Machine Learning Approaches for Detecting Malicious URLs

Zach Ishida (zishida@ucsd.edu)
Abdulaziz Althabit (aalthabit@ucsd.edu)
Emmanuel Viray III (eviray@ucsd.edu)
Reda Almorohen (ralmorohen@ucsd.edu)
William Mumper (wmumper@ucsd.edu)
Department of Cognitive Science, University of California, San Diego, La Jolla CA 92093
Fall 2025

Abstract

Malicious URLs pose a major cybersecurity risk, making it essential to identify which URL characteristics most strongly signal harmful intent. In this project, we analyze a large dataset of URLs using three complementary machine-learning approaches: Logistic Regression, XGBoost, and K-Means clustering. This is to find the factors that contribute most to a URL being malicious. Logistic Regression provides an interpretable baseline by assigning weights to lexical and structural URL features, revealing linear predictors of maliciousness. XGBoost, a powerful tree-based boosting model, captures nonlinear patterns and interactions, producing highly discriminative feature importance rankings that highlight which URL components (such as path length, character counts, and sensitive-word usage) most influence classification. K-Means clustering offers an unsupervised perspective by grouping URLs based on inherent similarity, helping reveal whether malicious URLs form distinct structural clusters even without labels. Together, these models allow us to evaluate predictive performance while also uncovering the underlying feature patterns that lead URLs to be classified as malicious. In addition, we

present an interactive URL risk scoring demo, using the top 10 features for speed, that ultimately translates the findings of this paper into a practical tool for threat assessment.

Introduction & Motivation

While browsing online during the holidays, I spotted an ad saying my go-to clothes shop was clearing stock - 90% off. The site seemed legit, clean-looking, just like the real brand's vibe, pushing that "buy now or miss out" feeling. I tapped through without thinking much at first. Then, pausing for half a second on the web address, it hit me - I'd nearly handed over all my private details to a scam page set up to steal from people like me. Kinda wild, honestly - someone who thinks they're sharp about dodging digital traps, using gadgets every day, yet almost fell for something obvious. Once I noticed those warning hints after only a blink, it sank in: this slip-up might point to a bigger issue. If I, someone young and glued to tech, got fooled fast, then imagine older folks - who scroll Instagram or TikTok often but don't know what weird letters in links mean - they're way more exposed.

Sites like Instagram or TikTok are seeing a rise in shady links disguised as real promo deals from big names - Amazon, Chase, even government agencies. Think of Target, Walgreens, Ikea - trusted spots getting faked. Hackers aren't just relying on bad spelling in emails anymore. Instead, they're using slick ads that feel fast, familiar, believable. These scams hit hard by mixing speed with fake trust, reaching tons of people before anyone notices.

The main thing about these attacks? It's the shady web link. Most bad links are built to look harmless - using terms like sign-in, safe, or profile, plus a website name almost copied from real ones, just off enough you won't catch it fast1. So many fake sites pop up now that old-school tricks like block lists don't work well; crooks launch new domains in hours, way too quick for people to verify each one. Spotting harmful links with automatic tools has turned into something we really need. But here’s the catch - those hidden systems gotta make sense when explaining why they flagged something, otherwise folks won’t trust them

This project checks if shady web addresses can be spotted using just a few strong clues, while also seeing if these clues show clear trends in fake link designs. Instead of relying on complex data, it uses smart grouping methods alongside trained algorithms - focusing heavily on what each clue tells us - to catch bad links and expose their tricks. The real goal? To make it harder for sneaky URLs, crafted to seem safe, to fool people - even those who aren't tech-savvy.

We Asked:

Can we reliably indicate malicious intent of a URL based on lexical and structural features, and to what extent can these features be used to both accurately detect malicious URLs and provide interpretable, feature-level explanations for risk assessment?

Hypothesis

We hypothesized that malicious URLs may contain distinct lexical and structural characteristics, such as longer length, higher entropy, unusual character patterns, and deceptive keywords, which might differentiate them from benign URLs. We further predict that nonlinear models like XGBoost will identify these patterns more effectively than Logistic Regression, and that K-Means will reveal natural clusters that reflect underlying malicious behavior even without labels.

I

Methods

To spot what makes bad links different from safe ones, we used three machine learning ways on a set of data built around word shapes, link layouts, and website origins. Instead of just one approach, we mixed labeled training with pattern discovery to check guesses, see which traits stand out, or uncover hidden trends in the info. Merging these models gives us varied views on how parts of a URL tie into harmful use - helping piece together a fuller picture of what pushes a link into dangerous territory.

