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

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.

1. **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.

2. Cleaning and Metadata Augmentation

To ensure satisfactory model performance**:**

- **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.

3. 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.

4. 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.

5. 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

6. 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.

7. 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.

1. 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.
- **Tokenization**
    - 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.

2. Model Architectures

Below is a detailed breakdown of the implemented model architectures, including their purpose, input requirements, and key design decisions:

a) Logistic Regression

- **Purpose**: A baseline model for binary classification, used as a benchmark for comparing more complex architectures.
- **Input**:
    - **Tokenized Inputs**: Embedded using a learned embedding layer.
    - **Metadata Features**: Numerical metadata features, normalized and concatenated with tokenized inputs if provided.
- **Architecture**:
    - A single linear layer for classification.
    - Input features are flattened and passed to the linear layer.
- **Output**:
    - A single node output representing logits for binary classification.

    - Sigmoid activation applied to logits during prediction to compute probabilities.
- **Key Features**:
    - Lightweight and interpretable.
    - Loss Function: Binary Cross-Entropy with logits.
    - Serves as a baseline for more advanced models.

b) Multilayer Perceptron (MLP)

- **Purpose**: A dense neural network for classification, with greater capacity to learn non-linear relationships than logistic regression.
- **Input**:
    - **Tokenized Inputs**: Embedded and flattened into a single vector.
    - **Metadata Features**: Concatenated with tokenized inputs if provided.
- **Architecture**:
    - Configurable hidden layers with GELU activation, dropout for regularization, and fully connected layers.
    - Final linear layer maps 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.

c) XGBoost

- **Purpose**: A tree-based gradient boosting model, ideal for structured data.
- **Input**:
    - **Flattened Tokenized Inputs**: Treated as numerical features.
    - **Metadata Features**: Directly combined with token inputs.
- **Architecture**:
    - Gradient-boosted decision trees with hyperparameters like max depth, number of estimators, learning rate, and more.
    - Handles 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.

d) k-Nearest Neighbors (k-NN)

- **Purpose**: A non-parametric, distance-based classifier.

- **Input**:
    - **Flattened Tokenized Inputs**: Treated as numerical features.
    - **Metadata Features**: Concatenated with token inputs if provided.
- **Architecture**:
    - Distance metrics configurable (minkowski, cosine, etc.).
    - Weighting strategies (uniform, distance) allow tuning the influence of neighbors.
- **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.

e) Transformer

- **Purpose**: A powerful sequence model that leverages attention mechanisms to understand complex patterns in tokenized inputs.
- **Input**:
    - **Tokenized Inputs**: Embedded and augmented with positional encodings to represent sequence order.
    - **Metadata Features**: Processed via a metadata layer and optionally concatenated with sequence embeddings.
- **Architecture**:
    - Stacked transformer blocks, each consisting of:
        - Multi-head self-attention layers.
        - Feed-forward layers with dropout and normalization.
    - 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.

f) LSTM

- **Purpose**: A recurrent neural network for sequence modeling, capable of learning order-dependent relationships in tokenized inputs.
- **Input**:
    - **Tokenized Inputs**: Embedded and packed into sequences, handling variable lengths efficiently.
    - **Metadata Features**: Optionally concatenated after pooling sequence features.

- **Architecture**:
  - Bidirectional LSTM layers with configurable depth and dropout.
  - Global average pooling over sequence outputs.
  - 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.

3. Training Pipeline

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

- **Custom Training Loop**
  - The training loop was implemented to include batching, gradient accumulation, mixed-precision training, and validation steps.
  - A scheduler was configured for learning rate adjustments, with early stopping mechanisms integrated for overfitting prevention.
- **Evaluation Metrics**
  - Models were evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC-ROC.
  - The test loss was tracked to ensure generalization.
- **Hyperparameter Optimization**
  - 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.

