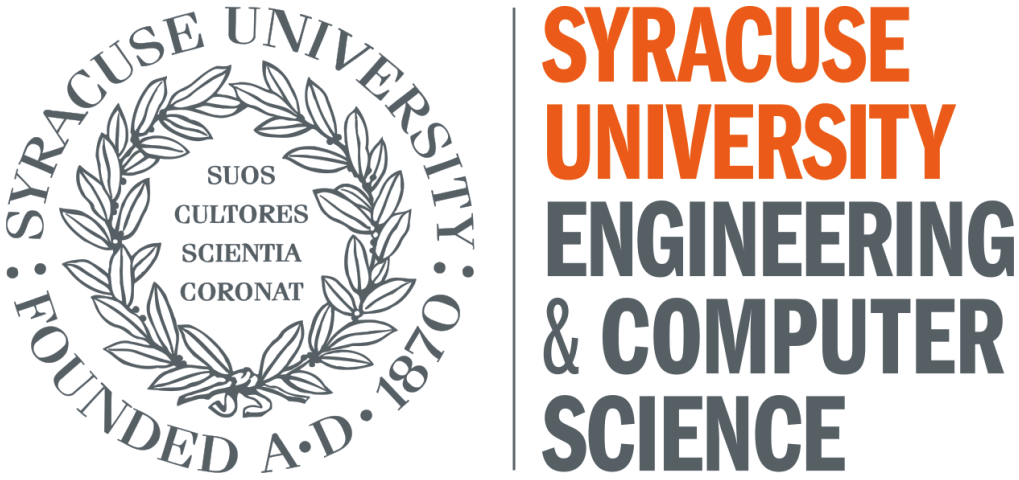
CIS600 Applied Natural Language Processing

Spring 2024 Term Research Project Report

"Malicious URL's detection using NLP and ML"

Team Information

Prasanth Sagar Kottakota	(SUID: 969978673)
Sri Krishna Chaitanya Chivatam	(SUID: 290797110)
Shyam Sudheer Nadella	(SUID: 364923152)
Vaibhav Krishna Joshi	(SUID: 556679674)
Uday Kumar Putta	(SUID: 317626035)



1. Abstract

This project explores the application of Natural Language Processing (NLP) techniques to classify URLs into categories such as benign, phishing, defacement, and malware. Utilizing a dataset composed of various URL types, we employed machine learning algorithms including Logistic Regression, K-Neighbors, Naive Bayes, Decision Tree, Random Forest, and XGBoost to develop predictive models. Key NLP techniques such as TF-IDF, Count Vectorization, BERT were implemented for feature extraction as tokens or embeddings from the URLs. The project was deployed in a cloud environment using the Django framework, enabling real-time classification and interaction through a web-based interface. Evaluation of the models based on accuracy, precision, recall, and F1-score demonstrated significant potential for enhancing cybersecurity measures. The deployment on the cloud further illustrates the practical scalability and accessibility of the solution for real-world applications. This report details the methodology, system architecture, model performance, and the practical implications of using NLP in cybersecurity.

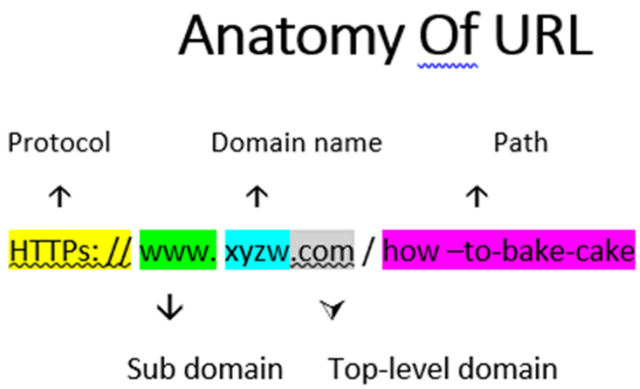
2. Introduction

In the realm of cybersecurity, URL classification emerges as a critical tool to preemptively identify and mitigate threats posed by malicious websites. Such classification efforts aim to discern between benign and harmful URLs, categorizing them into specific types such as phishing, defacement, malware, and others. Given the vast expanse of the internet and the continuous creation of new URLs, automated systems for quick and accurate URL classification are essential for protecting users and maintaining network security.

A Uniform Resource Locator (URL) is the address of a resource on the internet. It provides the means to retrieve any resource on the web, which could be a webpage, a file, or a downloadable resource. A URL can be broken down into several components:

Scheme: Indicates the protocol used to access the resource, such as HTTP, HTTPS, FTP, etc.
Subdomain: A prefix to the domain which can specify a particular server or a service, such as ‘www’ or ‘mail’.
Domain Name: The resource's primary identifier, linked to the server's IP address.

Top-Level Domain (TLD): The last part of the domain name, such as '.com', '.org', '.net', which can indicate the purpose or geographic area of the domain.
Path: Specifies a specific resource on the server, like a page or a file.
Query: Optional parameters that may provide additional instructions or information to the server.
Fragment: An internal page reference, usually identifying a specific part of the page.