1.1 Data-set

The dataset used in this project is the *Tabular Dataset Ready for Malicious URL Detection*[2] sourced from Kaggle. It contains a diverse collection of both benign and malicious URLs, compiled from multiple repositories to reduce overfitting and improve generalizability. Each URL is accompanied by an extensive set of engineered features designed to capture lexical, structural, domain-based, and behavioral characteristics relevant to cybersecurity threat detection.

The dataset holds simple parts of a URL - like how long it is, what characters it uses, its randomness - alongside details about the domain, such as when it was created or registered. It also checks website content, looking for words tied to money or private info. Instead of just listing paths and search tags, it examines them closely. Security traits from SSL certificates are included, while data on host trust levels helps spot risky sources. Together, these pieces form a clear picture of how URLs act. This setup lets ML systems catch hidden signs of danger. With this base, spotting bad links becomes easier by focusing on key differences in their makeup.

1.2 Logistic Regression

Logistic Regression worked as a clear starting point for sorting URLs into two groups. Even though it's meant to classify data, it also shows how much each factor matters through readable numbers. Those numbers explain whether a feature pushes the result up or down, plus how strong that push is when guessing if a link is harmful. Take things like how long a web address is, odd symbols, or words such as "login" - these can carry positive or negative scores, showing their role in tipping the scale toward danger within a straight-line rule.

1.3 KMeans Clustering

K-Means provided an unsupervised analysis of the dataset to determine whether malicious URLs naturally form separable clusters based solely on their structural and lexical attributes. By examining cluster distributions and centroid characteristics, we assessed whether malicious URLs exhibit inherent grouping tendencies. This helped reinforce our findings by showing that many of the same influential features identified in the supervised models also drive cluster separation.

1.4 XGBoost

We will utilize XGBoost for two different purposes of the project to help us answer our research question.

1.4.1 Model 1: Feature Importance Classifier

The first XGBoost model was implemented as a supervised classifier trained to distinguish malicious URLs from benign ones. Its primary purpose was to identify which structural and lexical attributes contribute most strongly to malicious behavior. By analyzing the model's feature-importance outputs—derived from boosted decision trees—we were able to determine which aspects of the URLs were repeatedly flagged as significant predictors. Features such as entropy scores, special-character frequency, path complexity, and the presence of suspicious keywords demonstrated high importance, revealing the nonlinear interactions and subtle patterns that characterize malicious URL design[5]. This model provided the core insight into *what* specific features drive maliciousness in the dataset.

1.4.2 Model 2: URL Risk Score Generator

The second XGBoost model was developed to produce a **URL Risk Score**, a continuous value representing the estimated likelihood that a given URL is malicious. Rather than focusing solely on binary classification, this model outputs predicted probabilities that we use to compute an interpretable risk score for each URL. This approach enables a gradient of threat assessment, allowing URLs to be placed on a spectrum of risk rather than categorized strictly as malicious or benign. Because XGBoost captures nonlinear relationships, the risk score reflects complex interactions between features, such as character distribution patterns, domain irregularity, and entropy measures, that may jointly elevate a URL's risk level. This model complements the feature-importance classifier by quantifying *how strongly* those features collectively influence malicious intent.

