order, which is then optionally concatendated with metadata features handling variable lengths efficiently.
- Architecture: Bidirectional LSTM layers with configurable depth and dropout, global average pooling over sequence outputs, and a metadata processing layer for concatenation with LSTM outputs.
- Output: Multi-class classification probabilities.
- Key Features:
    - Captures sequential dependencies effectively.
    - Loss Function: Cross-Entropy Loss.
    - Limited scalability compared to Transformers for very long sequences.

Training Pipeline

The training pipeline was designed to ensure efficient and accurate model optimization.

The training loop was implemented to include batching, gradient accumulation, mixed-precision training, and validation steps, with a scheduler being configured for learning rate adjustments, with early stopping mechanisms integrated for overfitting prevention. Models were evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC-ROC, and test loss (for generalization reasons). Optuna was used to tune hyperparameters for each model. Search spaces included parameters such as vector length, dropout rates, and hidden layer configurations for some of the PyTorch models.

Three types of adversarial perturbations were generated to test model robustness: replacing characters in URLs with visually similar alternatives (character replacement),randomly altering character cases or adding special symbols (case alteration, and randomly altering characters with visually similar Unicode equivalents (Random Unicode Character Replacement)

Adversarial examples were tokenized and evaluated against trained models, and like the original dataset's model results, results such as accuracy, precision, recall, and F1 were computed for each adversarial test. This comprehensive methodology ensured a robust framework for experimentation and evaluation, enabling a systematic comparison of model performance across various architectures and conditions.

Results

This section presents the results of the various models trained and evaluated on the prepared dataset. Each model was tested on clean data and then subjected to the adversarial attacks stated below.

The following table summarizes the performance of the models:

Model	Original Metrics	Similar Conversions Adversarial Metrics	Casing and Symbol Addition Adversarial Metrics	Random Unicode Character Replacement Adversarial Metrics
Transformer Model	Loss: 0.0058 Accuracy: 99.90% Precision: 1.00 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Loss: 0.0104 Accuracy: 99.73% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Loss: 0.0056 Accuracy: 99.89% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Loss: 0.0162 Accuracy: 99.50% Precision: 0.99 Recall: 1.00 F1 Score: 1.00
LSTM Model (Best)	Loss: 0.0024 Accuracy: 99.98% Precision: 1.00 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Loss: 0.0024 Accuracy: 99.98% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Loss: 0.0024 Accuracy: 99.98% Precision: 1.00 Recall: 1.00 F1 Score 1.00	Loss: 0.0024 Accuracy: 99.98% Precision: 1.00 Recall: 1.00 F1 Score: 1.00
MLP Model	Loss: 0.0072 Accuracy: 99.82% Precision: 1.00	Loss: 0.5172 Accuracy: 87.28% Precision: 0.98	Loss: 0.0183 Accuracy: 99.42% Precision: 1.00	Loss: 0.5848 Accuracy: 80.63% Precision: 0.99

	Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Recall: 0.80 F1 Score: 0.88	Recall: 0.99 F1 Score: 0.99	Recall: 0.67 F1 Score: 0.80
XGBoost SciKit Model	Loss: 0.4759 Accuracy: 100.00% Precision: 1.00 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Loss: 0.5013 Accuracy: 92.07% Precision: 1.00 Recall: 0.86 F1 Score: 0.93	Loss: 0.4762 Accuracy: 99.98% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Loss: 0.4918 Accuracy: 94.23% Precision: 1.00 Recall: 0.90 F1 Score: 0.95
KNN SciKit Model	Loss: 0.3208 Accuracy: 99.26% Precision: 0.99 Recall: 1.00 F1 Score: 0.99 AUC: 0.99	Loss: 0.4322 Accuracy: 87.88% Precision: 0.93 Recall: 0.86 F1 Score: 0.89	Loss: 0.3218 Accuracy: 99.22% Precision: 0.99 Recall: 1.00 F1 Score: 0.99	Loss: 0.5462 Accuracy: 75.58% Precision: 0.87 Recall: 0.68 F1 Score: 0.76
Logistic Regression Model	Loss: 0.0286 Accuracy: 99.48% Precision: 0.99 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Loss: 0.6620 Accuracy: 80.12% Precision: 0.96 Recall: 0.68 F1 Score: 0.80	Loss: 0.0677 Accuracy: 97.75% Precision: 0.99 Recall: 0.97 F1 Score: 0.98	Loss: 1.3223 Accuracy: 63.52% Precision: 0.97 Recall: 0.37 F1 Score: 0.54

Analysis of Results

The results obtained across the models demonstrate overall robust performance, but the adversarial performance was quite a bit different between models This is a result of both the strengths and weaknesses of different models in handling possible real-world variations in data.

