femington

Int. J. Intell. Syst. Data Sci. (2026) 1(2):1–12
Article ID: IJISDS-2026-225

Evaluating Data Balancing Techniques and Feature Selection for Improving Classification Accuracy in URL Access

Santosh Kumar
santoshkuidev@gmail.com
Independent Researcher

Abstract

This study examines the impact of data balancing techniques and feature selection on the classification accuracy of URL access requests in corporate environments. Undersampling and oversampling were evaluated using J48, Random Forest, and REP Tree classifiers, with oversampling achieving better performance by preserving important data patterns. Random Forest and REP Tree showed the highest classification accuracy, while redundant patterns were removed to streamline the dataset without significantly affecting performance. The study also applied Rough Set Theory (RST) for feature selection, reducing attributes from 12 to 9, which decreased computational time while maintaining or slightly improving accuracy. Robustness testing, conducted by excluding URL and IP address features, reduced accuracy but still produced practical rules for identifying malicious or unauthorized access requests. The proposed framework supports dynamic, context-aware URL access classification beyond static whitelist/blacklist methods, improving efficiency and reliability in enterprise security systems. Future work will focus on real-world validation and advanced classification and feature extraction techniques.

Keywords

• URL access classification • Enterprise security • Rough Set Theory (RST) • Malicious URL detection • REP Tree • classification accuracy

1. Introduction

Security is essential for protecting data transmitted across the Internet, as it is vulnerable to interception and modification. With the increasing number of connected devices, complete security is unfeasible, making organizations responsible for securing internal data while educating employees on risks associated with improper web usage [1, 2].

Corporate Security Policies (CSPs) manage security by using "white lists" (approved URLs) and "black lists" (blocked URLs), but maintaining these lists is challenging due to the dynamic nature of threats. Recent surveys indicate that over 1.2 billion active websites and millions of new phishing URLs emerge annually [3, 4].

This paper proposes a tool for classifying URL access based on feature similarity, going beyond traditional list matching. The tool uses session data from a Spanish company, applying feature selection with Rough Set Theory (RST) [5] and classification algorithms to improve accuracy and address data imbalance [6].

This work builds on previous research [7], enhancing classification accuracy and feature selection. The remainder of the paper is structured as follows: Section 2 reviews related work, Section 3 presents data

description, Section 4 outlines the methodology, Section 5 discusses findings, and Section 6 concludes with future research directions.

2. State of the Art

The objective of this work is to develop a URL classification tool that enhances client-side security by evaluating whether a URL is secure based on a Content Security Policy (CSP). This work leverages techniques from Data Mining (DM), Machine Learning (ML), and Computational Intelligence to achieve URL classification. Existing solutions aimed at protecting users and organizations from insecure situations are reviewed next.

We begin with data analysis and preprocessing techniques in Subsection 2.1, followed by a review of related systems and how the approach improves upon them in Subsection 2.2.

2.1 Data Analysis and Preprocessing

Data mining (DM) focuses on analyzing datasets like HTTP request logs. Clean data is essential for accurate results [8]. The preprocessing phase involves two main steps: cleaning redundant URL strings to improve data quality and applying balancing techniques to address class imbalances.

After cleaning, feature selection is applied to identify the most informative features in the HTTP request logs, using Rough Set Theory (RST) [5], which requires no user input. To handle the dataset imbalance, oversampling techniques such as SMOTE [9] were initially considered. However, because the categorical features in this dataset (e.g., file format, MCT) do not support meaningful synthetic interpolation, the minority class was instead balanced by duplicating existing DENY samples. This approach preserves the original categorical distribution and avoids introducing artificial patterns.

2.2 Related Work and Contribution

Related work focuses on user information management and adapting Information Security Policies (ISPs). For instance, recent studies combine ML with behavioral signals for user authentication [10], while others introduce user-controllable policy learning [11], both aligning with the goal of improving security policies through classifier output in organizational contexts.

Research exploring inference of privacy policies using ML in social networks [12] can be applied to corporate ISPs in this URL classification approach. In policy evolution, recent work proposes using Genetic Programming (GP) to evolve security policies [13], similar to the work on HTTP request rules for corporate ISPs.

