EMAIL PHISHING URL DETECTION

BEKZADAYEVA KYMBAT

SIS-2208

INTRODUCTION

- PHISHING IS ONE OF THE MOST COMMON CYBERATTACKS, IN WHICH ATTACKERS IMPERSONATE LEGITIMATE ONLINE SERVICES IN ORDER TO DECEIVE USERS AND OBTAIN SENSITIVE INFORMATION. WITH THE RAPID GROWTH OF ONLINE PLATFORMS AND DIGITAL COMMUNICATION, PHISHING ATTACKS HAVE BECOME INCREASINGLY WIDESPREAD AND SOPHISTICATED.
- TRADITIONAL PHISHING DETECTION APPROACHES ARE MAINLY BASED ON BLACKLISTS AND MANUALLY DEFINED RULES. WHILE SUCH METHODS CAN SUCCESSFULLY BLOCK KNOWN THREATS, THEY ARE INEFFECTIVE AGAINST NEWLY GENERATED PHISHING URLS THAT CONSTANTLY CHANGE THEIR STRUCTURE TO EVADE DETECTION.
- MACHINE LEARNING PROVIDES AN EFFECTIVE SOLUTION TO THIS PROBLEM BY ENABLING SYSTEMS TO LEARN STRUCTURAL PATTERNS FROM DATA AND GENERALIZE TO PREVIOUSLY UNSEEN URLS. IN THIS PROJECT, A MACHINE LEARNING–BASED SYSTEM FOR PHISHING URL DETECTION IS DEVELOPED. THE MODEL CLASSIFIES URLS AS LEGITIMATE OR PHISHING USING URL-BASED FEATURES AND PROVIDES PROBABILITY-BASED PREDICTIONS. THE TRAINED MODEL IS DEPLOYED USING A FLASK WEB APPLICATION WITH A SIMPLE USER INTERFACE THAT ALLOWS REAL-TIME URL ANALYSIS AND ADJUSTABLE DECISION THRESHOLDS.

DATASET AND EXPLORATORY ANALYSIS

The dataset used in this project was obtained from a public Kaggle repository called “Phishing Site URLs”. It originally contained over 549,000 URLs labeled as legitimate or phishing. After removing duplicate entries during preprocessing, approximately 507,000 unique URLs were used for training and evaluation. The dataset contains no missing values and is fully anonymized, making it suitable for academic research.

Since machine learning models require numerical target values, the original categorical labels were converted into binary format. Phishing URLs were encoded as 1, while legitimate URLs were encoded as 0, enabling supervised binary classification.

Exploratory data analysis showed that legitimate URLs dominate the dataset, while phishing URLs form a smaller but meaningful class, reflecting real-world conditions. Phishing URLs tend to be longer and contain more digits, special characters, and deeper path structures compared to legitimate URLs. These observations confirm that URL-level structural features provide strong signals for phishing detection and directly motivated the selected feature engineering approach.

https://www.kaggle.com/datasets/taruntiwarihp/phishing-site-urls

kaggle

Search

Create

Home

Competitions

Datasets

Models

Benchmarks

Game Arena

Code

Discussions

Learn

More

Your Work

VIEWED

Phishing Site URLs

Phishing URLs Datas...

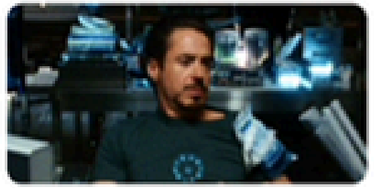
Web page Phishing D...

TARUN TIWARI · UPDATED 5 YEARS AGO

174 | Code | Download

Phishing Site URLs

Malicious and Phishing attacks ulrs



Data Card | Code (36) | Discussion (0) | Suggestions (0)

About Dataset

Phishing site Predict dataset

Youtube Explaination

Content

- Data is containg 5,49,346 entries.
- There are two columns.
- Label column is prediction col which has 2 categories

A. Good - which means the URLs is not containing malicious stuff and **this site is not a Phishing Site.**

B. Bad - which means the URLs contains malicious stuff and **this site is a Phishing Site.**

- There is no missing value in the dataset.

Usability

10.00

License

Database: Open Database, Cont...

