INTEL-UNNATI AI&ML JOURNAL
Number 2 (2025), pp. 1–9

Code:IUIT-2025

MACHINE LEARNING

Fake Link Detection Using `Python` and Machine Learning

Nakul J S, Keerthana P A , and Sneha Suresh

Saintgits Group of Institutions, Kottayam, Kerala

Abstract: With the increasing dependency on web infrastructures and virtual services, cyber attacks involving malicious links have emerged as a serious security issue. Cyber attacks usually target vulnerabilities within web infrastructures to obtain unauthorized access, alter data, or interrupt services. Malicious URLs are usually designed to mimic legitimate ones and thus prove challenging to detect manually. Consequently, discovering and blocking such threats necessitates intelligent and automated methods. In this work, we outline an artificial link detection system based on machine learning algorithms, developed in Python. The system is trained on a labeled data set that contains both safe and risky URLs. Through the examination of structural patterns and behavior indicators in the URLs, the model can identify legitimate and malicious links. This method offers a scalable and efficient solution for the early identification of malicious URLs, thus enhancing the security stance of web applications.

Keywords: fake links, malicious URLs, benign links, phishing detection, URL classification, TF-IDF, tokenization, stemming, lemmatization, vectorization, machine learning, logistic regression, decision tree, random forest, Naive Bayes, SVM, model evaluation, URL-based features, Flask web app, cybersecurity, feature extraction, Python, scikit-learn, threat detection

1 Introduction

In the modern digital age, web applications and web platforms pervasively permeate everyday life, enabling perpetual information exchange on the internet. Although digitalization increases convenience and productivity, it also leaves systems vulnerable to an enormous set of cyber threats. One of the most enduring and menacing is the exploitation of malicious URLs, which are typically designed to be used to attack vulnerabilities through SQL injection (SQLi). These attacks exploit unsecured input fields or URL parameters to run unauthorized SQL statements, which can cause data breaches, broken confidentiality, and total system failure [8].

These URLs are often well-masked to look legitimate, using strategies such as URL obfuscation and social engineering to avoid detection [6]. Manual detection of these threats has become increasingly fallible with the complexity of contemporary attacks and the volume at which they take place. Automation is therefore necessary.

Machine learning (ML) provides a scalable and effective solution. Through learning on labeled sets of safe and unsafe URLs, ML models are able to detect patterns of attacks with high accuracy. The current work introduces a Python system that is intended to identify false and dangerous URLs, that is, threats based on SQLi. Expanding upon previous research employing query structure analysis [1], hybrid deep learning models [2], GRU models [4], and transformers such as URLTran [5], our system seeks to advance the benchmark by including real-time analysis and enhanced feature extraction.

In determining the ability of the system, we tested its performance on five well-used classification models:Logistic Regression, Random Forest, Naive Bayes (MultinomialNB), Support Vector Machine (SVM), and Decision Tree Classifier. Each model was assessed using a range of standard performance metrics commonly used in classification tasks, providing a comparative view of their strengths in identifying SQLi-based threats. This multi-model approach enabled us to highlight the most promising algorithm for real-world deployment based on its balance of detection power and efficiency.

In addition, newly emerging innovations, including feature selection methods based on Gray Wolf Optimization(GWO) [3] and Mechanisms inspired by reinforcement learning like Human-Agent Transfer(HAT) are also being increasingly adopted for dynamic threat landscapes [7]. Our system is inspired by such progress with an emphasis on lightweight, deployable configurations compatible with integration into contemporary web architectures.

2 Libraries Used

In the project for various tasks, following packages are used.

```
flask
scikit-learn
pandas
joblib
```

3 Methodology

In this work, five classical machine learning models are implemented and evaluated for the detection of malicious URLs. The models used include Logistic Regression, Random Forest Classifier, Naive Bayes (MultinomialNB), Support Vector Machine (SVM), and Decision Tree Classifier. Among these, the Random Forest Classifier demonstrated superior performance in terms of accuracy and other evaluation metrics. Various stages in the implementation process are:

Data Loading: The data is a publicly available dataset of URLs, labeled as benign, malware, phishing, or defacement. The data is structured for supervised learning.

Pre-processing & Data cleaning: The URLs are preprocessed by removing duplicates and extracting essential components (length, special characters, subdomains, etc.) using Python libraries.

Feature extraction: Lexical features like @ presence, URL length, IP usage, and suspicious keywords are extracted to create the feature set.

Dataset Preparation: The data is divided into training and testing sets of 80% and 20%, respectively, to enable efficient model evaluation.

Model Training: Five machine learning modelsâĂŤLogistic Regression, Random Forest, Naive Bayes, Support Vector Machine (SVM), and Decision TreeâĂŤwere trained and evaluated to analyze the performance of various classification approaches.