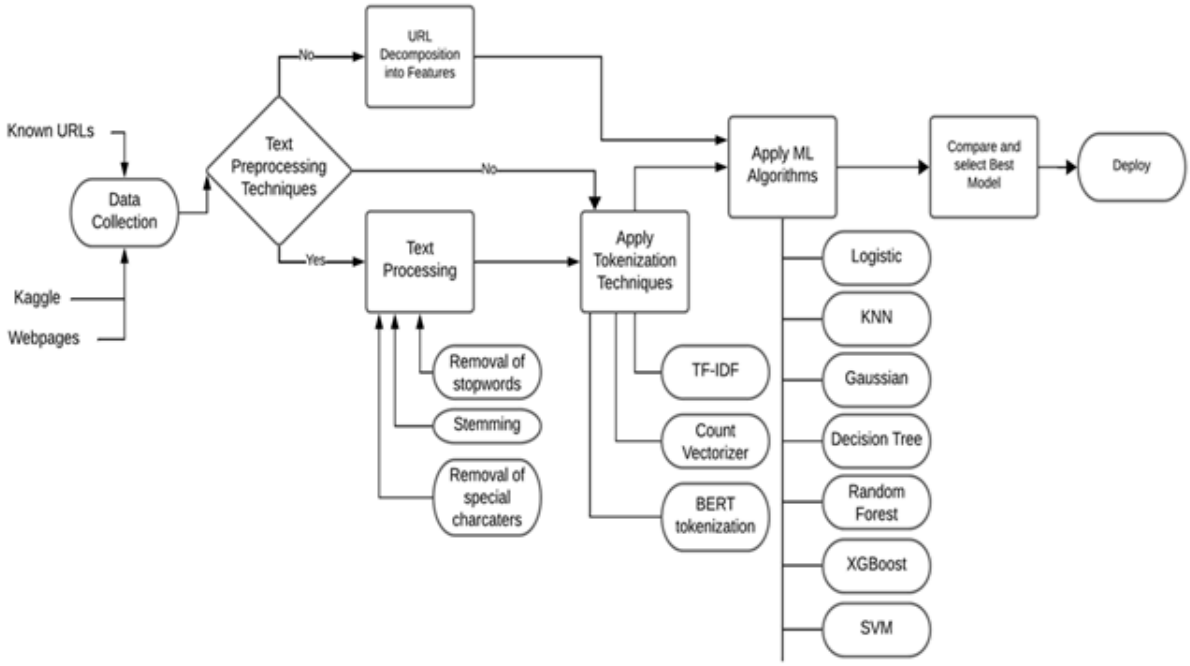
The classification of URLs using traditional methods often involves significant manual effort and domain knowledge. However, with advancements in Natural Language Processing (NLP), it is now feasible to automate and enhance this process. NLP facilitates the extraction of features from URLs in a manner that captures their lexical and semantic nuances, thereby aiding in the effective training of machine learning models. The application of NLP techniques such as Tokenization, TF-IDF (Term Frequency-Inverse Document Frequency), Count Vectorization, BERT embeddings can transform raw URLs into informative, actionable features that can be utilized by various algorithms to classify URLs reliably.

In this project, we leverage these NLP techniques to develop a series of machine learning models aimed at classifying URLs based on their likelihood of being safe or unsafe. By deploying this system in the cloud with the Django framework, we provide a scalable, accessible solution that can be integrated into broader cybersecurity systems, offering real-time URL classification capabilities.

This report details our approach from the initial data preparation to deployment, discussing the successes and challenges encountered in applying NLP techniques to cybersecurity. Aims to uncover hidden patterns and trends in the data that can aid researchers and policymakers in better understanding public perceptions and responses to the pandemic. These insights can ultimately contribute to the creation of more effective mitigation efforts.

The project aims to contribute to the ongoing efforts to understand and mitigate the impact of the pandemic on society.

3. FlowChart



4. DATA

4.1 Data Gathering:

The dataset used for this project was sourced from Kaggle, specifically from a CSV file located at /kaggle/input/malicious_phish.csv. This dataset is comprehensive, containing 600,000 URL entries categorized under various labels such as benign, phishing, defacement, and malware. The diversity and volume of the data ensure a robust foundation for training machine learning models capable of classifying URLs with high accuracy.

4.2 Data Preprocessing –

In the data preprocessing stage, two essential steps were taken to prepare the data for analysis: Data Cleaning and Transformation and Data Splitting.

4.2.1 Data Cleaning and Transformation –

The raw data underwent several preprocessing steps essential for the subsequent machine learning tasks:

Label Mapping: Each URL category was converted from a textual label into a numeric code to facilitate the processing by machine learning algorithms. This conversion was achieved using a predefined dictionary: {'benign': 0, 'phishing': 1, 'defacement': 2, 'malware': 3}. This step is crucial as it transforms categorical labels into a format that can be easily handled by various classification algorithms.

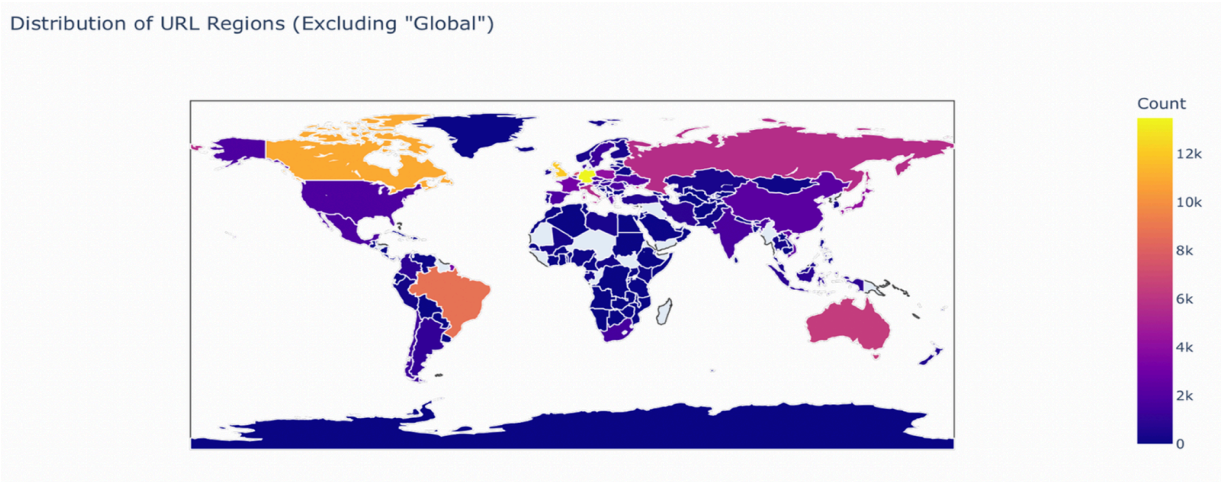
Feature Extraction: To convert URL strings into a numerical format suitable for machine learning, the TfidfVectorizer and CountVectorizer from the sklearn library and BERT from transformers were employed. The TfidfVectorizer was configured to analyze the URLs at the character level (analyzer='char') without converting characters to lowercase (lowercase=False). This configuration helps preserve the original structure of URLs, where case differences and specific character sequences can provide significant information for classification purposes.