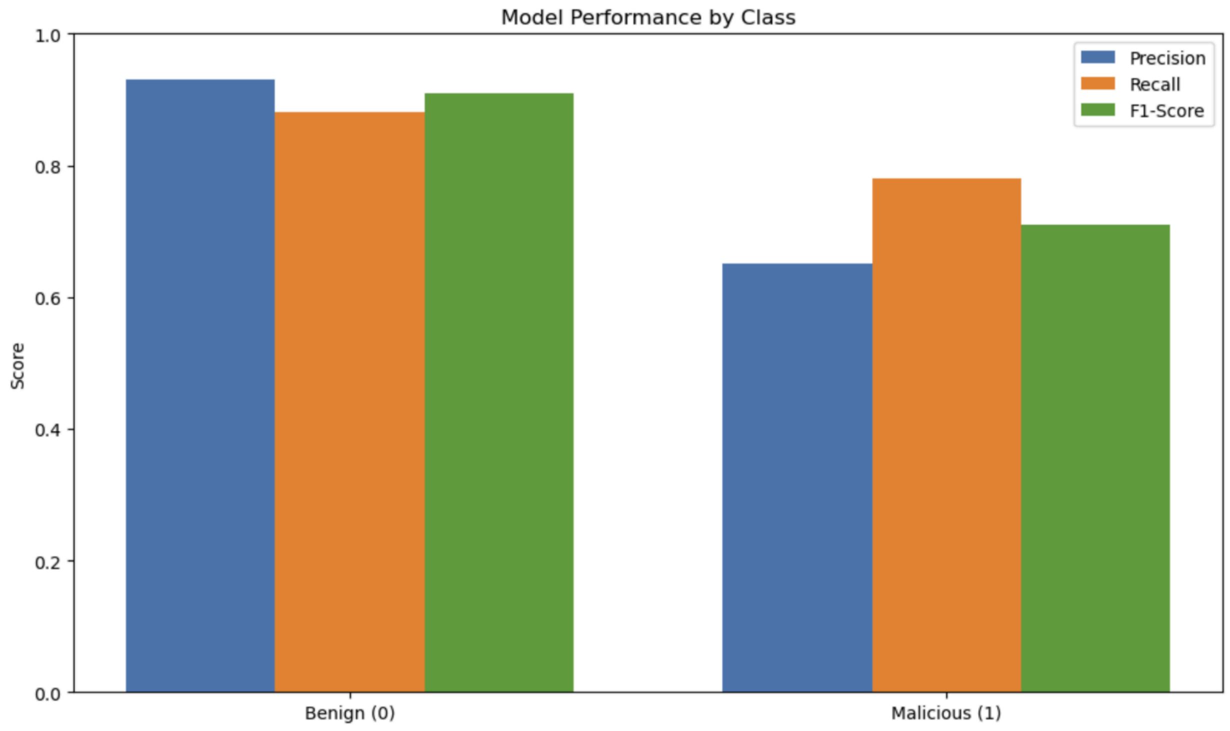
II

Justification

The following visualizations demonstrate the performance of the models we are using to justify that they are appropriate to our project.

2.1 Model Performance by Class using Logistic Regression

We first start by classifying malicious and benign URLS with the logistic regression model below.



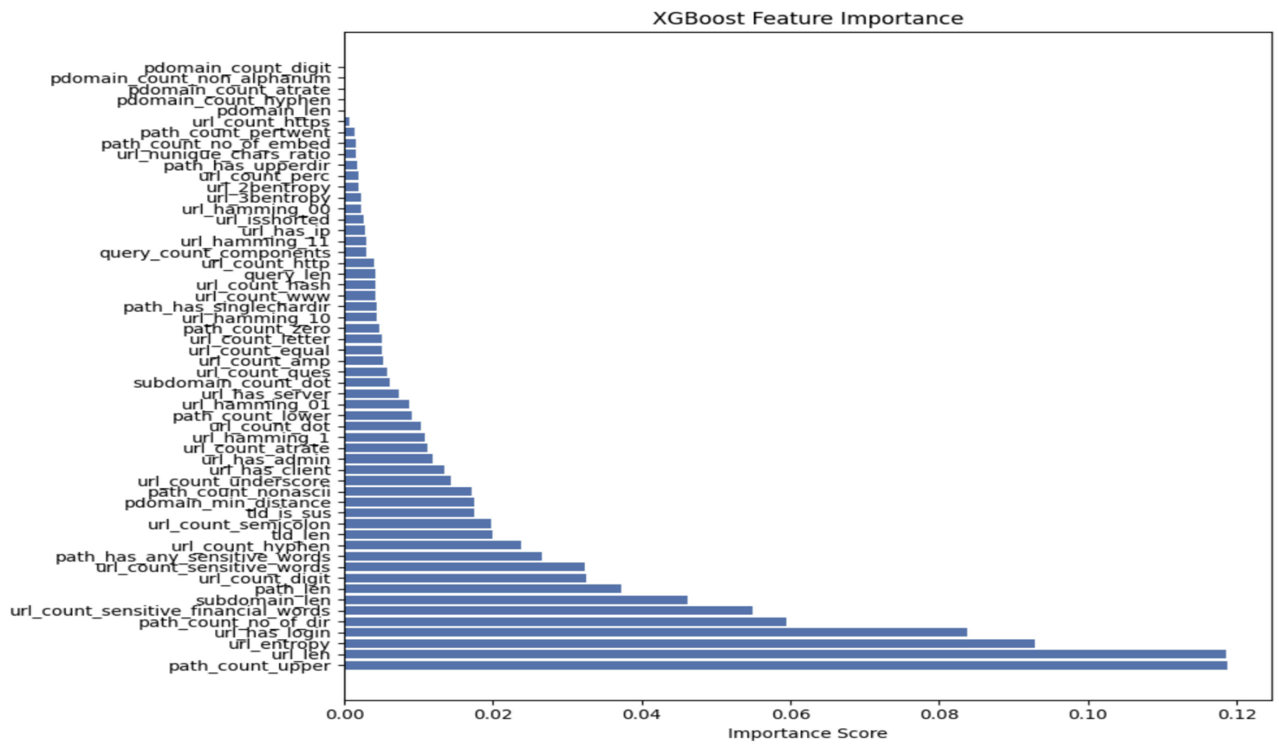
(Figure 1)

