BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

OPTIMIZATION OF CLASSIFICATION MODELS FOR MALICIOUS DOMAIN DETECTION

OPTIMALIZACE KLASIFIKAČNÍCH MODELŮ PRO DETEKCI MALIGNÍCH DOMÉN

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. PETR POUČ**
AUTOR PRÁCE

SUPERVISOR **Ing. RADEK HRANICKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

154617

Institut:	Department of Information Systems (DIFS)
Student:	**Pouč Petr, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Cybersecurity
Title:	**Optimization of Classification Models for Malicious Domain Detection**
Category:	Security
Academic year:	2023/24

Assignment:

1. Study machine learning methods. Focus on classification and optimization of existing models.
2. Learn about current research in malicious domain detection under the FETA project.
3. Analyze the success of current classification models and identify their weaknesses, e.g., related to the generation of false positives.
4. In consultation with the supervisor, propose optimizations to selected classification models to eliminate the identified problems.
5. Implement the proposed optimizations.
6. Experimentally verify the benefits of your optimizations on a suitably selected dataset and discuss the results.

Literature:

- S. R. Dubey, S. Chakraborty, S. K. Roy, S. Mukherjee, S. K. Singh and B. B. Chaudhuri, "diffGrad: An Optimization Method for Convolutional Neural Networks," in IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 11, pp. 4500-4511, Nov. 2020, doi: 10.1109/TNNLS.2019.2955777.
- Z. Jin, W. Chaorong, H. Chengguang and W. Feng, "Parameter optimization algorithm of SVM for fault classification in traction converter," The 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, 2014, pp. 3786-3791, doi: 10.1109/CCDC.2014.6852839.
- L. Sun, "Application and Improvement of Xgboost Algorithm Based on Multiple Parameter Optimization Strategy," 2020 5th International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Harbin, China, 2020, pp. 1822-1825, doi: 10.1109/ICMCCE51767.2020.004.

Requirements for the semestral defence:
Points 1 to 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Hranický Radek, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	30.10.2023

Abstract

This thesis focuses on the development of advanced methods for malicious domain name detection using optimization techniques in machine learning. The thesis investigates and evaluates the effectiveness of different optimization strategies for classification. As evaluation tools, I selected classification algorithms that differ in their approach, including deep learning, decision tree techniques, or hyperplane search. These methods are investigated in terms of their ability to effectively classify domain names depending on the implemented optimization techniques. Optimization strategies include the creation of ground-truth datasets, application of data processing methods, advanced feature selection, solving the class imbalance problem, and hyperparameter tuning. The final part of the paper presents a detailed analysis of the benefits of each optimization approach. The experimental part of the study demonstrates exceptional results by combining several methodologies. The top CNN models obtained up to 0.9926 F1 while lowering FPR to 0.3%. The contribution of this study is to provide specific methodologies and tactics for the successful identification of malicious domain names in the cybersecurity area.

Abstrakt

Tato diplomová práce se zaměřuje na rozvoj pokročilých metod pro detekci škodlivých doménových jmen s využitím optimalizačních technik v oblasti strojového učení. Práce zkoumá a hodnotí účinnost různých optimalizačních strategií pro klasifikaci. Jako nástroje pro hodnocení jsem vybral klasifikační algoritmy, které se liší v jejich přístupu, včetně hlubokého učení, techniky rozhodovacích stromů, nebo hledání hyperrovin. Tyto metody byly posouzeny na základě schopnosti efektivně klasifikovat doménová jména v závislosti na použitých optimalizačních technikách. Optimalizace zahrnovala vytvoření přesně označených datových sad, aplikaci technik zpracování dat, pokročilou selekci atributů, řešení nerovnováhy tříd a ladění hyperparametrů. Experimentální část práce prokazuje vynikající úspěšnost kombinováním jednotlivých metod. Přičemž nejlepší modely CNN dosahovaly až 0.9926 F1 při současném snížení FPR na hodnotu 0.300%. Přínos práce spočívá v poskytnutí konkrétních metod a strategií pro efektivní detekci škodlivých doménových jmen v oblasti kybernetické bezpečnosti.

Keywords

Optimization, Classification, Feature Engineering, Machine Learning, FETA Project, Hyperparameter Tuning, Malware, Phishing, Cybersecurity, Ground-truth, Imbalance handling, Data processing, Virus Total, Domain verification

Klíčová slova

Optimalizace, Klasifikace, Extrakce příznaků, Strojové učení, Projekt FETA, Ladění hyperparametrů, Malware, Phishing, Kybernetická bezpečnost, Ground-truth, Nerovnováha tříd, Zpracování dat, Virus Total, Verifikace domén

Reference

POUČ, Petr. *Optimization of Classification Models for Malicious Domain Detection.* Brno, 2024. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Hranický, Ph.D.

Rozšířený abstrakt

Ve své diplomové práci jsem se věnoval rozvoji a optimalizaci klasifikačních modelů pro detekci škodlivých doménových jmen, což představuje klíčovou oblast v rámci zajištění kybernetické bezpečnosti. Tento výzkum neprobíhal izolovaně, ale byl integrován do širších aktivit projektu FETA, který je součástí Fakulty informačních technologií. Projekt FETA se dlouhodobě zaměřuje na vývoj metod pro identifikaci a neutralizaci kybernetických hrozeb, což poskytlo pevný základ pro můj výzkum. Má práce tak stavěla na předchozích úspěších naší skupiny, což mi umožnilo přistupovat k problému s předem ověřenými metodami a technikami. V práci tedy využívám osvědčené metody a rozšiřuji je o nové poznatky a techniky. Význam tohoto výzkumu je umocněn narůstajícím počtem kybernetických útoků zaměřených na zneužití doménových jmen pro škodlivé činnosti.

Práce je strukturovaná do několika klíčových kapitol, z nichž každá se věnuje specifickému aspektu detekce škodlivých domén. Po úvodní kapitole, která nastiňuje cíle a motivaci práce, následuje teoretický přehled současných metod a technologií používaných v kybernetické bezpečnosti. Dále je kladen důraz také na objasnění metodologií, které jsou klíčové pro vyhodnocování efektivity vybraných klasifikačních modelů. Pozornost věnuji zejména metrikám skóre F1 a množství falešně pozitivních výsledků, které jsou klíčové pro praktické nasazení v reálných systémech kybernetické bezpečnosti. Převážnou část teoretického začátku se věnuji podrobnému zkoumání klasifikačních algoritmů, jako jsou Support Vector Machines (SVM), XGBoost a konvoluční neuronové sítě (CNN), Rozhodovací stromy, Lineární regrese, či Bayesovy klasifikátory. V dalších částech práce se budu omezovat více na první tři zmíněné, které byly zvoleny jako základní klasifikačních algoritmy pro naše experimenty. Tyto metody byly vybrány kvůli jejich prokázané efektivitě v předchozích studiích a potenciálu pro adaptaci na specifické požadavky v oblasti detekce škodlivých domén.

V části návrh se věnuji optimalizačním strategiím, které byly vyvinuty na základě identifikovaných nedostatků existujících řešení. Tyto strategie zahrnují pokročilé metody pro předzpracování dat, jako jsou techniky normalizace a odstranění anomálií, a techniky pro řešení nerovnováhy datových sad, včetně použití techniky SMOTE. Důraz byl také kladen na sofistikovaný výběr příznaků, který umožňuje efektivnější klasifikaci a identifikaci škodlivých domén. V této části se dále věnuji návrhu vlastní verifikační pipeline pro tvorbu groundtruth datasetů. Tento proces byl nezbytný pro zajištění, že datové sady používané pro trénink a testování modelů jsou vysoce kvalitní a spolehlivě reprezentují skutečné distribuce škodlivých a benigních domén. Závěrečná část práce se věnuje samotné implementaci a testování navržených optimalizací na vybraných klasifikačních modelech. Experimentální ověření těchto modelů probíhalo na pečlivě připravených datových sadách, což umožnilo praktické ověření efektivity jednotlivých optimalizačních přístupů.

Práce je zakončena interpretací výsledků a diskusi o významu zjištěných poznatků pro praktické aplikace v oblasti kybernetické bezpečnosti. Tato část zdůrazňuje přínosy a dopady optimalizovaných metod, navrhuje směry pro další výzkum a reflektuje možnosti implementace v praxi. Podařilo se mi nejen zdokonalit existující metody detekce, ale také poskytnout cenný základ pro budoucí inovace v této oblasti. Celkově přináší moje práce nový pohled na problém detekce škodlivých domén a otevírá cestu k dalším pokrokům v boji proti kybernetickým hrozbám.

Optimization of Classification Models for Malicious Domain Detection

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Radek Hranický, Ph.D. The supplementary information was provided by my friends and coworkers, Bc. Jan Polišenský and Ing. Adam Horák, whose insights and assistance enriched the content of this work.

I have listed all the literary sources, publications, and other sources, which were used during the preparation of this thesis.

.....
Petr Pouč
May 11, 2024

Acknowledgements

I am deeply grateful for the guidance and expertise provided by Dr. Radek Hranický, my Master's thesis advisor, whose professional support, insightful consultations, and valuable advice significantly enhanced the quality of this thesis. My gratitude goes to Jan Polišenský and Adam Horák, on whose work my thesis is based, they provided me with a lot of useful advice during its preparation. Additionally, I extend my thanks to Virus Total for granting access to the academic version of their API. I also want to acknowledge my girlfriend and family for their unconditional patience, persistent encouragement, and continuous assistance during this academic adventure. I'd like to express my thanks to my friends Honza, Honza, Honza, Michal, and Michal, affectionately known as Honza^3 and Michal^2. Through several meetings with them over beers and coffee, I was able to assess the level of my dumbness, which was often the driving force that compelled me to continue working on this thesis.

Contents

List of Figures

Chapter 1

Introduction

With the digital environment continually expanding, the importance of protecting internet domain names from cyber attacks has never been greater. In a world where digital presence is essential to both personal and corporate operations, cybersecurity relies on the capacity to discern between benign and malicious domains. This thesis targets this important need by attempting to improve the approaches employed in domain categorization, greatly contributing to the larger efforts to provide a safer digital environment for everybody. This project aims to explore optimizing classifiers leveraging a combination of various machine learning methods.

The research focuses on determining the effectiveness of classification approaches in detecting harmful domains, with an emphasis on establishing what types of domains these methods work best and most correctly for. A comprehensive approach utilizing multiple methods facilitates the creation of strong classifiers capable of effectively identifying potentially dangerous sites on the internet. The study critically examines the efficiency of current classification models in identifying dangerous internet domain names, building on the foundation of current research under the Flow-based Encrypted Traffic Analysis (FETA), VJ02010024 for the Ministry of the Interior of the Czech Republic[1]. The FETA project is built on a domain-centric strategy, because it remains successful even with encrypted communication, where domain names are exposed but other details remain hidden, enabling for threat monitoring and identification in such situations.

The thesis aims to evaluate the classifier's performance in real-world situations and to provide a comprehensive understanding of their capabilities. The primary focus of this thesis is on optimizing strategies and creating enhanced classifiers, in addition to the existing ones. By identifying the approaches that yield the most efficient improvements in performance, this thesis aims to provide valuable insights into the optimization of classifiers.

The thesis commences by introducing current research in the field of Internet domain security, then delving into theoretical approaches to detecting malicious websites. Based on the information gathered, optimization algorithms are implemented rather than a set of classifiers, which are then experimentally verified. The findings of these experiments are used to assess the efficacy of the implemented tactics and indicate areas for potential improvement. The results of these experiments serve to evaluate the effectiveness of the implemented methods and identify areas for possible expansion and refinement of the proposed classifiers. Specifically, the research concentrates on reducing false positives, re-

[1] https://www.fit.vut.cz/research/project/1530/.en

vitalizing domain collections, and conducting a comparative analysis of the most common classification models to enhance the state-of-the-art in domain classification.

The combination of optimization methodologies yielded exceptional results. Using ground truth datasets in conjunction with appropriate data preprocessing, managing class imbalance, and enhanced hyperparameter selection, has demonstrated the potential for significant improvements in existing classifiers. The results have shown improved F1 scores of up to 0.9926 (CNN) with reduced false positive rate of 0.00062 (XGBoost). These experiments have practical implications for future work on this project, particularly in the area of real-time detection of malicious domain names.

The thesis has been methodically structured into a clear and all-encompassing framework. It comprises five interrelated sections, each adding a distinctive viewpoint and depth to the research. The thesis begins with Chapter 2, which is a comprehensive assessment of the domain security landscape, establishing a structure for the ensuing investigation. This is followed by Chapter 3, a detailed comparison of existing categorization models, which provides insights into their strengths and limits. The subject matter then changes in Chapter 4 to the creative construction and preliminary design of optimization strategies for these models, demonstrating the integration of several machine learning technologies. The specific implementation and practical use of these tactics is described in depth in the following Chapter 5, where theory is transformed into action. The last Chapter 6 focuses on experimental analysis and presenting the obtained results, highlighting achieved progresses.

The insights gathered from this thesis can help to design more effective malign domain classifiers, which might be essential in today's ever-changing digital environment.

Chapter 2

An Overview of Related Machine-learning Principles

This chapter provides an in-depth exploration of a few basic topics essential to comprehending the internet domain name (later referred to as "domains") classification. Its goal is to provide the reader with a complete knowledge of the theoretical and practical components of machine learning and domain classification and establish the groundwork for the rest of the thesis's exhaustive investigation.

2.1 Core Principles of Machine Learning for Predictive Data Analysis

`Machine Learning (ML)` is an automated technique that extracts patterns from data. ML algorithms employ statistical approaches to allow computers to improve over time. The main part of ML is creating models, that examine and understand data patterns, allowing for the prediction and categorization of future data. Models are capable of making decisions, creating analysis, or data interpretation without the need for explicit programming.

Predictive data analysis is the process of creating and using a machine learning model that makes predictions based on patterns extracted from data. Predictive analysis is used to classify data automatically. [42]

2.1.1 Importance of Malicious Domain Detection and Current Research Landscape

Malicious domains present significant cybersecurity dangers, spanning a wide variety of illicit operations such as malware distribution, phishing assaults, and botnet recruiting. These domains are intended to deceive users by pretending to be legitimate web pages or services, making them difficult to discover. Data breaches, ransomware attacks, and malware infections are all possible threats linked with malicious domains. New versions of malware are developed every day, expanding the threat landscape and making cyber attacks an inevitable concern for both organizations and individuals.

Machine learning plays an important role in recognizing and categorizing dangerous URLs due to how it allows computers to evaluate by themselves data and make predictions. Classification, a subset of supervised machine learning, has been extensively utilized for

recognizing known and unknown harmful URLs, helping to build efficient techniques and strategies for malicious URL detection [5]. There are several advantages to utilizing machine learning for malware detection, including:

- **Dynamic threat detection** - The adaptable nature of machine learning has made it a valuable tool in identifying unknown malware threats. Its ability to evolve and learn from new data continuously has made it a powerful solution for detecting and preventing emerging malware threats. Its effectiveness in dealing with previously unknown malware makes it a preferred choice in the field of cybersecurity.
- **Scalability** - With the increase in the amount of data and complexity of malware threats, it has become crucial to develop machine learning (ML) algorithms to handle these challenges effectively. These algorithms can process larger datasets and detect more sophisticated attacks, providing solutions to the evolving needs of cybersecurity over time.
- **Real-time detection** - Real-time classification of malware and benign files can be efficiently achieved by leveraging machine learning models. This approach results in quicker response times and minimizes the damage caused by malware-related attacks. Furthermore, it not only improves the accuracy of malware detection but also has the potential to automate decision-making and enable proactive defense mechanisms against cyber threats.

Recent research has focused on developing advanced machine-learning techniques for identifying and mitigating malicious domains. For instance, DeepDom, an intelligent malicious domain detection system, utilizes scalable and heterogeneous graph convolutional networks to analyze the DNS scene and enhance the identification of malicious domains. [39].

BERT-based approaches for identifying malicious URLs are also frequently used. This approach works with deep-learning models and bidirectional encoder representation from transformers to tokenize URL strings and categorize them as malicious or benign [72].

2.2 Classification Pipeline

In the context of machine learning, data that is utilized to train the model is stored in collections, commonly referred to as `training datasets`. Each individual data point within the dataset is known as a `training instance` and encompasses a set of `features`. These features correspond to specific measurable attributes or characteristics of the phenomena being observed.

2.2.1 Data Collection and Preparation

The process of data collection and preparation typically comprises three main stages. These stages involve a systematic approach to gathering and organizing data for subsequent analysis. These stages are:

- gathering the relevant data that will be utilized in the process of training and testing the model,
- conducting the exploratory data analysis,

- preparation of data for analysis, cleaning and preprocessing the data, handling any missing values, outliers, or anomalies.

This phase in the classification process is mandatory and particularly important as it significantly impacts the performance of the classification algorithms.

The goal of this phase is to collect appropriate data for our classification problem. Our data should ideally be credible, accurate, consistent, accessible, complete, reliable, interpretable, and potentially useful. This is often not the case.

The data can be collected from multiple sources, with many format inconsistencies.

This phase aims to clean the data and refine it to the most suitable form of usage by learning algorithms. There are plenty of data preprocessing steps like handling missing values, data reduction, data normalization, data transformation, anomaly detection, etc.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an important step in the data analysis process that entails using numerous methodologies to better comprehend the information of the dataset being used. EDA assists in the identification of relevant variables, the detection of outliers, the comprehension of variable correlations, and, ultimately, the maximization of insights from the dataset while preventing any mistakes that may emerge later in the process [50]. All the following parts, such as handling missing values, identifying outliers, noise reduction, etc. can be considered as a part of EDA.

Handling Missing Values

In this section, I will cover a fundamental component of data preparation in machine learning that involves missing data handling. Missing values in datasets are prevalent and provide substantial difficulties in predictive modeling. Incomplete data may bias findings and impair the efficacy of machine learning systems. As a result, the treatment of these missing values is critical to the integrity and correctness of any analysis.

In the following Figure 2.1 we can obtain all the possible approaches used for handling the missing values in the datasets. This section highlights only several strategies used in the field of machine learning to address this problem. While the approaches and complexity of these solutions differ, each provides a distinct strategy for coping with incomplete datasets. We will go deeper into the most often used ways for dealing with missing data in the next subsections.

The easiest approach for handling missing values is their deletion. Three main strategies using this approach are:

1. **Listwise deletion (complete-case analysis)** - entails eliminating whole rows of data that have missing values. While this strategy can reduce the dataset and keep comprehensive information, it may result in severe data loss.

2. **Pairwise deletion** - in contrast to listwise deletion, it keeps instances with missing data for certain analyses while discarding them for others. This strategy makes the best use of available data, but it could end up resulting in varied sample sizes for various studies, which might bring bias to the results.

3. **Column-wise deletion (variable-wise deletion)** - this method includes eliminating entire columns or features with a large proportion of missing data. This strategy

may be appropriate when some qualities have a significant quantity of missing data and are not regarded as critical for the classification task.

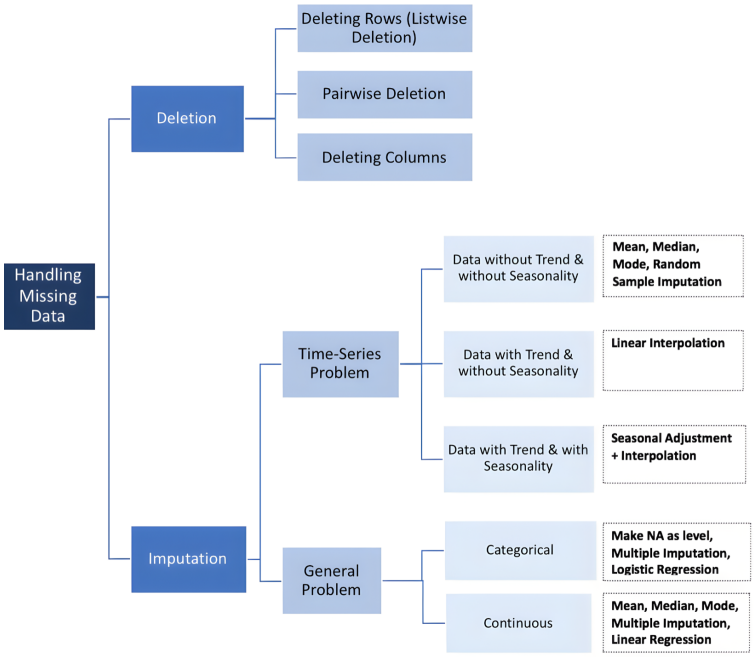


Figure 2.1: Methods of handling missing values [56]

The most common method to eliminate missing values is `missing values replacement`. This approach is about filling the empty values by calculating new values. It can be done using [78]:

1. **Imputation mean** - The missing value is replaced with the mean value of the entire feature column. This approach is used when missingness is random and data is normally distributed.

2. **Imputation median** - The missing value is replaced with the median value of the entire feature column. This method is more resistant to outliers and data distortion than mean imputation since the median is not as impacted by extreme values.

3. **Imputation most frequent (Imputation mode)** - This approach is particularly useful for categorical or numerical data, as it assumes that the most frequent value is the most likely to occur.

   More in-depth replacing missing values breakdown is described in Figure 2.2.

One of these methods is chosen based on the nature of the data, the structure of missingness, and the missing rate. The format in which the data is missing varies from each data source and type. Basic categories for missingness are: 1) `MCAR` (missing completely at random), where each variable and observation possesses the same chance of being absent, 2) `MAR` (missing at random), where the absence of a value corresponds to any additional variables values in the dataset and 3) `MNAR` (missing not at random). [78, 20]

In the extreme case of the absence of a large number of values, it is common to delete the entire column of data.

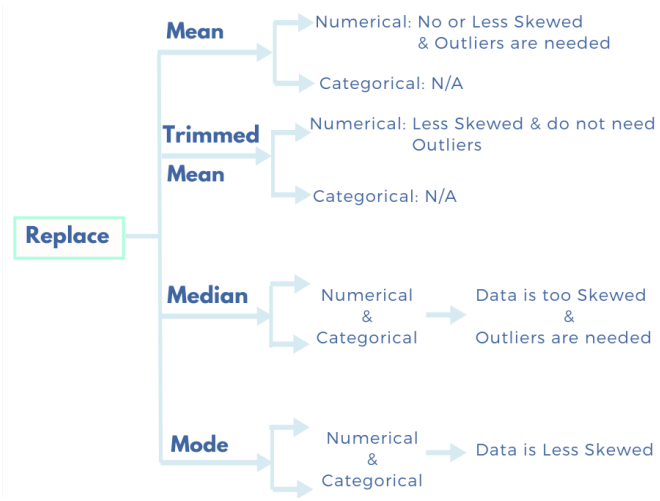


Figure 2.2: Methods of handling missing values [62]

There are also various techniques for automated data preprocessing, including automated tools, such as `Auto-Sklearn`[1] or `Learn2Clean`[2], for this specific purpose [78].