4.2.2 Data Splitting -

The dataset was divided into training and testing sets using an 80-20 split, ensuring that the models are validated against unseen data to avoid overfitting. The random state was set to 32 during the splitting process to ensure that the results are reproducible and consistent across different runs.

4.3 Data Visualization and Exploratory Data Analysis -

World Map: Introduce a choropleth world map that visualizes the origin or target demographic of different URL categories if geographic data is available. Explain how this visualization helps in understanding the geographical distribution and potential targeted attacks.
URL Length Distribution: Provide a histogram of URL lengths to show the variance and common patterns in the dataset. Discuss how URL length might influence the classification and the insights this graph provides about the dataset.



5. METHODOLOGIES

5.1 URL Decomposition

5.1.1 Objective and Methodology

The process of URL decomposition is fundamental to isolating critical components of URLs, which often bear indicators of their nature, whether benign or malicious. This step involves parsing each URL into its constituent parts: the scheme, subdomain, domain, top-level domain (TLD), path, query parameters, and fragment. The granularity achieved through this decomposition facilitates detailed analysis and feature extraction, which are pivotal in distinguishing between URL categories effectively.

5.1.2 Implementation Detail

For precise parsing of URLs, the tldextract library was employed. This Python library excels in extracting detailed URL components while adeptly handling variations in URL structures and anomalies often encountered across different domains.

5.2 Tokenization Techniques

5.2.1 Approach and Rationale

In the analysis of URLs, character-level tokenization was deemed more appropriate than word-level due to the structural and syntactical significance of each character. This approach allows for the detailed examination of patterns within URLs, including file extensions, specific directories, and delimiter characters, which are vital for identifying potentially malicious URLs.

Here we approach tokenization and embedding using TF-IDF, Count vectorizer, BERT libraries or transformers. And the same can be found in flow charts and code files for more detail.

5.2.2 Tools and Configuration

The project utilized TfidfVectorizer and CountVectorizer from the sklearn library, configured for character-level analysis, and BERT from transformers to understand the semantic meaning for the words in URL. This configuration was chosen to highlight unique and significant characters within URLs through TF-IDF scoring, and to account for the frequency of character sequences using count vectorization, which are statistical techniques. While BERT is more of semantic meaning based, where we use a BERT transformer to tokenize the input and generate embeddings, and to give a bit of context of bert here it is “BERT is not trained on an existing "word model" but rather learns its own word embeddings through the Transformer architecture during pre-training".

5.3 Selection of Libraries

5.3.1 Justification for Library Choice

The selection of TfidfVectorizer and CountVectorizer was predicated on their robustness, flexibility, and the depth of feature extraction capabilities they offer for text data. These tools are particularly adept at transforming text into a numerical format that can effectively be used by machine learning algorithms, making them ideal for the text-rich components of URLs.

And the same applies to BERT which is taken from the transformer library.

5.3.2 Integration into Data Processing Pipeline

These vectorization tools were seamlessly integrated into the data processing pipeline subsequent to the URL decomposition phase. This arrangement ensures that the structural and semantic richness of the raw URL data is fully harnessed, transforming it into a potent set of features ready for machine learning classification. And this applies to all tokenizations.

6. Directory Structure and Implementation Details

The organization of the project repository is pivotal for ensuring effective data management, model development, and evaluation. This section provides a concise overview of the directory structure and the functional role of each component within the project.

6.1 Directory Layout

Based on the functionalities discussed in the provided code snippets and the inclusion of Jupyter notebooks, here's a detailed breakdown of the probable directory structure:

/kaggle/input: This directory contains the dataset utilized in the project, notably the malicious_phish.csv file. This arrangement is typical in environments where datasets are pre-configured or pre-loaded, such as on the Kaggle platform.

Jupyter Notebooks:
The project comprises three Jupyter notebooks that encapsulate the entire process from data loading, preprocessing, model training, and evaluation. These notebooks serve as both a development environment and a detailed documentation tool that outlines the methodology in an executable format.

Functionality and Implementation:
The Jupyter notebooks likely include all necessary code segments to perform the following operations seamlessly within an interactive environment:

Data Loading: Data is directly loaded from the /kaggle/input directory using the pandas library, ensuring straightforward access and manipulation of dataset components.
Preprocessing: The preprocessing steps are implemented within the notebooks, including the conversion of categorical labels to numeric form and the preparation of the data for machine learning models.
Feature Extraction: Feature extraction is conducted using techniques such as TF-IDF vectorization, BERT where URL data is transformed into a suitable numeric format for model training.
Model Training and Evaluation: Various machine learning models are trained and evaluated directly within the notebooks. This includes using libraries from sklearn to apply algorithms like Logistic Regression, Decision Trees, and others.