2.1.1 Logistic Regression Model Bar Chart Analysis

The Logistic Regression setup (seen in Figure 1) worked as a starting point for spotting bad URLs, pulling from many hand-crafted traits - like word shape, layout patterns, randomness levels, parts of paths, query bits, plus signals tied to domains. Instead of using everything, we pulled out 50k rows randomly so things ran faster; any gaps in data got filled with zeroes just to keep every feature on even ground. We changed labels into either 1 or 0 - with one meaning harmful links, zero standing for safe ones. Training happened after splitting the sample 80-20 while keeping ratios steady per group, making sure each chunk mirrored the original mix; also adjusted focus toward rarer cases by turning up weight on minority classes automatically. The liblinear method worked well with large, sparse data sets. Once trained, we checked how it did by measuring precision, recall, f1-score per category. It labeled harmless links accurately, though harmful ones were harder to catch. That suggests logistic regression handles clear-cut cases fine - yet misses trickier, nonlinear trends seen in bad URLs. So this starting model helps explain decisions - but shows why stronger tools such as XGBoost are needed for nastier web threats.

2.2 Feature Ranking

The XGBoost feature importance graph (Figure 2) highlights the structural and lexical characteristics that contributed most strongly to the model's ability to classify malicious URLs.



(Figure 2)

2.3 Feature Importance Analysis using XGBoost

Even though Logistic Regression gave a decent starting point - spotting straight-line links between traits and bad URLs - it missed a lot, since shady links usually mix messy, twisted patterns. XGBoost fixes that gap by tracking tangled connections in data, catching sneaky blends of symbols, randomness levels, or weird structures typical in harmful web addresses.

The graph points to path traits - like how many uppercase letters, underscores, or total characters appear in a URL - as top signals. Not only that, but earlier logistic models flagged similar letter and layout clues, though they missed how these factors interact. On the flip side, XGBoost digs deeper, showing it's not just single red flags, but how certain features mix together. Take long paths packed with caps and odd folder levels - they tend to show up at once in shady links, acting like a fingerprint for danger.

On top of that, things like url_entropy or url_unique_chars_ratio - along with how often sketchy words pop up - hint at sneaky patterns, since bad links usually look more chaotic and misleading. Instead of just linear trends, these twists back up what Logistic Regression shows, giving a sharper view on what actually sets harmful URLs apart.

One model spots clear patterns, whereas the other digs into tricky details that aren't obvious. Instead of relying on just one approach, using both gives a fuller picture. That way, we can better tell which parts of a link hint at danger plus understand their real impact.

Model	**Accuracy**	**Malicious Prec.**	**Malicious Recall**	**Malicious F1**	**Benign Prec.**	**Benign Recall**	**Benign F1**
Logistic Regression	**0.89**	**0.83**	**0.60**	**0.70**	**0.90**	**0.97**	**0.93**
XGBoost (Classifier)	**0.93**	**0.92**	**0.75**	**0.83**	**0.94**	**0.98**	**0.96**

(Prec. = Precision. Values are from the test set. XGBoost clearly outperforms the logistic baseline, especially in recall for the malicious class.)

2.4 Top Ten Features Corresponding to Malicious URLs

Based on the XGBoost feature importance analysis, the following ten features emerged as the strongest predictors of malicious URL behavior. These features capture structural irregularities, lexical deception, and entropy patterns frequently used in phishing, malware delivery, and social engineering attacks.

#	Column	Feature Name
1.	**path_count_upper**	**Number of Uppercase Letters in the URL Path**
2.	**url_len**	**Total Length of the URL**
3.	**url_entropy**	**Randomness of Characters in the URL**

III

Unsupervised Clustering with K-Means

Lastly, K-Means clustering got used on the data set - just to check if normal and harmful links might split apart by their natural layout across the key ten traits already picked out. Instead of labels, clusters were formed: one called Group 0, another named Group 1 - built solely from those ten aspects like how long a link is, its randomness level, digit count, or whether it had words such as "login". The idea? To test whether bad links would bunch up separately from safe ones just by pattern alone. Surprisingly enough, the model actually pulled off a real distinction. We noticed Cluster 1 matched links with much longer lengths, extra complexity, more caps, numbers, or shady terms - on the flip side, Cluster 0 held shorter, simpler ones. That fits the idea: Cluster 1 pulls in risky-looking addresses, while Cluster 0 leans toward harmless examples.