Expected update frequency

Monthly

Tags

Internet | Social Networks | Crime | Public Safety | Search Engines

FEATURES.PY

As part of this project, a lexical feature extraction module was implemented in the file features.py. The purpose of this module is to transform a raw URL string into a fixed-length numerical feature vector suitable for machine learning classification.

The URL is first parsed using the urlparse function from the Python standard library to extract its main components, including the protocol, hostname, path, and query parameters. If a URL cannot be parsed correctly, a zero feature vector is returned to ensure system robustness and prevent runtime errors.

features.py

```python
import re
import numpy as np
from urllib.parse import urlparse

# This function checks whether the hostname is an IP address
# Phishing URLs often use IP addresses instead of domain names
def has_ip_address(hostname: str) -> int:
    pattern = r'^(\d{1,3}\.){3}\d{1,3}$'
    return int(re.match(pattern, hostname) is not None)


# This function extracts numerical features from a URL string
# These features are later used by machine learning models
def extract_url_features(url: str) -> np.ndarray:
    try:
        # Parse the URL into its components (scheme, hostname, path, query, etc.)
        parsed = urlparse(url)
    except:
        # If URL parsing fails, return a zero vector
        return np.zeros(11)

    # Extract hostname and path; use empty strings if missing
    hostname = parsed.hostname or ""
    path = parsed.path or ""

    # Create a feature vector based on URL characteristics
    return np.array([
        len(url),                                   # Total length of the URL
        len(hostname),                              # Length of the hostname
        len(path),                                  # Length of the path
        sum(c.isdigit() for c in url),              # Number of digits in the URL
        sum(c in ['@', ':', '-', '?', '=', '%', '.', '#', '&'] for c in url),
                                                    # Number of special characters
        hostname.count('.'),                        # Number of dots in the hostname
        int('@' in url),                            # Presence of '@' symbol
        int('-' in hostname),                       # Presence of '-' in hostname
        int(parsed.scheme == "https"),              # Whether HTTPS protocol is used
        has_ip_address(hostname),                   # Whether hostname is an IP address
        len(parsed.query)                           # Length of query parameters
    ], dtype=float)
```

TF-IDF REPRESENTATION

- TF-IDF WITH CHARACTER-LEVEL N-GRAMS WAS USED TO CONVERT URLS INTO NUMERICAL FEATURE VECTORS SUITABLE FOR MACHINE LEARNING. CHARACTER-LEVEL REPRESENTATION IS PREFERRED OVER WORD-LEVEL TOKENIZATION BECAUSE URLS CONTAIN DIGITS, SYMBOLS, AND OBFUSCATED PATTERNS RATHER THAN NATURAL LANGUAGE.
- TF-IDF ASSIGNS HIGHER WEIGHTS TO DISTINCTIVE CHARACTER SEQUENCES THAT FREQUENTLY APPEAR IN PHISHING URLS WHILE REDUCING THE IMPACT OF COMMON PATTERNS. IN THIS PROJECT, N-GRAM LENGTHS FROM 3 TO 6 CHARACTERS WERE SELECTED TO BALANCE EXPRESSIVENESS AND COMPUTATIONAL EFFICIENCY. THIS APPROACH PROVIDES A ROBUST AND SCALABLE REPRESENTATION THAT GENERALIZES WELL TO PREVIOUSLY UNSEEN PHISHING URLS

TRAIN_MODEL.PY

train_model.py

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    roc_auc_score,
    average_precision_score
)
import joblib


# Path to the phishing URL dataset
DATA_PATH = r"C:\Users\Huawei\Documents\project\phishing_site_urls.csv\phishing_site_urls.csv"


# Load the dataset from CSV file into a DataFrame
df = pd.read_csv(DATA_PATH)


# Ensure consistent naming of the label column
# Some datasets use "Label", others use "label"
if "Label" in df.columns and "label" not in df.columns:
    df = df.rename(columns={"Label": "label"})


# Remove rows with missing values and duplicate URLs
df = df.dropna(subset=["URL", "label"]).drop_duplicates(subset=["URL"])


# Convert all URLs to string type
df["URL"] = df["URL"].astype(str)

```

In the train_model.py module, I implemented the complete training and evaluation pipeline for phishing URL classification. The dataset is loaded from a CSV file using the pandas library and contains URLs with corresponding phishing or legitimate labels.

During preprocessing, I ensured data consistency by standardizing the label column name, removing rows with missing values, and eliminating duplicate URLs to avoid biased evaluation. All URLs were explicitly converted to string format to ensure compatibility with text-based feature extraction methods.

These preprocessing steps improve data quality and ensure that the model is trained on clean, consistent, and reliable input data before feature extraction and model training.