Identifying Outliers and Anomalies Detection

Outliers are data points that exist at the outermost ranges of a data set, significantly deviating from the majority of observations. [10]. Anomalies are observations that deviate from the common data format or behavior. Anomalies and outliers can have a negative impact on the model's performance, so it is important to detect and handle them properly [15].

Correctly identifying outliers is an important step in data analysis given that outliers can have a major influence on statistical analyses and distort the results of hypothesis testing. If they are not discovered and dealt with properly, they can lead to inaccurate conclusions.

An essential step in this part of the classification pipeline is to determine which are `true outliers` and which are the other ones. True outliers are important for our classification, as they represent natural data variability in our sample. Outliers that are not labeled as true ones are often measurement errors, data processing errors, wrong data transformations, and others.

Commonly used techniques for identifying outliers are:

- Visualization Methods

  Outliers are often identified by miscellaneous visualization methods. Well-known examples are `box plots, histograms, and scatterplots`.

- Statistical Outlier Detection

  This method involves using various statistical tests. Common statistical methods are `z-score` and some of its modifications, `Interquartile Range Method (IRQ)` and `Mahalanobis Distance`. [12]

[1] https://github.com/automl/auto-sklearn
[2] https://github.com/LaureBerti/Learn2Clean

In the following Figure 2.3, we can see the difference between multi-class classifiers and one-class classifiers outlier detection.

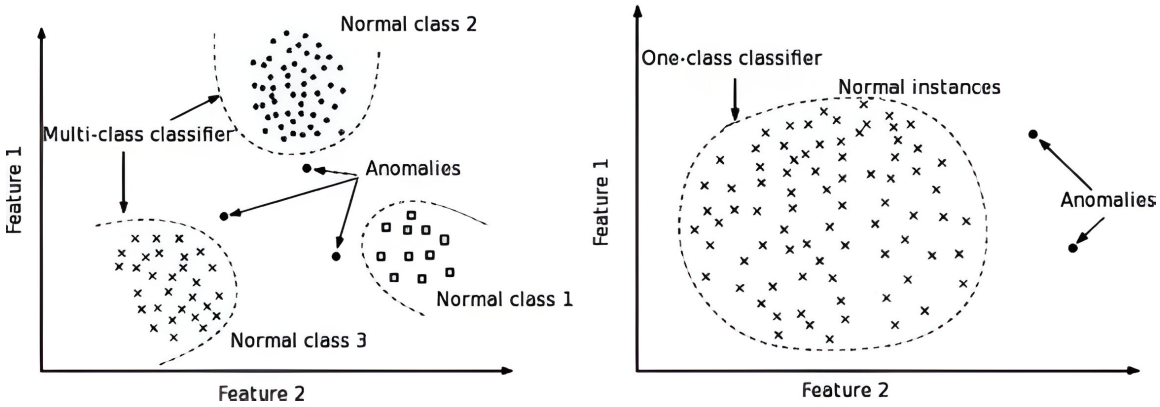


Figure 2.3: Visualization methods to determine outliers [18]

Statistical detection of outliers is the process by which to identify extreme values in a dataset by applying statistical tests or techniques. The use of z-scores is a frequently occurring statistical treatment for detecting outliers, particularly in situations where the data or individual variables in the dataset follow a normal distribution. By computing the z-score for each data point, it is feasible to discover values that deviate from the mean. For instance, setting the lower and upper limits to three standard deviations below and above the mean, respectively, can help identify outliers. Data points that fall outside this range substantially differ from the majority of observations and are qualified as outliers. [12, 69, 16]

The usage of z-score is visualized in the Figure 2.4 below.

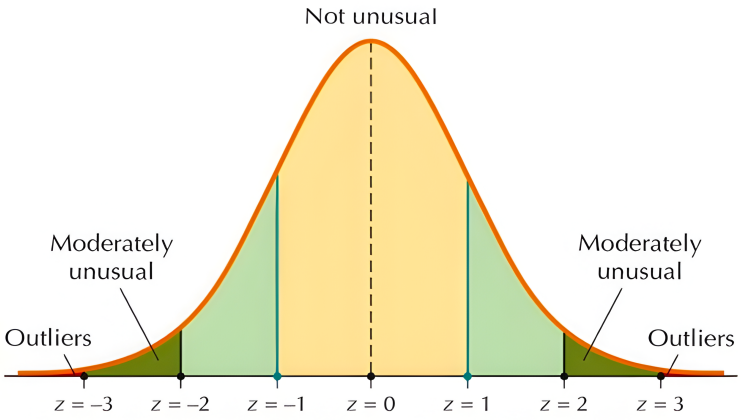


Figure 2.4: Usage of z-score method to identify outliers in dataset [12]

Another method for outlier detection is the Interquartile Range (IQR) method, which involves calculating the range between the first and third quartiles of the data. Data points that fall below the lower bound (Q1 - 1.5 * IQR) or above the upper bound (Q3 + 1.5 * IQR) are identified as outliers [69].

The next mentioned method is Mahalanobis distance, which is a multivariate statistical technique. As depicted in Figure 2.5, it calculates the distance between a point and

a data distribution while taking into consideration variable correlation and assuming an anisotropic Gaussian distribution [31]. Because this technique takes into account the correlation between features, it is more beneficial for datasets with correlated variables [36].

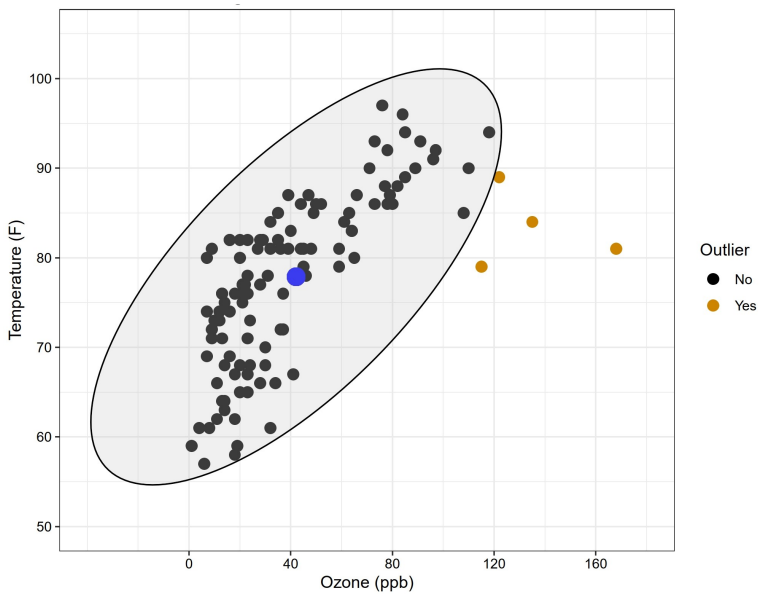


Figure 2.5: Mahalanobis distance for New York's air quality [31]

It is necessary to keep in mind that the underlying distribution of the data affects the ability to identify outliers. As a result, it is strongly recommended to evaluate the data's assumption of normality prior to doing outlier testing. Histograms, box plots, and normal probability plots are a few examples of graphic methods that can help verify the presumed normality. [12, 69, 16]

To identify anomalies we can choose different approaches using unsupervised learning algorithms, such as Isolation Forest. This method detects anomalies by isolating them in the data. Isolation Forest randomly selects a feature and then randomly selects a split value between the minimum and maximum values of the selected feature. The process is repeated recursively, leading to the isolation of anomalies. Isolation Forest is particularly useful for high-dimensional data and can handle both numerical and categorical features. [13, 40]

Noise Reduction

Noise reduction involves removing irrelevant or redundant data from the dataset. Noise can be introduced into the dataset due to various reasons such as measurement errors, data processing errors, and data transformations.

Feature selection is a prominent approach for noise reduction that includes picking a subset of important features from a dataset and rejecting the rest. Feature extraction is a different strategy for noise reduction that includes changing the original features into a new collection of features that capture the most essential information in the dataset.

2.2.2 Data Splitting

Splitting of data into discrete sets for training, validation, and testing is a fundamental procedure in machine learning. This process, known as data splitting, is critical to the construction of models capable of not just learning from current data but also generalizing their learning to new, previously unknown data. The significance of data splitting stems from its capacity to reduce the potential risks of overfitting and underfitting, resulting in balanced model performance [60].

Data can be split into 2 or 3 portions. A common approach is splitting into 2 portions, one for creating a prediction model and the other for assessing the model's effectiveness [60], which can be seen in the following Figure 2.6.

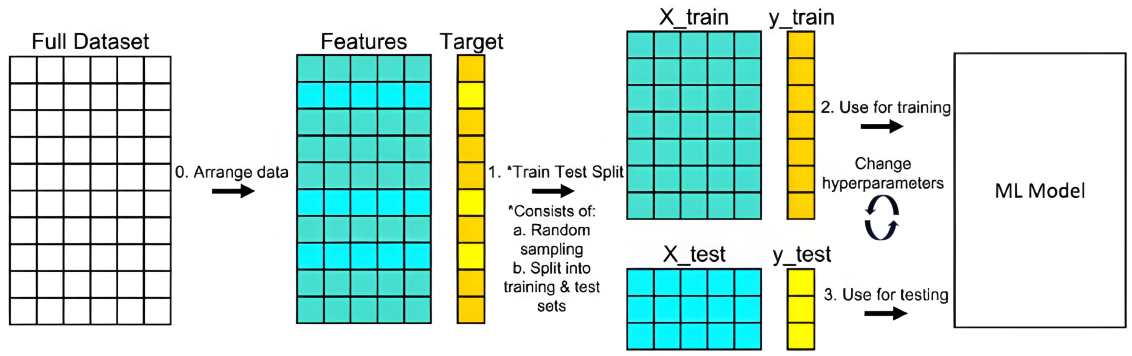


Figure 2.6: Splitting dataset into 2 portions [59]

There are several data splitting techniques used, each adapted to the particular needs of the dataset and the issue at hand. The simplest method is random splitting, in which data points are allocated at random to the testing, validation, or training sets. Conversely, stratified splitting guarantees that every split preserves the percentage of classes that were in the original dataset, which makes it very useful for managing unbalanced datasets. When dealing with time series data, time-based splitting is essential since the chronological order of the data points is essential. [60]

2.2.3 Selection of the Learning Algorithm

In machine learning, choosing a learning algorithm is a significant choice that is impacted by a wide range of variables. Important factors in this decision-making process include the kind of issue (classification, regression, clustering, etc.), the size and properties of the dataset, computing limitations, and the required accuracy and interpretability of the model [27].

Algorithms like Decision Trees, Support Vector Machines (SVMs), and Neural Networks are widely employed in the fields of regression and classification. Decision trees are preferred because of their ease of interpretation and simplicity, particularly in situations when it is essential to comprehend the decision-making process. SVMs are useful in situations when the number of features far outweighs the number of samples because of their well-known efficacy in high-dimensional domains. Deep learning models among them have become quite popular because of neural networks' unmatched ability to handle complicated, nonlinear interactions in big datasets. [27, 11]

An algorithm's selection also depends on how bias and variance are traded off [11]. High variance models overcomplicate the model, resulting in overfitting, whereas high bias models oversimplify the problem, resulting in underfitting. Finding a balance is key to developing a model that performs well when applied to newly obtained data [78]. This is further discussed in the next chapter, in the Section 3.3, called "Identification of Problematic Areas".

2.2.4 Feature Engineering, Selection and Extraction

Two crucial methods that can be used to reduce the occurrence of false positives in classification models are `feature engineering` and `feature selection`. The effectiveness of the model's capacity to distinguish between classes is improved by both techniques, which also enhances the model's overall predictive performance.

`Feature engineering` is a machine learning technique of selecting and transforming raw data into features[3].This process is visually summarized in the following Figure 2.7. To lower the number of false positives and simplify data transformations, it is necessary to either develop new features or change current ones. False positives are less likely to occur thanks to effective feature engineering tactics that emphasize the real underlying trends in the data.
[58]

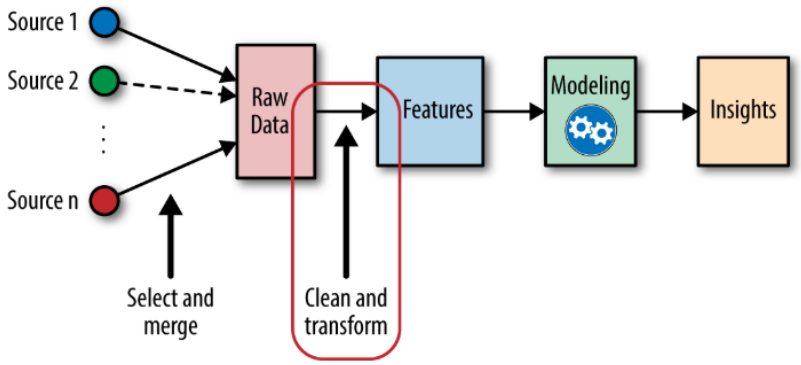


Figure 2.7: Features engineering process [79]

`Feature selection` is the process of choosing the most relevant features. Using this technique we are keeping only the most relevant features, so the overall complexity of the model is reduced, making it less likely to misclassify due to overfitting.

In contrast, dimensionality reduction is at the core of `feature extraction`. `Principal Component Analysis (PCA)` and `Linear Discriminant Analysis (LDA)` are two important techniques that help reduce the dimensionality of the feature space while preserving a significant amount of the dataset's variation. The way this method works is visualized in the Figure 2.8. This not only improves computing efficiency and model performance but also helps to alleviate problems such as the curse of dimensionality [11, 55].

[3]Any measurable input that can be used with a predictive model is referred to as a „feature".

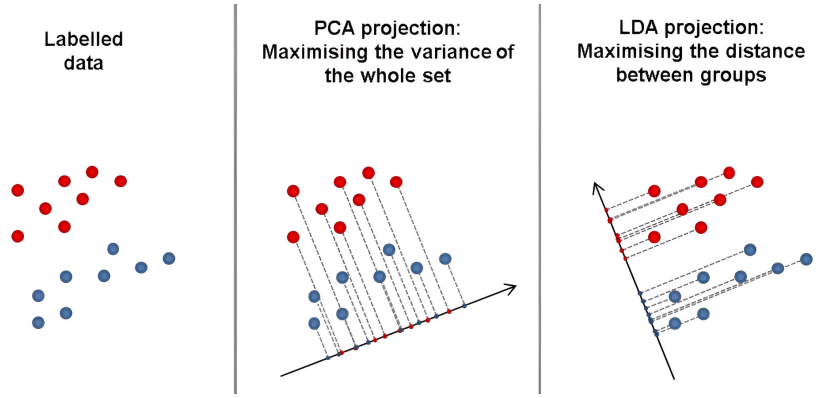


Figure 2.8: PCA and LDA, 2 feature extraction methods [55]

The particular needs of the dataset and the task will determine which option from feature selection or extraction to choose. When handling high-dimensional data, where direct interpretation is less important, feature extraction is crucial even if feature selection is frequently chosen for interoperability [28]. The goals of both approaches are to lower computing costs, increase model accuracy, and reduce overfitting.

2.2.5 Training the Model

A key component of the machine learning process is training a model, which entails modifying the model's parameters to enable it to incorporate knowledge from the training set [27].

Before anything else the parameters of the model are initialized, usually at random, to start the training process. These parameters are then iteratively adjusted using an optimization method, usually gradient descent or one of its variations [11].

Depending on the unique characteristics of the situation, selecting the right `optimization function` and `loss function` is extremely important. For instance, a typical loss function for regression issues is a mean squared error, but a typical loss function for classification tasks is cross-entropy loss [66]. Likewise, because of their computing speed, optimization methods such as stochastic gradient descent are favored for enormous data sets.

2.2.6 Evaluation of the Model's Performance

The conventional approach to assess this is to see how well the model performs on a dataset (training set) that it has never encountered before. Model performance is evaluated using a variety of criteria, the selection between which is based on the specifics of the issue at present. Metrics including accuracy, precision, recall, and F1 score are frequently employed in classification tasks.

It's also crucial to take into account the idea of overfitting, which occurs when a model performs well on training data but badly on fresh, untested data. To provide a more accurate assessment of the performance and generalizability of the model, methods such as cross-validation are employed. This metric is also further discussed later (see Section 3.3).

All metrics related to the subject matter are covered in detail in the following section. For a comprehensive understanding of the different performance metrics, it is recommended that you seek Section 2.4.

2.2.7 Model Tuning and Optimization

In order to improve a model's performance, machine learning requires the processes of model optimization and tuning. This procedure entails modifying the hyperparameters of the model, which are the pre-training parameters that are not acquired from the data [9]. The performance of a machine learning model may be greatly impacted by this hyperparameter adjustment.

Data Preparation: The outset of this methodology involves the meticulous preparation of the dataset, essential for ensuring compatibility with various machine learning and deep learning algorithms. This step includes encoding class labels into numeric values and partitioning the dataset into features (`X`) and labels (`y`). A stratified split is then executed to divide the data into training and testing sets, utilizing the `train_test_split` function to maintain a proportional representation of each class across the splits.

Initial Parameter Configuration: For the initial configuration, baseline models for XGBoost, CNN, and SVM are set up with preliminary parameters. These include the learning rate (eta) for XGBoost, the number of convolutional layers for CNN, and the kernel type for SVM. The selection of these initial parameters is informed by exploratory data analysis and empirical insights specific to each model type.

Defining the Parameter Space: A comprehensive hyperparameter space is established for each model, incorporating a wide range of values for key parameters. For XGBoost, this includes `max_depth`, `min_child_weight`, and `gamma`. For CNNs, the parameter space spans the number of layers, kernel sizes, and activation functions. SVM's space covers `C` (regularization parameter) and `gamma` (for RBF kernel). This approach ensures a broad exploration of configurations during the tuning phase.

Exhaustive Search and Cross-Validation: Employing the grid search module from `sklearn.model_selection`, an exhaustive search is conducted over the defined hyperparameter space for each model. This process is paired with a 5-fold StratifiedKFold cross-validation to ensure an impartial evaluation of each parameter set. This combination facilitates a detailed assessment of model performance across a diverse array of configurations.

This systematic technique for selecting hyperparameters integrates exhaustive grid search with cross-validation, encompassing a broad examination of potential hyperparameter combinations for XGBoost, CNN, and SVM models. Grid search meticulously assesses performance across all specified hyperparameter sets, ensuring an exhaustive exploration of the space, as depicted in the following Figure 2.9.

K-fold cross-validation, employed alongside grid search, divides the data into 'k' segments for a balanced examination of each hyperparameter set. This stratagem significantly mitigates the risk of overfitting and offers a comprehensive evaluation of the model's generalization capabilities, proving invaluable for ensuring the model's efficacy across diverse data segments.

By adopting this multifaceted approach, not only do we identify the most effective hyperparameters for enhancing model performance, but we also ensure the adaptability and reliability of models across varied datasets.

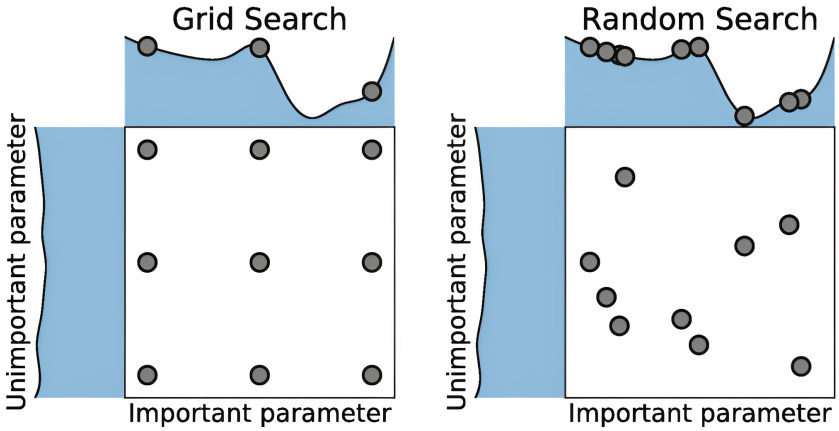


Figure 2.9: Visualization of hyperparameter optimization strategies [3]

2.3 Machine Learning Approaches for Domain Classification

Classification models are a form of machine learning technique that applies a class label to domain input instances. In machine learning, there are several types of classification problems, including binary, multi-class, multi-label, and unbalanced classifications.

Binary classification includes predicting one of two classes, whereas multi-class classification entails predicting one of at least two classes. Multi-label classification involves estimating one or more classes for each sample, whilst unbalanced classification works with datasets whereby the number of cases in each class is not equal [4, 54].

There are Several well-known ML techniques in the domain classification area, that have been extensively researched and utilized. Models include Convolutional Neural Networks (CNN), Support Vector Machines (SVM), Decision Trees (DS), XGBoost, Naive Bayes, and Logistic Regression. These algorithms have shown great potential in correctly classifying textual material into predetermined domain categories. In the forthcoming sections, I will systematically examine each of these methods individually.

2.3.1 Convolutional Neural Network (CNN)

CNNs have been popular for domain classification tasks due to their capacity to learn hierarchical representations of written content automatically, capturing detailed patterns and characteristics within the input text. CNNs are well-suited for domain classification tasks due to their hierarchical feature learning capabilities, which allow them to successfully distinguish domain-specific properties within text input [25]. CNN architecture is illustrated below in Figure 2.10.

Common CNN architecture is composed of [1, 25]:

- **Convolutional layers** - Textual data may have its spatial hierarchies of characteristics effectively extracted by CNNs thanks to the convolutional layers. These layers efficiently capture local relationships and patterns in the incoming data by applying different filters to it [1]. This feature is very helpful for classifying domains since the way letters and subsequences are arranged can give a clear indication of the kind of domain—malicious or benign.