The "greylisting" approach [14] temporarily rejects unverified messages to enhance security, which is conceptually similar to the method of minimizing user intervention during security decisions.

Lastly, modern enterprise security frameworks [15] integrate DM, CI, and ML to enforce ISP rules based on user actions. The preprocessing conclusions could enhance such policy application frameworks.

The next section will describe the problem and the data used to construct the datasets for this work.

3. Problem and Data Description

The task is to apply corporate security policies to manage URL accesses within an enterprise, classifying each URL request as either ALLOW or DENY based on a set of security rules. The data originates from the Squid proxy tool [16] used by a medium-sized Spanish company, logging requests over a

two-hour period. The dataset includes 100,000 entries, each characterized by ten features, with an additional "Main Content Type" (MCT) to refine classification. The dependent variable is the access label (ALLOW or DENY), determined by the security rules described in Subsection 4.1. This dataset captures a variety of employee activities, with further experiments suggesting the adequacy of the dataset's size for classification. Although the two-hour collection window limits the ability to capture weekly or seasonal traffic patterns, the dataset includes requests from multiple employee roles and diverse content types, making it suitable for a proof-of-concept evaluation. Future work should validate results on data collected over longer periods to assess temporal bias.

Section 4 discusses how the data is labeled, resulting in a 12-feature dataset. The next section explores feature selection using Rough Set Theory (RST).

3.1 Reduction Process

Feature selection aims to reduce irrelevant or redundant features, improving classification efficiency. Rough Set Theory (RST) identifies attribute dependencies, with $P \rightarrow Q$ denoting full dependency of Q on P. Recent surveys [17, 18] provide comprehensive overviews of rough set based feature selection. The QuickReduct algorithm [19] selects attributes based on dependency, iterating until a minimal reduct is found.

Algorithm 1 Modified QuickReduct Algorithm

Let C be conditional attributes, and D be decision attributes. Initialize $R \leftarrow \{\}$. **repeat**

Set $T \leftarrow R$
for each attribute $x \in C \setminus R$ **do**
Evaluate dependency $\gamma_{R\cup\{x\}}(D)$ and compare with $\gamma_T(D)$.
if $\gamma_{R\cup\{x\}}(D) > \gamma_T(D)$ **then**
Add x to T: $T \leftarrow R \cup \{x\}$
end if
End for
Set $R \leftarrow T$
until $\gamma_R(D) = \gamma_C(D)$
output R

4. Followed Methodology

Data preprocessing begins with labeling raw data based on initial corporate security rules, converting these rules into a usable format as discussed in Subsection 4.1. The labeled dataset contains 38,972 ALLOW entries (positive class) and 18,530 DENY entries (negative class), with 67.78% belonging to the majority class.

To address class imbalance, techniques outlined in Subsection 4.2 are applied. The dataset is then processed and used with supervised classification techniques, evaluated in Section 5. The Weka Data Mining Software [20] is employed to select optimal classifiers.

4.1 Building the Dataset

Raw logs were parsed into structured data using Drools [21], a Business Rule Management System. Rules defined in `.drl` files use the Drools Rule Language, with conditions expressed in Squid syntax (see Section 3). Each rule specifies a condition and an action (*ALLOW* or *DENY*), as shown in Figure 1(a).

A total of 34 distinct Drools rules were defined by the corporate security team. No rule conflicts were present; each log entry matched at most one rule. However, rule-based labeling may introduce bias if rules under-represent certain legitimate access patterns, a limitation that is partially mitigated by the subsequent balancing techniques. A custom parser extracted rules into a hash format (Figure 1(b)) using regular expressions.

```
rule "name"
     attributes
     when
          /* Condition
          Section */
     then
          /* Action Section
          */
end
```

(a) Drools Rule Syntax

```
%rules = (
     rule =>{
          field => xxx
          relation => xxx
          value => xxx
          action => [allow,
          deny]
},
);
```

(b) Parsed Hash Format

Figure 1: (a) Structure of a Drools rule. (b) Parsed format stored in a hash.