Integration and Workflow:
The project's notebooks are designed to encapsulate the full lifecycle of the project within a single, coherent narrative. From initial data analysis to complex model evaluations, each notebook is structured to provide insights into specific stages of the project:

Code files can be found in folder Notebook/
Notebook1(nlp-project-tf-idf.ipynb) focuses on data loading, preprocessing, generating tokens using tf-idf, applying Ml models and evaluation

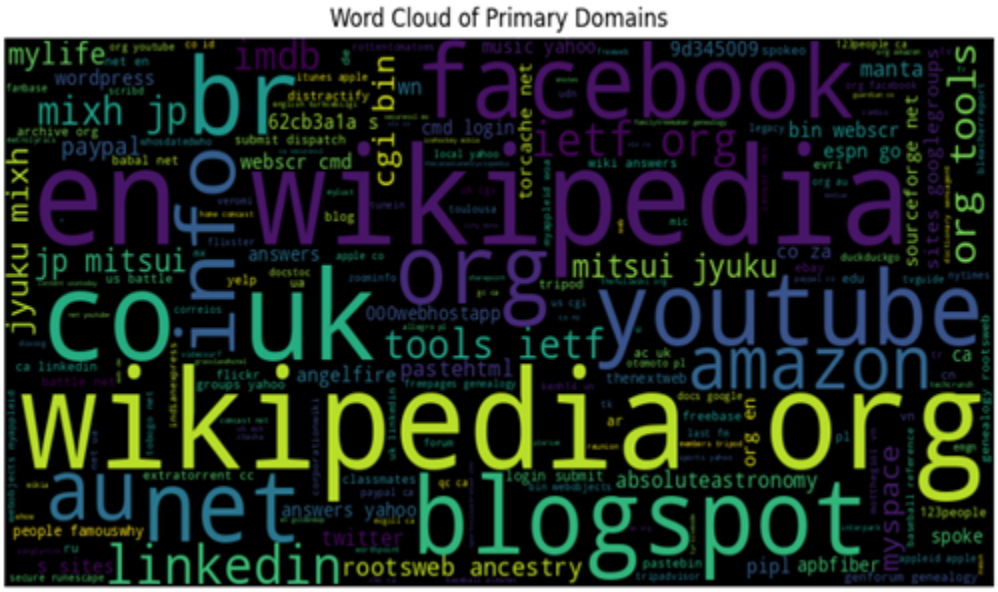
Notebook2(nlp-project-count-vectorizer.ipynb) focuses on data loading, preprocessing, generating tokens using count vectorizer, applying ML models and evaluation.

Notebook3(NLP_Final_Project_URLdecomposition_&_BERT.ipynb) specifically focuses on Implementing or creating features from URL(No NLP) which is called URL decomposition, it focuses on url meta data, and then sing BERT(NLP) as well. So, in both ways, we focus on data loading, preprocessing, generating tokens or embeddings and applying Ml models and evaluation.

Notebook4(nlp-final-project-data-visualizations.ipynb) focuses on data visualization like finding the number of urls in each category, geographical origin of urls from countries etc..

6.2 Word Cloud

The words are visually represented in a word cloud. The word cloud displays the presence of the most widely used and frequent words. Each word's size is determined by how many times it appears in the text used to generate the word cloud. The size of frequent words is larger than that of less frequently used terms.



7. MACHINE LEARNING MODELS

The selection of machine learning models for the URL classification project was guided by the need to effectively distinguish between benign(safe) and malicious(unsafe) URLs. The models are chosen to represent a blend of simple to complex approaches, each with unique strengths in handling the intricacies of textual and character-based data typical of URLs. Below is a detailed

analysis of each model, including their theoretical foundations, practical applications, and specific reasons for their selection and configuration in this project.

7.1. Logistic Regression

Theoretical Background: Logistic Regression is a fundamental statistical model that estimates the probabilities using a logistic function, which is particularly useful for binary classification tasks. It operates by fitting a logistic curve to the decision boundary, where the probabilities of the dependent variable (label) change from 0 to 1.
Practical Applications: Widely used in fields such as credit scoring, medicine for disease diagnosis, and social sciences for predicting categorical outcomes. Its popularity stems from the model's transparency and the ease with which its results can be interpreted.
Configuration Details: In this project, Logistic Regression was implemented with a high iteration count (max_iter=1000) to ensure the model converges properly, given the potentially complex and large feature space derived from the TF-IDF vectorization of URL data. The solver was set to 'liblinear' due to its effectiveness in handling binary classification problems.

7.2. K-Nearest Neighbors (KNN)

Theoretical Background: KNN belongs to the family of instance-based, non-parametric learning algorithms. It assumes that similar things exist in close proximity, in other words, nearer objects are more closely related to each other than objects farther away. This principle is particularly powerful in classification tasks where the similarity between instances directly influences their labeling.
Practical Applications: KNN is extensively utilized in recommendation systems, image recognition, and genetic pattern identification where the similarity between instances significantly impacts the outputs.
Configuration Details: For the URL classification, the Euclidean distance metric was chosen to quantify the similarity between feature vectors. The optimal number of neighbors (k) was selected through cross-validation techniques, ensuring that the model does not overfit by considering too many or too few neighbors.