- **Pooling layers** - To decrease the number of dimensions in the data, pooling layers are used after convolutional layers. In addition to lowering the chance of overfitting,

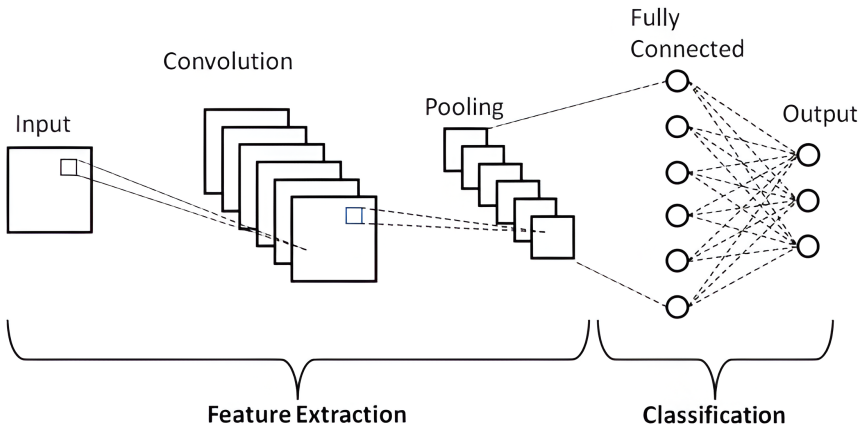


Figure 2.10: 5 layer CNN architecture [1]

this step improves the computational efficiency of the network. By concentrating on the most important characteristics that influence the categorization choice, pooling layers in domain classification can assist in extracting the most pertinent information from the textual input [25].

Max pooling, one of the most common pooling operations, is defined as:

$$P(i,j) = \max_{m,n \in \text{Window}} I(i+m, j+n)$$

where I is the input, and P is the pooled output.

- **Activation function** - In a neural network, an activation function is a mathematical operation that is performed on the output (also known as „activation“) of a node prior to it being forwarded to the subsequent layer [25]. An activation function's main goal is to add non-linearity to the network so that it may learn and mimic intricate patterns and functions that linear operations are unable to.

  The Rectified Linear Unit (ReLU) is a commonly used activation function in CNNs, defined as:

  $$f(x) = \max(0, x)$$

  This function introduces non-linearity into the network, allowing it to learn more complex patterns.

- **Fully-conected layer** - Every input neuron in a fully linked layer is coupled to every output neuron. In a fully linked layer, the output of a neuron is provided by

  $$O_j = \sigma\left(\sum_i W_{ij} X_i + b_j\right)$$

  where W and b are the weights and biases, X is the input, O_j is the output, and σ is the activation function.

2.3.2 Support Vector Machines (SVM)

SVMs flourish at dealing with high-dimensional feature spaces, making them ideal for domain classification problems requiring complicated textual input [64].

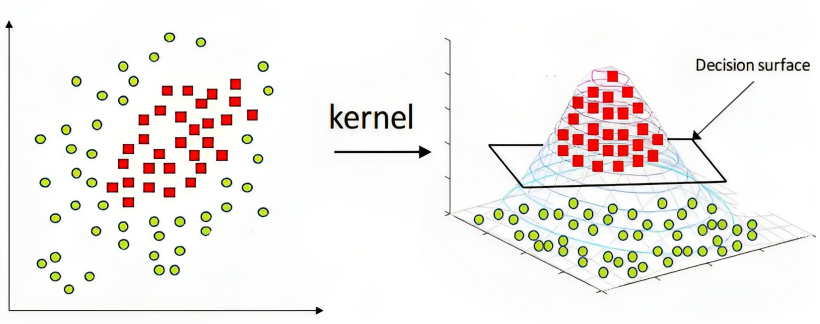


Figure 2.11: SVM [37]

Provided in Figure 2.11 is a graphical representation of SVM that seeks the hyperplane in a high-dimensional space that best separates classes. The primary objective is to achieve the greatest class margin, which is the amount of space between the hyperplane and the nearest data points (support vectors).

Assume the given data are linearly separable, and the line $w^T x + b = 0$ indicates the decision boundary between the two classes, where w represents a weight vector, b represents the bias or threshold, and x indicates the training sample. The hyperplane divides the space into two spaces: (1) positive half-space where the samples from the first/positive class (ω^+) are located, and (2) negative half-space where the samples from the second/negative class (ω^-) are located. [73]

The objective of SVM is to determine the values of w and b to position the hyperplane as far away from the nearest samples as doable. Additionally, SVM seeks to generate the two planes, H_1 and H_2, as follows [73]:

$$H_1 \rightarrow w^T x_i + b = +1 \text{ for } y_i = +1 \tag{2.1}$$

$$H_2 \rightarrow w^T x_i + b = -1 \text{ for } y_i = -1 \tag{2.2}$$

where $w^T x_i + b \geq +1$ is the plane for the positive class and $w^T x_i + b \leq -1$ represents the plane for the negative class (see Fig. 4). These two equations can be combined as follows:

$$y_i(w^T x_i + b) - 1 \geq 0 \quad \forall i \ = 1, 2, ..., N \tag{2.3}$$

The SVM margin represents the sum of d_1 and d_2 as follows, margin $= d_1 + d_2 = \frac{2}{\|w\|}$, where d_1 and d_2 represent the distance from the first and second plane, respectively, to the hyperplane, and $d_1 = d_2$ as shown in Fig. 4. In the SVM classifier, the margin width needs to be maximized as follows:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i(w^T x_i + b) - 1 \geq 0 \quad \forall i \ = 1, 2, ..., N \tag{2.4}$$

The first term denotes the function that needs to be minimized. It involves the square of the weight vector's Euclidean norm. Because a bigger value suggests a lower margin, minimizing this term corresponds to maximizing the margin. [73]

2.3.3 Decision Trees (DS)

For domain classification tasks, decision trees have been applied, providing a visible and interpretable process for classifying textual input into domain-specific categories. Decision trees can capture complicated boundaries for decision-making and have been used in a variety of domain categorization scenarios.

Class labels are represented by leaves in the tree structure, conjunctions of features are represented by branches, and tests on features are represented by nodes. This structure is outlined in the Figure 2.12.

When using decision tree learning, a decision tree for classification or regression is employed as a predictive model to make inferences about a target variable that has a defined range of values. We may use Gini impurity to quantify the impurity in decision tree algorithms of a dataset and find the best split from a non-terminal node [48]. Gini impurity is computed as follows

$$\text{Gini}(D) = 1 - \sum_{i=1}^{n}(p_i^2) \tag{2.5}$$

Where:

$\text{Gini}(D)$ is the Gini index of the dataset D.

n is the number of classes in the dataset.

p_i is the probability of an item being classified to a particular class.

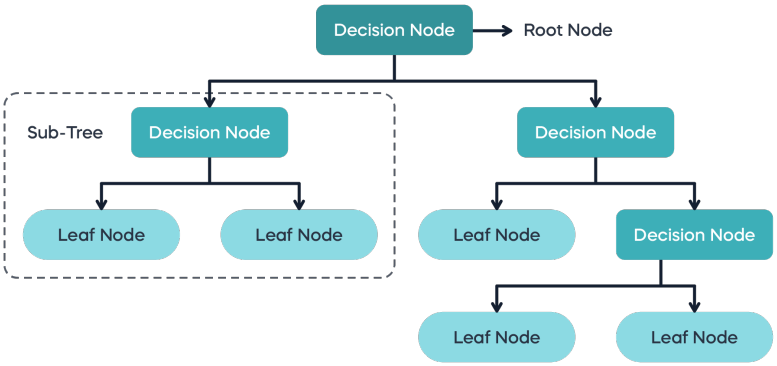


Figure 2.12: Decision tree [74]

2.3.4 Naive Bayes (NB)

Naive Bayes is a probability-based method that uses Bayes' Theorem to determine the likelihood of a data point belonging to a specific class. The Naive Bayes theorem calculates the probability of a hypothesis H given the data D as [68]:

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)}$$

Where:

- $P(H|D)$ is the posterior probability of hypothesis H given the data D.
- $P(D|H)$ is the likelihood, the probability of data D given that the hypothesis H is true.
- $P(H)$ is the prior probability of hypothesis H.

- $P(D)$ is the probability of the data.

The Naive Bayes classifier assumes that the features are conditionally independent given the class label. This assumption simplifies the calculation of the likelihood term [68]:

$$P(D|H) = P(x_1, x_2, \ldots, x_n|H) = \prod_{i=1}^{n} P(x_i|H)$$

The Naive Bayes method serves as both a classifier and a generative model that implies substantial feature independence. It is well-known for its straightforwardness of use, efficiency, and capacity to handle multidimensional data. It is commonly used in the classification of text, spam filtering, sentiment analysis, and others.

It operates as a classifier by revealing log odds and defining linear decision boundaries. Probabilistic techniques for classification problems frequently require modeling the conditional probability distribution $P(C|D)$, where C represents classes and D defines objects in a specific language for classification [24].

The joint probability distribution, on the other hand, presents difficulties due to its exponential growth in terms of the number of attributes (n) and the demand for a complete training set with many instances for each potential description. [68, 24]

These problems can be avoided by establishing attribute independence for the class (n). If the assumption is correct [24]:

$$\text{argmax}_c P(a_1, a_2|c) = \text{argmax}_c P(a_1|c)P(a_2|c)$$

The naive Bayes assumption does not degrade prediction accuracy for all values of attributes A_1 and A_2, even if the actual probability differs.

This assumption indicates that the combined attributes A_1 and A_2 correlate with the class in the same way as the individual attributes A_1 and A_2 do. [68, 24]

2.3.5 Linear and Logistic Regression (LR)

Linear Regression is a supervised learning approach that predicts a continuous result variable using one or more predictor variables. It is a linear way of modeling the interaction between several independent variables and a dependent variable. Linear regression is a parametric method that assumes a linear connection exists between the input and output variables as depicted in Figure 2.13. Linear regression is used for predicting continuous-valued outputs and is not suitable for binary classification tasks like malign domain classification. [35]

Therefore we use logistic regression, which is more suitable for malign domain classification, as it is specifically designed for binary classification tasks. Logistic regression is usually used when the dependent variable is categorical, and it works well for binary classification tasks such as predicting whether an email is spam or not, or if a domain is malicious or not. It is an efficient approach for dealing with classification issues that is simple to implement, understand, and train. [34]

It is frequently utilized in many fields, including finance, economics, and social sciences. Logistic regression models have been proven to be successful in binary and multiclass domain classification problems.

The linear predictor function for a given data point in logistic regression is defined as the linear combination of the explanatory variables and a set of coefficients used for regression. The logistic function, also known as the sigmoid function, is used to transfer the linear

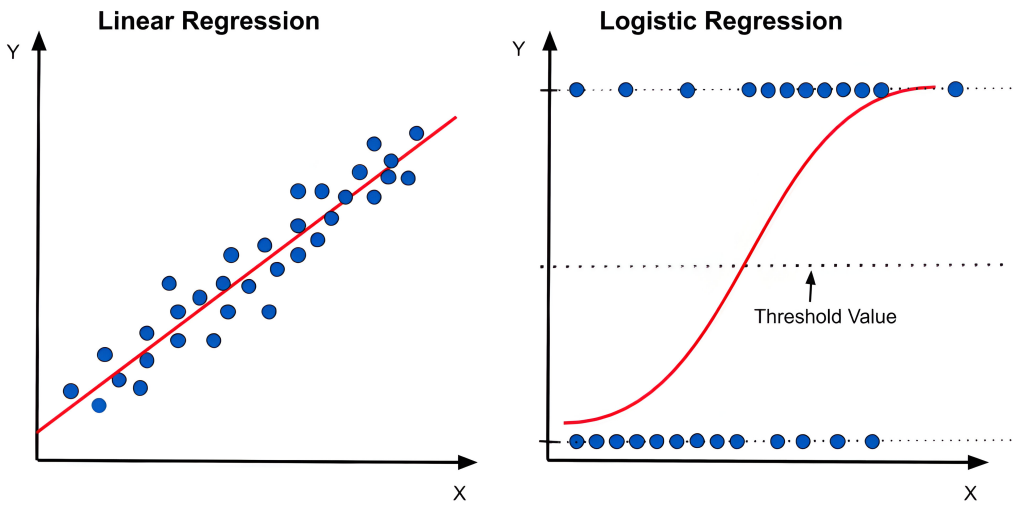


Figure 2.13: Logistic regression vs linear regression [53]

combination to the range, which represents the chance of the input falling into one of many classes. [35, 30]

The threshold is an important factor in establishing projected class labels in logistic regression. Rather than discrete predictions, logistic regression models produce continuous probability for the target classes. These probabilities, which vary from 0 to 1, show the likelihood or confidence that an instance belongs to a specific class.

The threshold is used to assign class labels to these continuous probabilities. In binary classification tasks, the threshold value is set at 0.5 by default. If a particular instance's anticipated probability is higher than or equal to 0.5, it is allocated to the positive class. Otherwise, it is assigned to the negative class. [35, 34]

However, the threshold used might have a major impact on the classification model's performance, especially when dealing with unbalanced datasets. After the computation of the model, it is recommended to evaluate how well the model predicts the variable that is the dependent one, which is known as goodness of fit. The Hosmer-Lemeshow test is a popular method for determining model fit [35].

2.3.6 Extreme Gradient Boosting (XGBoost)

XGBoost is a popular and efficient ensemble machine-learning technique, especially when it comes to classification and regression tasks. XGBoost is often used in domain difficulties with classification, because of its ability to build a strong classification model by merging numerous weak learners.

XGBoost is known for its great prediction accuracy. It successfully handles complicated data linkages and can capture nonlinear patterns, which is useful in identifying sophisticated harmful domain behavior. XGBoost can handle huge datasets and is parallelizable, making it suited for scalable solutions in cybersecurity applications with massive datasets [41].

The structure of the XGBoost model is highlighted by the visual representation Figure 2.14.

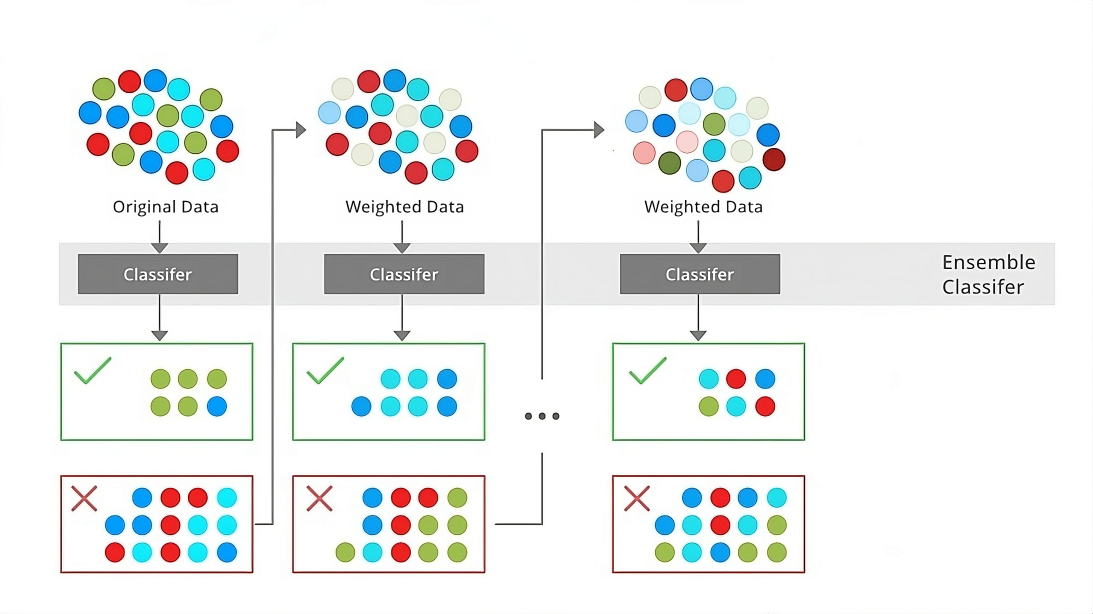


Figure 2.14: XGBoost structure [26]

2.4 Performance Metrics and Evaluating Model Efficiency

The performance of different classification approaches is evaluated in a comparative examination of existing classification models, notably in the context of domain categorization. In the context of domain classification, the F1 score and accuracy are crucial measures for determining a model's dependability [45, 47].

Other frequent classification measures, in addition to F1 score and accuracy, include precision, recall, and the area under the ROC curve (AU-ROC) [29].

2.4.1 F1 score and Accuracy

The F1 score is one of the most important metrics in machine learning. This metric is used to evaluate the model's accuracy in case you deal with imbalanced datasets or you observe more data points of one class than of another, meaning there is an uneven distribution between the two classes being predicted [45].

F1 score is a combination of 2 metrics, known as `precision` 2.6 and `recall score` 2.7 [47]. Precision is a class-balanced metric, meaning, it can be only used if the dataset has the same amount of samples for each class. It's an easy-to-understand indicator that gives a rough idea of how well a model is doing. However, most of the time this metric cannot be used. Let's take this example. In the real world, the vast majority of domains are benign. An accuracy rate of 99% for a simple prediction model that classifies all domains as benign does not guarantee its effectiveness. Additional factors must be considered to evaluate its efficacy. [47]

The recall is often known as the True positive rate. This metric indicates how many of the positive class samples in the dataset the model successfully identified.

$$\text{Precision} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{\text{TP} + \text{TN}}{\text{Total}} \tag{2.6}$$

$$\text{Recall (Sensitivity)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

On our FETA project (see Section 3.4), we use the F1 score as the main accuracy indicator, because benign datasets are often significantly larger than malicious ones, and we aim for a higher F1, as an increased F1 score indicates an improved balance between recall and precision. This is particularly crucial because a higher F1 score implies that the model is getting better at correctly identifying both benign and malicious instances, which is fundamental to the system's reliability and effectiveness. In a binary classification problem F1 is calculated as follows 2.8:

$$\begin{aligned} F1 &= \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \\ &= \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \end{aligned} \tag{2.8}$$

2.4.2 ROC Curve and AUC

ROC (Receiver Operating Characteristic) curves and AUC (Area Under the Curve) are metrics used to assess the effectiveness of classification models, especially in binary classification situations. They participate in comprehending the trade-offs between the `true positive rate (TPR)` 2.9, also known as sensitivity, and the `false positive rate (FPR)` 2.10, known as specificity [29].

$$\text{True Positive Rate (Sensitivity)} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2.9}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} \tag{2.10}$$

The ROC curve is a depiction of the true positive rate versus the false positive rate at different threshold levels. The AUC is determined as the area under the ROC curve and indicates the degree or level of separateness between the classes. A higher AUC suggests that the model is better at differentiating between classes. [29, 57]

Visual representation of these 2 metrics is shown in the following Figure 2.15.

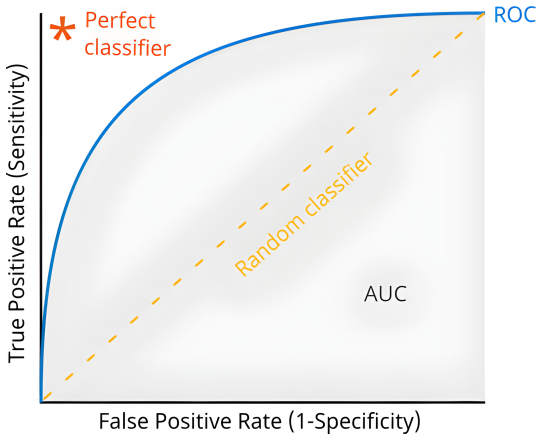


Figure 2.15: ROC curve [52]

Chapter 3

Comparative Analysis of Current Classification Models

This portion of the thesis expands deeply into the landscape of classification models. This comprehensive review examines a variety of classification models, evaluating their efficacy and application in the context of domain categorization. This section devotes a large amount of time to addressing various performance indicators, giving an overview for evaluating the efficacy of these models. The emphasis is not just on the theoretical features of these models, but also on the practical consequences of applying these models to real-world data, notably in the domain name classification field. This involves identifying places where existing research has revealed obstacles or constraints.

This section also includes an overview of the work done under the FETA project, providing insights into how these models were implemented and advanced in a focused research context.

3.1 Background

For malicious domain classification, we can use plenty of classification methods, but we can use a few of them that will provide us sufficient results [80, 7]. The model used is often determined by several criteria, including the quantity and quality of the dataset, the features used for classification, computing resources, and the problem's specific requirements. [54].

The most suitable models for domain name classification are `Neural Networks`, `XGBoost` and `SVM` [7, 23], these classification methods used for domain classification are deeply analyzed and summarized in the following Section 3.2.

The approach we specialize in is referred to as the domain-centric approach [8], and it focuses on identifying domains based on their names—which is especially effective when the traffic is encrypted and only the domain name is accessible. Aside from the domain-centric method, additional approaches include web scraping and website classification based on their visual appearance [21]. These alternate techniques might enhance the overall domain classification by adding more layers for analysis.

3.2 Existing Research on Domain Names Classification

Current domain name classification research involves a variety of methodologies, approaches, different model usage, and many problems. Malicious domains, such as phishing domains,

have been the subject of several studies that have examined and suggested detection techniques based on domain name data. In this section, I will analyze some of the recent research, strengths, and weaknesses of these approaches, including critical issues, which I will address in the following Section 3.3.

Faroughi et al. [22] focus on categorizing internet domain names into specified categories using graph-based semi-supervised learning, where the nodes represent domains and the edges represent the various commonalities across domains.

They presented six different approaches, each validated through 5-fold cross-validation on the training set [22]. The overall results of this experiment are presented in the following Figure 3.1.

Performance of the different classifiers obtained on the 5-fold cross validation set.

Method	Accuracy	$Precision^{macro}$	$Recall^{macro}$	$F-Measure^{macro}$
TFIDF-Supervised-DomainsName	0.359	0.342	0.358	0.313
NFA-Supervised-DomainsName	0.410	0.414	0.331	0.348
SVM-Supervised-DomainsSequence	0.404	0.335	0.367	0.334
SSDN-SemiSupervised-DomainsName	0.471	0.486	0.390	0.404
SSDS-SemiSupervised-DomainsSequence	0.441	0.390	0.344	0.344
SSB-SemiSupervised-both	0.522	0.528	0.456	0.465
Naive-Most-Frequent	0.133	0.005	0.040	0.008
Naive-Uniform	0.033	0.064	0.063	0.061

Figure 3.1: Graph-based semi-supervised learning results [22]

Ravi et al. [65] propose an approach for identifying and categorizing domain names generated through Domain Generation Algorithms (DGA) utilizing deep learning architectures and traditional machine learning methods, such as CNN or SVM. The results are shown in the Figure 3.2 below.