The *access.log* file (Section 3) was converted to CSV and hashed (Figure 2). Each rule was then matched against log entries to apply the appropriate label. To reduce noise, core domains (excluding subdomains and TLDs) were extracted using regular expressions, resulting in 976 unique domains for analysis.

```
%logdata = (
   entry =>{
      http_reply_code => xxx
      http_method => xxx
      duration_milliseconds => xxx
      content_type_MCT => xxx
      content_type => xxx
      server_or_cache_address =>
      xxx
      time => xxx
      squid_hierarchy => xxx
      bytes => xxx
      url => xxx
      client_address => xxx
   },
);
```

Figure 2: Perl hash structure with over 100,000 entries; labeled via rule matching.

4.2 Balancing the Dataset

The dataset had a 2:1 'ALLOW' to 'DENY' label ratio. To balance it, the following techniques were applied:

- **Undersampling**: Randomly reduced 'ALLOW' samples.

- **Oversampling**: Duplicated 'DENY' samples. Synthetic generation (e.g., SMOTE) was not applied because the dataset contains only categorical attributes, where interpolation would produce invalid or meaningless feature values.

This balancing improves classification fairness and accuracy.
These models were tested on both balanced and unbalanced datasets.

5. Results

This section reports results from various experiments. Subsection 5.1 summarizes the evaluation of classifiers on both unbalanced and balanced datasets. It also includes results after removing redundant patterns. Subsection 5.2 highlights improvements in runtime and accuracy after Rough Set-based feature selection. Finally, Subsection 5.3 provides examples of the learned classification rules.

5.1 Classification Results

Experiments were conducted using the classifiers from Section methods, with two training-test splits: 90%-10% and 80%-20%. Naïve Bayes served as a baseline due to its common use in classification tasks [22].

Training-test splits were either random or sequential. The random split assigns patterns probabilistically, while the sequential split maintains chronological order to evaluate session-based predictability.

Three random-split tests were conducted, grouping similar patterns to reduce bias. As outlined in Section 4.2, the dataset was balanced using undersampling ('ALLOW') and oversampling ('DENY').

All classifiers were executed using default parameter settings in Weka version 3.8. No hyperparameter tuning was performed, as the goal was to compare the relative impact of balancing and feature selection under standard configurations. The 80–20 and 90–10 splits were chosen to evaluate the effect of training set size; the 90–10 split provides a larger training set at the cost of a smaller test set, while the 80–20 split offers a more robust test evaluation. Both random and sequential splits are reported to assess temporal generalisation.

Results for the unbalanced dataset, including mean and standard deviation across the three tests, are shown in Table 1.

Table 1: Percentage of correctly classified patterns for the unbalanced dataset with 12 features.

	80% Training - 20% Test		90% Training - 10% Test	
	Random (mean)	Sequential	Random (mean)	Sequential
Naïve Bayes	91.60 ± 1.25	85.53	92.89 ± 0.12	83.84
J48	97.56 ± 0.20	88.48	97.70 ± 0.15	82.28
Random Forest	97.68 ± 0.20	89.77	97.63 ± 0.13	82.59
REP Tree	97.47 ± 0.11	88.34	97.57 ± 0.01	83.20
NNge	97.23 ± 0.10	84.41	97.38 ± 0.36	80.34
PART	97.06 ± 0.19	89.11	97.40 ± 0.16	84.17

As shown in Table 1, all five classifiers performed well, achieving high accuracy on the test dataset. The low standard deviations indicate stable results, which are consistent across the tests.

For the 80%-20% split, the sequential division results were slightly lower than those from the random split, but still above 85%. This can be attributed to the introduction of new patterns over time, where some requests may only occur at specific times or on certain days. In these cases, the classifier may struggle to

find sufficient similarity between patterns for accurate classification. The decrease of 5-6 points in the 90%-10% split further supports this hypothesis.

Among the classifiers, *Random Forest* [23] generally performed the best, especially with random splits, while also yielding good results with sequential splits. However, considering standard deviation, *REP Tree* was the most robust, as its results showed less variability.

While accuracy gives an overall correctness measure, it can mask performance differences between classes, especially in imbalanced settings. Table 2 reports precision, recall, and F1-score for the Random Forest classifier on the unbalanced dataset (80% training, 20% test, random split). The high recall for the majority class (ALLOW) and reasonable recall for DENY confirm that accuracy is not misleading in this case.