7.3. Decision Trees

Theoretical Background: Decision Trees are a type of supervised learning algorithm that is used for classifying problems. They work by splitting the data into two or more homogeneous sets based on the most significant attributes making the groups as distinct as possible.
Practical Applications: Decision Trees are commonly applied in operational research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. They are also used in the manufacturing and production industries to manage decisions on the shop floor.

Configuration Details: The Decision Tree in this project utilized the 'entropy' criterion to measure the quality of splits. Entropy is a way of measuring impurity or randomness in the dataset, and by selecting the splits that decrease entropy the most, the tree grows in a way that incrementally increases homogeneity.

7.4. Random Forest

Theoretical Background: Random Forest improves on the decision tree algorithm. It creates an ensemble of decision trees usually trained with the bagging method. The general idea of the bagging method is that a combination of learning models increases the overall result.
Practical Applications: Used extensively in banking, stock trading, medicine, and e-commerce, Random Forests are pivotal for risk assessment, predicting stock behavior, patient diagnosis, and inventory classification.
Configuration Details: The Random Forest model was configured with 100 trees (n_estimators=100) and employed the 'sqrt' method to select the maximum number of features considered at each split. This approach helps in reducing training time and improving model performance by introducing randomness in the feature selection process, which decorrelates the trees and makes the forest more robust to noise.

7.5. XGBoost

Theoretical Background: XGBoost stands for Extreme Gradient Boosting and is a more efficient implementation of the Gradient Boosting framework. XGBoost improves upon traditional Gradient Boosting methods by introducing a more regularized model formalization to control over-fitting, which provides better performance.
Practical Applications: XGBoost has been at the heart of winning solutions in ML competitions due to its scalability and good performance across a variety of problems. It is broadly used in industries for challenges like predictive maintenance, anomaly detection, and customer segmentation.
Configuration Details: Configured to maximize computational speed and model efficiency, the XGBoost model was fine-tuned with grid search, focusing on hyperparameters like the learning rate and the depth of the trees. The learning rate controls the impact of each tree on the final outcome, while the depth regulates the complexity of the decision rules.

7.6. Support Vector Machine (SVM)

Theoretical Background: SVM is a powerful classifier that works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane.

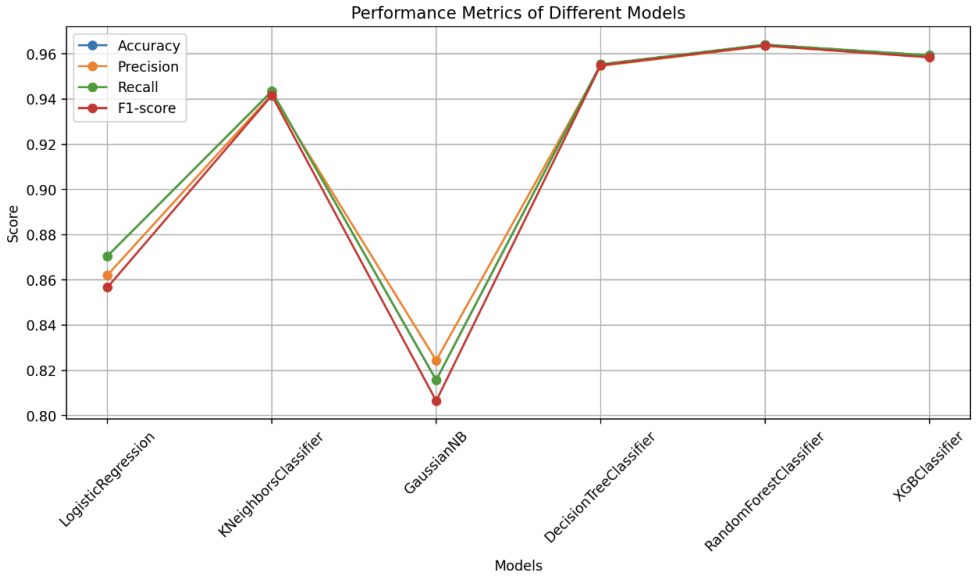
Practical Applications: SVMs are widely used in the industry for image classification, text categorization, and bioinformatics. They are highly preferred due to their effectiveness in high-dimensional spaces.
Configuration Details: For URL classification, an SVM with a linear kernel was tested due to the binary nature of the output and the high dimensionality of the input space. The C parameter, which controls the penalty for misclassifying data points, was optimized using cross-validation.