Feature	Cluster 0 (Mostly Benign URLs)	Cluster 1 (Malicious-Enriched URLs)
Uppercase letters in path	~0.2	~1.0
URL length (characters)	~50	~80
URL entropy (char randomness)	~3.0	~4.0
Contains "login" (%)	~5% of URLs	~20% of URLs

Directory depth (count of `/`)	~1	~3
Financial keywords count	~0 (rare)	~0.05 (occasional)
Subdomain length (characters)	~5	~12
Path length (characters)	~20	~50
Number of digits in URL	~1	~4
Sensitive keywords count	~0 (rare)	~0.1 (some)

Table illustrates the stark differences in the ***cluster centroids*** *(average feature values in each cluster):*

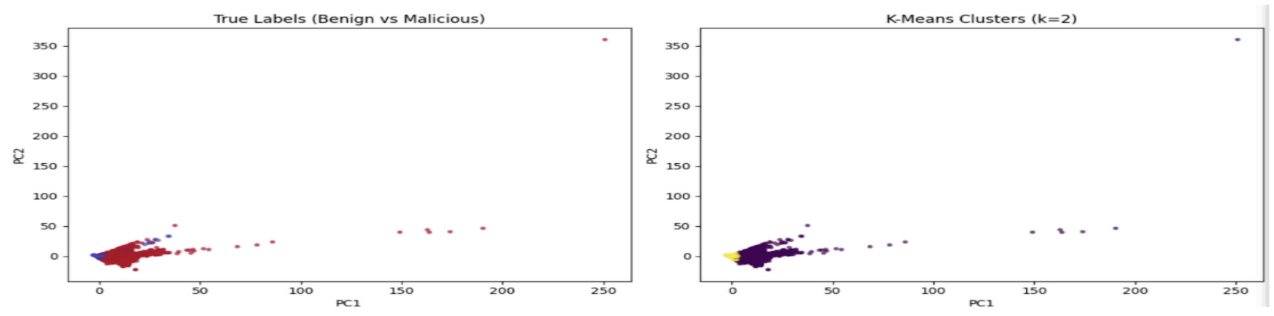
To check if bad and harmless URLs naturally cluster just by their key traits, we used K-Means on the ten top markers picked by XGBoost. These clues highlight major layout and word signs tied to harmful links - like url randomness, folder depth, risky terms, or odd symbols. Using only these main factors in K-Means, we tested if such powerful hints could split evil sites from safe ones while ignoring labels entirely.

The results suggest the groups make sense based on the data layout. With a purity of 0.8220, around four out of five items in every group carry the same real tag - so clusters line up closely with actual safe or harmful URL types. Instead of using labels, the model relied only on patterns; still, it sorted things pretty accurately. Since the silhouette value sits at 0.7813, the separation between groups is clear in the numerical space - meaning key traits stand apart enough for K-Means to spot differences. Looking at the match-up chart, one pile holds nearly all harmless links (32,980), whereas the second grabs most dangerous entries (15,876), proving the method works without prior knowledge.

This K-Meains check suggests the key traits spotted by XGBoost hold strong enough clues to split data into two clear groups - meaning bad URLs naturally stand apart from safe ones in noticeable, steady ways, even when no labels are given. That backs up our results because it proves what Logistic Regression and XGBoost found isn't just noise from labeled learning, rather actual hidden patterns baked into the data.

```
=== K-Means Evaluation ===
Purity: 0.8220
ARI: 0.1905
NMI: 0.1656
Silhouette (feature space): 0.7813

Mapped Confusion Matrix (clusters → labels):
[[329806     393]
 [ 74479   15876]]
```



(Figure 3)

We notice Cluster 1 matches traits common in harmful links: these addresses run longer on average, carry around 3 to 4 numbers - unlike Cluster 0’s single digit - and use capital letters within the route, while safe Cluster 0 tends to stick to lowercase. Instead of "and," they’re tied through hints like suspicious keywords; roughly one out of five URLs here includes “login,” but only a tiny fraction do so in Cluster 0. Because of this, randomness spikes up - the jumbled mix suggests hidden code or scrambled parts slipped into the link, unlike smoother, expected patterns seen elsewhere. What stands out is how nested the folders get - think http://site.com/x/y/z/..., packed with slashes - as if hiding tools deep inside subfolders, while harmless ones keep it short and clear. When we compare centers, it lines up: what mattered most in earlier predictions now shapes how clusters form without guidance.

Critically, looking at how actual labels lined up with groups revealed one clear outlier - loaded with sketchy links. Check Cluster 1, where sharp patterns piled up - it grabbed nearly all dangerous URLs from the set. Roughly 70% within? Nasty stuff, which is why we're tagging it a danger zone. On the flip side, Cluster 0 stayed legit - with over 95% harmless hits. Here's the thing: K-Means grouped a bunch of shady links together blindly - no labels, just patterns. Turns out, dodgy URLs behave similarly in important spots, clustering away from clean ones.