Classification: Trained models are used to classify URLs into one of the categories: Benign, Malware, Phishing, or Defacement.

Model Evaluation: Models are compared using various classification metrics such as Accuracy, Precision, Recall, and F1-score. These performance metrics are then used to determine the best-performing classifier for a given task.

Model selection and reporting Every URL is classified as Benign, Phishing, Malware, or Defacement. The system identifies the best performing model for real-time threat detection

4 Implementation

As the first step of the task, the dataset `url_data.csv`, containing approximately 600,000 pre-labeled URLs, was imported into a local development environment using Visual Studio Code (VS Code). The dataset was sourced from a public URL repository. Each entry comprises a raw *URL* accompanied by a corresponding *label* categorized into one of four classes: benign, malware, phishing, or defacement. The data was loaded into a `pandas` DataFrame and properly encoded for machine learning operations. An initial inspection of the dataset confirmed that the class balance was adequately maintained across categories.

The preprocessing and cleaning phase was tailored for URL-based input, as opposed to the typical text classification task. Preprocessing was conducted in well-defined steps, which included:

- **Length extraction:** Overall character length of the URL string.
- **Symbol flagging:**Availability of special characters like @, //, -, and _.
- **IP and subdomain detection:**Whether the URL features a raw IP address or more than one subdomain.
- **Suspicious keyword flags:** Occurrence of URLs that include words like login, secure, or verify.

This process of transformation took around $25\ to\ 30$ minutes on a machine available locally. Everything was done through Python libraries like pandas, numpy, and re.

During the feature extraction phase, the `TfidfVectorizer` from `sklearn.feature_extraction.text` was used on the URL strings. This transformed the raw URLs into a sparse high-dimensional matrix suitable for classification. The obtained `TF-IDF` matrix had a shape of about 50000 X 80000 and represented the URL corpus utilized for training.

The model training phase entailed the implementation and assessment of five traditional machine learning algorithms: Logistic Regression, Naive Bayes (MultinomialNB),

Support Vector Machine (SVM), Random Forest Classifier, Decision Tree Classifier. All models were trained on the full dataset without train-test splitting since the aim was to compare classification performance on all available data. Training and test were carried out with the `scikit-learn` library, and model performance was measured in terms of accuracy, precision, recall, and F1-score. The trained models were serialized with `joblib` and saved for usage within a web interface.

Aside from comparing individual model performances, the system was intended to facilitate dynamic model selection via an interface over the web. This makes it easy to try different algorithms and see how a variety of models react to one and the same input. The accuracy of each model is even presented in the interface to help users make a well-informed decision. The objective was not merely to obtain high precision but also to present a transparent and easy-to-use setting for threat identification. The integration of traditional machine learning algorithms with an easy-to-use frontend provides a guarantee that the system is efficient as well as user-friendly for applicability.
Results of these deployments, such as performance measures for every model and the architecture of the interactive web interface, are presented in the following section.

5 Results & Discussion

The standard classical machine learning algorithms from the `scikit-learn` Python library were used to train and test the models. The entire system was designed in VS Code and trained on $600,000$ pre-labeled URLs without any external acceleration.

Metrics of evaluation including accuracy, precision, recall, F1-score, root mean square error (RMSE), and inference time were logged for comparison. The results summary is presented in Table 1. While deep learning methods were at first considered, conventional models were more efficient and realistic for this URL-based structured classification task.

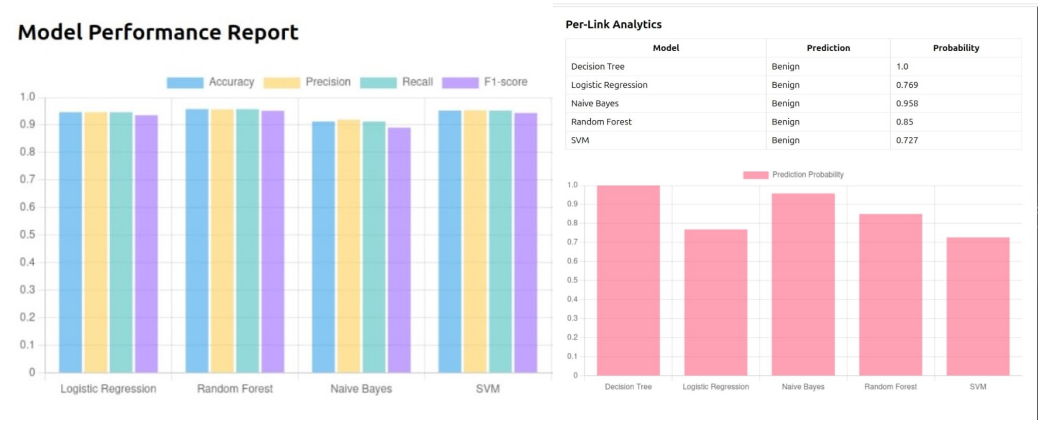
Model No.	Model Name	RMSE	Precision	Accuracy	Recall	F1-Score	Time
1	Logistic Regression	0.603	0.947	0.947	0.947	0.936	0.008 sec
2	Random Forest	0.533	0.957	0.958	0.958	0.952	0.818 sec
3	Naive Bayes	0.735	0.920	0.913	0.913	0.891	0.001 sec
4	SVM	0.579	0.954	0.953	0.953	0.944	10.717 sec
5	Decision Tree	0.555	0.951	0.955	0.955	0.950	0.004 sec