7.7. Multinomial Naive Bayes

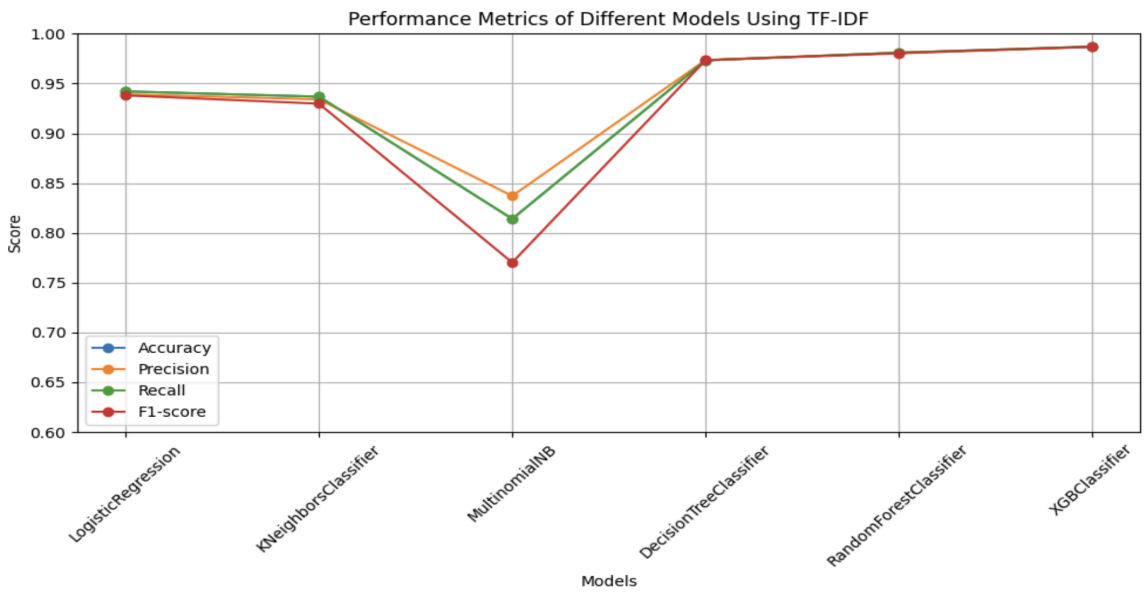
Theoretical Background: This model extends Naive Bayes to more than two classes. It is particularly suited for classification with discrete features (e.g., word counts for text classification).
Practical Applications: Beyond text classification, it is also used in medical diagnosis and spam detection, where it performs well under the assumption that the features are conditionally independent given the class.
Configuration Details: Applied directly to the frequency of the characters obtained from the Count Vectorizer, this model was set up with default parameters, assuming independence between features, which simplifies computation and is particularly effective in high-dimensional spaces.applications.