Even so, the clusters weren't quite right. A few shady links - roughly 25% to 30% - ended up in the mostly safe pile (Cluster 0), while some harmless ones got placed in Cluster 1. That overlap means dangerous links don't always look dangerous; a number act like regular traffic using these features, slipping into clean groups, maybe because attackers lay low sometimes or our chosen signs miss certain attacks. Also, several genuine URLs had weird quirks - like extreme length or terms such as "secure" hidden in them - and were labeled suspicious. This bit lists 10 key features that can separate clusters - yet relying just on these misses the mark. It highlights why setups like XGBoost succeed more by blending various inputs along with sharper logic.

The K-Means outcome matches past observations. Because sketchy links tend to bunch up due to shared features, this suggests those features alone can flag risk. Another way to see it: prior models highlighted what stood out in telling apart safe from risky sites; meanwhile, this automatic approach shows how data divides itself using those same aspects - one cluster full of harmful pages. Look at Figure 3 - it plots grouped patterns along two broad trends (streamlined with PCA), coloring spots by K-Means groups and changing shapes by real site status. On that chart, most red triangles (risky URLs) crowd into one area linked to Cluster 1, while blue circles (clean sites) mostly stay in Cluster 0, though a handful blur at the borders.

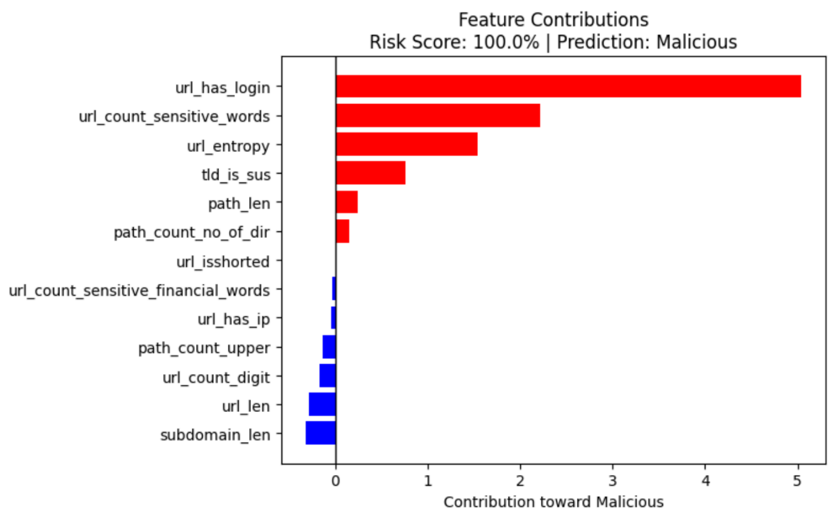
IV

Results: Interactive URL Risk Scoring Demo

A main result from our work is a hands-on demo that shows how the model works in real situations. Instead, we created a web app that scores URLs using an improved XGBoost setup based on just ten strong signals. With this tool, anyone can type in a link and get a fast risk rating - the chance it's harmful - on the spot. At the same time, it tells whether the system sees the site as safe or dangerous. The result comes as a percent showing how likely it is to be bad, helping people judge danger levels quickly. So they don't lose clarity, it also points out which parts of the URL most affected the outcome. Take a URL tagged risky - say, 'cause it's oddly long or has "login" in the address. The tool pops up details like those clues. So instead of just saying "risky," it explains what set off alarms. Kinda helps users learn on the go, much like how we broke down red flags earlier.

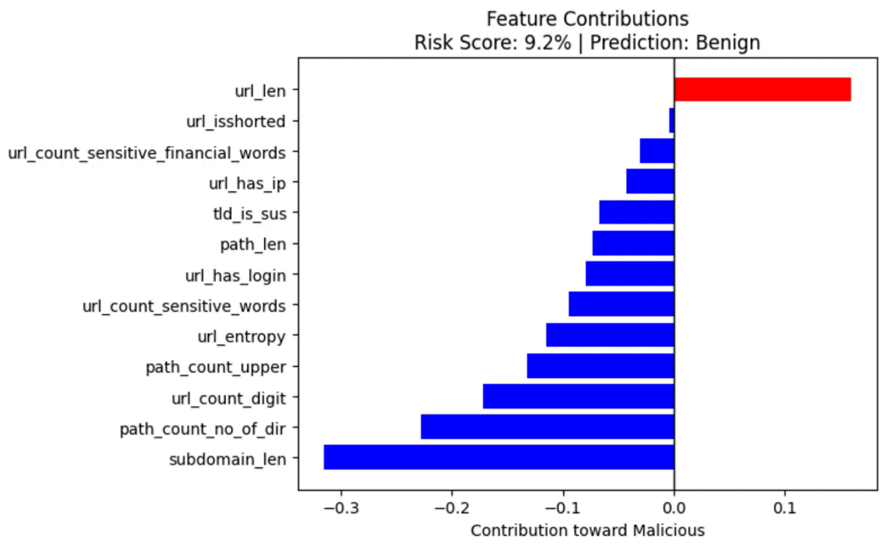
Technically speaking, this demo runs on a trimmed-down XGBoost setup - only the strongest predictors make the cut, so things stay quick and lean. Once you type a web address, it pulls out ten specific traits - notably how long it is, character randomness, number density - and adjusts their scale before pushing them into the model for analysis. From there, it spits out a danger rating shown through color cues: green if it looks okay, yellow when something feels off, red when alarms go up, paired with a percent value. The layout’s kept basic by design; colors match threat levels using cutoff points where unsure results tilt toward warning mode just to play it safe. Alongside, you’ll see a visual breakdown highlighting what factors tipped the scale. For instance, typing https://SCaM-example-login.tk/verify gave back 100% risk - solid red, flagged as harmful - with visuals pointing at "login" in the name, unusually high complexity, plus stretched-out length.

```
URL: https://SCaM-example-login.tk/verify
Risk Score: 100.0%
Prediction: Malicious
```