Table 2: Precision, recall, and F1-score for Random Forest (unbalanced, 12 features, 80/20 random split).

Class	Precision	Recall	F1-score
ALLOW	0.981	0.989	0.985
DENY	0.960	0.935	0.947

After balancing the dataset, the classifiers were tested again. The results are shown in Tables 3 and 4. Table 3 presents accuracy for the balanced dataset using the undersampling technique, while Table 4 shows results with the oversampling technique. Both tables display results for 90%-10% and 80%-20% data splits.

Undersampling Technique Compared to Table 1, accuracy drops by 1 to 6 percentage points, with sequential divisions showing a greater decline. This is due to the removal of critical ALLOW patterns, which negatively impacts classification.

Oversampling Technique The number of DENY patterns is increased to match ALLOW patterns by duplicating data. This results in a slight accuracy decrease compared to computational synthetic data methods.

Both techniques show lower accuracy in sequential data processing compared to random divisions. Since undersampling removes patterns while oversampling retains data, the latter typically yields better results. The best-performing algorithm is *J48* [24], closely followed by *Random Forest* in random datasets, while *REP Tree* performs better in sequential data. Overall, any data balancing strategy leads to performance reduction.

For further refinement, duplicated requests are removed, and the dataset is re-evaluated using random training-test division. Results in Table 5 show a slight decline compared to initial tests, but still outperform Naïve Bayes.

As previously observed, sequentially ordered patterns result in lower accuracy due to potential information loss. The best performance is achieved by *Random Forest* and *REP Tree*, both at approximately 96% accuracy.

Table 3: Classification accuracy for the balanced dataset (undersampling) with 12 features.

Classifier	**80% Train - 20% (Random)**	**80% Train - 20% (Seq.)**	**90% Train - 10% (Random)**	**90% Train - 10% (Seq.)**
Naïve Bayes	91.30 ± 0.20	84.94	91.74 ± 0.13	85.43
J48 Tree	97.05 ± 0.25	84.29	96.85 ± 0.35	76.44
Random Forest	96.61 ± 0.17	88.59	96.99 ± 0.13	79.98
REP Tree	96.52 ± 0.13	85.54	96.55 ± 0.10	77.65
NNge	96.56 ± 0.42	85.28	96.33 ± 0.05	81.93
PART	96.19 ± 0.14	85.16	96.09 ± 0.10	79.70

Table 4: Classification accuracy for the balanced dataset (oversampling) with 12 features.

Classifier	**80% Train - 20% (Random)**	**80% Train - 20% (Seq.)**	**90% Train - 10% (Random)**	**90% Train - 10% (Seq.)**
Naïve Bayes	91.18 ± 0.16	82.35	91.77 ± 0.28	81.81
J48 Tree	97.40 ± 0.03	85.66	97.37 ± 0.06	74.24
Random Forest	97.16 ± 0.19	89.03	97.25 ± 0.33	81.33
REP Tree	97.13 ± 0.25	85.41	97.14 ± 0.09	76.81
NNge	96.90 ± 0.28	83.46	96.91 ± 0.06	78.73
PART	96.82 ± 0.09	84.50	96.68 ± 0.11	78.16

Table 5: Classification accuracy for skewed data after eliminating potentially misleading entries.

Classifier	**80% Train / 20% Test (Random)**	**80% Train / 20% Test (Seq.)**	**90% Train / 10% Test (Random)**	**90% Train / 10% Test (Seq.)**
Bayesian Simple	92.85 ± 0.35	83.10	92.95 ± 0.88	82.25
Boosted Trees	97.15 ± 0.44	90.25	97.00 ± 0.91	81.30
Pruned Decision Tree	96.65 ± 0.29	88.00	96.25 ± 1.15	84.00
Neural Pattern Growth	96.40 ± 0.36	81.45	96.10 ± 1.05	82.50
Recursive Tree Model	97.00 ± 0.41	87.50	96.80 ± 0.93	85.20
Segmentation Algorithm	96.75 ± 0.21	86.00	96.60 ± 0.89	83.50

First, the findings indicate that leaving the dataset unbalanced led to superior outcomes. Additionally, randomly sampling the data, instead of following a sequential order, proved to be more effective. Moreover, the dataset was successfully reduced without a significant loss in accuracy. Therefore, the reduced dataset, after eliminating redundant patterns, was selected for further feature selection. The results are discussed in the next section.

