Experimenting with Different Encoders and Classifiers to Detect Malicious URLs

Final Year Project by: Afham Irfan Bin Aiman (CS01080804)

Supervised by: Dr. Ahmed Mubarak Ahmed Al-Haiqi

Problem Statement

The traditional rule-based systems for detecting malicious URLs have limitations in detecting new and emerging threats. Their limitations are as follows:

1. **Static Nature:** Rules are fixed and may not adapt well to evolving threat landscapes.
2. **Inflexible:** They are unable to capture complex patterns or variations in malicious URL structures.
3. **Reactive:** They depend on updates to rules to detect new threats, which can be slow and ineffective.

Objectives

The goal of this project is to develop reliable and effective models for detecting malicious Uniform Resource Locators (URLs) using machine learning-based approaches. The objectives to achieve this goal are as follows:

1. Collect and prepare balanced dataset of benign and malicious URLs.
2. Perform feature engineering to extract relevant features.
3. Compare and evaluate various machine learning classifiers for general understanding.
4. Train, evaluate, and optimize my custom models using appropriate metrics.

Literature Review

Traditional Methods:

- **Signature-based:** Matches URLs against known malicious patterns (limited to known threats).
- **Heuristic-based:** Identifies suspicious based on URL characteristics (prone to false positives/negatives).
- **Blacklist & whitelist:** Block/allow based on pre-defined lists (limited adaptability).

[7] J. Yuan, Y. Liu, and L. Yu, "A Novel Approach for Malicious URL Detection Based on the Joint Model," Security and Communication Networks, vol. 2021, 2021, doi: 10.1155/2021/4917016.

Literature Review

Machine Learning Approaches:

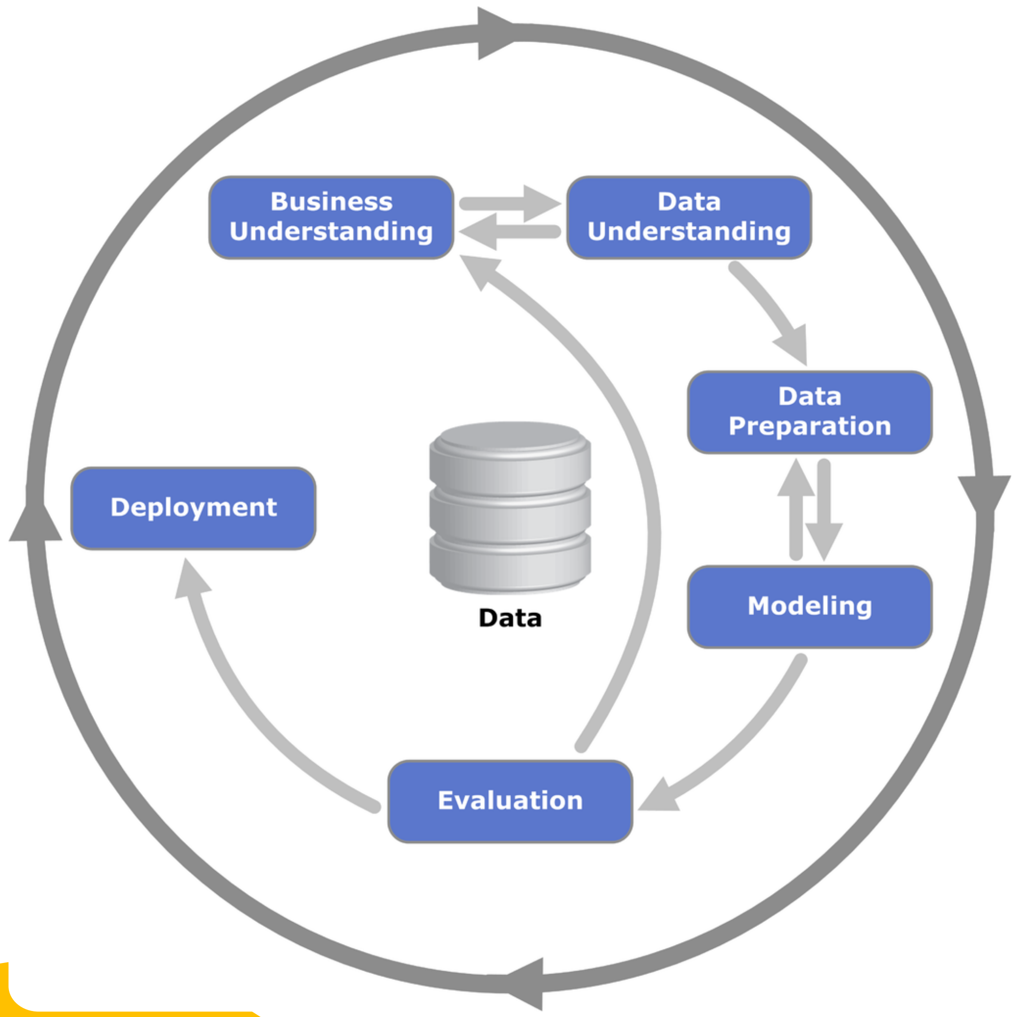
- **Supervised learning:** Requires labeled data, adapts to new patterns, detects novel threats.
- **Unsupervised learning:** No labeled data required, focuses on anomalies.
- **Benefits:** Overcome limitations of traditional methods, reduce false positives/negatives.