8. Results

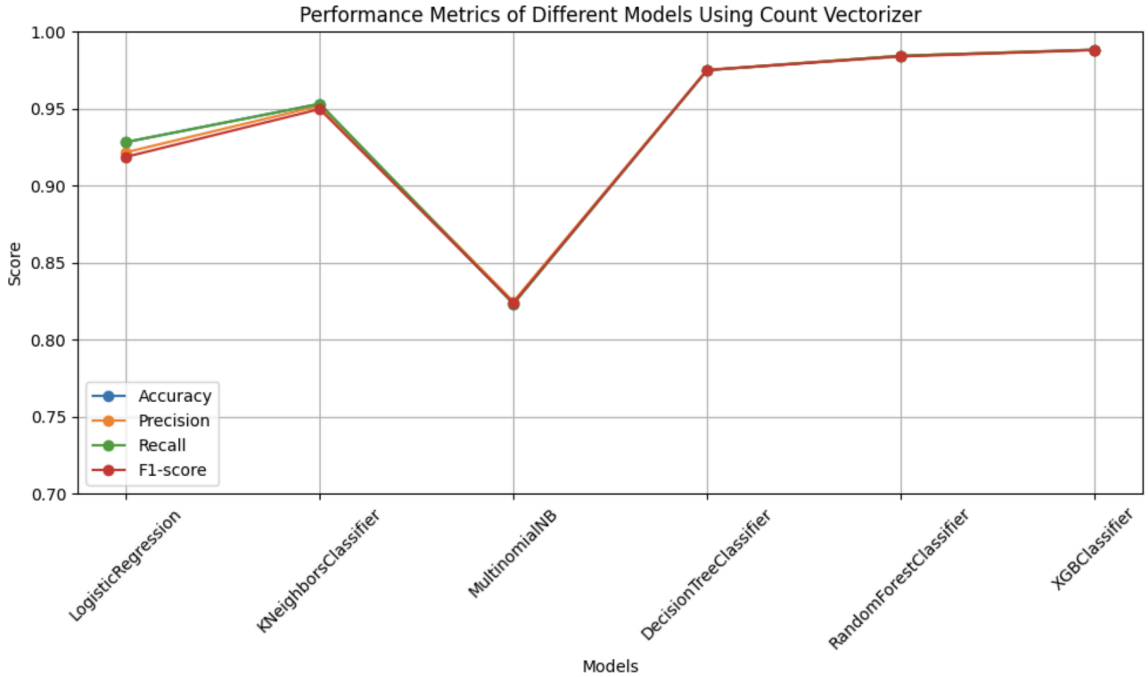
8.1. URL Decomposition



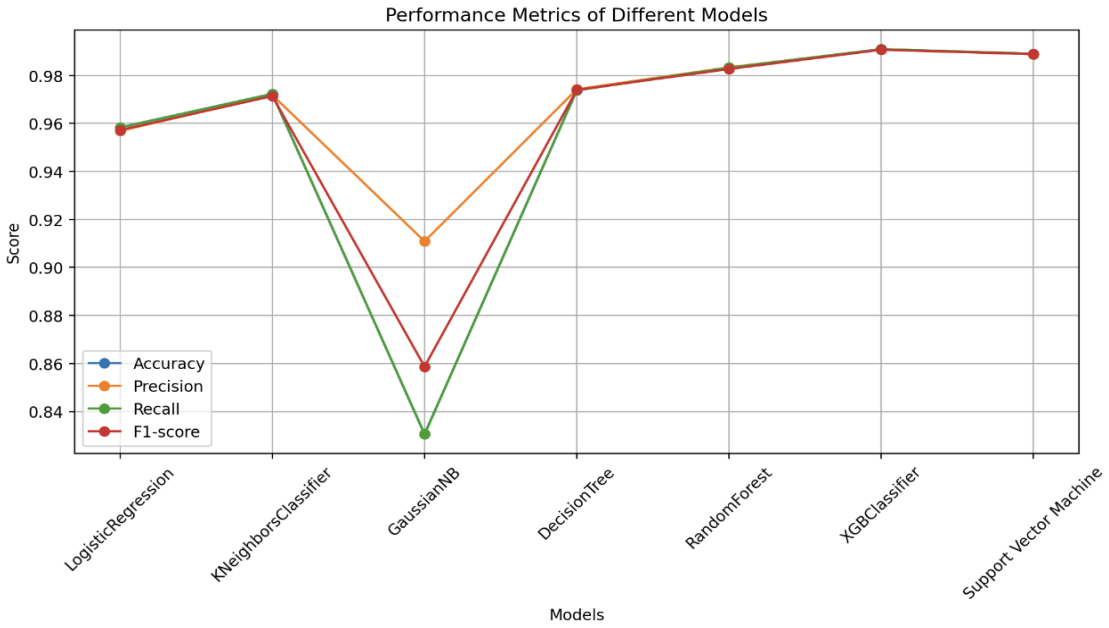
8.2. TF-IDF



8.3. Count - Vectorizer



8.4. BERT



8.5. Observations

Successful Utilization of Performance Metrics:

- **Well-Performing Algorithms:**

  RandomForestClassifier and XGBClassifier: These demonstrated exceptional performance across accuracy, precision, recall, and F1-score, as evidenced by the analysis of four different graphs. Their ability to effectively handle complex data patterns ensures high reliability and robustness in various predictive tasks.

  Support Vector Machine (SVM): Maintains competitive performance, particularly in accuracy and precision, making it a stable choice for scenarios that demand high reliability.

- **Moderately Performing Algorithms:**

  LogisticRegression and KNeighborsClassifier: These models show some fluctuations in performance metrics across different feature representations. This variability points to a sensitivity to preprocessing techniques, which could limit their applicability across diverse scenarios.

DecisionTreeClassifier and GaussianNB: Significant performance dips in recall and F1-score suggest potential overfitting or a lack of generalization from training data. The influence of preprocessing methods necessitates careful model tuning and validation.

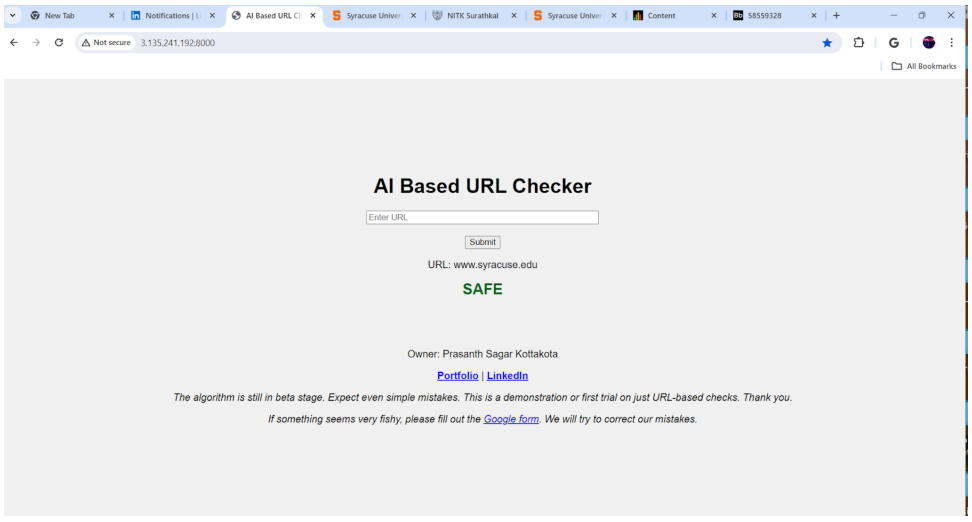
Leveraging Insights for Predictive Modeling:

- In evaluating various models, the consistent superior performance of **RandomForestClassifier** and **XGBClassifier** resembles how **BERT** outshines traditional models in natural language processing. Their robust handling of complex data underscores their effectiveness and reliability, positioning them as optimal choices for high accuracy in predictive tasks. This underscores their potential as preferred models for complex data analysis within projects.

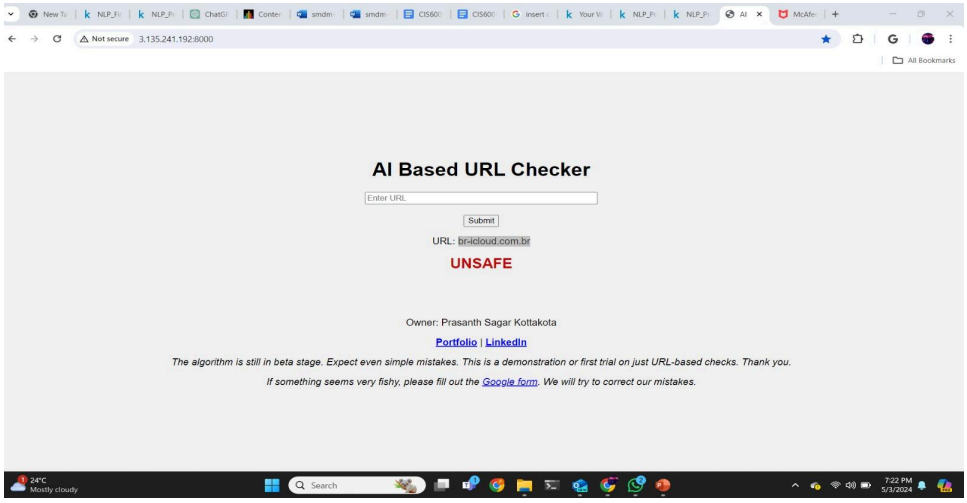
Addressing Data Challenges:

- **Data Imbalance Issue:** The predominance of safe URLs in the dataset creates an imbalance, potentially skewing the training process and causing the model to underperform in identifying the minority class of unsafe URLs.
- **Overfitting Scenario:** High accuracy and scores during testing indicate overfitting, where models perform well on training data but fail to generalize effectively. This becomes a significant issue in real-time applications, where the model's performance in new environments does not align with test results.