Table 1: Summary of Classification Models Tested on the URL Dataset

As can be seen from Table 1, Random Forest, SVM, and Decision Tree classifiers also excelled with more than 95% accuracy each. Although Naive Bayes provided quicker runtime, its relatively lower recall and precision render it less preferable for security-critical applications. Logistic Regression demonstrated consistent performance in all parameters.

Given the limited training time (about 25–30 minutes for all models) and high accuracy obtained, traditional machine learning methods prove both effective and scalable to identify malicious links based on structural features of URLs.

Along with the model-wide metrics, the system also gives real-time prediction confidence per input. Figure 1(a) displays the per-link analytics, showing how various models classify the same URL input, and their corresponding probability scores.



((a)) Model Performance Report: Accuracy, Precision, Recall, and F1-score

((b)) Per-Link Analytics: Classification and Probability Scores

Figure 1: Model-wise and Link-wise Evaluation Results

This double-layered examination allows users not just to select a model on the basis of worldwide performance but also to examine how a single prediction differs by algorithm providing both transparency and adaptability in actual-world deployment.

6 Conclusions

The fake link detection system exhibited excellent performance by leveraging traditional machine learning techniques. Logistic Regression, Naive Bayes, Support Vector Machine, Decision Tree, and Random Forest models were able to attain high accuracy and consistent classification over various threat types such as benign, phishing, malware, and defacement. These models being computationally efficient were highly effective in detecting malicious patterns by considering lexical and structure-based features of URLs. Considering the data nature and the project restrictions, traditional machine learning methods were used instead of deep learning models because of their speed of training, fewer resources used, and understandability. All models were executed and trained locally on VS Code, and preprocessing and training were finished within 30 minutes.The framework was further integrated into a web application that is user-friendly, with support for real-time analysis and model comparison. The combination of rapid computation, flexible deployment, and reliable classification makes these traditional models a feasible and scalable solution for early threat detection in real-world environments.

Acknowledgments

We would like to express our heartfelt gratitude and appreciation to Intel© Corporation for providing an opportunity to this project.First and foremost, we would like to extend our sincere thanks to our team mentor Ms.Akshara Sasidharan for her invaluable guidance and constant support throughout the project.We are deeply indebted to our college

Saintgits College of Engineering and Technology for providing us with the necessary resources,and sessions on machine learning. We extend our gratitude to all the researchers, scholars, and experts in the field of machine learning and natural language processing and artificial intelligence, whose seminal work has paved the way for our project. We acknowledge the mentors, institutional heads, and industrial mentors for their invaluable guidance and support in completing this industrial training under Intel© -Unnati Programme whose expertise and encouragement have been instrumental in shaping our work.

References

[1] ANJAN, K., ANDREOPOULOS, W., AND POTIKA, K. Prediction of higher-order links using global vectors and hasse diagrams. In *2021 IEEE International Conference on Big Data (Big Data)* (2021), pp. 4802–4811. 10.1109/BigData52589.2021.9671432.

[2] ANJAN, K., ANDREOPOULOS, W., AND POTIKA, K. Prediction of higher-order links using global vectors and hasse diagrams. In *2021 IEEE International Conference on Big Data (Big Data)* (2021), pp. 4802–4811. 10.1109/BigData52589.2021.9671432.

[3] ARASTEH, B., AGHAEI, B., FARZAD, B., ARASTEH, K., KIANI, F., AND TORKAMANIAN-AFSHAR, M. Detecting sql injection attacks by binary gray wolf optimizer and machine learning algorithms. *Neural Computing and Applications 36*, 12 (2024), 6771–6792.

[4] DASARI, N. S., BADII, A., MOIN, A., AND ASHLAM, A. Enhancing sql injection detection and prevention using generative models. *arXiv preprint arXiv:2502.04786* (2025).