4. Adversarial Testing

- **Adversarial Examples**

Three types of adversarial perturbations were generated to test model robustness:

  - **Character Replacement:** Replacing characters in URLs with visually similar alternatives.
  - **Case Alteration and Symbol Injection:** Randomly altering character cases or adding special symbols.
  - **Unicode Substitution:** Replacing characters with visually similar Unicode equivalents.

- **Testing Procedure**

- Adversarial examples were tokenized and evaluated against trained models.
- Metrics 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 adversarial attacks, specifically:

1. **Similar Conversions Adversarial Metrics**: Substituting characters with visually similar alternatives.
2. **Casing and Symbol Addition Adversarial Metrics**: Introducing variations in casing and appending random symbols.
3. **Random Unicode Character Replacement Adversarial Metrics**: Replacing characters with random Unicode characters.

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	Loss: 0.0058	Loss: 0.0104	Loss: 0.0056	Loss: 0.0162

	Accuracy: 99.90% Precision: 1.00 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Accuracy: 99.73% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Accuracy: 99.89% Precision: 1.00 Recall: 1.00 F1 Score: 1.00	Accuracy: 99.50% Precision: 0.99 Recall: 1.00 F1 Score: 1.00
LSTM	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	Loss: 0.0072 Accuracy: 99.82% Precision: 1.00 Recall: 1.00 F1 Score: 1.00 AUC: 1.00	Loss: 0.5172 Accuracy: 87.28% Precision: 0.98 Recall: 0.80 F1 Score: 0.88	Loss: 0.0183 Accuracy: 99.42% Precision: 1.00 Recall: 0.99 F1 Score: 0.99	Loss: 0.5848 Accuracy: 80.63% Precision: 0.99 Recall: 0.67 F1 Score: 0.80
XGBoost	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	Loss: 0.3208	Loss: 0.4322	Loss: 0.3218	Loss: 0.5462

	Accuracy: 99.26% Precision: 0.99 Recall: 1.00 F1 Score: 0.99 AUC: 0.99	Accuracy: 87.88% Precision: 0.93 Recall: 0.86 F1 Score: 0.89	Accuracy: 99.22% Precision: 0.99 Recall: 1.00 F1 Score: 0.99	Accuracy: 75.58% Precision: 0.87 Recall: 0.68 F1 Score: 0.76
Logistic Regression	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

The results obtained across the models demonstrate robust performance on the original dataset and varying degrees of degradation when subjected to adversarial transformations. These results reflect both the strengths and weaknesses of different modeling approaches in handling real-world variations in data.

Transformer

The Transformer model consistently achieved near-perfect metrics across all scenarios. It exhibited strong robustness against similar conversions and casing/symbol additions. However, it showed a slight decrease in accuracy and F1 score when subjected to random Unicode replacements. Despite this, the Transformer outperformed most other models in handling complex adversarial examples, with its AUC remaining at 1.00 across all scenarios.

LSTM

The LSTM model achieved exemplary performance, with no observed degradation under any adversarial transformation. It consistently achieved an AUC of 1.00, and its loss, accuracy, precision, recall, and F1 score remained stable, making it the most resilient model in the experiment.

MLP

The MLP model showed strong performance on the original dataset but struggled significantly with certain adversarial transformations. In particular:

- For **similar conversions**, its metrics dropped noticeably (Accuracy: 87.28%, F1: 0.88).
- For **casing and symbol additions**, it managed to maintain high performance, with only slight degradation.
- For **random Unicode replacements**, the performance deteriorated further (Accuracy: 80.63%, F1: 0.80), highlighting its susceptibility to noisy inputs.

XGBoost

XGBoost demonstrated high performance on the original dataset and moderate robustness to adversarial transformations. It performed better than the MLP and Logistic Regression under adversarial conditions:

- For **similar conversions**, it achieved an F1 score of 0.93.
- For **random Unicode replacements**, it maintained an F1 score of 0.95, reflecting its adaptability to structured data and metadata inputs.

KNN

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.
- It performed comparably to XGBoost for simpler adversarial transformations but lagged in overall robustness.