```python
# Convert textual labels into binary numerical labels
# 1 represents phishing, 0 represents legitimate
def encode_label(x):
    x = str(x).strip().lower()
    return 1 if x in ["bad", "phishing", "malicious", "defacement", "1"] else 0


# Apply label encoding to the dataset
df["label"] = df["label"].apply(encode_label).astype(int)


# Split the dataset into training and testing sets
# Stratification preserves the class distribution
X_train, X_test, y_train, y_test = train_test_split(
    df["URL"],
    df["label"].values,
    test_size=0.2,
    random_state=42,
    stratify=df["label"].values
)


# Initialize TF-IDF vectorizer using character-level n-grams
# Character n-grams are effective for detecting phishing patterns in URLs
vectorizer = TfidfVectorizer(
    analyzer="char",
    ngram_range=(3, 6),
    min_df=2,
    max_features=300000
)


# Fit the vectorizer on training data and transform both sets
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

- In this stage, I converted the original textual labels into binary numerical values, where phishing URLs are encoded as 1 and legitimate URLs as 0. This enables supervised binary classification and ensures compatibility with standard machine learning models and evaluation metrics.
- To evaluate the model's generalization ability, the dataset was split into training and testing sets using an 80/20 ratio with stratification. Stratified splitting preserves the original class distribution in both sets, while a fixed random seed ensures reproducibility and fair evaluation without data leakage.
- After the split, URLs were transformed into numerical feature vectors using character-level TF-IDF n-grams with lengths from 3 to 6. The TF-IDF vectorizer was fitted only on the training data and then applied to both training and test sets, ensuring that no information from the test set was used during training.

train_model.py

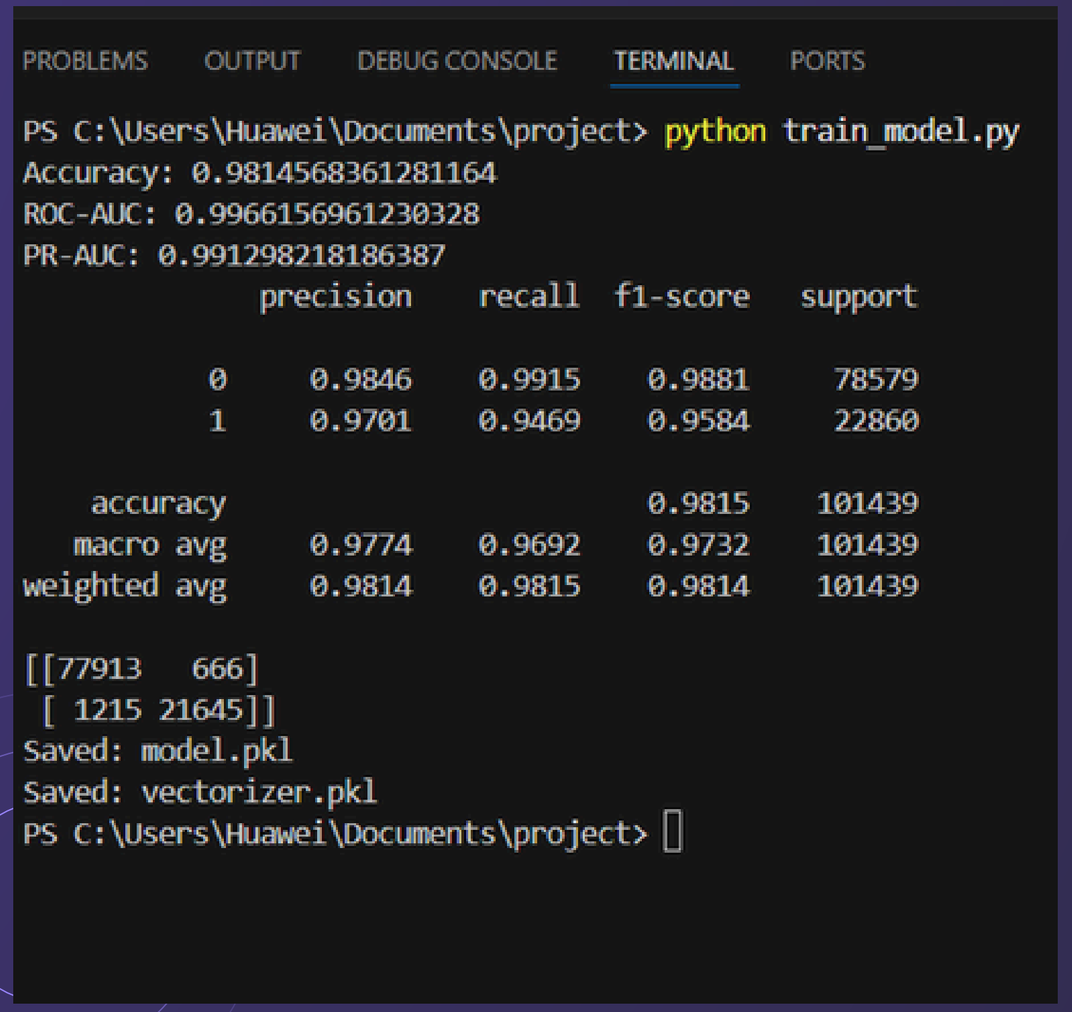
```python
# Create a Linear Support Vector Machine classifier
# Class balancing compensates for class imbalance in phishing datasets
base = LinearSVC(class_weight="balanced")


# Calibrate the SVM classifier to enable probability estimation
model = CalibratedClassifierCV(
    base,
    method="sigmoid",
    cv=3
)


# Train the calibrated model on the training data
model.fit(X_train_vec, y_train)


# Predict phishing probabilities for the test set
proba = model.predict_proba(X_test_vec)[:, 1]


# Convert probabilities into binary predictions using threshold 0.5
pred = (proba >= 0.5).astype(int)


# Print evaluation metrics
print("Accuracy:", accuracy_score(y_test, pred))
print("ROC-AUC:", roc_auc_score(y_test, proba))
print("PR-AUC:", average_precision_score(y_test, proba))
print(classification_report(y_test, pred, digits=4))
print(confusion_matrix(y_test, pred))


# Save the trained model and TF-IDF vectorizer to disk
# These files are later used in the web application
joblib.dump(model, "model.pkl")
joblib.dump(vectorizer, "vectorizer.pkl")


print("Saved: model.pkl")
print("Saved: vectorizer.pkl")
```