Method	Testing 1				Testing 2			
	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
CNN - LR	0.590	0.623	0.590	0.553	0.627	0.652	0.627	0.596
CNN - NB	0.530	0.581	0.530	0.504	0.582	0.617	0.582	0.551
CNN - KNN	0.607	0.624	0.607	0.581	0.620	0.649	0.620	0.587
CNN - DT	0.557	0.582	0.557	0.513	0.598	0.634	0.598	0.563
CNN - RF	0.594	0.611	0.594	0.544	0.629	0.631	0.629	0.582
CNN - SVM-L	0.582	0.573	0.582	0.529	0.636	0.617	0.636	0.586
CNN - SVM-RBF	0.206	0.195	0.206	0.073	0.388	0.201	0.388	0.218

Figure 3.2: Results of the proposed method, CNN-classical machine learning algorithms [65]

Liu et al. [49] seek to enhance the identification of algorithmically created domain names by addressing the issue of imbalanced sample distribution. They employed an improved borderline synthetic minority over-sampling approach (Borderline-SMOTE) and achieved an average F1-Score of 83.45 using RCNN and Borderline-SMOTE.

For DGA identification and categorization, **Vinayakumar et al.**[76] conducted an in-depth examination of several deep learning architectures such as architectures linked to recurrent structures, CNN, and combinations of CNN and recurrent structures.

A study presented by **Aljofey et al.** [6] offered an innovative technique for anti-phishing by integrating several variables such as URL character sequence, hyperlink information, and textual content. On their dataset, the study obtained an accuracy of 96.76% with a false-positive rate of 1.39%, while on a benchmark dataset, the study achieved an accuracy of 98.48% with a false-positive rate of 2.09%.

In his master's thesis, **P. Kocinec** [43] presents a Convolutional Neural Network (CNN) classifier that he developed for the purpose of DGA classification. The implementation of this classifier resulted in a significant reduction of false positives, with a True Positive Rate (TPR) of 97%.

In a study conducted by **Torroledo et al.** [75], it was found that classification based on TLS features can be effectively used to detect phishing and malware domains. The study reported a precision rate of 0.8963, using 30 TLS features, indicating the potential of this approach.

Chatterjee et al. proposed a lexical-based approach that utilizes 14 lexical features extracted from URLs to train their model. Their model achieved an F1 score of 0.867 on a dataset of 73,575 URLs.

Lastly, in order to improve the quality of categorization by addressing the lack of ground-truth labels, **Sims et al.** [70] introduced a novel method that estimates a probabilistic ground-truth label mask for domain name categorization using a majority vote among an ensemble of different spatial point spread functions.

Despite the various and complex approaches described above, there is still a lot of space for improvement. Most studies emphasize the relatively low F1 scores, which frequently fall below 0.9, suggesting that the balance between accuracy and recall is not optimal. Furthermore, these studies were mainly done on smaller datasets (10,000 to 100,000 samples), which may not provide the adaptability and generalizability necessary for deploying these models in a variety of real-world scenarios.

3.3 Identification of Problematic Areas

During my research on domain name classification (see Section 3.2), I noticed recurring issues. The most common problems include:

- **inaccurate classification**,
- **overfitting (overtraining)**,
- **imbalanced datasets**,
- **high rate of false positives**,
- **problem of not having ground-truth datasets**.

We have faced some of these problems also in our project FETA, some of these issues were addressed (see Section 3.4). Recognizing the importance of large datasets and feature selection, we focused on both of these during the development process. Although these efforts were mainly successful, several difficulties were not successfully addressed. My future research has concentrated on addressing the remaining challenges in order to improve our model's robustness and accuracy (see Chapter 4 and Chapter 5).

3.3.1 Poor F1 Results

The classification of domain names is complicated due to the wide range of methodologies for producing domain names. The accuracy and reliability of classification models are affected by this complexity.

The accuracy of a model, often quantified through the F1 score (see Subsection 2.4.1, can be influenced by a variety of factors. Imbalanced datasets can result in skewed predictions and low recall for minority classes [46], whereas overfitting can reduce a model's capacity to generalize to new datasets [77]. Feature selection is equally important. Irrelevant features may fail to capture important data patterns, resulting in worse precision and recall. Furthermore, the model's complexity involves a trade-off, while more sophisticated models may detect small nuances in data, they are also prone to overfitting and need a large amount of data and processing resources to train. Finally, the size of the training dataset has a considerable effect on model performance. Research has demonstrated that models developed for small datasets frequently fail to generalize successfully when used for bigger datasets (see Section 3.2). Larger datasets provide more comprehensive examples and variations, allowing models to learn and generalize better, thereby potentially improving the F1 score.

3.3.2 Model's Overfitting

Creating computational models with strong prediction and generalization capabilities is one of the main goals in the field of machine learning. These models are trained to predict the outputs of an unknown target function within the context of supervised learning. A finite set of training data T, which consists of instances of input pairings and the accompanying intended outputs, encapsulates this target function: $T = \{[\mathbf{x}_1, \mathbf{d}_1], \ldots, [\mathbf{x}_n, \mathbf{d}_n]\}$, where $n > 0$ denotes the number of these arranged input/output pairs or patterns. The training phase's ultimate objective is for the model to successfully generalize to new, untested data in addition to correctly predicting the outputs for the input samples in T. `Overfitting` is a common indicator of poor generalization when the model learns the training instances by repeated learning and is unable to make predictions for patterns that exist outside of the training dataset.

Moreover, overfitting is a serious problem in model training since it teaches the model to recognize noise and random fluctuations in addition to the training data's underlying patterns. This usually happens when the model is trained for an excessive amount of time or when it is unreasonably complicated in comparison to the quantity of training data that is available [77]. Overfitting has negative effects on the model's capacity to generalize to new, unknown data, which makes it perform poorly on real-world datasets even when the model has high accuracy on the training set [77, 17].

The well-known machine learning trade-off known as the `Bias-Variance problem` is represented by these two imperatives: high generalization and reliable prediction on T. [66]

The widely used approach to balance the model's minimal Bias and minimal Variance is using `cross validation`. Its main purpose is to evaluate the generalizability of a statistical analysis's findings to a different set of data. K-fold cross-validation is the most prevalent type of cross-validation [44]. In k-fold cross-validation, the dataset is partitioned into k equal-sized subsets or folds. For each fold i , the model is trained on the data from all folds except fold i , and then it is tested on fold i [38]. This process is repeated for each fold, and the results are averaged to produce a single estimation. The process can be represented as follows [38, 44]:

$$CV_k = \frac{1}{k} \sum_{i=1}^{k} E_i \tag{3.1}$$

where CV_k is the cross-validation score over k folds, and E_i is the evaluation metric (such as accuracy or mean squared error) computed on the i^{th} fold.

The optimal balance between variance and bias can be seen in the Figure 3.3.

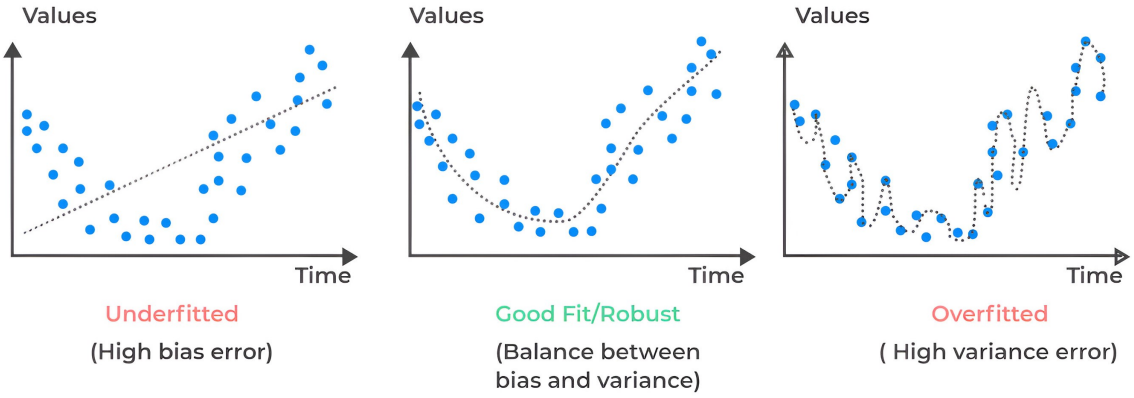


Figure 3.3: Demonstration of generalization and overfitting [17]

3.3.3 Imbalanced Datasets

One of the significant issues encountered in domain classification is dealing with unbalanced datasets. This problem is widespread because there are usually far more benign domains than malignant ones, leading to a skewed distribution of data.

This problem notable across many articles and research on this topic is `imbalanced datasets`. Imbalanced datasets occur when one class is overrepresented in comparison to another, illustrated in Figure 3.4.

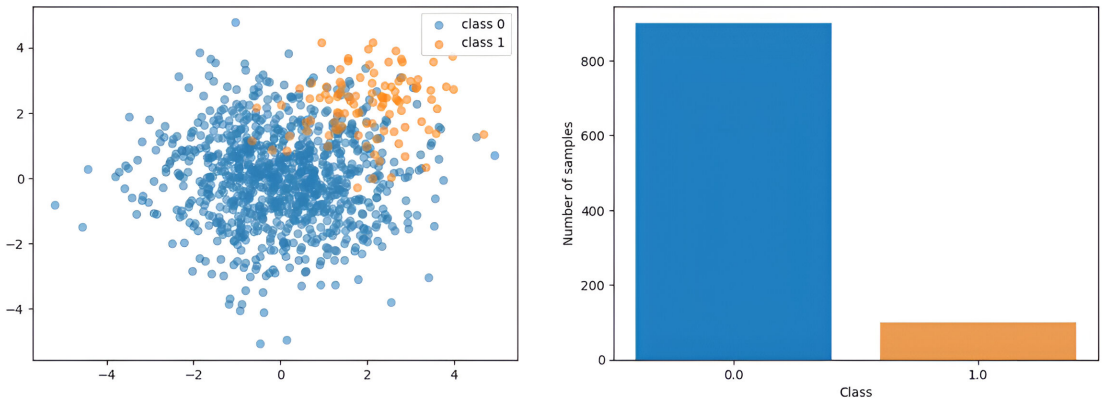


Figure 3.4: Imbalanced datasets [19]

As a result, classifiers are biased toward the dominant class and misclassify minority classes. The processing of an unbalanced dataset is divided into two phases. First, reduce the imbalance ratio in data sets, and then choose the most important features from the

dataset. Imbalance is often stated as a ratio of the total number of occurrences in the majority and the total number of occurrences in the minority classes. Overlapping minor disjunctions, a lack of density, noisy data, and dataset variation are all characteristics of unbalanced data that make categorization extremely challenging. [46]

To overcome this issue, several solutions are suggested, including sampling methods such as oversampling the minority class and undersampling the majority class, algorithmic methods. [46] This approach is depicted in the following Figure 3.5.

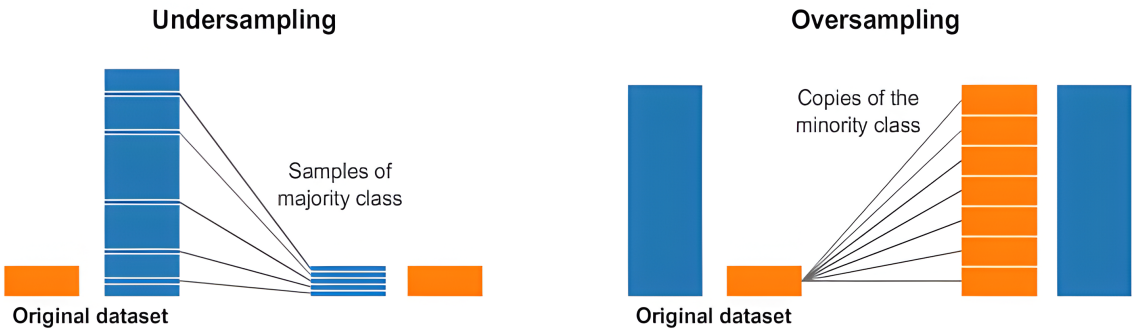


Figure 3.5: Datasets sampling methods [19]

A few examples of sophisticated oversampling methods include heuristic approaches or synthetic sampling. An example of this approach is minority oversampling (SMOTE), demonstrated in the Figure 3.6 below. SMOTE is a technique that creates synthetic samples to oversample the item of the minority class.

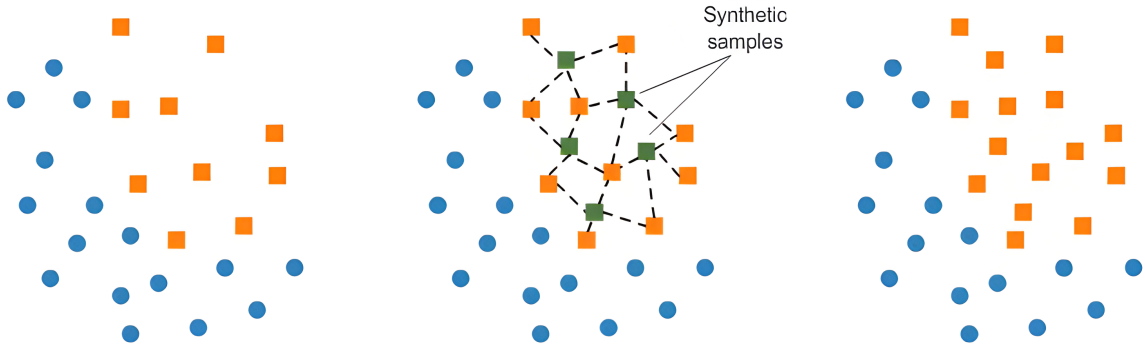


Figure 3.6: SMOTE sampling oversampling method [19]

Common classification methods for handling imbalanced datasets are [46]:

- **Random Forest** - Aggregates results from several classifiers. Performance on unbalanced datasets may be affected by bias due to non-linear correlation between independent and dependent variables.
- **Decision Trees** - Divides training data randomly and condenses sub-trees upon encountering errors. It may require several divisions to adequately handle imbalanced datasets.
- **Neural Networks** - Constantly modifies weights to reduce mistakes. However, these models may not perform well with unbalanced data as they focus on minimizing mistakes for frequent classes.

- **Naive Bayes** - Takes feature flexibility into account but may classify data incorrectly.
- **Support Vector Machine (SVM)** - Finds the ideal hyperplane dissociation using a maximum margin. Performance is impacted by the decision boundary's tendency to favor the minority class despite being aware of unbalanced datasets.

3.3.4 High FPR

The issue of false positives, wherein benign domains are incorrectly categorized as malicious, poses a significant threat to the reliability of classification models. This challenge can lead to inaccurate assessments and can negatively impact decision-making processes.

The most straightforward approach to visualize the amount of false positives is to use a `confusion matrix`. A confusion matrix is a matrix, or table, used to assess the impact of a classification model. It contains the number of:

1. **True Positive (TP)** - The model predicted the positive class correctly,
2. **True Negative (TN)** - The model predicted the negative class correctly,
3. **False Positive (FP)** - The model predicted a positive class, but it was incorrect,
4. **False Negative (FN)** - The model predicted the negative class, but it was really positive.

. A high FPR (false positive rate) indicates that a significant proportion of benign domains have been erroneously identified as malicious. This can be seen in the Figure 3.7. Such misidentification may result in the unreasonable banning or flagging of legitimate websites, interrupting normal web operations and causing irritation to both users and domain owners.

When misclassifications occur often, it undermines the classification system's credibility and trustworthiness. Users and network executives may lose faith in the system's capacity to reliably detect threats, resulting in under-utilization or outright abandonment of the utility. As such, effective mitigation strategies are needed to reduce the incidence of false positives to maintain a high level of trust and confidence in the security of online communications.

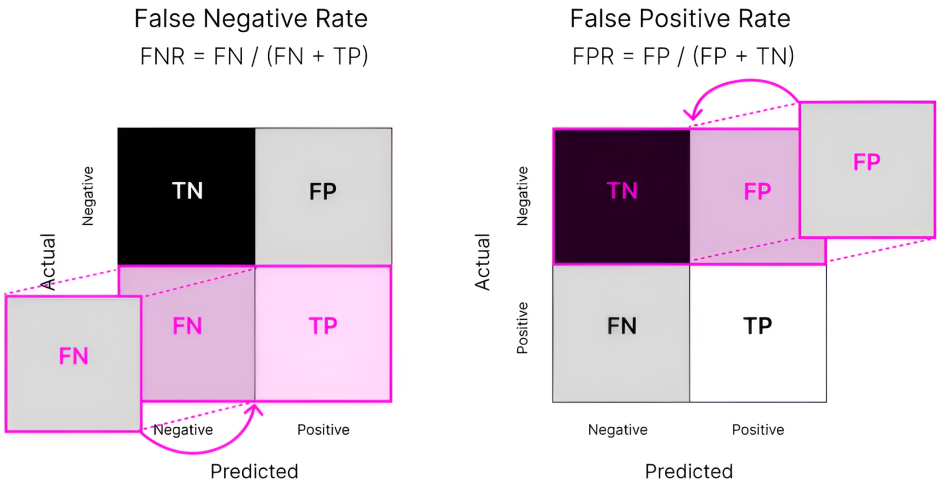


Figure 3.7: False positive rate

One of the approaches to reduce FPR is improving the `feature selection`, `balancing the datasets`, adjusting the threshold value, or implementing some post-processing of the results, for example using a whitelist of known benign domains.

3.3.5 Not Having the Ground-truth Dataset

Ground-truth datasets are essential for domain name classification. Ground-truth data is labeled data that serves as a standard for training and assessing classification algorithms. They provide reliable labeling for machine learning models. The quality of these datasets impacts the accuracy of models. Having high-quality datasets is essential for developing reliable models [2]. This section addresses the significance of a robust ground-truth dataset in domain name classification, as well as its influence on the performance and reliability of classification models.

A ground-truth dataset is primarily used for training and verifying machine learning algorithms. Ground-truth data that is accurate and comprehensive guarantees that the model learns to distinguish between benign and harmful domains. The model's capacity to generalize and perform properly on unknown data is directly influenced by the model's training quality. This approach can have a dual benefit of not only mitigating overfitting but also minimizing the incidence of false positive domains.

A known ground truth is necessary for the calculation of metrics like accuracy, precision, recall, and F1 score (see Subsection 2.4.1), which indicate how effectively the model is working.

This section has examined the problematic areas related to the classification of domain names and has also discussed potential solutions to address such issues. The following chapter will look at the design and implementation of these suggested solutions, which aim to enhance the overall efficiency of domain name classification.

3.4 Overview of Work on The "FETA" Project

The project FETA (Flow-based Encrypted Traffic Analysis) is a collaboration project among three teams (CESNET, ČVUT in Prague, and Brno University of Technology - FIT). The project is funded by the Ministry of the Interior of the Czech Republic under the VJ Program – Strategic Support for the Development of Security Research in the Czech Republic 2019-2025 (IMPAKT 1).

The overarching objective of the FETA project is to pioneer novel technologies, tools, and methodologies to monitor encrypted communication, identifying security threats to network infrastructure, devices, and services, thereby bolstering their cybersecurity resilience. This four-year project commenced in 2022 and is committed to advancing the country's security research initiatives.

The project has already successfully devised an innovative and highly accurate method for detecting IoT malware. This breakthrough methodology has been documented and published in the esteemed IEEE Internet of Things (IEEE IoT) journal under the article titled "BOTA: Explainable IoT malware detection in large networks"[1].

[1] https://ieeexplore.ieee.org/document/9983820

3.4.1 Domain Dataset Collections

The FETA project's Domain Dataset Collections have been cautiously chosen from a variety of sources that are well-known for their extensive archives of domain-related data.

The data were obtained from a number of reliable sources, including Cisco Umbrella[2], CESNET[3], ThreatFox[4], and MISP (Malware Information Sharing Platform)[5].

Are collections are stored in MongoDB[6]. MongoDB's document-oriented model aligns well with the nature of domain datasets, facilitating the storage of semi-structured or unstructured data prevalent in threat intelligence feeds. This design choice enables efficient querying and indexing, crucial for expedited retrieval and analysis of domain-related information required for threat detection and security analysis within the FETA project. MongoDB stands out as an ideal choice for our domain dataset collections within the FETA project, particularly due to its exceptional capability in managing large volumes of data.

The domain dataset collections amassed for the FETA project are substantial, comprising hundreds of thousands, and in certain cases, millions of individual data entries.

Collector Module

The data we collect differs from source to source. The common format of the data is domain names with some flag signaling if they are benign or malign. We need those data in a united format before storing them in our database. For managing and preprocessing the domain-related data we use a module called `dr-collector`[32, 33]. Script establishes a connection with the database, using commands we handle the loading of domain data from different sources and storing this data into the specified MongoDB collection.

The function `run_parallel_resolving` resolves domain-related information (DNS, RDAP, TLS, IP RDAP, geo data, reputation data, etc.) for each domain name.

It is crucial to run this function before saving the domains, ensuring data integrity, and storing the information in a structured way in the database. This provides the database with relevant information and enables subsequent domain-related data analysis.

For creating these collections different sources were used (see Table 3.1).

The benign collection is the primary domain name provided by CESNET. Phishing domains were collected from OpenPhish and PhishTank. Malware was collected primarily from ThreatFox, and a few of the domains were also collected from MISP. DGA collection was created as a proportion pick from Fraunhofer DGArchive. In the following Table 3.1 we can see the collections that are currently present in our database:

As previously stated, the utilization of the `dr-collector` script has allowed for the uniform collection of data. This, in turn, has enabled us to more effectively work with the data and train our model.