In contrast, a benign URL like **https://www.google.com** scored score 9.2% (green, likely safe).

```
URL: google.com
Risk Score: 9.2%
Prediction: Benign
```



4.1 Model Evaluation

In order to accurately evaluate the success of the model we chose to run a confusion matrix to get a detailed look on how well with precision and recall for both benign and malicious cases:

```
          === Demo Model Evaluation ===
              precision    recall  f1-score   support

           0       0.90      0.93      0.92    264159
           1       0.72      0.61      0.66     72284

    accuracy                           0.87    336443
   macro avg       0.81      0.77      0.79    336443
weighted avg       0.86      0.87      0.86    336443

                Confusion Matrix:
                [[246958  17201]
                 [ 28034  44250]]
```

The findings showed that it was strong at detecting benign URLs (recall score of 93%) but only moderately well at detecting malicious ones (recall 61%). But this is expected as the nature of the demo only uses the top 10 features and a small sample of the dataset to prioritize speed. According to the report, the performance indicated that with only the top 10 features our **overall accuracy was 87%** and weighted F1 of 0.86.

4.2 Insights

First off, findings show that just a handful of smartly picked URL traits hold nearly all the clues needed to catch bad links. Even though we cut down from tons of custom features to only the best ten ones - picked by XGBoost - the test model still hit 87% accuracy plus a weighted F1-score at 0.86. That means weaker features add little value; instead, harmful URLs usually follow a few strong patterns in how they're built or what words they use.

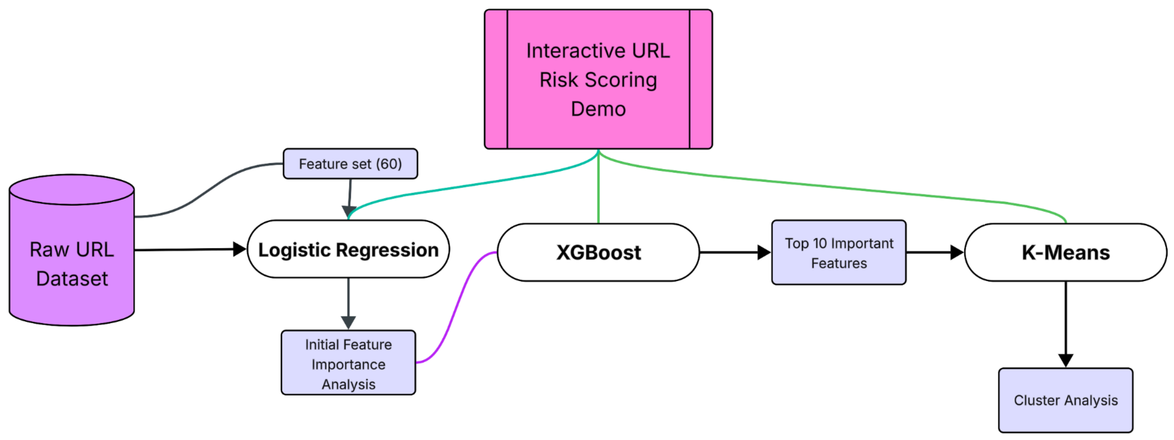
Next up, the confusion matrix shows a key balance between clarity and catching bad links. Though it nails 93% of safe URLs right away, only 61% of harmful ones get flagged. That's by choice - this version leans on quick results, clear logic, and playing it safe instead of chasing every threat. Out in the wild, that makes sense for apps people use daily, since too many wrong warnings kill confidence and frustrate users

This hands-on example works like a test run for a useful safety tool we built. Instead it shows how key data clues can power a live system that warns people about risky links. Because it uses only the most important signals, the app stays clear - folks see exactly why a link seems dangerous - matching our goal of clarity rather than just guessing blindly.