XGBoost model showed high high performance on the original dataset and moderate robustness to adversarial transformations. For similar conversions, it achieved an F1 score of 0.93. For random Unicode replacements, it maintained an F1 score of 0.95, as a result of the model's adaptability to different types of inputs. Hence, it had a better performance than previous models.

The KNN model displayed acceptable performance on the original dataset but was highly sensitive to adversarial attacks: Under random Unicode replacements, the F1 score dropped to 0.76, indicating difficulty in handling high-dimensional adversarial variations. However, for simpler transformations, it performed similarly to XGBoost. Logistic Regression model, while being an interpretable model, struggled significantly with adversarial transformations. For similar conversions, its accuracy dropped to 80.12%, with an F1 score of 0.80.For random Unicode replacements, the metrics declined drastically with an accuracy of only 63.52% and F1 of only .54.

The Transformer model consistently achieved near-perfect metrics across all scenarios, with great performance in most adversarial examples. However, with Unicode adversarial dataset, the F1 and Accuracy was reduced. Regardless, it was still a better choice, with a perfect AUC in all conditions as opposed to some other models. The LSTM model achieved amazing performance, with no observed degradation under any transformation induced. It performed better than all remaining models. It consistently achieved an AUC of 1.00, and its loss, accuracy, precision, recall, and F1 score near-perfect, making it a perfect fit for all conditions.

The MLP model showed strong performance on the original dataset but struggled significantly with certain adversarial transformations. For similar conversions, its metrics dropped noticeably with an accuracy of 87.28%/ F1 score of 0.88.. For random Unicode replacements, the performance deteriorated further with an accuracy of 80.63% and an F1 score of 0.80 highlighting its susceptibility to noisy inputs. For other cases, it managed to maintain a high performance similar to that of the original dataset.

Discussion

Transformer and LSTM models proved to be the most robust, due to their sequence modeling features aiding the robustness of models. This is due to ability to learn complex relationships in data, especially our tokenized data. The MLP model, while good on clean data, lacked resilience under adversarial transformations, as a result of their limited sequential modeling capability. XGBoost and KNN demonstrated moderate performance especially with these adversarial datasets, with a better performance for XGBoost as a result of its gradient boosting architecture. Logistic Regression, being a simpler model, failed to generalize well to adversarial transformations, showing its true limitations in handling advanced cases.
The inclusion of normalized metadata features was a reason why models like XGBoost and MLP performed pretty well on clean data. However, these models struggled when adversarial changes modified token inputs, leaving them effectively in the dark. Likewise, like XGBoost and KNN could not utilize the token embedding layers at all which unfortunately allowed thme to use only token indices. This limitation highlights the limitations of such ready-made models for enhanced performance as opposed to ones made from scratch, as you are not able to use an end-to-end pipeline to extract the full performance of this model.

Overall, these results greatly showed the effectiveness of different models with different capabilities in handling robust, adversarial settings.

Division of Work

The project was a collaborative effort, with each team member contributing to specific aspects of the pipeline.

- Amr:
    - Designed and implemented the Transformer model and the custom BPE tokenizer for handling URL and metadata inputs. Worked on the KNN , including its integration and testing with other models.
- Elyas:
    - Conducted the data analysis above to understand the dataset's structure and characteristics before passing it on to Ahmed. Also, and implemented the LSTM model with data and grading.
- Humaid:
    - Managed Dataset Preparation, as stated in the section above while also developing MLP and Logistic Regression models. Packaged and preprocessed metadata features for integration into the models.

- Ahmed:
    - Focused on the implementation Hyperparameter Finetuning using Optuna for optimizing model performance. In addition, he designed and implemented the XGBoost model, ensuring its compatibility with the pipeline and performing detailed evaluations.

This division ensured that all aspects of the project were addressed effectively, fairly each team member's strengths and areas of expertise.

Conclusion

This study explored the performance of various machine learning models in classifying URLs and metadata while additionally evaluating their robustness to adversarial transformations. Transformer and LSTM models were the most reliable , with amazing metrics against all conditions above and a slight edge to LSTM.
However, their models unfortunately did not show as good of a performance in adversarial conditions. Limitations in Scikit-learn meant that XGBoost and KNN were not able to accept the layers meant for token embedding. Due to this, this work shows just how important an efficient end-to-end pipeline model for model training can do to the results of such models.

Overall, it shows the importance of models that act well under adversarial conditions. This work can be further enhanced via the use of well-architected hybrid models and the use of adversarial training.