- For classification, a Linear Support Vector Machine was selected because it performs well on high-dimensional sparse TF-IDF features. Class weighting was applied to address imbalance between phishing and legitimate URLs.
- Since LinearSVC does not provide probability estimates, the model was calibrated using CalibratedClassifierCV to produce phishing probabilities for threshold-based decision making.
- The model was evaluated on a held-out test set using accuracy, ROC-AUC, PR-AUC, and the confusion matrix, demonstrating strong performance without data leakage.

MODEL EVALUATION RESULTS

After training, the model was evaluated on a held-out test set. The achieved accuracy is **98.15%**, indicating a high overall correctness of predictions. The ROC-AUC score of 0.9966 demonstrates excellent discriminative ability between phishing and legitimate URLs, while the PR-AUC score of **0.9913** confirms strong performance under class imbalance.

The classification report shows balanced precision and recall for both classes, and the confusion matrix indicates a low number of misclassifications. Importantly, these results were obtained without data leakage, as feature extraction and model training were performed only on the training set, and evaluation was conducted on unseen test data.



APP.PY

- After training the model, I deployed it as a web application using the Flask framework. The application loads the trained machine learning model and the TF-IDF vectorizer from disk using joblib and performs real-time phishing URL classification.
- Users can submit a URL through the web interface, after which the system predicts the phishing probability and converts it into a human-readable risk level (Low, Medium, or High). Loading the model once at application startup ensures efficient inference and avoids retraining for each request.