[5] MANERIKER, P., STOKES, J. W., LAZO, E. G., CARUTASU, D., TAJADDODIANFAR, F., AND GURURAJAN, A. Urltran: Improving phishing url detection using transformers. In *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)* (2021), IEEE, pp. 197–204.

[6] MUDULI, D., SHOOKDEB, S., ZAMANI, A. T., SAXENA, S., KANADE, A. S., PARVEEN, N., AND SHAMEEM, M. Sidnet: A sql injection detection network for enhancing cybersecurity. *IEEE Access* (2024).

[7] SOMMERVOLL, Å. Å., ERDŐDI, L., AND ZENNARO, F. M. Simulating all archetypes of sql injection vulnerability exploitation using reinforcement learning agents. *International Journal of Information Security 23*, 1 (2024), 225–246.

[8] THANG, N. M., AND LUONG, T. T. Algorithm for detecting attacks on web applications based on machine learning methods and attributes queries. *Journal of Science and Technology on Information security* (2021), 26–34.

A Main code sections for the solution

A.1 Environment Setup and Dependencies

All the development was done in VS Code on a local machine. The Python environment was set up using pip with dependencies specified in requirements.txt:

```
pip install -r requirements.txt
```

Main Libraries Used:

- **pandas, numpy**
  For data manipulation, preprocessing, and numerical operations.
- **scikit-learn**
  Used for model training, evaluation metrics, vectorization (TF-IDF), and machine learning algorithms.
- **joblib, flask**
  `joblib` is used for saving and loading trained models and vectorizers. `Flask` is used to build the backend for the web-based URL classification interface.

A.2 Loading the Dataset

The dataset is loaded into a pandas DataFrame, and the labels are encoded as numerical values so that they can be used by machine learning models.

```
import pandas as pd

# Loading the dataset (sampled for testing)
df = pd.read_csv('url_data.csv').head(50000)

# Map string labels to integers
df['type'] = df['type'].map({
    'benign': 0,
    'malware': 1,
    'defacement': 2,
    'phishing': 3
})
```

A.3 Feature Extraction Using TF-IDF

TF-IDF is employed to transform raw URL strings into numerical features that represent token importance within the dataset.

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Vectorizing the URL column
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['URL'])  # URLs as input features
y = df['type']  # Labels as target
```

A.4 Dataset Preparation

The dataset is divided into training and test sets using an 80:20 ratio:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42)
```

A.5 Model Definition and Training

Five traditional ML models are trained from the dataset.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "Naive Bayes": MultinomialNB(),
    "SVM": SVC(probability=True),
    "Decision Tree": DecisionTreeClassifier()
}
```

A.6 Model Evaluation

The following metrics were computed for each classification model:

- **Accuracy:** Number of correct predictions out of all samples.
- **Precision:** âĂŞ Number of true positives out of total predicted positives.
- **Recall:** âĂŞ Number of true positives detected out of total actual positives.
- **F1-Score:** âĂŞ Harmonic mean of recall and precision.
- **Root Mean Squared Error (RMSE):** âĂŞ Quantifies standard deviation of errors in prediction.
- **Inference Time:** âĂŞ Time consumed by the model for prediction on the test set.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
                                        f1_score, mean_squared_error
import time, numpy as np
```

```
model_metrics = {}

for name, model in models.items():
    model.fit(X_train, y_train)
    start = time.time()
    y_pred = model.predict(X_test)
    elapsed = time.time() - start
```

```
model_metrics[name] = {
    "rmse": round(np.sqrt(mean_squared_error(y_test, y_pred)), 3),
    "precision": round(precision_score(y_test, y_pred, average='weighted',
                                       zero_division=0), 3),
    "accuracy": round(accuracy_score(y_test, y_pred), 3),
    "recall": round(recall_score(y_test, y_pred, average='weighted', zero_division
                                       =0), 3),
    "f1": round(f1_score(y_test, y_pred, average='weighted', zero_division=0), 3),
    "time": f"{elapsed:.3f} sec"
}
```

A.7 Model Persistence

Trained models and the TF-IDF vectorizer are saved for deployment:

```
from flask import Flask, render_template, request
import joblib
```

```
import joblib

for name, model in models.items():
    joblib.dump(model, f'models/{name}.pkl')

joblib.dump(vectorizer, 'models/vectorizer.pkl')
joblib.dump(model_metrics, 'models/accuracies.pkl')
```

A.8 Flask Web Interface

The web application was coded with **Flask** for the backend and **TailwindCSS** for the frontend design. Users can carry out the following activities:

- **Submit a URL** for processing.
- **Select the machine learning model** among various options available.
- **See predictions** together with classification confidence.
- **Evaluate performance** using interactive modal tables and charts showing measurement parameters.

```
if __name__ == '__main__':
app.run(debug=True)
```