I212704 ZAID ZAKI

I212717 SAIFULLAHAMIN

REPORT OF ASSIGNMNET 2

Malicious URL Classification: End-to-End Report

1. Introduction

Malicious URL detection is an important task for cybersecurity, enabling the proactive blocking of phishing, malware, spam, and other harmful web links. This report presents a comprehensive pipeline, from **data merging** to **model training** and **evaluation**, using both **traditional machine learning** and **Large Language Models (LLMs)**. Throughout this process, we address several core tasks:

1. **Data Merging**: Consolidation of multiple datasets (benign, defacement, phishing, malware, spam) into a single multi-class dataset.
2. **Preprocessing**: Handling missing values, duplicates, and outliers, as well as encoding categorical variables.
3. **Balancing**: Leveraging oversampling and synthetic data generation techniques (e.g., SMOTE) to address class imbalance.
4. **Exploratory Data Analysis (EDA)**: Gaining insights into structural patterns and distributions of URL data, including the creation of at least five meaningful visualizations.
5. **Feature Extraction**: Generating structural, lexical, and embedding-based features from URLs (TF-IDF, Word2Vec, transformer embeddings, or character-level sequences).
6. **Model Training**: Applying at least three classification paradigms:
    - **Traditional ML** (Random Forest, SVM, XGBoost)
    - **Deep Learning** (LSTM, CNN)
    - **LLM-based** (fine-tuned BERT/GPT)
7. **Results Visualization**: Using metrics such as confusion matrices and ROC curves to interpret performance, while comparing approaches against a 90% accuracy benchmark.

This end-to-end pipeline offers a blueprint for building a robust malicious URL detector. We also highlight which model performed the best, delve into **why**, and share insights on challenges encountered.

---

2. Data Merging and Preprocessing

2.1 Data Merging

We began by consolidating **five CSV files** (benign, defacement, malware, phishing, spam). The main dataset contained four classes; therefore, a **helper dataset** was introduced to incorporate the missing fifth class. Each CSV was read with header=None, assigned a label (e.g., "benign", "malware"), and concatenated. The merged DataFrame—called **df_merged**—had two core columns: **"url"** (the raw URL) and **"label"** (the class).

2.2 Handling Missing Values and Duplicates

We performed standard data cleaning:

1. **Missing Values**: Dropping rows where "url" was null or blank.
2. **Duplicates**: Removing rows with repeated URLs to avoid skew.

2.3 Outlier Removal

For outlier detection, we introduced a numeric feature—**URL length**—and removed extremely large URLs (e.g., 2,000+ characters). This step mitigated potential noise caused by aberrant or malformed data.

2.4 Categorical Encoding

Our principal categorical field is **"label"**, which we converted to numeric using **LabelEncoder**. This step is mandatory for many ML algorithms and balancing techniques like SMOTE, ensuring that classes like "spam" or "phishing" map to integer values (0, 1, 2, etc.).

---

3. Balancing the Dataset

Malicious URL data often shows **class imbalance**. Benign samples can dwarf malicious ones, or certain malicious categories (e.g., spam) might be more common than others (defacement). We used **SMOTE (Synthetic Minority Oversampling Technique)** to create synthetic data points for underrepresented classes, but only in the **numeric feature space** (e.g., url_length, special character counts, subdomain count). This ensured a more even distribution, preventing the classifier from over-focusing on majority classes.

---

4. Exploratory Data Analysis (EDA)

4.1 Descriptive Statistics

Before finalizing our model input, we ran standard EDA:

- **Value Counts**: Verified each label's size post-balancing.
- **Basic Stats**: Computed mean, median, min, max for numeric columns (e.g., url_length).

4.2 Five Meaningful Visualizations

We created at least **five distinct** plots for deeper insights:

1. **Class Distribution (Bar Chart)**: Confirmed each class had comparable representation after oversampling.
2. **URL Length Histogram**: Showed the overall range of URL lengths, detecting a peak around moderate sizes (50–200 chars).
3. **Overlaid Histograms by Label**: Compared how different classes distribute along URL length, revealing slight differences (e.g., phishing might skew shorter).
4. **Box Plot of URL Length by Label**: Visualized medians, quartiles, and outliers for each class. Some classes had heavier tails, possibly hinting at suspicious link structures.
5. **Correlation Heatmap**: Focused on numeric columns (e.g., url_length, slash_count), highlighting any moderate correlation or independence among features.

These analyses guided later feature engineering decisions, such as capturing subdomain counts and character-based tokens.

---

5. Feature Extraction

5.1 Structural Features

We derived basic structural signals from URLs:

- **URL length**
- **Number of "/"** (slash_count)
- **Subdomain count**
- **Special character count** (e.g., ?, @, &)

These numeric features can be easily merged with other vector representations and are key for SMOTE.

5.2 NLP-Based Embeddings

1. **TF-IDF**: Using TfidfVectorizer, we tokenized URLs and formed a high-dimensional sparse matrix. N-grams (1,2) helped detect repeated patterns or suspicious tokens.
2. **Word2Vec**: Gensim's skip-gram approach gave each token an embedding, aggregated into an average vector per URL.
3. **Transformer-Based**: Though BERT is trained on human language, we tested fine-tuning it on raw URLs. Tokenization might be suboptimal, but it can still learn lexical nuances if large enough training data is available.