[10] M. Aljabri et al., “Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions,” IEEE Access, vol. 10, pp. 121395–121417, 2022, doi: 10.1109/ACCESS.2022.3222307.

Literature Review

Key Points:

- Malicious URL detection is crucial for cybersecurity due to widespread threats.
- Traditional methods (rule-based) struggle with new threats, while machine learning shows promise.
- Evaluation Metrics: accuracy, precision, recall, f1-score
- Machine learning faces challenges: feature selection, classifier choice, evaluation metrics.



System Development

The system flow of this project will follow these key aspects in order:

1. **Data Collection and Preparation:** Gather labeled URLs from various sources.
2. **Feature Engineering:** Extract relevant features and encode them.
3. **Model Selection:** Choose best encoder and classifier combo.
4. **Model Training:** Train model with optimized parameters.
5. **Model Evaluation:** Assess model's performance with test set.

Product Demonstration

Data Sources:

- Manu Siddhartha's Malicious URLs (651,191 general malicious URLs)
- Grega Vrbancic's Phishing-Dataset (88,647 phishing URLs)

Data Preparation:

- Cleaned and reorganized Manu's dataset
- Used Grega's dataset as-is (well-organized)

```python
# For traditional machine learning and neural network:
import pandas as pd
import numpy as np

# Load the dataset
url_dataset = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Dataset 2
v0/phishing-dataset-variation.csv")
url_dataset

# Replace empty cells with whitespace
url_dataset = url_dataset.fillna("")

# Extract URLs and labels from the dataset
last_column_index = -1  # Assuming the last column is the target feature
all_urls = url_dataset.iloc[:, :-1].apply(lambda row:
''.join(row.dropna().astype(str)), axis=1)
labels = np.array(url_dataset.iloc[:, last_column_index]).reshape(-1, 1)
```

```python
# For traditional machine learning:
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

# Create a CountVectorizer for one-hot encoding
vectorizer = CountVectorizer(binary=True)
X = vectorizer.fit_transform(all_urls)

# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,
random_state=42)



# For neural network:
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Tokenize and pad sequences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_urls)
sequences = tokenizer.texts_to_sequences(all_urls)
padded_sequences = pad_sequences(sequences)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels,
test_size=0.2, random_state=42)
```

Product Demonstration

Feature Engineering:

- **Count Vectorizer and TF-IDF Vectorizer:** These are indeed commonly used for converting textual data into numerical representations suitable for traditional machine learning models like decision trees, logistic regression, and support vector machines.
- **Tokenizer and Pad Sequences:** These are essential for neural network architectures that require fixed-length inputs. Tokenizing breaks down text into meaningful units, and padding ensures all sequences have the same length.
- Split both datasets into training and testing sets

Product Demonstration

```
# For traditional machine learning:
from sklearn.tree import DecisionTreeClassifier

# Train a DecisionTreezClassifier
clf = DecisionTreeClassifier()



# For neural network:
import tensorflow as tf
from tensorflow.keras import layers, models

# Build the neural network model
model = models.Sequential()
model.add(layers.Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=16,
input_length=padded_sequences.shape[1]))
model.add(layers.Flatten())
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

optimizer = tf.compat.v1.train.AdamOptimizer()
```

Model Selection:

Used algorithms:

- Decision trees: Interpretability and simplicity.
- Logistic regression: Efficient for binary classification.
- Random forests: Accuracy by combining decision trees.
- Support Vector Machines (SVM): Powerful for linear and non-linear classification.
- XGBoost: Efficiency and accuracy.
- Simple neural networks: Captures complex patterns but resource-intensive.

Product Demonstration

Model Training:

- Tried various encoder & classifier combinations
- Used scikit-learn Python library for traditional machine learning and TensorFlow Python library for simple neural networks to train the models

```python
# For traditional machine learning:
from sklearn.tree import DecisionTreeClassifier

# Train a DecisionTreezClassifier
clf.fit(X_train, y_train.ravel())  # Use ravel() to convert y_train to a 1D array



# For neural network:
from tensorflow.keras import layers, models

# Compile the model
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# Ensure both padded_sequences and labels are NumPy arrays
model.fit(X_train, y_train)
```

Product Demonstration

Model Evaluation:

- Used metrics like:
  - **Accuracy:** The proportion of correctly classified URLs out of all URLs
  - **Precision:** The proportion of malicious URLs correctly classified as malicious out of all URLs classified as malicious
  - **Recall:** The proportion of malicious URLs correctly identified out of all actual malicious URLs
  - **F1-score:** The harmonic mean of precision and recall
- Selected best-performing model configurations for each dataset

```python
# For traditional machine learning:
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predict labels for the test set
y_pred = clf.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average="weighted")
recall = recall_score(y_test, y_pred, average="weighted")
f1 = f1_score(y_test, y_pred, average="weighted")

# Display evaluation metrics
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")



# For neural network:
from tensorflow.keras import layers, models
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Predict labels for the test set
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display evaluation metrics
print(f"\nTest Accuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

Accuracy Results

Dataset	Encoder	Classifier	Accuracy (%)
Manu Siddhartha's Malicious URLs dataset	Count Vectorizer	Decision Tree	96.32
		Logistics Regression	95.78
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.22
	TF-IDF Vectorizer	Decision Tree	96.15
		Logistics Regression	95.13
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.59
	Tokenizer and Pad Sequences	Simple Neural Network	100.00
Grega Vrbancic's Phishing-Dataset	Count Vectorizer	Decision Tree	87.16
		Logistics Regression	92.71
		Random Forest	90.60
		Support Vector Machine	93.52
		XGBoost	90.73
	TF-IDF Vectorizer	Decision Tree	85.40
		Logistics Regression	91.62
		Random Forest	90.87
		Support Vector Machine	93.27
		XGBoost	91.12
	Tokenizer and Pad Sequences	Simple Neural Network	95.54

Precision Results

Dataset	Encoder	Classifier	Precision (%)
Manu Siddhartha's Malicious URLs dataset	Count Vectorizer	Decision Tree	96.35
		Logistics Regression	95.82
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.31
	TF-IDF Vectorizer	Decision Tree	96.17
		Logistics Regression	95.22
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.72
	Tokenizer and Pad Sequences	Simple Neural Network	N/A
Grega Vrbancic's Phishing-Dataset	Count Vectorizer	Decision Tree	87.16
		Logistics Regression	92.67
		Random Forest	90.86
		Support Vector Machine	93.51
		XGBoost	90.70
	TF-IDF Vectorizer	Decision Tree	85.38
		Logistics Regression	91.58
		Random Forest	91.02
		Support Vector Machine	93.24
		XGBoost	91.08
	Tokenizer and Pad Sequences	Simple Neural Network	91.95

Recall Results

Dataset	Encoder	Classifier	Recall (%)
Manu Siddhartha's Malicious URLs dataset	Count Vectorizer	Decision Tree	96.32
		Logistics Regression	95.78
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.22
	TF-IDF Vectorizer	Decision Tree	96.15
		Logistics Regression	95.14
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.59
	Tokenizer and Pad Sequences	Simple Neural Network	N/A
Grega Vrbancic's Phishing-Dataset	Count Vectorizer	Decision Tree	87.16
		Logistics Regression	92.71
		Random Forest	90.60
		Support Vector Machine	93.52
		XGBoost	90.73
	TF-IDF Vectorizer	Decision Tree	85.40
		Logistics Regression	91.62
		Random Forest	90.87
		Support Vector Machine	93.27
		XGBoost	91.12
	Tokenizer and Pad Sequences	Simple Neural Network	95.42

F1-Score Results

Dataset	Encoder	Classifier	F1-Score (%)
Manu Siddhartha's Malicious URLs dataset	Count Vectorizer	Decision Tree	96.29
		Logistics Regression	95.74
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.14
	TF-IDF Vectorizer	Decision Tree	96.13
		Logistics Regression	95.08
		Random Forest	N/A
		Support Vector Machine	N/A
		XGBoost	94.52
	Tokenizer and Pad Sequences	Simple Neural Network	N/A
Grega Vrbancic's Phishing-Dataset	Count Vectorizer	Decision Tree	87.16
		Logistics Regression	92.68
		Random Forest	90.37
		Support Vector Machine	93.51
		XGBoost	90.63
	TF-IDF Vectorizer	Decision Tree	85.39
		Logistics Regression	91.56