[2] https://umbrella.cisco.com/
[3] https://www.cesnet.cz/
[4] https://threatfox.abuse.ch/
[5] https://www.misp-project.org/
[6] https://mongodb.com/

Collection	Source	Records
dga_2310	Fraunhofer DGArchive	228,000
benign_2307, benign_2310	Cisco Umbrella	480,000 (approx.)
benign_cesnet2_intersect, benign_cesnet_intersect_2307, benign_cesnet_union_2307, cesnet2, cesnet2_2310, cesnet_2307	CESNET	Up to one million (approx.)
malware	ThreatFox, MISP	110,000
misp_2307, misp_2310	MISP, OpenPhish, PhishTank	77,000
unique_root_cas	-	-

Table 3.1: Data Collections for Malicious Domain Detection

Each record, which corresponds to the collected domain name, is stored in our database in a consistent format. Specifically, the structure of each record adheres to a predetermined format, ensuring the uniformity of the data collected. This is shown in Listing 3.1

```
{
  "_id": {"$oid": "64add36187a68cb65e178a44" },
  "domain_name": "srvupdate.duckdns.org",
  "category": "unknown",
  "dns": {
    "dnssec": {... },
    "remarks": {... },
    "sources": {... },
    "ttls": {A, AAAA, SOA, ... },
    "A": ["141.147.78.236"],
    "MX": {priority, TTL, value, ... },
    "TXT": [""],
    "zone_SOA": {primary_ns, resp_mailbox_dname, expire, min_ttl, ...}
  },
  "evaluated_on": {"$date": "2023-08-27T01:03:46.727Z" },
  "ip_data": [
    {
      "ip": "141.147.78.236",
      "from_record": "A",
      "remarks": {rdap_evaluated_on, geo_evaluated_on, icmp_evaluated_on,
          ... },
      "rdap": {whois_server, expiration_date, last_changed_date, ... },
      "asn": {as_org, network_address, prefxi_len, ... },
      "geo": {country, region, city, latitude, ... },
      "rep": null,
      "ports": []
    },
```

```
    {... },// Other IP data entries
    {... }
  ],
  "label": "malware",
  "rdap": {... },
  "remarks": {... },
  "source": "malware.txt",
  "sourced_on": {"$date": "2023-07-12T00:10:41.288Z" },
  "tls": null,
  "url": "srvupdate.duckdns.org"
}
```

Listing 3.1: JSON example of our data structure

3.4.2 Classification Model

The Figure 3.8 below depicts the several processes in the process of constructing our classification model for the FETA project.

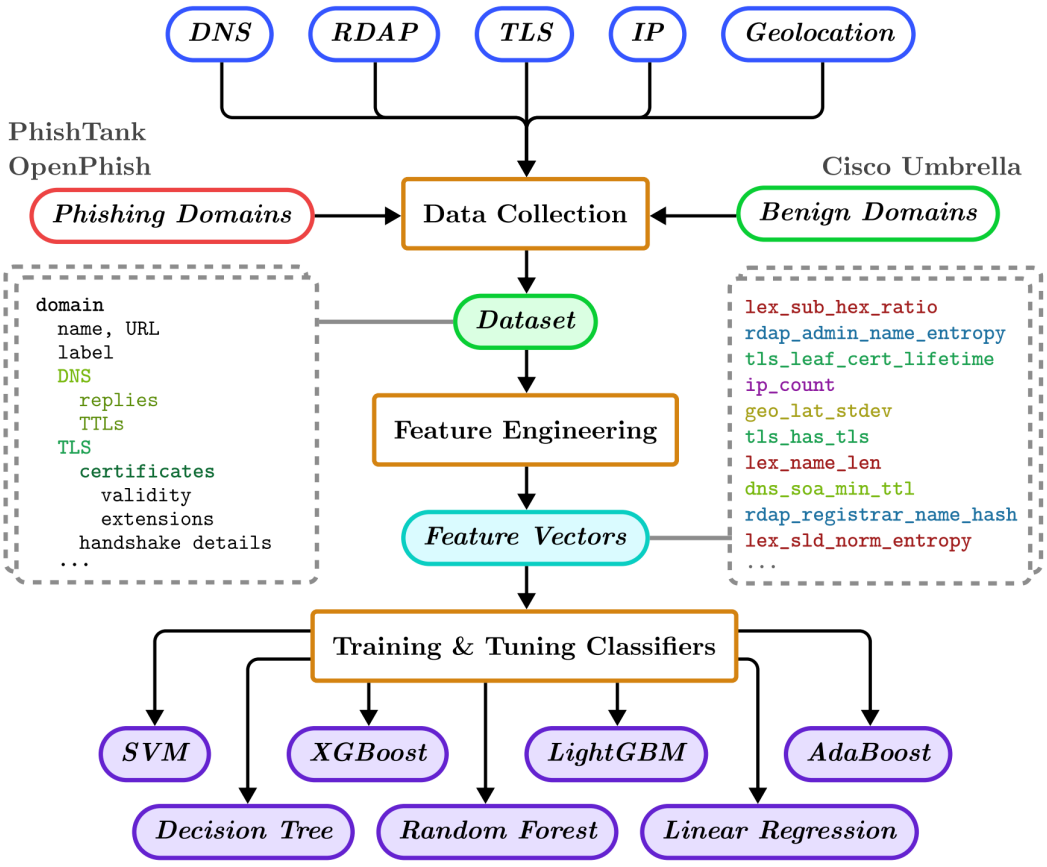


Figure 3.8: Overview of the classifier creation [33, 61]

1. **Data collection**: The initial stage involved acquiring a labeled dataset. This dataset included instances of various categories, such as both phishing and benign domains, which the model was trained to recognize.

2. **Feature engineering**: After the data was collected, it was necessary to design features suitable for the classification tasks. For phishing classification, features such as DNS characteristics (like the number of DNS records or time to live [TTL]), lexical features (like length, hyphenation, subdomain count), and TLS fields (including root certificate validity and leaf certificate validity) were developed.

3. **Feature selection**: Following the development of the features, a subset of the most informative features was selected for the classification task. This selection was made using various feature selection techniques, but mainly using correlation analysis. There were 169 characteristics in the dataset when I started my research. I had assisted with creating a few of these features before writing this research. In my study, I go into depth on how these elements were further examined in order to determine which ones were inefficient and how to remove them.

4. **Training and tuning the classifier**: Once the features were selected, the classifier was trained and tuned. Various machine-learning techniques were used for classification, including support vector machines (SVMs), decision trees, random forests, XGBoost and LightGBM. XGBoost was primarily utilized for phishing classification.

5. **Evaluation**: After training, the performance of the classifier was assessed on a held-out test set. This assessment assisted in making sure the classifier did not overfit the training set.

3.4.3 Data Overview

In the Figure 3.9 there is a clear visualization of the entire phishing dataset, while the second plot 3.10 represents the benign one. These plots contain some of the most important features on the left and their values on the right. Due to the limitations of space, only a few features are depicted, and a comprehensive view of the feature vector is available in Appendix C. Each rectangle represents the value of one feature. In this case, the dataset is a set of domains for which the values of each feature are known. Features are properties of these domains.

The rectangles in the fence are arranged according to the value of the leftmost feature. Rectangles with higher values are further away from the origin of the x-axis. The rectangles in different colors represent different classes of domains. In this case, the classes are malicious and benign domains.

The plot can be used to identify trends in the data and easily spot the difference between benign and malign datasets. For example, it may highlight missing geographical data or show that malicious domains frequently have greater values for specific features like the number of IP addresses or TLS certificates.

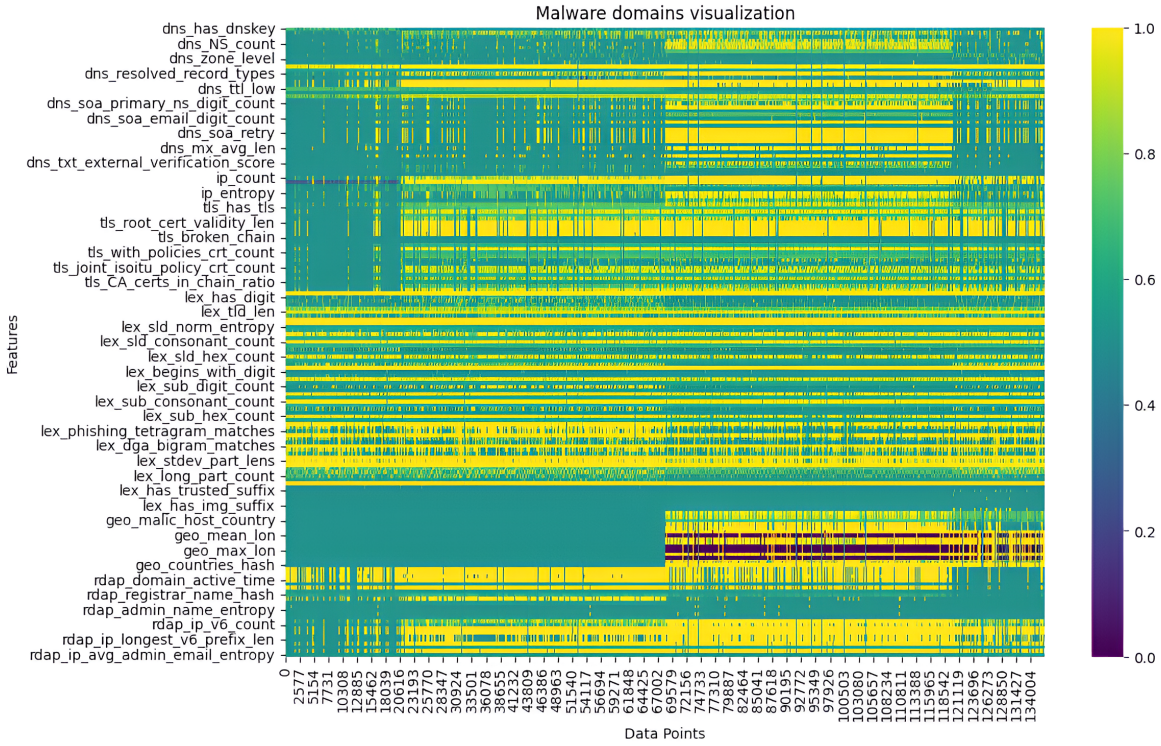


Figure 3.9: Vizualization of Phishing domains

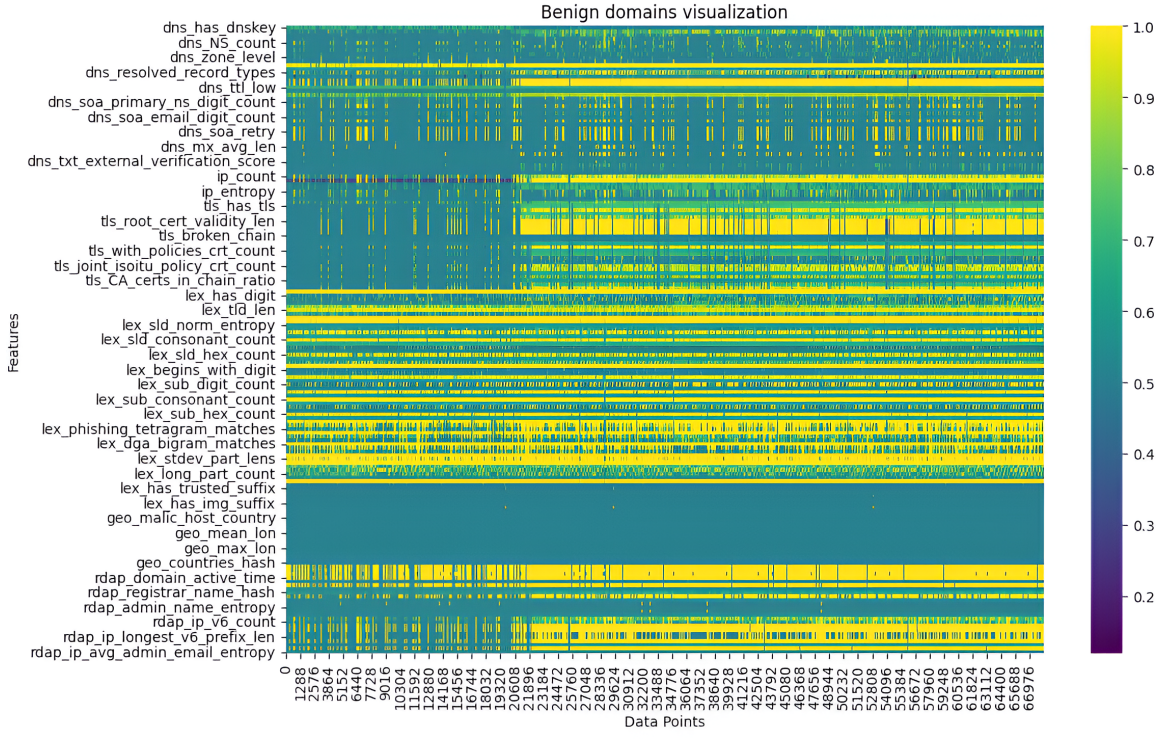


Figure 3.10: Vizualization of Benign domains

3.4.4 Summary of The Achieved Results

The project's unique findings caught the attention of the professional community, resulting in the publication of the article in the Feisty Duck website newsletter—a valuable cybersecurity knowledge base organized by renowned security expert and researcher Ivan Risti. This website is well-known in cybersecurity circles for providing insights into SSL/TLS protocol security via services such as SSL Labs and Hardenize, which are helpful when evaluating web server security and communication.

In December of 2023, a research paper titled **"Spotting the Hook: Leveraging Domain Data for Advanced Phishing Detection"** was accepted for presentation as a short paper at the 37th IEEE/IFIP Network Operations and Management Symposium (NOMS 2024). The paper describes a novel approach to detecting and mitigating phishing attacks using domain-specific data and presents experimental results that demonstrate the effectiveness of the proposed method. The paper's acceptance at NOMS 2024 is a testament to the significance of the research and its potential to make a valuable contribution to the field of network security.

Chapter 4

Preliminary Design of Strategies to Enhance the Effectiveness of Classification Models

In the previous chapter, I have discussed many limitations and problems when using domain names classification. This chapter outlines and discusses possible solutions and strategies, that I propose, for optimizing the classification process. Based on these problems I have designed several optimization approaches, that should resolve these problems, or reduce their limitations. The proposed solution would include:

- process of creating ground-truth datasets,
- further feature selection,
- data preprocessing,
- imbalanced data handling,
- advanced hyperparameters tuning.

The following chapter examines further the implementation, including all the important components and designed solutions.

4.1 Enhanced Feature Selection

The concept of this process was already described in the previous chapters (see Subsection 2.2.4). This section will aim to understand the logic behind the theory and show multiple methods of how the feature vector can be examined to eliminate redundant features, or those with negative impact.

Choosing the most relevant features can be done by using different approaches, the way I am choosing the relevant ones is by calculating its `SHAP values`[1]. Another "classical method of determining the the correct set of features consisted of prolonged EDA with the additional feature engineering. [14]" Another traditional approach to selecting the right collection of features is known as EDA, which stands for exploratory data analysis. I incorporated this approach as a part of data preprocessing later on (see Section 4.3).

[1] Shapley Additive Explanations - values that give a model's features a respective relevance value

SHAP values indicate how much each feature is important in the context of the whole model. This concept assumes that features act like players in the group. The goal of this approach is to predict the overall contribution of each feature. [14]

To further examine and illustrate the influence of each feature, I created multiple Python scripts that generate plots of the model's SHAP values. These visualizations assist in understanding not just which features have the most impact on the model's predictions, but also how each feature's contribution varies over data points. The scripts provide a range of visualizations, including summary plots that aggregate SHAP values throughout the dataset and detailed plots that focus on individual predictions, giving a better understanding of the model's behavior under different circumstances.

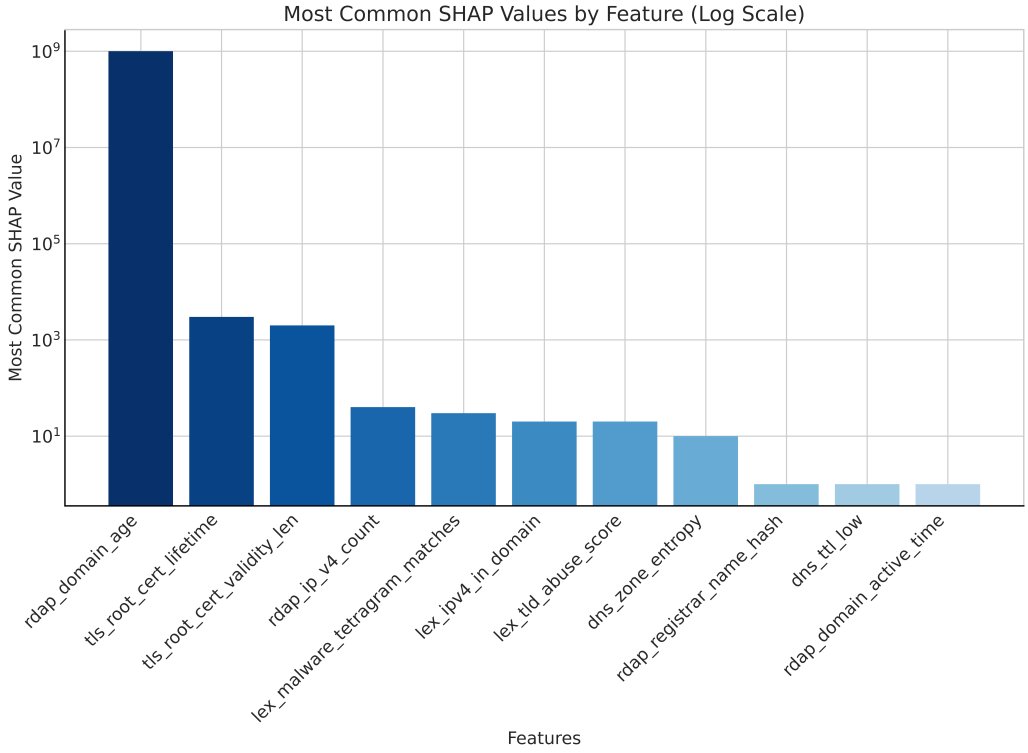


Figure 4.1: Features distribution based on SHAP values

The idea behind the Figure 4.1 is to attempt to find out what features have the highest impact on the final prediction. If a feature has the highest SHAP values, it does not necessarily imply a negative impact, it is important to further identify whether the effect of a feature is positive or negative.

I created a script 4.1, that runs training jupyter notebook multiple times, each time dropping different features with abnormal SHAP values, meaning values from the previous Figure 4.1.

```
def run_notebook_and_extract_fp(notebook_path, feature=None):
    # Load the notebook
    with open(notebook_path, 'r', encoding='utf-8') as f:
        notebook_content = f.read()

    # Parse the notebook
    notebook = nbformat.reads(notebook_content, as_version=4)

```

```
    # Modify the feature_to_drop variable with chosen features
    if feature:
        for cell in notebook.cells:
            if cell.source.startswith("feature_to_drop"):
                cell.source = f'feature_to_drop = "{feature}"'
                break

    # Jupyter notebook execution
    ep = ExecutePreprocessor(timeout=600, kernel_name='python3')
    ep.preprocess(notebook, {'metadata': {'path': './'}})

    # Extract the false positive value contained in the last cell
    last_cell_output = notebook.cells[-1].outputs[0].text
    return int(last_cell_output)

# List of features to drop
features_to_drop = ['rdap_domain_age' ... 'tls_root_cert_lifetime']

# Execute the notebook for the first time without dropping any feature
initial_fp_value = run_notebook_and_extract_fp('Playground.ipynb')

# Execute the notebook for each feature
for feature in features_to_drop:
    fp_value = run_notebook_and_extract_fp('Playground.ipynb', feature)
    # Calculate the percentage change in false positives
    percentage_change = ((fp_value - initial_fp_value) / initial_fp_value) * 100
    change_direction = "increased" if percentage_change > 0 else "decreased"
```

Listing 4.1: Multiple runs of Jupyter Notebook for Feature Dropping

The first run of the script above is without any feature drops, meaning the model training is performed on the origin feature vector. Other runs are compared to the first one, so we can see, how exactly the amount of false positives changed. The output of the script is shown in the Table 4.1.

Original Feature Vector		369
Feature Dropped	Percentage Change in FP	New FP Value
rdap_domain_age	**5.33% decrease**	**352**
lex_phishing_tetragram_matches	2.36% decrease	360
rdap_ip_v4_count	1.57% decrease	365
tls_root_cert_lifetime	70.41% increase	633
rdap_domain_active_time	4.73% increase	389
rdap_time_from_last_change	2.96% increase	382
dns_zone_entropy	0.79% increase	374

Table 4.1: Output of the script provided above

To see how each feature contributes to the final score more in detail, we can use quantitative visualization commonly known as `SHAP force plots`. Using this plot 4.2 we can also see the ordering of used features, meaning the most positive and most negative ones [14].

The force plot 4.2 shows value, which is the prediction for the chosen observation. The positive SHAP values are shown on the left side of this figure, while the negative values

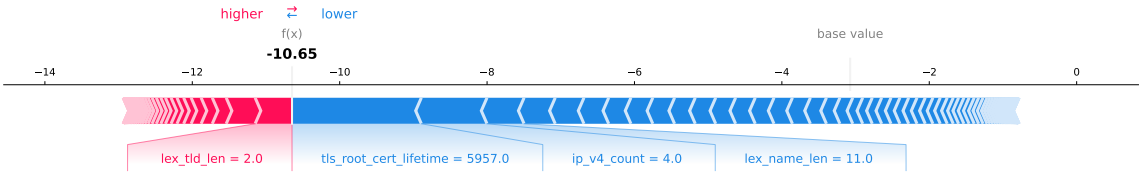


Figure 4.2: Force plot of features for 1 randomly chosen domain name

are shown on the right, as though they were in competition. The lines connected to the features show how much each feature deviates from the base value in the prediction.

A waterfall plot 4.3 is the other option to see the feature's contribution in detail. This plot contains almost the same information, but a different style of features plot can be useful for us to see other relations and connections between them.

This plot presents these contributions sequentially. Once the base prediction has been made, each feature's contributions are added or subtracted one at a time in descending order of their magnitude until it reaches the final prediction value. This is evident from the visual representation in the following Figure 4.3.