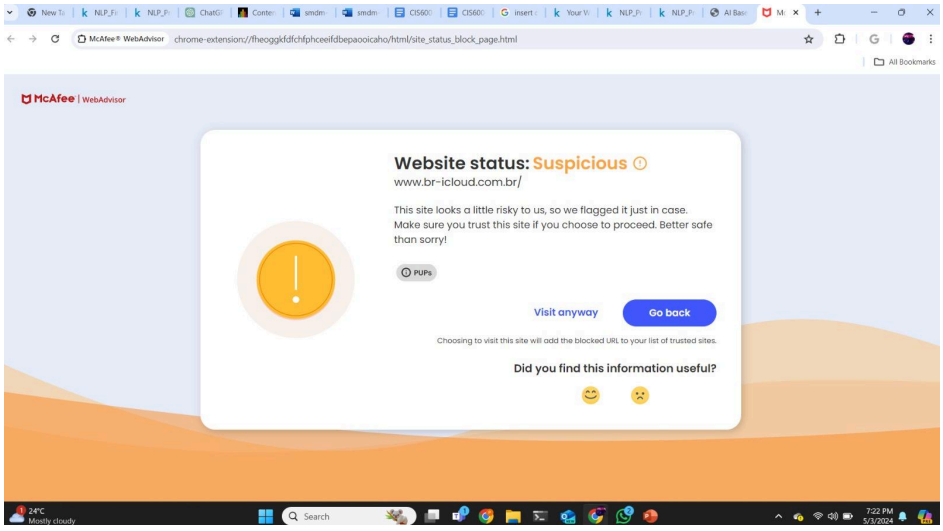
8.6. Safe URL



8.7. Unsafe URL



8.8. Verified Unsafe URL (using McAfee)



8.9. Evaluation of Deployed Model

The **Malicious URL detector** underwent rigorous evaluation to assess its precision and reliability in differentiating **'safe'** from **'unsafe'** URLs. The model accurately classified "www.syracuse.edu" as 'safe,' demonstrating its capability to reliably identify legitimate and secure websites, consistent with expectations for well-established educational domains. This

performance underscores the robustness of the model's training and its precision in real-world scenarios.

Furthermore, the URL "www.br-icloud.com.br" was identified as 'unsafe' by the system. To confirm this finding, the URL was also assessed using **McAfee's** security service, which marked it as 'suspicious.' This agreement between the system's outputs and a recognized security service validates the model's effectiveness in detecting potentially hazardous URLs.

These results highlight the practical utility of the model in real-world applications and emphasize the need for ongoing updates and rigorous testing. This is particularly crucial for URLs that may not be straightforward to classify, ensuring the safety and accuracy of the web browsing experience. Such continuous improvement and validation are essential for maintaining the relevance and effectiveness of the model in a dynamically changing threat landscape.

Link to the deployed model: “http://3.135.241.192:8000/”

9. Conclusion

Integration of Advanced NLP and ML Techniques:

- Combines cutting-edge NLP techniques like **BERT** for semantic understanding with traditional methods such as **TF-IDF** and **Count Vectorizer**, ensuring comprehensive analysis of URL data.
- Employs robust ML algorithms, including Random Forest and Logistic Regression, to adapt swiftly to new threats and enhance predictive capabilities.

Real-Time Application and User Accessibility:

- Deployed using the **Django framework**, the project offers a user-friendly interface that allows for real-time URL safety assessments, making it accessible and practical for everyday use.

Continuous Learning and Adaptation:

- Features a dynamic learning component that updates the model with new data, maintaining the system’s effectiveness against evolving cyber threats.

Contribution to Cybersecurity Practices:

- Provides valuable insights and tools for cybersecurity firms and web developers, aiming to enhance security protocols and protect users from potential harm.
- Sets a new standard in the domain of cybersecurity by prioritizing early detection and prevention of threats through advanced technological integration.

10. Implications and Future Directions

Continuous Data Update and Model Re-training:

- Given the dynamic nature of malware URLs, which often change to evade detection, it is crucial to implement continuous data updating mechanisms. Employing automated pipelines and data scraping techniques will ensure that the model stays updated with the latest URL patterns.
- Regular re-training of the machine learning models with fresh data will help maintain high accuracy and adaptiveness, making the system robust against newly emerging cyber threats.

Expansion of Data Sources and Real-time Analysis:

- Expanding the sources from which data is scraped can provide a more comprehensive dataset, encompassing a wider array of potential threats. This would allow for a broader understanding of malicious tactics and improve detection strategies.
- Implementing real-time data analysis and threat detection within the system can significantly decrease response times and enhance the protective measures provided by cybersecurity applications.

Integration into Cybersecurity Infrastructure:

- The project's technology can be seamlessly integrated into cybersecurity firewalls to enhance early detection capabilities for malicious URLs. This integration would provide a robust defense mechanism, reducing the vulnerability window against new threats.

11. References

1. https://ar5iv.labs.arxiv.org/html/2310.05953
2. https://ieeexplore.ieee.org/abstract/document/9633889