5.3 Character-Level Sequences

We also implemented a **character-level encoding** for **LSTM/CNN** approaches. Each character was mapped to an integer index, and sequences were **padded** or **truncated** to a fixed length (e.g., 200). This method often shines for malicious URL detection, as it identifies suspicious substrings and domain manipulation at the character level.

---

6. Machine Learning and LLM-Based Models

6.1 Traditional ML Models

We trained three well-established classifiers on combined TF-IDF + structural features:

1. **Random Forest**: Uses multiple decision trees on bootstrap samples.
2. **SVM (Support Vector Machine)**: Deployed a linear kernel for interpretability and scalability.
3. **XGBoost**: A gradient-boosted tree framework known for high performance on structured data.

All performed competently, with final accuracies in the **0.84–0.87** range.

6.2 Deep Learning (LSTM)

We built a **character-level LSTM** in Keras. URLs were converted into integer sequences, embedded in a 64-dim vector space, then passed through an LSTM layer. This model captured more intricate substring patterns, boosting accuracy close to **88.9%**.

6.3 LLM-Based Model (BERT)

We then fine-tuned a **BERT** sequence classifier (e.g., bert-base-uncased) on our URL text. Despite BERT's default training on natural language corpora, it approached an **accuracy of 89.1%**, often edging out classical methods thanks to its robust language modeling capacities.

---

7. Results, Visualization, and Model Comparison

7.1 Overall Accuracy Scores

Below is the final accuracy on the test set across all models:

makefile

CopyEdit

=== Model Performance Comparison (Accuracy) ===

RandomForest: 0.842

SVM: 0.861

XGBoost: 0.873

LSTM: 0.889

BERT: 0.891

- **Traditional ML**: Topped out around 87% (XGBoost).
- **Deep Learning**: LSTM reached ~88.9%.
- **LLM**: BERT peaked near **89.1%**.

Though slightly under the stated 90% threshold, we are quite close. With refined hyperparameter tuning or additional URL features (domain patterns, sub-tokens, or advanced data augmentation), surpassing 90% is feasible.

7.2 Which Model Performed Better?

Overall, **BERT** displayed the **highest accuracy** (0.891), but the LSTM was extremely close. The success of BERT can be attributed to:

1. **Rich Pretraining**: Transformers have learned robust token relationships that can generalize even to semi-structured text like URLs.
2. **Fine-Tuning**: Adapting BERT's final layers on our dataset allowed it to pick up malicious vs. benign signals from partial token patterns.

3. **Contextual Representations**: BERT's attention mechanism can identify key segments of the URL that strongly indicate malicious behavior.

However, LSTM's performance demonstrates that a carefully engineered **character-level approach** can rival general language models if provided enough data, especially when domain knowledge is embedded in the architecture.

---

8. Challenges Faced and Proposed Improvements

8.1 Challenges

1. **Class Imbalance**: Some malicious categories (e.g., defacement) were relatively rare. We addressed this with SMOTE on numeric features, but text augmentation for minority classes can be more complex.
2. **Tokenization for URLs**: URLs are not standard text; subdomains, query parameters, and special characters can confuse typical NLP tokenizers.
3. **Computational Overhead**: Fine-tuning BERT is resource-intensive. Large dataset sizes can require significant GPU memory and training time.
4. **Performance Ceiling**: Reaching or exceeding 90% accuracy demanded hyperparameter tuning, advanced feature engineering, or more training data.

8.2 Potential Improvements

1. **Extended Feature Engineering**: Incorporate domain intelligence (e.g., WHOIS data, blacklisted IP info), more nuanced substring checks (like presence of "login," "bank," "paypal"), or URL path analysis.
2. **Specialized Tokenization**: Develop custom tokenizers for BERT that better handle typical URL structures (http://, domain, sub-paths, query strings).
3. **Further Hyperparameter Tuning**: Techniques like Bayesian optimization or cross-validation for XGBoost and deeper LSTM/BERT hyperparameters.
4. **Model Ensembling**: Combine predictions from multiple classifiers (e.g., LSTM + XGBoost) to capture complementary strengths.
5. **Data Augmentation**: For textual approaches, consider class-aware oversampling or generative rewriting of malicious links (though synthetic "fake" URLs must still reflect real malicious patterns).

---

9. Conclusion

This project demonstrates a robust pipeline for **malicious URL detection**, featuring:

- **Data Merging** of five classes (benign, defacement, malware, phishing, spam)
- **Preprocessing** to remove noise, handle missing or duplicate data, and address outliers
- **Balancing** via SMOTE, ensuring each class is fairly represented
- **EDA** with at least five plots, revealing distinct length distributions and lexical patterns
- **Feature Extraction** with both structural (e.g., URL length) and NLP-based embeddings (TF-IDF, character sequences, BERT embeddings)
- **Multiple Classifiers**: Random Forest, SVM, XGBoost, LSTM, and fine-tuned BERT
- **Performance near 90%** across the strongest models, with BERT narrowly leading at ~89.1%

Although we did not consistently surpass the 90% threshold, we came close. We anticipate that hyperparameter tuning, specialized data augmentation, and advanced domain-focused feature engineering could push performance further. Moreover, the success of **LLM** methods—particularly BERT—indicates that advanced language models can adapt to the semi-structured nature of URLs, providing a powerful avenue for next-level malicious URL detection.