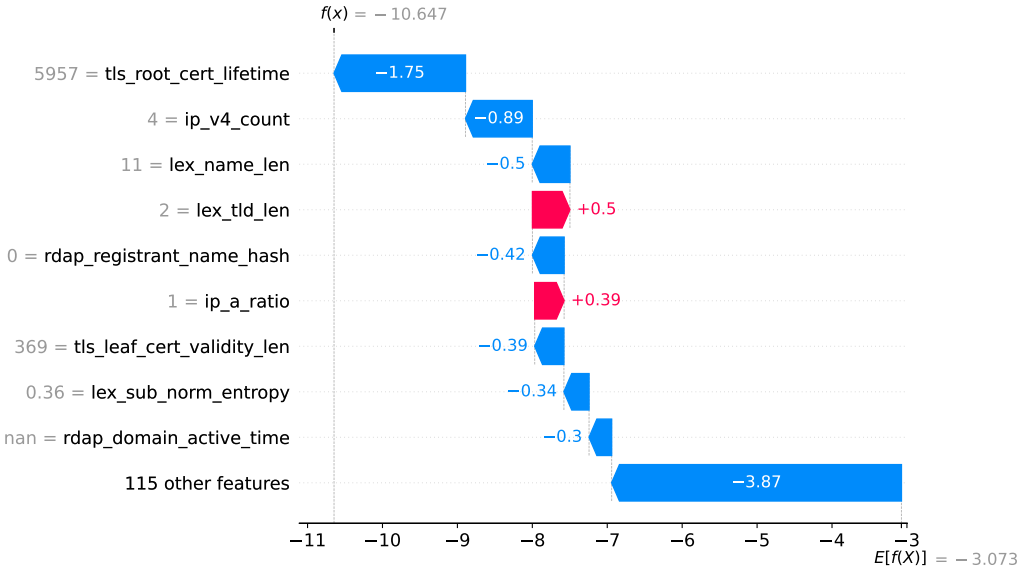


Figure 4.3: Waterfall plot of features for 1 random domain

In these plots, we can see two different values, `"base value E[f(X)]"` and `"output value f(x)"`.

As a point of comparison, the base value, also known as Baseline Value/Expected Value, displays the model's average prediction over all instances in the dataset. In other words, this value means the average prediction that you would get if you were to run every possible input `X` through the model `f` and then average the results.

The output value, often denoted as `f(x)`, is the prediction of the model for the specific instance being analyzed. Meaning when you feed an input `X` into the model `f`, it outputs a prediction `f(x)`.

The difference between these two values is called `prediction difference`, or `baseline deviation`.

$$\text{Prediction Difference} = f(x) - E[f(X)] \tag{4.1}$$

A bigger prediction difference signifies that the provided instance is significantly different from „average“ instances in terms of how the model perceives it.

The following Table 4.2 presents a list of domain names that have been assessed as false positives. Despite being benign, these domains have been erroneously categorized by the model as malicious.

Domains	Possible Reasons for Being Evaluated as Malicious
yeeterracing.com ticks2.bugsense.com www.zhcsm.cn	High `rdap_domain_active_time`: It is essential to notice that extended periods of inactivity followed by sudden spikes in activity may indicate that a domain is being used maliciously. These domains might have remained inactive for a long time before being reclaimed for nefarious reasons.
torrentsnows.com connect.proxytx.cloud torrentz2.eu	High `tls_root_cert_lifetime`: Malicious domains can employ long-lived certificates to limit the risk of detection and avoid the need for reissues. This is a significant concern since it allows such domains to remain active and possibly dangerous over extended periods of time.
cflare.io value-wolf.org urchintelemetry.com	High `ip_v4_count`: Load balancing or disguising the real origin of harmful traffic may necessitate the usage of numerous IP addresses. This implies that many separate IP addresses may be involved in transmitting traffic, making it more difficult to trace the source and identify the responsible party.
ibaraki-84764.herokussl.com os-api.test.os.qkcorp.net awrcaverybrstuktdybstr.com	Low `rdap_domain_age`: Threat actors may utilize newly formed domains for malicious purposes before quickly disposing of them to evade detection and prevent their activities from being tracked.
www.google.com.fritz.box www.loyalbooks.com www.manpower.gov.om	`Domain Names that Seem Ordinary`: Mimicking popular or well-known services might be a phishing tactic used to deceive users.

Table 4.2: Some of the evaluated false positive domain names

These domains could be incorrectly classified due to various reasons. One of the reasons could be a limited feature set, but that's not the case. The feature vector that was made during the FETA project is very complex and it captures the complete behavior or characteristics of malicious domains (see Appendix C).

Another reason is most likely the dynamic content of these sites. Some of them could initially be benign during the model training but might become malicious later.

Another likely reason to include some typical malicious domains in the false positives can be that the model is not calibrated precisely enough to identify specific sorts of malicious behavior because it is too generalized, so domains do not follow typical malicious behaviors they might be overlooked.

When I was going further through the false positives, to examine the exact reason for the false classification of domains being malicious, I noticed, that there was a possibility that these domains could be malicious. The issue may be related to our benign and malignant domain datasets, as indicated in Subsection 3.4.1. These datasets may not be as reliable as previously assumed, given their inclusion of domain names that should not be present. As a result, it is vital to reevaluate the quality of these datasets to ensure their correctness and trustworthiness. This topic is further discussed in the next section (see Section 4.2).

Using SHAP values is only one of the possible techniques for determining important features. For the classification of malignant domains, the most successful of our models is XGBoost, for which we can use another very popular technique. XGBoost has a built-in function to plot feature importance directly using `model.feature_importances_`. This variable indicates how much each feature contributes to the model's decision-making process throughout training. In the Figure 4.4 we can see the top features evaluated based on this metric.

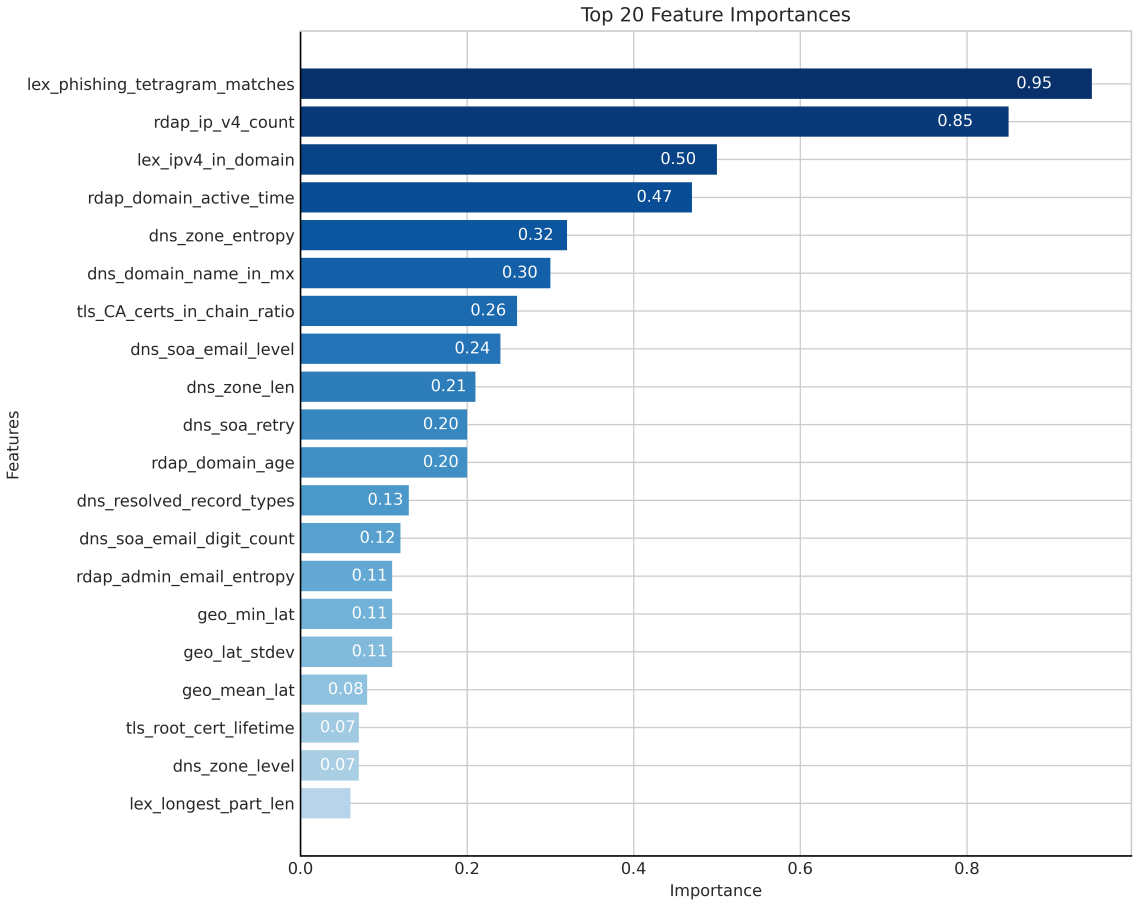


Figure 4.4: Most important features for XGBoost using feature_importances_

We can see there are some differences between top features based on SHAP values and built-in feature importance. This is quite common in machine learning models. The top feature determined by SHAP value emphasizes each instance effect (impact of a feature on individual prediction), whereas the top feature determined by feature_importances shows overall feature importance throughout the whole dataset. Both of these measurements can be beneficial in their respective contexts.

The feature examination also included inspection of domain-specific feature distributions on different datasets—comprising benign and phishing domain instances. This was conducted in order to identify characteristics that may impose time-intensive processing requirements during classification tasks.

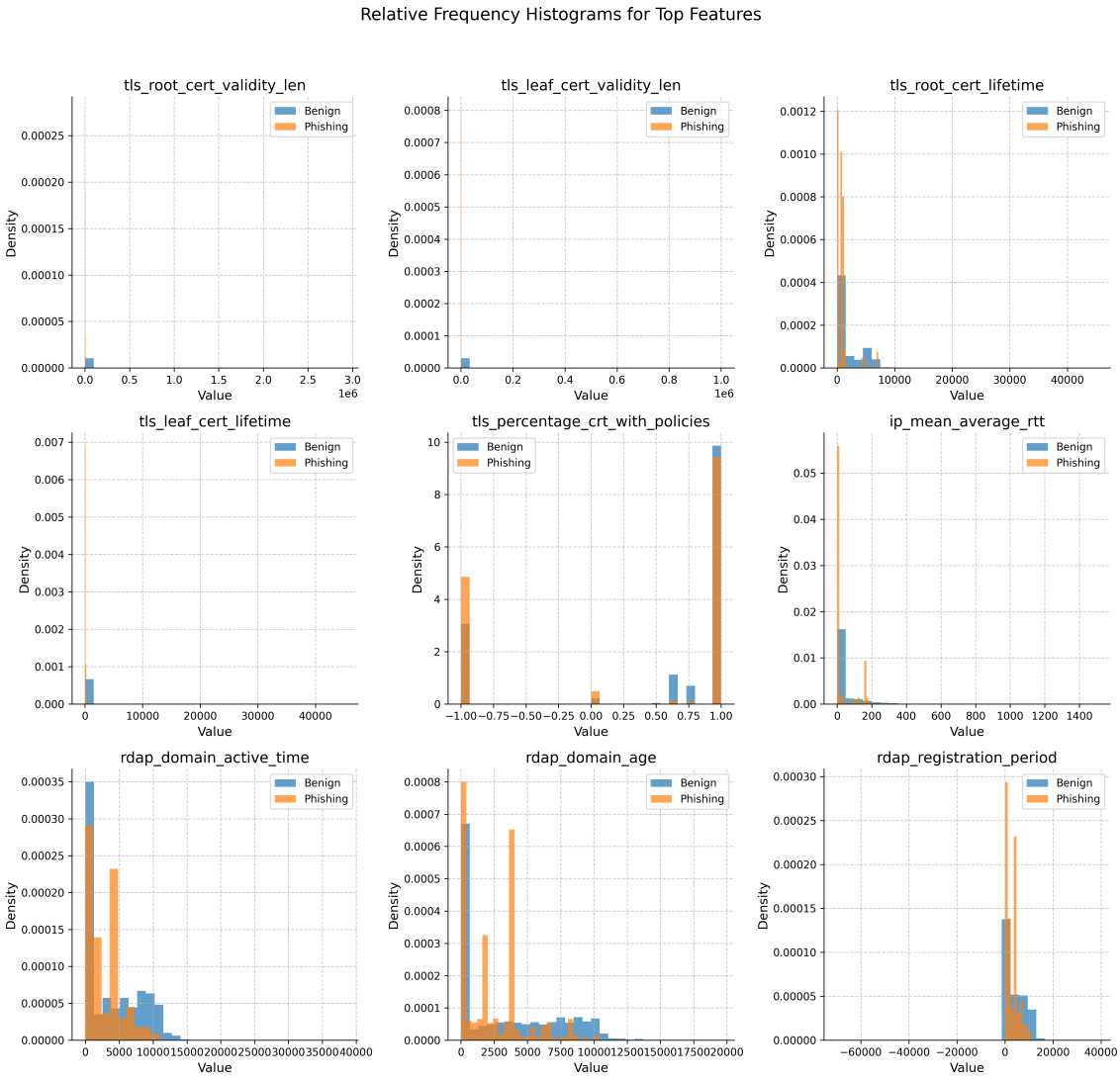


Figure 4.5: Histograms depicting the relative frequencies of certain features of TLS and RDAP

The selected features include those related to domain name `validity length, lifetime, age, active time, and period`. The histograms in Figure 4.5 show the distribution of these features for both benign and phishing domain names, allowing for visual comparison and identification of potential differences.

Upon examining these histograms, we were able to discern intriguing inconsistencies between the datasets with regard to several aspects, particularly:

- `tls_root_cert_lifetime`,
- `tls_leaf_cert_lifetime`,
- `tls_root_cert_validity_len`,
- `tls_leaf_cert_validity_len`,
- `rdap_domain_age`.

The distributions of these characteristics demonstrated a significant variation between the benign and phishing datasets, which suggests that the processing requirements may be time-intensive. The observed inconsistencies can be attributed to the following code snippet Listing 5.1.

```
for certificate in item['certificates']:
    cert_counter += 1  # Incrementing the certificate counter

    # Calculate the validity length of the certificate in days
    validity_len = round(int(certificate['valid_len']) / (60*60*24))
    if validity_len < 0:
        broken_chain = 1
        break

    try:
        # Compute the lifetime of the certificate from the collection date
        lifetime = round((collection_date - certificate['validity_start']).total_seconds()
                    / (60*60*24))
    except OutOfBoundsDatetime:
        # Handle dates outside the permissible range
        print(certificate['validity_start'], collection_date)
        lifetime = -1

    try:
        # Calculate the time remaining before the certificate expires
        time_to_expire = round(
                            (certificate['validity_end'] - collection_date).total_seconds()
                            / (60*60*24))
    except OutOfBoundsDatetime:
        # Handle dates outside the permissible range
        print(certificate['validity_end'], collection_date)
        time_to_expire = -1

    # Check if the certificate has already expired
    if time_to_expire < 0:
        expired_chain = 1
        break
```

Listing 4.2: Python code that illustrates the creation of TLS features

All of the TLS features mentioned above rely on the collection date to make their calculations. This means that if malignant domains were collected during a different time period than benign domains, the classifier would learn that the ones collected in June were bad and the ones collected in September were good. This underscores the importance of ensuring that data collection is done consistently to avoid unintentional bias in the classifier.

This temporal disparity in the data-collecting method may result in a faulty understanding within the classifier's learning mechanism and unnecessary over-fitting of the model. To address this issue the simple solution was just removing these wrong features.

4.2 Correcting Collection Errors: Forming Reliable Ground Truth

This section is a continuation of the previous Section 4.1, in which I discovered that a significant portion of false positives originates from the data we have collected. It is evident that the accuracy of our results is heavily influenced by the quality of the data we collect. Moreover, this claim is supported by several studies presented earlier (see Section 3.2).

To eliminate the problem of having collections with data inaccuracies, I have successfully contacted `Virus Total (VT)`[2], to gain access to their academic API, so I can make all our collections ground-truth. Virus Total is a web-based service that obtains a variety of antivirus scanners thereby making it easier for users to examine files and URLs for malware and dangerous material by scanning them using several antivirus engines and other tools.

The primary objective of this section is to create the design architecture for script, or maybe pipeline that ensures this functionality - inspecting the domain names and creating the correct collections. The proposed architecture for the script is visualized in Figure 4.6.

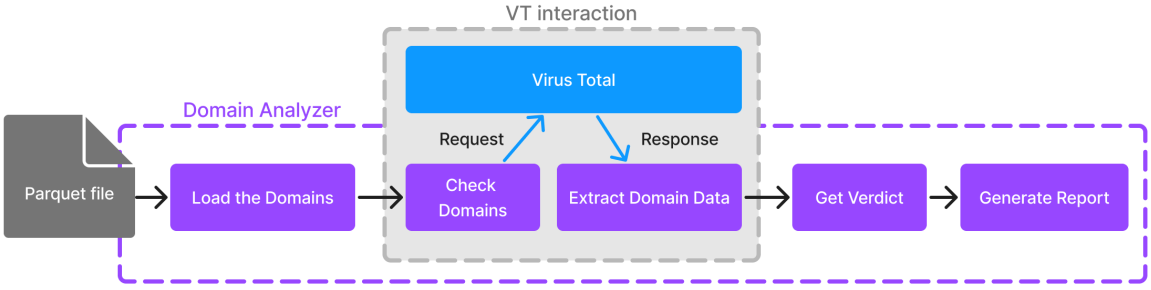


Figure 4.6: Pipeline of the domain names inspection

This design showcased in Figure 4.6 illustrates the different parts working together to carefully validate domains. The output of this script should be a PDF report, marking all domain names as `benign` or `malign`, as depicted in Figure 4.7.

Domain	Verdict	Detection Ratio	Detection Timestamp	Harmless	Malicious	Live Status
hkoptimize.com	Malign	2/73	2023-07-25 22:10:19	71	2	Dead
leavehomesafe.gov.hk	Malign	2/74	2023-08-02 22:10:11	72	2	Dead
yeeterracing.com	Malign	4/72	2023-10-21 16:39:16	68	4	Dead
call.easy2convert4.me	Malign	1/79	2022-12-14 10:30:01	78	1	Alive
natureapi.com	Malign	2/83	2022-04-17 08:09:38	81	2	Alive
solnicka.sk	Malign	1/77	2023-01-26 00:29:22	76	1	Alive
video.browser.tvall.cn	Benign	0/79	2022-10-05 16:46:27	79	0	Dead
www.manpower.gov.om	Benign	0/77	2023-02-14 08:35:25	77	0	Dead
discordapp.page.link	Benign	0/82	2022-04-18 17:48:23	82	0	Alive
wifihotspot.hu	Benign	0/58	2018-12-31 20:44:52	58	0	Alive
	Benign count	**4/10**				
	Malign count	**6/10**				

Figure 4.7: Outcome of the script explained above

Here are brief overviews of the responsibilities and actions of the functions described in the following pseudo-code 1. For the sake of brevity and clarity, the specifics of implementation, including error handling and data manipulation, have been omitted.

[2] https://www.virustotal.com/gui/home/upload

The script architecture presented here delineates the structure for domain validation within Parquet files through the utilization of the Virus Total academic API.

Algorithm 1 DomainAnalyzer

procedure INITIALIZE
    Initialize API key and headers
end procedure
function CHECKDOMAIN(`domain`)
    Input: Domain to be checked
    Output: Information for the specified domain
    Retrieves data associated with a specified domain by making a request to the VirusTotal API.
end function
function EXTRACTDOMAINDATA(`domain, result`)
    Input: Domain and its result
    Output: Extracted domain data
    Extracts essential information from the domain result obtained
end function
function LOADPREVIOUSDATA(`mode`)
    Input: Mode for loading data
    Output: Previously processed domain data
    Loads the domain data that has been processed before, depending on the mode selected
end function
function SAVEDATA(`df, mode`)
    Input: DataFrame and mode
    Output: None
    Saves the DataFrame containing domain analysis data for future reference
end function
function GENERATEREPORT(`df, output_filename, rows_per_page`)
    Input: DataFrame, output filename, rows per page
    Output: None
    Generates a report based on the DataFrame and saves it as a PDF
end function
function PROCESSSELECTEDDOMAINS(`input_mode, mode, batch_size`)
    Input: Input mode, mode, batch size
    Output: Processed domain data
    Processes selected domains based on the mode in batches
end function

The detailed implementation of this proposed schema is explicated in the subsequent chapter (see Subsection 5.1.3).

4.3 Data Preprocessing

Improving data quality through proper preparation and cleaning approaches may considerably improve model performance. Normalization, scaling, imputation of missing values, and resolving class imbalance are some of the most often utilized ways for achieving this desired

result. A well-designed preprocessing pipeline also guarantees that the training data is of excellent quality and that any potential biases or confounding variables are considered.

For example, Figure 4.8 presented below depicts a comprehensive framework for preprocessing steps and strategies that can be used to maximize the effectiveness of machine learning models.