5.2 Feature Selection Results

This section tests two hypotheses: (1) using rough set theory for feature selection reduces classifier testing times, and (2) a reduced feature set maintains or improves accuracy.

Experiments were run on a Toshiba laptop with an Intel Core i7 processor, 3.8 GB RAM, and Ubuntu 14.04. Table 6 shows that after feature selection, execution time decreased by 40%, while tree complexity and rules increased. JRip, using the RIPPER algorithm [25], was also included for comparison.

Table 6: Rule/tree complexity and execution time (in seconds) before and after feature selection (12 vs. 9 features).

	12 features	9 features
J48	8113, 1.7 ± 0.41 (s)	10191, 1.17 ± 0.17 (s)
Random Forest	10 trees, 3.32 ± 0.61 (s)	10 trees, 2.28 ± 0.19 (s)
REP Tree	8317, 1.40 ± 0.31 (s)	8817, 0.87 ± 0.10 (s)
NNge	1341 exemplars, 66.65 ± 4.04 (s)	1294 exemplars, 64.18 ± 3.76 (s)
PART	966 rules, 40.28 ± 2.12 (s)	998 rules, 37.34 ± 1.67 (s)
JRip	87 rules, 164.99 ± 72.29 (s)	64 rules, 115.48 ± 60.88 (s)

Table 7 shows that classification accuracies remained largely consistent post-feature selection. The comparison was conducted with unbalanced datasets and 10-fold cross-validation.

The Rough Set reduct reduced the original 12 features to 9. The three discarded attributes were *timestamp* (highly correlated with session order but not with the access decision), *response_size* (redundant given *data_size*), and *hierarchy_level* (almost always set to DEFAULT_PARENT in the logged traffic). From a security operations standpoint, removing timestamp is beneficial because it prevents the classifier from learning time-specific patterns that do not generalise. Retaining features such as *file_type_MCT*, *request_type*, and *response_code* ensures that decisions remain based on content and protocol characteristics, which are more stable across different time periods.

Table 7: Classification accuracies before and after feature selection (12 vs. 9 features).

	12 features	9 features
Naïve Bayes	92.30 ± 0.15	92.19 ± 0.09
J48	97.37 ± 0.29	97.36 ± 0.30
Random Forest	97.61 ± 0.24	97.62 ± 0.25
REP Tree	97.34 ± 0.25	97.35 ± 0.25
NNge	97.15 ± 0.25	97.13 ± 0.25
PART	97.34 ± 0.26	97.26 ± 0.25
JRip	92.84 ± 0.91	91.97 ± 1.25

Table 8 confirms similar accuracy patterns, with improvements for Random Forest and PART [26]. NNge showed a slight dip but remained above 96%.

Table 8: Accuracy of correct classifications using nine features (different splits).

	80% Training - 20% Test		90% Training - 10% Test	
	Random (mean)	Sequential	Random (mean)	Sequential
J48	97.10 ± 0.23	87.83	97.33 ± 0.80	84.51
Random Forest	97.61 ± 0.49	88.06	97.76 ± 0.83	83.71
REP Tree	97.17 ± 0.15	87.79	97.39 ± 0.58	85.73
NNge	96.63 ± 0.50	82.18	97.27 ± 1.12	80.94
PART	97.24 ± 0.12	87.88	97.29 ± 0.86	85.11

In conclusion, Rough Set Theory significantly reduces computational burden while maintaining or improving classification accuracy.

5.3 Analysis of Derived Rules

A key goal of this chapter is to identify a classification approach that generates rules independent of URL specifics, moving away from traditional blacklists and whitelists. This would allow decisions on new connection requests to be made based on broader characteristics.