app.py

```python
from flask import Flask, render_template, request
import joblib
# Create Flask application instance
app = Flask(__name__)
# Load the trained machine learning model from disk
# This model was trained earlier using TF-IDF features
model = joblib.load("model.pkl")
# Load the TF-IDF vectorizer used during training
# It converts a URL string into numerical feature vectors
vectorizer = joblib.load("vectorizer.pkl")

def risk_level(p):
    #Convert phishing probability into a human-readable risk level.
    # This improves interpretability for end users.
    if p >= 0.75:
        return "High"
    if p >= 0.40:
        return "Medium"
    return "Low"

@app.route("/", methods=["GET", "POST"])
def index():
    #Main route of the web application.
    #Handles both displaying the form (GET)
    #and processing user input (POST).

    # Initialize variables passed to the HTML template
    result = None
    probability = None
    risk = None
    url_input = ""
    threshold = 0.5   # Default decision threshold
```

```python
    # If the user submits the form
    if request.method == "POST":

        # Read URL entered by the user
        url_input = request.form.get("url", "").strip()
        # Read threshold value entered by the user
        t = request.form.get("threshold", "0.5").strip()
        # Try to convert threshold to float
        try:
            threshold = float(t)
        except:
            threshold = 0.5

        # If a URL was provided
        if url_input:
            # Convert the URL into TF-IDF feature vector
            X = vectorizer.transform([url_input])
            # Predict phishing probability using the trained model
            proba = float(model.predict_proba(X)[0][1])
            # Apply decision threshold to get final class
            pred = 1 if proba >= threshold else 0
            # Convert numerical prediction to label
            result = "Phishing" if pred == 1 else "Legitimate"

            # Save probability and risk level for display
            probability = proba
            risk = risk_level(proba)
# Render HTML template with prediction results
    return render_template("index.html", result=result, probability=probability, risk=risk, url_input=url_input, threshold=threshold)
# Run Flask development server
if __name__ == "__main__":
    app.run(debug=True)
```

USER INTERFACE

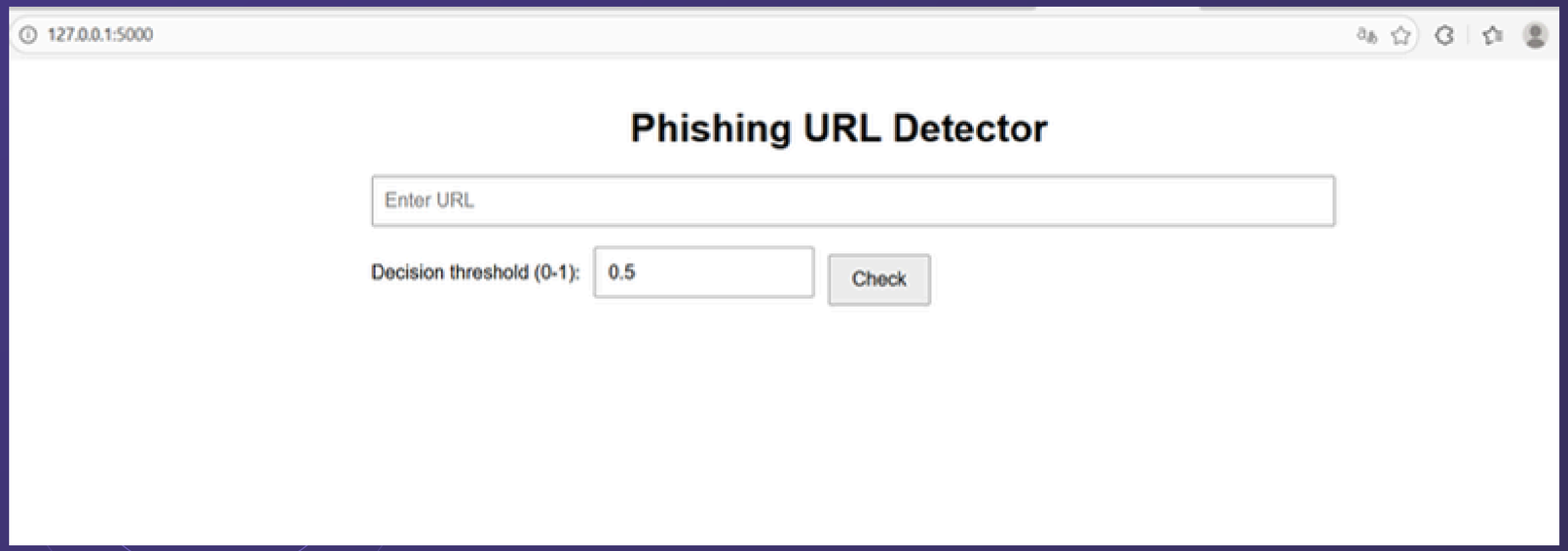
- After model inference, the application renders the index.html template and displays the prediction results to the user. The frontend receives the predicted class label, phishing probability, risk level, submitted URL, and the selected decision threshold from the backend.
- The user interface is implemented using HTML with embedded CSS and follows a simple, minimalistic design. It allows users to submit a URL, adjust the decision threshold, and view results only after a prediction is made, ensuring clarity and ease of use. This separation between backend logic and frontend presentation makes the application clean, interpretable, and easy to maintain.

templates > index.html > ...