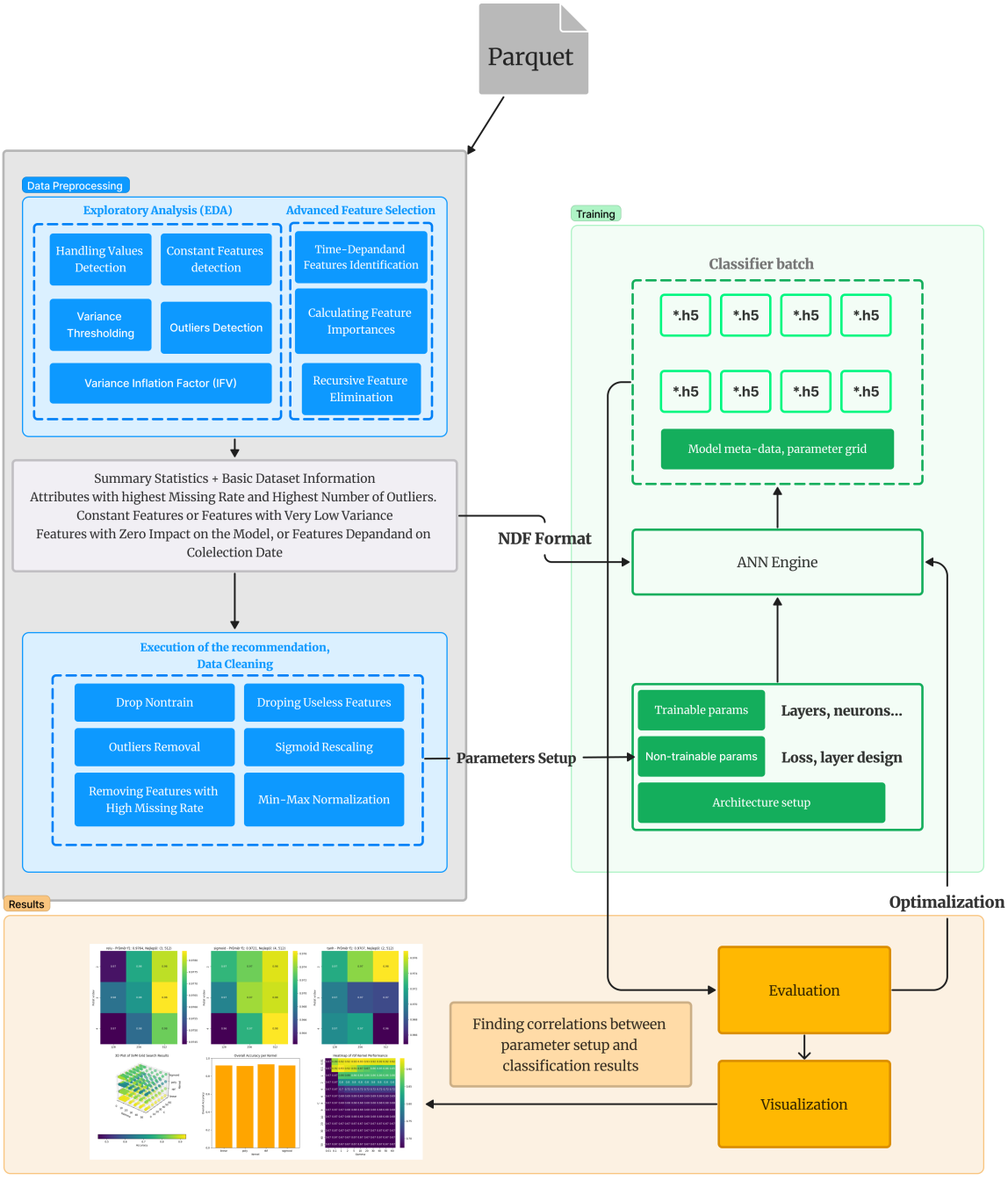


Figure 4.8: Data preprocessing steps

The preprocessing steps are visualized in the grey box, which occurs prior to the training stage, represented by the green box.

During the data preprocessing stage, we provide recommendations for modifying the data and then execute those suggestions. These recommendations are generated through two approaches: feature selection and exploratory analysis, both of which were previously discussed in this Section 4.1.

In my research, I employed a systematic approach to both Exploratory Data Analysis (EDA) and Feature Selection, leveraging various statistical and machine learning techniques. These techniques were systematically incorporated into my Python code to ensure thorough data analysis and feature optimization. Below is a comprehensive description of each technique:

1. **EDA**:

   For obtaining basic information about dataset structure I am using `df.info()` and `df.sescribe()`. These two functions provide information about the number of attributes (which is also the size of our feature vector), and data types, for example, `dtypes: bool(3), float64(93), int64(72), object(2), timedelta64(4)`. It also contains count, mean, standard deviation, minimal value, maximal value, etc. for each attribute.

   - **Handling Missing Values** - In my implementation, missing values are identified using the *df.isnull().sum()* function within the *explore_data* method. This approach provides a comprehensive count of missing values across all columns in the dataset.
   - **Constant Features Detection** - To identify features that have a single unique value throughout the dataset, I used *df.columns[df.nunique() == 1]* within the *explore_data* method. Detecting and removing constant features is crucial as they offer no variability and hence no predictive power for machine learning models.
   - **Outliers Detection** - For outlier detection, I have chosen `Tukey's method using the Interquartile Range (IQR)`, which is a common technique for outlier detection. This approach is illustrated in the following image 4.9.

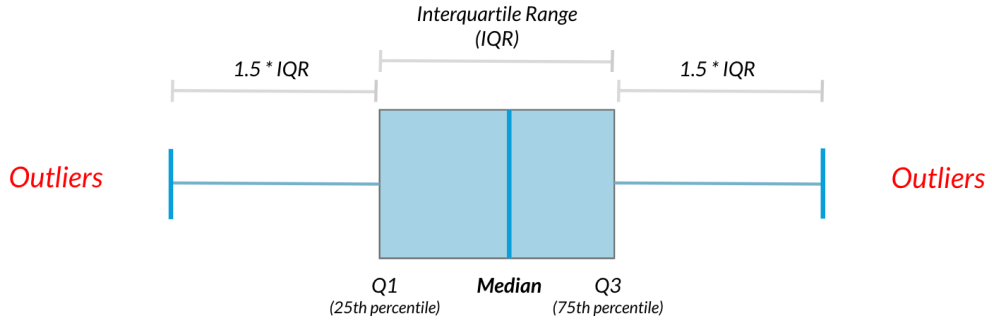


Figure 4.9: Outlier detection using chosen method

   This approach within the *explore_data* method calculates the IQR as a measure of statistical dispersion and identifies outliers as data points that fall signifi-

cantly outside the interquartile range, thus enabling a robust outlier detection mechanism.

- **Variance Thresholding** - I employed the *VarianceThreshold* class from *sklearn.feature_selection* in the *explore_data* method. This technique enabled me to filter out features with variance below a predetermined threshold, ensuring that only the most relevant features, which show sufficient variability, are retained for model training.
- **Values Scaling** - For scaling numerical features in the dataset, I utilized a flexible and dynamic approach within the *apply_scaling* method of my *FeatureEngineeringCLI* class. Depending on the dataset's characteristics and the chosen machine learning model, different scaling techniques such as *StandardScaler*, *MinMaxScaler*, and *RobustScaler* from *sklearn.preprocessing* were applied. The choice of scaler is determined based on the presence of outliers and the distribution of the data.

2. **Feature Selection**:

   - **Identification of Time-Dependent Features** - In the *perform_feature_engineering* method, I manually identified features that exhibited dependency on time. This was done by examining features with timestamps or durations, recognizing that such temporal aspects can significantly influence the behavior and characteristics of the data.
   - **Calculation of Feature Importance** - Using the *feature_importances_* attribute of an XGBoost model in the *perform_feature_engineering* method, I quantified the importance of each feature. This model-based approach helped in identifying features that most significantly contribute to the predictive power of the model.
   - **Recursive Feature Elimination (RFE)** - The *perform_RFE* method incorporated Recursive Feature Elimination using the RFE technique from *sklearn.feature_selection*. This process involved iteratively building a model and choosing the best or worst performing feature, setting it aside, and then repeating the process with the rest of the features. This technique was instrumental in refining the feature set to include only the most impactful features.

The preprocessing step would be implemented as a command-line interface (CLI) script using poetry to facilitate ease of use, including package installation.

Because of their appropriateness for the desired purpose, parquet files would be the preferred input format. Following the completion of the program run, the user would be given the choice to make the proposed adjustments. If you select this option, a new parquet file will be created in accordance with the recommendations. This method will entail the removal of features that are constant, time-dependent, or have no obvious benefits, as well as outliers. It will also handle any missing values in the data.

A uniform format of a parquet file is created, denoted as NDF (normalized domain-name format, see Figure 4.8). This format is used by all classification models, ensuring consistency and accuracy in the results.

4.4 Handling Categorical Features

As part of the preprocessing phase, it was necessary to address the issue of categorical variables. After exploring various approaches, including innovative ones, that were considered suitable for our intended purposes, it became apparent that it would be beneficial to dedicate a separate section to this aspect of the preprocessing process.

Our dataset comprises various features that significantly impact the model's accuracy. This section explores the original strategy and the modifications that were made to address categorical features within our datasets: `geo_continent_hash`, `geo_countries_hash`, `rdap_registrar_name_hash`, `tls_root_authority_hash`, `lex_tld_hash` and `tls_leaf_authority_hash`.

4.4.1 Initial Strategy for Categorical Features

The initial preprocessing strategy involved a differentiated approach towards each categorical feature based on its characteristics and the expected impact on the classification model.

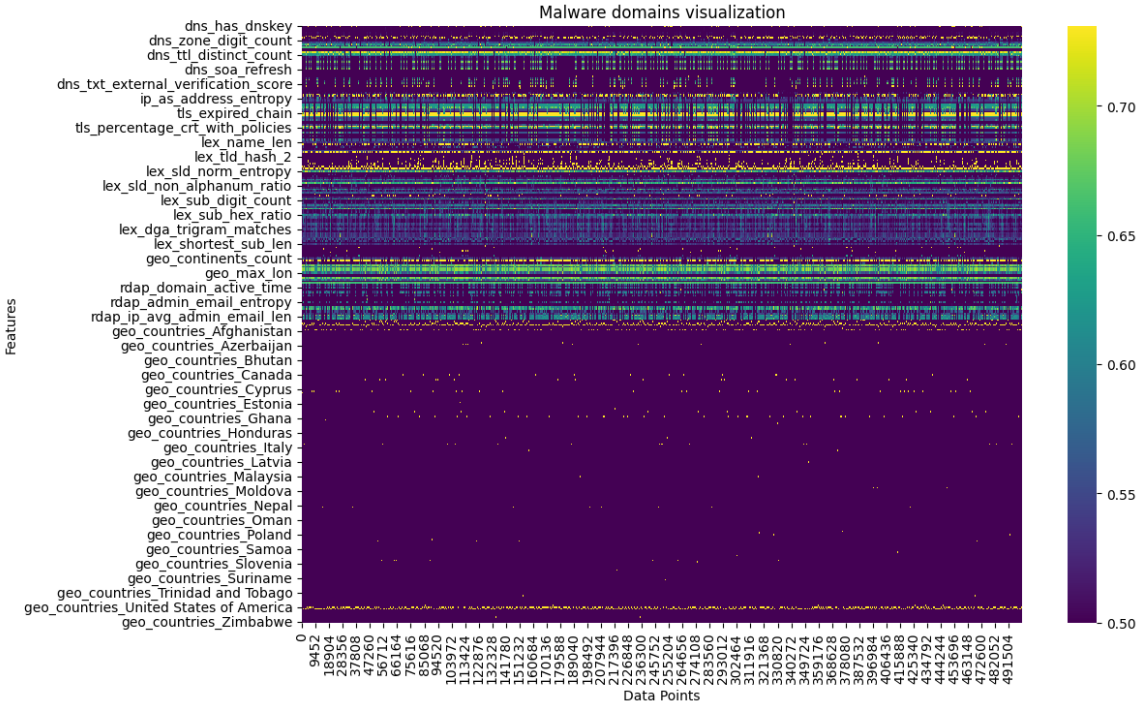
- **One-Hot Encoding for Geographic Features** - The `geo_continent_hash` and `geo_countries_hash` features were initially processed using one-hot encoding. This method was chosen to transform these categorical variables into a format that could be provided to ML models to better understand the geographic distribution of domains. The rationale behind this choice was to capture the geographical variance, presuming that certain continents or countries might exhibit higher tendencies towards hosting malign domains.
- **Binary Encoding for `lex_tld_hash`** - Given its considerably higher cardinality, the `lex_tld_hash` feature, representing the top-level domain hashes, was encoded using binary encoding. This technique was preferred over one-hot encoding to mitigate the dimensionality explosion, thereby reducing the feature space complexity while retaining significant information about the top-level domain's influence on a domain's malignancy.
- **Dropping `tls` and `rdap` Features** - The `tls_root_authority_hash`, `tls_leaf_authority_hash`, and `rdap_registrar_name_hash` features were excluded from the model due to their extensive unique values. Including these features would have led to an overwhelming increase in dimensions when encoded, which could potentially dilute the model's effectiveness and increase computational complexity without a commensurate gain in predictive power.

4.4.2 Reevaluation and Strategy Adjustment

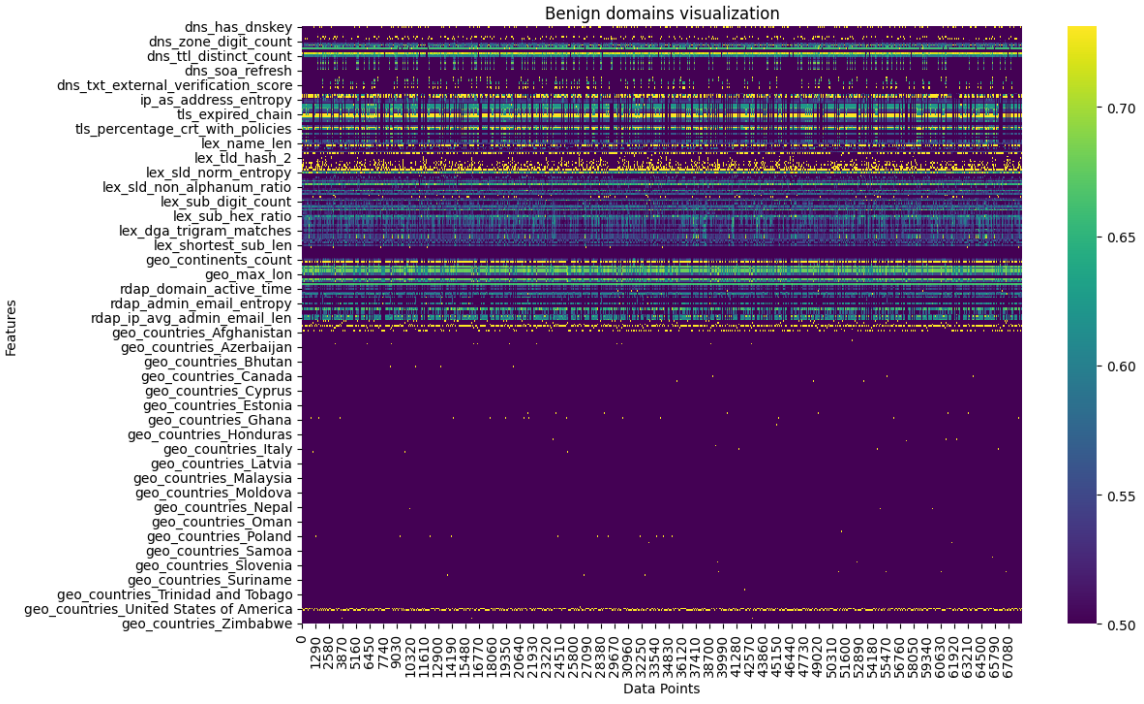
Upon further analysis, the initial strategy's effectiveness was reassessed, leading to significant adjustments in handling categorical features.

- **Ineffectiveness of One-Hot Encoding for Geographic Features**: It was discovered that the one-hot encoding applied to geographic features was not optimal. The lack of variance in the `geo_continent_hash` and `geo_countries_hash` features meant that these one-hot encoded vectors were almost static, offering little to no

predictive value to the model. This low variance indicated that the geographical location, as captured by these features, did not significantly influence the likelihood of a domain being malign. This can be spotted in the following figures 4.10a4.10b.



(a) Visualization of malign domains after categorical encoding



(b) Visualization of benign domains after categorical encoding

Figure 4.10: Visualizations of domain categorizations after encoding

- **Integrated Feature Creation Using Decision Tree**: To address the limitations observed with the initial approach, a novel strategy was adopted. Rather than treating each categorical feature separately, a decision tree was utilized to synthesize all categorical features into a single, comprehensive feature. This method leveraged the decision tree's ability to capture nonlinear relationships and interactions between features, thus creating a more potent and informative feature that encapsulates the essence of all categorical data. This integrated feature approach aimed to enhance model performance by focusing on the most informative aspects of the categorical data, reducing dimensionality, and simplifying the model's structure. The diagram depicting the architecture of the decision tree is presented in the Figure 4.11 below.

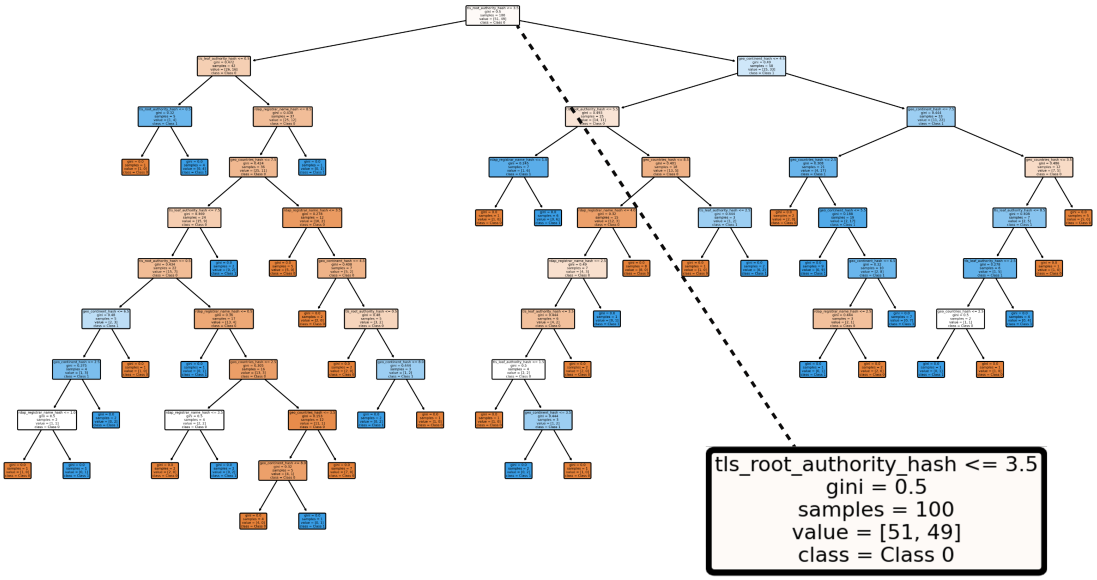


Figure 4.11: Creating new feature using decision tree

One of the most significant advantages of decision trees is their ability to describe nonlinear connections and interactions between features without the need for considerable data preparation. After training, the decision tree model was used to estimate the likelihood of each case in the dataset falling into the positive class (referred to as class 1). This probability was calculated using the decision tree classifier's `predict_proba` method, which yields probabilities for all classes. The positive class's probability was then extracted and added as a new feature, `dtree_prob` back to the dataset. This new feature summarizes the model's decision-making of the chance that a particular instance fits into the positive category, using the information provided in the categorical features. The details regarding the implementation are included in the following chapter (see Section 5.4).

Decision trees are notable for their ability to intuitively capture complicated connections between attributes, often with no data preparation. In the subsequent implementation chapter (see Section 5.4), this study examines the usefulness of merging decision tree models with previously discussed encoding strategies, which may improve prediction accuracy.

4.5 Improved Hyperparameters Tuning

Tuning hyperparameters, as seen in Figure 4.12, is an important step in building machine learning and deep learning models. This method involves identifying the best values for the model's parameters that were not learned directly from the training data. Hyperparameters are preset prior to the training phase of various classification algorithms. At first, I intend to use conventional methods like grid search and random search, which are comprehensive but computationally expensive and time-consuming. To potentially improve model performance and reduce tuning costs, I am going to explore more novel approaches like Bayesian optimization and Keras Tuner. These sophisticated algorithms are intended to efficiently navigate the hyperparameter field, potentially yielding better outcomes by responding more dynamically to training feedback [71, 67].

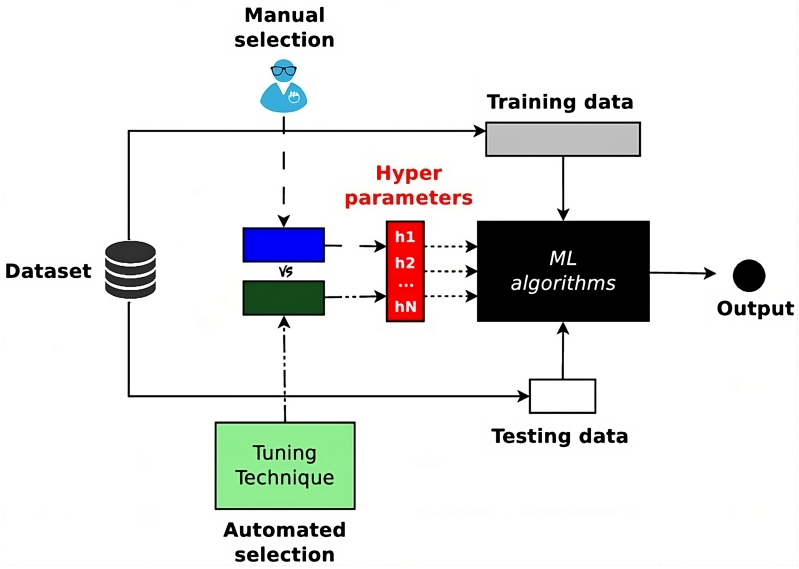


Figure 4.12: General scheme of hyperparameter tuning process [51]

This section advances into hyperparameter tuning strategies for XGBoost, Convolutional Neural Networks (CNNs), and Support Vector Machines (SVMs), all of which have produced promising results in domain classification tasks.

XGBoost Hyperparameters:

For XGBoost, I plan to use a Bayesian optimization framework that directly models the function space of hyperparameters using a Gaussian Process. This probabilistic model estimates hyperparameter performance (e.g., cross-validation accuracy) prior to actual evaluations and dynamically modifies its assumptions based on the observed outcomes [67]. XGBoost parameters to be tuned:

- **Learning Rate (eta)**: Influences the step size shrinkage used to prevent overfitting. Typically set between 0.01 and 0.3.
- **Max Depth**: Controls the maximum depth of a tree. Values usually range from 3 to 10.

- **Subsample**: Determines the fraction of samples to be used for each tree. Values less than 1 can lead to a reduction of overfitting.
- **N_estimators**: Number of trees in the ensemble. More trees can increase accuracy but also computational cost. Usually, a few hundred to a few thousand.
- **Alpha** (L1 regularization term on weights): Increases the model's regularity and reduces overfitting by adding a penalty on the magnitude of the coefficients. Typically ranges from 0 to 1.
- **Gamma** (Minimum loss reduction required to make a further partition on a leaf node): Controls whether a node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Suggested values range from 0 to 0.5.
- **Objective**: Specifies the learning task and the corresponding objective function, for example, 'binary:logistic' for binary classification.