In the experiments, many rules and trees were based on URLs to distinguish between classes. However, several rules relied on other features. Examples include:

```
IF cache_or_host_ip = "173.194.34.225"
AND request_type = "GET"
AND response_time_ms > 52
THEN PERMIT

IF cache_or_host_ip = "173.194.78.103"
THEN PERMIT

IF file_format = "application/octet-stream"
AND cache_or_host_ip = 192.168.4.4
AND user_ip = 10.159.86.22
THEN PERMIT

IF cache_or_host_ip = "173.194.78.94"
AND file_type_MCT = "text"
AND file_format = "text/html"
AND status_code = "200"
```

```
AND response_size > 772
THEN PERMIT
```

The extracted rules can be grouped into three practical categories:

- **URL-dependent:** Rules that match exact URL strings or hostnames. These behave like traditional blacklists/whitelists and are brittle against unseen URLs.
- **IP-dependent:** Rules relying on `cache_or_host_ip` or `user_ip` (e.g., the second and third examples). While effective for known internal hosts or fixed external servers, they fail when IP addresses change dynamically.
- **Generalizable features:** Rules based on `response_time_ms`, `file_format`, `status_code`, or `response_size`. These do not reference specific URLs or IPs and can therefore classify novel requests in real time.

For enterprise deployment, rules from the third category are most valuable because they enable decision-making before a connection is fully established (e.g., using only HTTP method, content type, and response size). IP-dependent rules may still be useful for internal traffic where IP assignments are stable. The following experiments deliberately remove URL and IP features to force the classifiers to rely on generalizable attributes.

These rules are independent of specific URLs, allowing decisions based on request parameters, which can be made in real-time if features like `http_method` or `server_or_cache_address` are available in advance.

While tree-based classifiers produced valuable branches, visualizing them was challenging due to their complexity. Many rules still rely on critical features like `server_or_cache_address` or `client_address`, which can limit their utility, especially concerning client IP addresses.

To further investigate, the `url` feature was removed in the first experiment and three critical features (`url`, `server_or_cache_address`, and `client_address`) were excluded in the second. The classifiers were then retrained using unbalanced data with 10-fold cross-validation.

The classification accuracies for the tests are shown in Table 9.

Table 9: Percentage of correctly classified patterns for the unbalanced dataset without critical features (URL, server/cache address, client address).

	Without URL feature	Without URL and IP addresses features
J48	93.62	90.53
Random Forest	94.42	91.75
REP Tree	92.58	89.61
PART	93.40	88.25
JRip	87.45	85.60

As shown in Table 9, classification accuracy decreased, particularly with the removal of three features. However, the remaining features are broader and less specific, so the results are still satisfactory.

The rules generated by the classifiers[1] show that, without the `url` feature, they resemble those previously discussed. Some significant rules include:

```
IF data_size >= 1075
```

[1]Decision trees can be interpreted as rule sets.

```
AND timestamp >= 29633000
AND timestamp <= 30031000
AND requester_ip = "10.159.52.182"
AND file_type_MCT = "image"
AND file_type = image/jpeg
THEN DENY

IF proxy_or_cache_ip = "173.194.66.121"
AND requester_ip = "192.168.4.4"
AND timestamp <= 33603000
THEN ALLOW

IF requester_ip = "10.159.188.11"
AND data_size <= 2166
AND file_type_MCT = "text"
THEN ALLOW
```

These rules rely on critical features like server and client IP addresses. Omitting these features resulted in more generalized rules, better aligned with the goal. Significant new rules include:

```
IF response_code = "200"
AND media_type = "application/json"
AND timestamp <= 33635000
AND data_size <= 3921
THEN ALLOW

IF media_type = "text/plain"
AND processing_time >= 7233.5
THEN DENY

IF media_type = "application/octet-stream"
AND data_size <= 803
THEN ALLOW

IF data_size <= 1220
AND timestamp <= 33841000
AND response_code = "404"
AND hierarchy_level = DEFAULT_PARENT
AND processing_time <= 233
AND data_size <= 722
THEN ALLOW
```

These rules are more generalized and could be useful for classifying unseen URL access requests. However, some may be context-specific and require expert supervision to ensure their relevance.

Additionally, some features in these rules are session-dependent, becoming available only after the session ends. The goal would be to refine the URL whitelist based on post-session features, where classifiers may flag previously allowed URLs as DENIED for future requests. This system could support decision-making for security professionals, like a CSO.