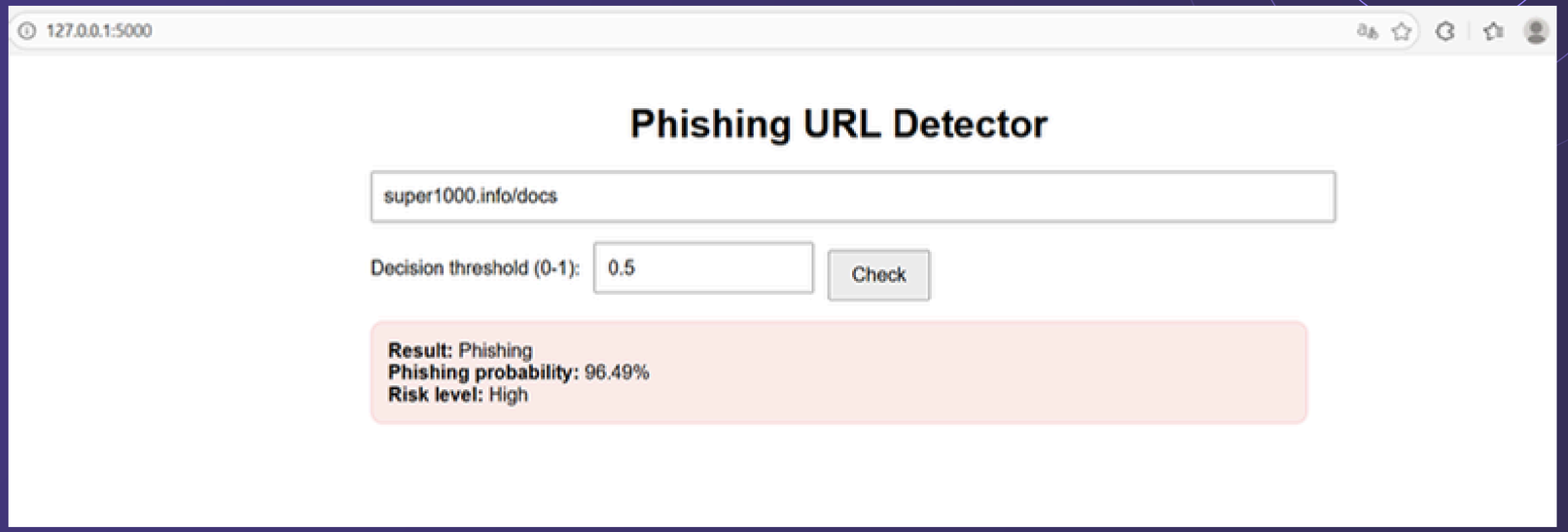
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Phishing URL Detector</title>
  <style>
    body { font-family: Arial, sans-serif; max-width: 780px; margin: 40px auto; }
    input { width: 100%; padding: 10px; font-size: 16px; }
    button { margin-top: 12px; padding: 10px 18px; font-size: 16px; cursor: pointer; }
    .row { display: flex; gap: 12px; margin-top: 12px; align-items: center; }
    .row input { width: 160px; }
    .box { margin-top: 18px; padding: 14px; border-radius: 10px; }
    .good { background: #e9f7ef; border: 1px solid #b7e1cd; }
    .bad { background: #fdecea; border: 1px solid #f5c6cb; }
  </style>
</head>
<body>
  <h1 style="text-align:center;">Phishing URL Detector</h1>

  <form method="POST">
    <input type="text" name="url" placeholder="Enter URL" value="{{ url_input }}">
    <div class="row">
      <div>Decision threshold (0-1):</div>
      <input type="text" name="threshold" value="{{ threshold }}">
      <button type="submit">Check</button>
    </div>
  </form>