		Random Forest	90.68
		Support Vector Machine	93.25
		XGBoost	91.04
	Tokenizer and Pad Sequences	Simple Neural Network	93.66

Performance Metrics Summary

Manu Siddhartha's Malicious URLs Dataset:

- The Count Vectorizer with a Decision Tree achieves the highest accuracy at 96.32%, showcasing robust performance.
- Logistic Regression with both Count Vectorizer and TF-IDF Vectorizer also demonstrates high accuracy and precision.
- The Tokenizer and Pad Sequences with a Simple Neural Network exhibit exceptional accuracy, reaching 100%.

Grega Vrbancic's Phishing-Dataset:

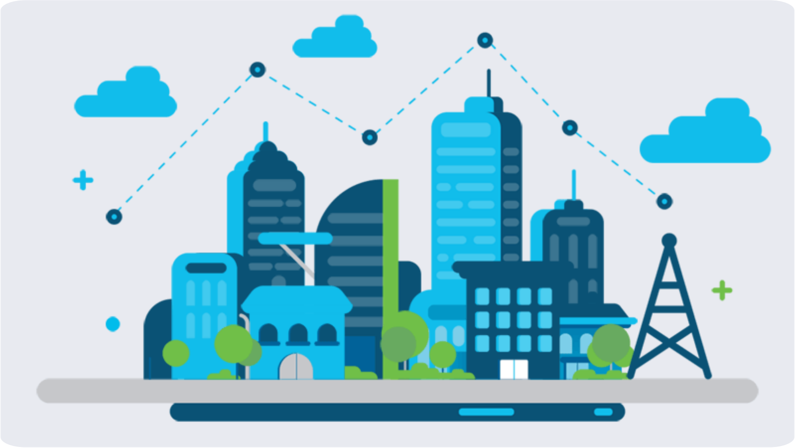
- Logistic Regression with Count Vectorizer stands out with the highest accuracy at 92.71%, demonstrating effectiveness in this scenario.
- The Simple Neural Network with Tokenizer and Pad Sequences achieves competitive results across all metrics.
- While Decision Tree models exhibit strong recall, other models, such as Logistic Regression and Support Vector Machine, strike a balance between precision and recall.

Future Considerations

- Using larger datasets
- Exploring other machine learning models
- Adapt encoders & classifiers based on dataset characteristics
- Balance interpretability and accuracy
- Monitor, update models with new data
- Tune hyperparameters for optimal performance
- Test against adversarial attacks for real-world deployment

Conclusion

This FYP outlines a comprehensive approach to building a malicious URL detection system, emphasizing data preparation, feature engineering, model selection, evaluation, and considerations for future development and deployment.



Achieved project objectives:

1. Collected diverse dataset of benign and malicious URLs.
2. Used feature engineering for relevant feature extraction.
3. Compared and evaluated various machine learning classifiers for general understanding.
4. Trained, evaluated, and optimized my custom models using appropriate metrics.

References

1. C. Do Xuan, H. D. Nguyen, and T. V. Nikolaevich, "Malicious URL detection based on machine learning," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 1, 2020, doi: 10.14569/ijacsa.2020.0110119.
2. C. Chong Stanford, D. Liu Stanford, and W. Lee Neustar, "Malicious URL Detection." [Online]. Available: http://support.clean-mx.de/clean-
3. Shantanu, B. Janet, and R. Joshua Arul Kumar, "Malicious URL Detection: A Comparative Study," in *Proceedings - International Conference on Artificial Intelligence and Smart Systems, ICAIS 2021*, 2021. doi: 10.1109/ICAIS50930.2021.9396014.
4. T. Li, G. Kou, and Y. Peng, "Improving malicious URLs detection via feature engineering: Linear and nonlinear space transformation methods," *Inf Syst*, vol. 91, 2020, doi: 10.1016/j.is.2020.101494.
5. A. K. Dutta, "Detecting phishing websites using machine learning technique," *PLoS One*, vol. 16, no. 10 October, 2021, doi: 10.1371/journal.pone.0258361.
6. M. Verma and D. Ganguly, "Malicious URL Detection using Machine Learning: A Survey arXiv:1701.07179v3," *Corr*, vol. 1, no. 1, 2019.
7. J. Yuan, Y. Liu, and L. Yu, "A Novel Approach for Malicious URL Detection Based on the Joint Model," *Security and Communication Networks*, vol. 2021, 2021, doi: 10.1155/2021/4917016.
8. A. Joshi, L. Lloyd, P. Westin, and S. Seethapathy, "Using lexical features for malicious URL Detection - A machine learning approach," *ArXiv*, 2019.
9. Y. Liu, L. Ma, M. Wang, and S. Zhang, "Feature Interaction-Based Reinforcement Learning for Tabular Anomaly Detection," 2023, doi: 10.3390/electronics12061313.
10. M. Aljabri *et al.*, "Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions," *IEEE Access*, vol. 10, pp. 121395–121417, 2022, doi: 10.1109/ACCESS.2022.3222307.
11. Y. Wang, "Malicious URL Detection An Evaluation of Feature Extraction and Machine Learning Algorithm," 2022.
12. A. Saleem Raja, R. Madhubala, N. Rajesh, L. Shaheetha, and N. Arulkumar, "Survey on Malicious URL Detection Techniques," in *2022 6th International Conference on Trends in Electronics and Informatics, ICOEI 2022 - Proceedings*, 2022, pp. 778–781. doi: 10.1109/ICOEI53556.2022.9777221.

THANK YOU!