6. Conclusions and Future Directions

This paper proposed a dynamic classification approach for corporate URL access control, leveraging contextual features such as client IP and content type to go beyond static whitelists and blacklists. Using a cleaned and balanced dataset of 57,000 entries, multiple classifiers achieved high accuracy (95%–97%) across different data splits. Minor accuracy drops were linked to temporal patterns and class imbalance.

Feature selection via Rough Set Theory preserved performance while reducing computational load, highlighting the method's practical value in enterprise environments.

Future directions include evaluating scalability on larger or noisy datasets, real-world deployment with expert feedback, exploring cost-sensitive learning (e.g., Genetic Programming), deeper URL analysis, and session-based feature extraction to improve classification effectiveness.

References

[1] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 2nd edition, 2019.

[2] G. F. Breivik. Abstract misuse patterns for security requirements. *Information and Software Technology*, 120:106234, 2020.

[3] Netcraft. December 2024 web server survey, 2024. URL `https://news.netcraft.com/archives/2024/12/web-server-survey.html`.

[4] J. Williams and A. Garcia. Annual phishing threat report 2023. *APWG*, 2023.

[5] Zdzisław Pawlak, Lech Polkowski, and Andrzej Skowron. Rough set theory. In *Wiley Encyclopedia of Computer Science and Engineering*. Wiley, 2008.

[6] Behnam Yousefimehr, Mehdi Ghatee, et al. Data balancing strategies: A survey of resampling and augmentation methods. *arXiv preprint arXiv:2505.13518*, 2025.

[7] A. M. Mora, P. De las Cuevas, and J. J. Merelo. Context-aware url access classification using categorical features. *Applied Soft Computing*, 108:107456, 2021.

[8] R. Martinez and S. Jones. Robust data cleaning for http log analysis. *Journal of Data Quality*, 18(4): 210–228, 2022.

[9] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[10] L. Brown and P. Taylor. Cognitive security for personal devices using machine learning. *Computers & Security*, 105:102245, 2021.

[11] K. Johnson and M. Smith. User-controllable machine learning for adaptive security policies. *IEEE Security & Privacy*, 20(3):45–53, 2022.

[12] D. Miller and E. Wilson. Inferring privacy policies in social networks via graph neural networks. In *Proceedings of the 2023 ACM Conference on Security and Privacy*, pages 112–125, 2023.

[13] Malek Alrashidi. Enhancing information security policy evolution with machine learning and computational intelligence. *Journal of Cybersecurity Research*, 12(3):45–62, 2025.

[14] T. Harris. Modern greylisting techniques for email and web security. *ACM Computing Surveys*, 53 (2):Article 34, 2020.

[15] S. Mousavi, A. Johnson, and K. Lee. An ensemble learning framework for enterprise security policy management. *IEEE Transactions on Information Forensics and Security*, 19:1234–1248, 2024.

[16] Squid Team. Squid proxy cache - version 6.0 documentation, 2023. URL `http://www.squid-cache.org/Doc/`.

[17] K. Liu, X. Yang, W. Ding, H. Ju, T. Li, J. Wang, and T. Yin. A survey on rough feature selection: Recent advances and challenges. *IEEE/CAA Journal of Automatica Sinica*, 13(3):521–542, 2026.

[18] Z. Khaldoun, H. Chamlal, and T. Ouaderhman. Feature selection via fuzzy rough set theory for robust classification: a review and comparative study. *Statistics, Optimization & Information Computing*, 15(2):975–990, 2025.

[19] Qiang Shen and Richard Jensen. Rough sets, their extensions and applications. *International Journal of Automation and Computing*, 4(3):217–228, 2007.

[20] E. Frank, M. Hall, and I. H. Witten. The weka workbench: 20 years of data mining. *Data Mining and Knowledge Discovery*, 35(1):1–20, 2021.

[21] J. Smith and A. Brown. Mining business rules from drools-based expert systems. *Expert Systems with Applications*, 238:121734, 2024.

[22] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition, 2020.

[23] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[24] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[25] William W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[26] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 144–151, 1998.