  {% if result %}
    <div class="box {% if result == 'Legitimate' %}good{% else %}bad{% endif %}">
      <div><b>Result:</b> {{ result }}</div>
      <div><b>Phishing probability:</b> {{ (probability * 100) | round(2) }}%</div>
      <div><b>Risk level:</b> {{ risk }}</div>
    </div>
  {% endif %}
</body>
</html>

```

USER INTERFACE OVERVIEW

The web application provides a simple interface for phishing URL detection. Users can enter a URL for analysis and adjust the decision threshold to control detection sensitivity. The interface follows a minimalistic design to ensure clarity and ease of use. It directly connects user input with the deployed machine learning model for real-time predictions.

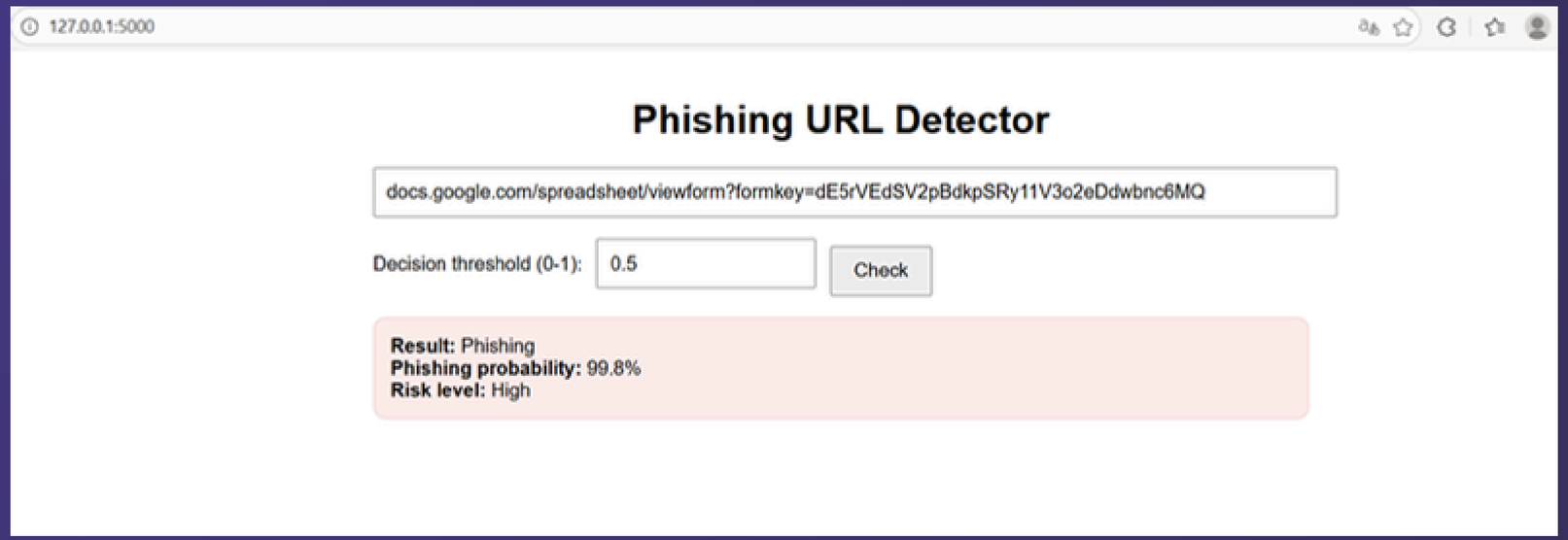


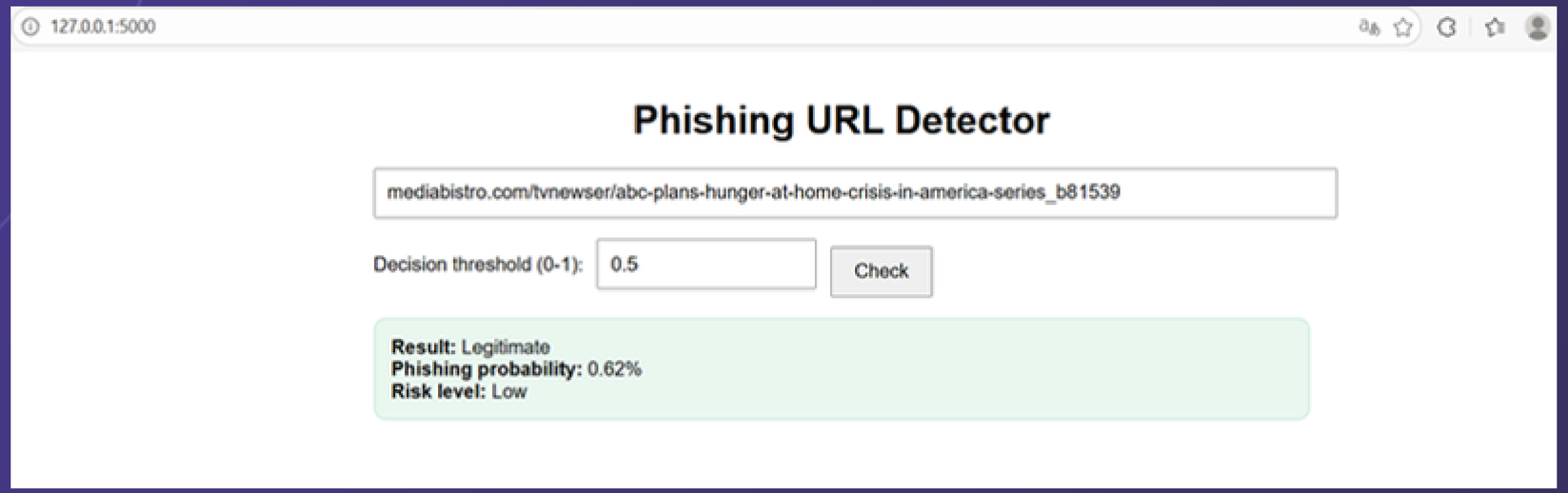
The figure shows an example of the system’s output when analyzing a suspicious URL. The model classifies the URL as phishing with a predicted probability of 96.49%, corresponding to a high risk level.



The result is clearly presented using the predicted label, probability score, and risk interpretation, with a red background highlighting the threat. This example demonstrates successful real-time integration of the trained machine learning model into the web application.

The next figure illustrates another example of the system’s output when analyzing a suspicious URL. The model classifies the URL as phishing with a predicted probability of 99.8%, corresponding to a high risk level. The result demonstrates the model’s ability to confidently identify highly suspicious URLs using probability-based classification. Visual highlighting emphasizes the severity of the detected threat and improves interpretability of the output.