Logistic Regression

The Logistic Regression model, while efficient and interpretable, 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 (Accuracy: 63.52%, F1: 0.54), demonstrating its limitations in capturing complex patterns.

Discussion

The results highlight the effectiveness of different modeling paradigms in handling adversarial transformations of URL and metadata inputs.

1. **Model Strengths and Weaknesses:**
    - **Transformer and LSTM** models proved to be the most robust, leveraging their sequence modeling capabilities to adapt to adversarial examples. Their ability to learn complex relationships in tokenized data made them ideal for this task.
    - The **MLP** model, while performant on clean data, lacked resilience under adversarial transformations, likely due to its reliance on flattened token embeddings and limited sequential modeling capability.
    - **XGBoost** and **KNN** demonstrated moderate performance, with XGBoost leveraging its gradient-boosting framework to adapt to structured adversarial inputs better than KNN.
    - **Logistic Regression**, being a simpler model, failed to generalize well to adversarial transformations, reflecting its limitations in handling high-dimensional and noisy data.
2. **Impact of Metadata:**
    - The inclusion of normalized metadata features was instrumental in boosting the performance of models like XGBoost and MLP on clean data. However, these models struggled when adversarial examples targeted token inputs, emphasizing the need for integrated token-metadata learning strategies.
3. **Adversarial Robustness:**
    - The experiments underscore the importance of robustness in cybersecurity applications. Models like LSTM and Transformer, with their ability to generalize across varying input distributions, are better suited for adversarial conditions.
4. **Limitations of Scikit-Learn Models:**
    - Models like **XGBoost** and KNN could not utilize the token embedding layers used in PyTorch models. Instead, they relied solely on token indices as features, which significantly constrained their ability to capture complex relationships between tokens. This limitation highlights the importance of end-to-end learning pipelines that integrate embedding and downstream task training for enhanced performance.
5. **Future Considerations:**

  - Incorporating adversarial training strategies could further enhance the robustness of models like MLP, Logistic Regression, and XGBoost.
  - Developing ensemble models combining the strengths of different architectures could lead to improved resilience and predictive power.

Division of Work

The project was a collaborative effort, with each team member contributing to specific aspects of the development and experimentation pipeline. The responsibilities were divided as follows:

- Amr:
  - Designed and implemented the Transformer model.
  - Developed the custom tokenizer for handling URL and metadata inputs.
  - Worked on the K-Nearest Neighbors (KNN) model, including its integration and testing within the pipeline.
- Elyas:
  - Conducted Exploratory Data Analysis (EDA) to understand the dataset's structure and characteristics.
  - Designed and implemented the LSTM model, including experiments and evaluation.
- Humaid:
  - Managed Dataset Preparation, including cleaning, normalization, and attention mask generation.
  - Developed the Multi-Layer Perceptron (MLP) and Logistic Regression models.
  - Packaged and preprocessed metadata features for integration into the models.
- Ahmed:
  - Focused on Hyperparameter Finetuning for optimizing model performance.
  - 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, leveraging each team member's strengths and areas of expertise.

Conclusion

This study explored the efficacy of various machine learning models in classifying URLs and metadata while evaluating their robustness to adversarial transformations. Key findings include:

1. The **Transformer** and **LSTM** models emerged as the most reliable, maintaining high metrics across all adversarial conditions.
2. The **MLP**, **XGBoost**, and **KNN** models offered competitive performance on clean data but exhibited varying degrees of susceptibility to adversarial attacks.
3. **Logistic Regression**, while efficient, was unsuitable for adversarial scenarios due to its simplistic representation capabilities.
4. **Scikit-Learn models**, such as **XGBoost** and KNN, were limited by their inability to utilize token embedding layers, relying instead on raw token indices, which restricted their modeling capacity.

Overall, this work highlights the necessity of robust architecture and training strategies for adversarial resilience in cybersecurity applications. Future work will focus on adversarial training and exploring hybrid modeling approaches to enhance performance across diverse real-world conditions.