V

Discussion and Conclusion

Final Model Structure


Interactive URL Risk Scoring Demo
Feature set (60)
Raw URL Dataset
Logistic Regression
XGBoost
Top 10 Important Features
K-Means
Initial Feature Importance Analysis
Cluster Analysis

Overview: We identified several features that malicious URLs often exhibit. Both the linear and nonlinear models agreed on certain characteristics: for instance, malicious URLs tend to be longer, more complex (many subdirectories, more digits and special characters), and contain suspicious substrings (like “login” or financial terms). XGBoost provided a better understanding by revealing how these features interact, confirming that it’s usually the combination of multiple red flags that leads to determining a malicious URL. The K-Means clustering further supported these findings, showing that by using the top features, unsupervised training gets a decently high accuracy when detecting malicious URLs.

Interpretability vs. Performance: A central goal of our project was to prioritize interpretability and insight over mere predictive performance. The Logistic Regression model, despite being less accurate, was useful for explaining feature effects in simple terms (positive/negative influence). The XGBoost model then struck a balance by significantly boosting detection rates while still allowing us to extract feature importance and other detection patterns. We intentionally constrained XGBoost to the most important features for the demo, sacrificing a small amount of accuracy in exchange for transparency and speed. The results show that this top-10-features model still performs impressively (87% accuracy) while being interpretable. This outcome suggests that a thoughtfully pruned model can retain the "essence" of what a more complex model learns, which is encouraging for deployment in security tools where explainability is crucial for user trust.

Limitations: There are some limitations to acknowledge. First, our analysis was based on a specific curated dataset. Real attackers constantly evolve their tactics, and new types of malicious URLs might not be represented well. For example, some modern phishing URLs are very short and use innocent-looking domains to evade detection. Such cases would challenge our models since many of our top features flag long or complex URLs. Additionally, our feature set was largely lexical, being based on the URL string itself. We did not incorporate features from the content of the webpage pointed to by the URL like HTML, scripts, or network behavior. In practice, comprehensive malicious URL detection might combine URL analysis with content analysis for even greater accuracy. Another limitation is that our K-Means clustering, while insightful, used a simple Euclidean distance and treated features equally. Some features like entropy need different scaling or distance measures to cluster optimally. Despite these limitations, our approach demonstrated that just by analyzing the URL string, you can yield a powerful detection mechanism.

Discussion: There are a number of ways to expand on the work we have completed. One immediate improvement would be to explore additional features or transformations that could catch edge cases. For instance, features using n-gram representations of the URL for models to gain more context. Another extension would be to incorporate real-time updating: malicious URLs appear and disappear quickly (such as one-time-use phishing links). A live system could continually retrain and update the model with fresh data from users in order to stay current. A for the modeling side, while we focused on interpretability, one could experiment with deep learning models (such as LSTMs or Transformers that treat the URL as a sequence of characters). These models might detect complex patterns beyond our engineered features, potentially improving recall on malicious URLs. That being said, explaining their decisions would be harder. Ensemble approaches could also be tried, combining our interpretable XGBoost with other classifiers to see if we can capture the remaining difficult examples. Lastly, deploying the interactive tool as a browser extension or integrating it into email clients could be a real next step. We would need to ensure low latency and possibly add a feedback loop, allowing users to report misclassifications to improve the model.

Conclusion

This project demonstrated how a combination of machine learning techniques can be used to not only detect malicious URLs with high accuracy but also to explain the decision factors behind those detections. We found that a relatively small set of features related to URL length, complexity, and the presence of suspicious tokens carries most of the information needed to identify malicious links. Extracting these features allowed us to build an interactive demo that is both effective and user friendly. Our findings reinforce general website security intuitions with quantitative evidence, and contribute to the goal of turning those insights into a practical tool. As cyber threats continue to evolve, approaches that balance predictive power with user friendliness will be vital, enabling security practitioners and end users alike to trust and understand machine learning backed threat detectors. Our project is a step in that direction, showing the value of malicious URL classification and finding why a model flags a link as malicious. We believe this approach can enhance the effectiveness of cybersecurity defenses and help users stay safer online.

Machine Learning Approaches for Detecting Malicious URLs

Citations

[1]https://en.wikipedia.org/w/index.php?title=Phishing&oldid=1323182952
[2]https://www.kaggle.com/datasets/pilarpieiro/tabular-dataset-ready-for-malicious-url-detection

Link to Slides:

https://docs.google.com/presentation/d/1mmcrCmXjE2PBAXqXyqnaETGtRfQ2d6KWHIiGWta

STWs/edit?usp=sharing