The figure presents an example of the system's output when analyzing a legitimate URL. The model classifies the URL as legitimate with a phishing probability of 0.62%, corresponding to a low risk level. This result demonstrates the model's ability to correctly identify benign URLs and avoid false positives. Together with the phishing detection examples, this confirms that the system provides balanced and reliable real-time predictions for both classes.

CONCLUSION

- IN THIS PROJECT, A MACHINE LEARNING–BASED SYSTEM FOR PHISHING URL DETECTION WAS DEVELOPED USING CHARACTER-LEVEL TF-IDF FEATURES AND A CALIBRATED LINEAR SUPPORT VECTOR MACHINE CLASSIFIER. THE PROPOSED APPROACH SUCCESSFULLY DISTINGUISHES BETWEEN PHISHING AND LEGITIMATE URLS USING ONLY URL-LEVEL INFORMATION.
- A COMPLETE MACHINE LEARNING PIPELINE WAS IMPLEMENTED, INCLUDING DATA PREPROCESSING, FEATURE ENGINEERING, MODEL TRAINING, EVALUATION, AND DEPLOYMENT. CHARACTER-LEVEL TF-IDF N-GRAMS PROVED EFFECTIVE FOR CAPTURING STRUCTURAL PATTERNS ASSOCIATED WITH PHISHING ATTACKS AND ACHIEVED STRONG PERFORMANCE WITHOUT DATA LEAKAGE.
- THE TRAINED MODEL WAS DEPLOYED AS A FLASK WEB APPLICATION THAT PROVIDES REAL-TIME PHISHING DETECTION WITH PROBABILITY-BASED RISK ASSESSMENT. THE RESULTS CONFIRM THE EFFECTIVENESS OF MACHINE LEARNING FOR URL-BASED PHISHING DETECTION, WITH FUTURE WORK FOCUSING ON ADDITIONAL FEATURES OR ADVANCED MODELS.

THANK YOU FOR

attention