CNN Hyperparameters:

For CNNs, I intend to use Keras Tuner[3], which combines Hyperband and Bayesian optimization approaches. This dual method not only accelerates the search for the ideal configuration but also assures that the tuning process is less prone to local minimums [71]. CNN parameters to be tuned:

- **Number of Layers** and **Kernel Size**: Both parameters will be tuned concurrently with a strategy that evaluates network depth and kernel sizes in tandem, optimizing for both feature extraction capabilities and computational efficiency.
- **Activation Function**: Different activation functions will be tested dynamically based on their performance in earlier layers versus deeper layers, using a tiered approach within the tuner to adaptively select the best function for each layer.
- **Pooling Size**: Pooling parameters will be optimized to balance between effective feature compression and loss of important spatial information, with real-time adjustments based on the complexity of features being processed.
- **Batch Size and Learning Rate**: These parameters will be tuned using a predictive model that estimates optimal batch sizes and learning rates based on training dynamics observed in initial epochs, allowing for adaptive learning rate schedules and batch sizes that evolve based on model behavior.

SVM Hyperparameters:

To optimize Support Vector Machines (SVMs), the method of Bayesian optimization will be employed to fine-tune the kernel type, C, and gamma:

- **Kernel Type**: The choice between type of kernels affects the decision boundary. Kernels define the transformation space approach. Types include:
  - Linear,
  - Polynomial,

[3] https://keras.io/

    - Radial Basis Function (RBF),
    - Sigmoid.
- **C (Regularization Parameter)**: Balances the trade-off between achieving a low error on the training data and minimizing the norm of the weights.
- **Gamma**: In the RBF kernel, it defines the influence of a single training example, with low values meaning 'far' and high values meaning 'close'.

Each hyperparameter plays a crucial role in the configuration of the model, influencing various aspects such as complexity, speed of training, and the model's ability to generalize. The optimal values for these hyperparameters were determined through a series of experiments, utilizing techniques such as grid search and cross-validation to systematically evaluate their effects on model performance (see Section 5.2).

4.6 Designing Experiments for Class Balancing

Class imbalance is a common problem in machine learning, particularly for models trained on datasets in which some classes are severely undervalued. This mismatch can result in biased models that perform badly on minority groups, affecting predictive modeling's overall effectiveness and fairness. To address this, I designed experiments to investigate several class balancing techniques, evaluating their impact on model performance, and determining the most successful ways for our specific needs and different clasifiers.

The major objective is to improve predicted accuracy and fairness in models coping with imbalanced classes. I intend to experiment with various ways to find the best balance between recognizing minority class instances and preserving overall model accuracy.

Experimentation with Weighting Techniques

One of the first techniques we will explore is the adjustment of class weights. This involves assigning a higher weight to the minority class in the loss function of the model, thereby increasing the cost of misclassifying the minority class. Techniques such as the `scale_pos_weight` parameter in XGBoost and similar parameters in SVM will be utilized. The effectiveness of these weighting strategies will be measured by their ability to improve minority class recognition without compromising overall accuracy.

Synthetic Sample Generation

In addition to weighting techniques, I will employ synthetic sample generation methods like SMOTE (Synthetic Minority Over-sampling Technique). SMOTE works by creating synthetic samples from the minority class to balance the class distribution.

Synthetic Sample Generation

To improve the class distribution even further, I want to use advanced synthetic sample creation techniques. While SMOTE is frequently used, I intend to give it a unique twist by combining it with selective undersampling of the majority class, resulting in a dual-strategy that not only increases the presence of the minority class but also refines the majority class to eliminate potential noise and overfitting. This hybrid technique, I would call it „Balanced-SMOTE," seeks to optimize the training dataset not only by adding synthetic examples, but also by removing redundant or less informative samples from the majority class. This might be done using different approaches, for example incorporating k-means to cluster data within each class and removing those far from cluster centroids, I would prefer removing outliers using IRQ method described earlier (see Section 2.2.1).

Comparative Analysis

Each method's impact on model performance will be evaluated using metrics such as precision, recall, F1 score, and the overall accuracy. Special attention will be paid to the False Positive Rate (FPR), as an increase in this metric could indicate overfitting to the minority class or to synthetic noise introduced by oversampling techniques.

Chapter 5

Implementation of Optimization Strategies for Classification Models

This chapter is dedicated to the practical implementation of theoretical optimization techniques outlined in previous chapters of this thesis. Following extensive investigation and analysis of existing categorization models, as well as the discovery of prospective improvements, this chapter describes the processes required to incorporate these advancements. The goal here is to convert breakthroughs in theory into actionable, empirical discoveries that significantly enhance machine learning models' capacity to detect malicious domains effectively.

The findings in this chapter are intended to offer a solid foundation for future study and practical applications in this specific cybersecurity field.

Since the original code is written in `Python language`[1], I will also use it to implement the follow-up scripts. The Python version used is 3.11. For the dependency management and packaging in the project, I chose `Poetry`[2].

5.1 Ground-truth Improvement With Domain Examination Pipeline

The Domains Examination Pipeline, as implemented in the „vt_checker.ipynb“ jupyter notebook file, constitutes a critical component of our cybersecurity research framework, particularly focusing on the analysis of domain characteristics to distinguish between benign and malign entities. This pipeline leverages asynchronous programming paradigms and integrates with the VirusTotal (VT) API, offering a comprehensive approach to domain examination.

A rigorous process was used to create a credible ground truth benign dataset, called `benign_2312`. In the beginning, CESNET provided us with a dataset of domains that were considered to be benign but needed validation. To verify the dataset's accuracy

[1] https://www.python.org/
[2] https://python-poetry.org/

and validity, a systematic approach was established that entailed an almost automatic creation pipeline depicted in Figure 5.1.

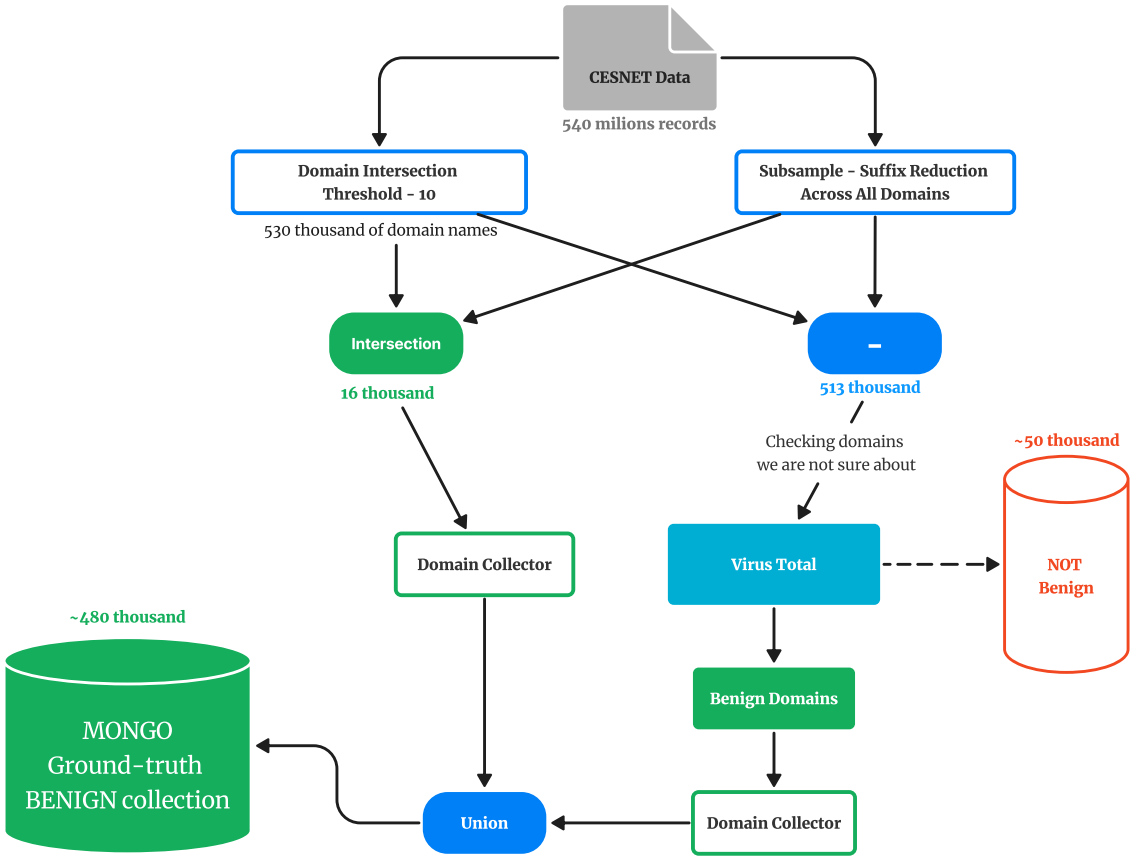


Figure 5.1: Pipeline of creating whole new ground-truth benign dataset

First, a domain intersection approach was used to identify domains that appeared often in CESNET data over several months. A threshold was established, limiting further investigation to domains that had been visited at least ten times within the specified timeframe, as these were thought to be more innocent. Simultaneously, suffix reduction was used to remove duplicates and related domains from the dataset.

Domains that were not included in the intersection procedure require extra verification. As mentioned earlier, I was able to connect with VirusTotal, a prominent cybersecurity organization, gaining access to their academic API. This allowed us to reliably validate up to 20,000 domains each day and successfully distinguish between malicious and benign domains.

By combining the original subset with the VirusTotal results, a final dataset was created that was comprehensive and reliable. The methodical approach and use of VirusTotal's API ensured that the finalized dataset was ground truth, therefor suitable for our later use in the FETA project group.

5.1.1 DomainAnalyzer Class

The pipeline in Figure 5.1 begins with the initialization of the Python environment for asynchronous execution, ensuring that the domain analysis can proceed efficiently without blocking operations. This setup is crucial for handling network requests to the VT API, which may have variable response times depending on the network latency and the API server's current load.

At the core of the pipeline is the `DomainAnalyzer` class, designed to encapsulate the functionality needed for querying domain information from VT, processing the responses, and analyzing the domains based on various security indicators. This class abstracts the complexity involved in making asynchronous API requests and parsing the returned data, thereby providing a streamlined interface for domain examination.

```
class DomainAnalyzer:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://www.virustotal.com/api/v3/domains/"
        # Initialize session for asynchronous requests

    async def fetch_domain_report(self, domain):
        """
        Asynchronously fetches the security report of the specified domain
        from VirusTotal's API.
        """
        url = f"{self.base_url}{domain}"
        headers = {"x-apikey": self.api_key}
        async with aiohttp.ClientSession() as session:
            async with session.get(url, headers=headers) as response:
                if response.status == 200:
                    return await response.json()
                else:
                    return None

    async def analyze_domain(self, domain):
        """
        Analyzes a~single domain by fetching its report and extracting
        key security indicators.
        """
        report = await self.fetch_domain_report(domain)
        # Process the report to extract security indicators
        # Example: Extracting malicious URL detections
        if report:
            detections = report['data']['attributes']['last_analysis_stats']
                                    ['malicious']
            return {'domain': domain, 'malicious_detections': detections}
        else:
            return {'domain': domain, 'malicious_detections': 'N/A'}

# Example usage:
# analyzer = DomainAnalyzer(api_key="YOUR_API_KEY")
# result = await analyzer.analyze_domain("example.com")
# print(result)
```

Listing 5.1: Code behind the domain verification process

5.1.2 Pipeline Operation

The operation of the pipeline is illustrated through an example usage scenario within the notebook, demonstrating how to instantiate the `DomainAnalyzer` class and invoke its analysis methods on a set of domains. This practical example underscores the pipeline's applicability in real-world research contexts, providing insights into both its implementation and usage.

```
# Example usage in a~Jupyter notebook cell:
analyzer = DomainAnalyzer(api_key="YOUR_VIRUSTOTAL_API_KEY")
domains = ["example.com", "malicious.com"]
for domain in domains:
analysis_result = await analyzer.analyze_domain(domain)
print(analysis_result)
```

5.1.3 Virus Total - Verification Box

The core component of the system architecture is represented by a blue box labeled as `Virus Total`. The functionality of this node has been explained previously (see Section 4.2). Now we'll go further into its implementation.

API Integration and Configuration

- Integrates with the `VirusTotal` API, a cornerstone for fetching domain-related data.
- Implements a method `_load_api_key` to securely fetch the API key from environmental variables.
- Establishes HTTP request headers through `_create_headers`, centralizing the API key and content type for subsequent requests.

```
def _load_api_key():
    load_dotenv()
    api_key = os.getenv('VT_API_KEY')
    if api_key is None:
        raise ValueError("API key is not set. Please set the VT_API_KEY.")
    return api_key

def _create_headers(self):
    return {"x-apikey": self.api_key, "Accept": "application/json"}
```

Listing 5.2: API Integration and Configuration

Domain Analysis Methods

- The `check_domain` method conducts the primary API call to `VirusTotal` for a given domain, handling HTTP responses and errors.
- Employs `_determine_verdict` to classify domains into 'Malign' or 'Benign' based on analysis statistics like the count of malicious and suspicious flags.

– Includes a method `_is_domain_live` to ascertain the current operational status of a domain, augmenting the analysis with live/dead status.

```
def check_domain(self, domain: str) -> Optional[dict]:
    url = f"https://www.virustotal.com/api/v3/domains/{domain}"
    response = requests.get(url, headers=self.headers)
    if response.status_code == 200:
        return response.json()
    elif response.status_code == 429:
        print(f"Quota exceeded")
        return "Quota Exceeded"
    else:
        print(f"Error: Unable to fetch information for {domain}.{response.text}")
        return None

def _determine_verdict(self, analysis_stats: dict) -> str:
    return "Malign" if analysis_stats.get('malicious', 0) > 0
                    or analysis_stats.get('suspicious', 0) > 1 else "Benign"
def _is_domain_live(self, domain: str) -> str:
    try:
        result = subprocess.run(['./livetest.sh', domain], capture_output=True,
                text=True)
        return "Alive" if result.stdout.strip() == '1' else "Dead"
    except Exception as e:
        print(f"Error: Unable to check if domain {domain} is live. {e}")
```

Listing 5.3: Domain Analysis Methods

Data Extraction and Formatting

– `extract_domain_data` method parses and extracts relevant data from the API response, including the domain's verdict, detection ratio, and last analysis timestamp.

– A utility method `_format_timestamp` converts UNIX timestamps into human-readable date and time formats.

```
def extract_domain_data(self, domain: str, result: dict) -> Optional[Tuple]:
    try:
        attributes = result['data']['attributes']
        analysis_stats = attributes['last_analysis_stats']
        verdict = self._determine_verdict(analysis_stats)
        detection_ratio = f"{analysis_stats['malicious']}/
        {analysis_stats['malicious'] + analysis_stats['harmless']}"

        last_analysis_date = attributes.get('last_analysis_date', attributes.get
                            ('last_submission_date', 0))
        formatted_timestamp = self._format_timestamp(last_analysis_date) if
                            last_analysis_date else 'N/A'

        domain_status = self._is_domain_live(domain)
        return (domain, verdict, detection_ratio, formatted_timestamp,
                analysis_stats.get('harmless', 0), analysis_stats.get
                ('malicious', 0), analysis_stats.get('suspicious', 0), domain_status)
```

Listing 5.4: Data Extraction and Formatting

Live Status

- Implementation incorporates a shell script (`livetest.sh`) to determine the live status of each domain.
- This script utilizes a simple ping command to assess if a domain is operational ('Alive') or not ('Dead'), enhancing the depth of the analysis.

```
ping -c 1 -W 1 $1 > /dev/null 2>&1 && echo 1 || echo 0
```

Data Management and Reporting

- Methods `load_previous_data` and `save_data` facilitate the storage and retrieval of analyzed domain data, supporting CSV file formats.
- The `save_checkpoint` function merges new analysis data with existing records, ensuring data integrity and continuity.
- `generate_report` generates comprehensive PDF reports from the analyzed data, incorporating pagination and summary statistics.

Batch Processing and Domain Selection

- The method `process_selected_domains` orchestrates the overall process, handling domain batch processing based on predefined limits to comply with API call quotas.
- Accommodates different input modes (e.g., 'txt', 'parquet') for domain name sources, enhancing the module's flexibility.
- Implements progress tracking and quota management, ensuring efficient and responsible use of API resources.

Report Generation

- The script efficiently calculates the count of benign and malign domains, integrating this summary into the DataFrame.
- Utilizes Matplotlib to generate a visually engaging report, represented as a styled table (see Figure 4.7).

Data Transformation and Transfer Script

Given the historical nature of the domain data, a Python script was created to facilitate the transformation of old collected data into a new MongoDB collection. This script plays a crucial role in ensuring the data's relevance and usability in ongoing research.

The script initiates by establishing a connection to a MongoDB instance through a specified URI. Subsequently, it proceeds to retrieve domain names from a designated file.

Regarding the data transfer process, it delineates between source and destination collections within the database `drdb`. The iteration process encompasses querying the

source collection for associated records corresponding to each domain name. Subsequently, it proceeds to insert each retrieved record into the destination collection.

5.2 Strategies for Parameter Selection

This chapter explores optimization strategies for three machine learning algorithms: XGBoost, Neural Networks, and Support Vector Machines (SVM). Each algorithm provides a unique collection of parameters, and the ideal configuration is critical to attaining the model's full potential in a variety of tasks. For a thorough evaluation and optimization of these parameters, the implementations are separated into three independent Python scripts: `params_tuning_XGBoost.py`, `params_tuning_CNN.py`, and `params_tuning_SVM.py`. These scripts serve as a foundation for our investigation, presenting a systematic way to determine the most optimal approaches for parameter adapting across different models.

5.2.1 XGBoost

The optimization of XGBoost parameters is critical for harnessing the full potential of this powerful algorithm. In the `params_tuning_XGBoost.py` Python script, three prevalent approaches for hyperparameter tuning are implemented: Random Search, Grid Search, and Bayesian Optimization using Hyperopt[3]. Each method offers a different strategy for exploring the hyperparameter space, balancing the trade-off between computational efficiency and the thoroughness of the search. All approaches need to first iniciate the space of hyperparameters to search. Params are almost the same between all these approaches with minor changes shown in the Listing 5.5.

```
# Parameters
params_space_random_search = {
    "max_depth": [6, 9, 12],         # maximum distance between the root and the leaf
    "eta": [0.1, 0.15, 0.2],         # learning rate
    "min_child_weight": [1.0, 2.0], # minimum sum of instance weight in a~child
    "subsample": [0.6, 0.8, 1.0],    # subsample ratio of the training instances
    "alpha": [0, 0.1, 0.5],          # L1 regularization
    "gamma": [0.1, 0.2, 0.3],        # loss reduction required to make a~further
                                     # partition on a~leaf
    "lambda": [1.0, 1.5],            # L2 regularization
    "n_estimators": [100, 200],      # number of trees
    "colsample_bytree": [0.8, 1.0], # subsample ratio of columns
    "max_bin": [256, 512]            # maximum number of bins that feature values will
                                     # be bucketed in
}

params_space_bayes_optimalization = {
    'max_depth': scope.int(hp.quniform('max_depth', 6, 12, 3)),
    'learning_rate': hp.choice('learning_rate', [0.1, 0.15, 0.2]),
    'min_child_weight': hp.choice('min_child_weight', [1.0, 2.0, 3.0]),
    'subsample': hp.choice('subsample', [0.6, 0.8, 1.0]),
    'alpha': hp.choice('alpha', [0, 0.1, 0.5]),
    'gamma': hp.choice('gamma', [0.1, 0.2, 0.3]),
    'lambda': hp.uniform('lambda', 1.0, 1.5),
```

[3] https://hyperopt.github.io/hyperopt/

```
    'n_estimators': hp.choice('n_estimators', [100, 200]),
    'colsample_bytree': hp.choice('colsample_bytree', [0.8, 1.0])
}

params_space_grid_search = {
    #same as random search
}
```

Listing 5.5: Parameters space initialization

Random Search

Random Search works on the assumption of random sampling from a specific hyperparameter field. This strategy, while possibly less efficient than systematic grid searches, can frequently result in near-optimal designs with substantially less rounds. The Random Search is used in the script to choose hyperparameter combinations at random, such as the learning rate, number of estimators, and max depth, and then evaluate the model's performance for each combination.

```
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, StratifiedKFold

xgb = XGBClassifier(
    objective="binary:logistic",
    eval_metric=["error", "logloss", "auc"],
    grow_policy="lossguide",
    max_delta_step=0,
)

# Cross-validation strategy
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Search across 100 different combinations, and use all available cores
random_search = RandomizedSearchCV(
                estimator=xgb, param_distributions=param_distributions,
                n_iter=100, n_jobs=-1, cv=kfold, scoring='f1', verbose=3,
                random_state=42)

# Fit the random search model
random_search.fit(X_train, y_train)

print(f"Best parameters found: {random_search.best_params_}")
```

Listing 5.6: Random Search

Grid Search

Grid Search takes a more extensive technique, evaluating every conceivable combination of hyperparameters inside a given grid. Despite its computational complexity, this technique ensures the identification of the best design within the grid restrictions.

Hyperopt with Bayesian Optimization

Bayesian Optimization via Hyperopt is a complex method that uses probability models to inform the search for the best hyperparameters. This approach seeks to reduce the number of required assessments by foreseeing the performance of various hyperparameter sets and strategically selecting the next set to examine. In the script, a defined search space for hyperparameters such as learning rate and max depth is sent into Hyperopt, which then uses Bayesian optimization to effectively determine the combination that produces the best model performance.

The Table 5.2 below summarize the parameters obtained from each optimization method used in the study. It is worth noting that the generated parameters vary minimally across the various techniques, showing a convergence towards a similar solution space, which strengthens the model's architecture's durability with regard to the data available.

In terms of time required for processing, Random Search was the most efficient technique, followed by Bayesian Optimization and, finally, Grid Search, which took the biggest amount of the computation time.

Optimization Method	Max Depth	Learning Rate (eta)	Min Child Weight	Subsample
Random Search	6	0.15	2.0	0.8
Bayesian Opt.	6	0.1	3.0	0.9
Grid Search	9	0.2	2.0	0.8

Table 5.1: Best parameter combinations returned by each approach

Optimization Method	Alpha	Gamma	Lambda	Max Bin	N Estimators	Col Sample
Random Search	0	0.2	1.0	256	300	0.8
Bayesian Opt.	0.05	0.1	1.57	N/A	200	0.8
Grid Search	0.1	0.1	1.5	256	300	0.8

Table 5.2: Best parameter combinations returned by each approach

The optimal method for tuning xgboost parameters is discussed in the subsequent chapter (see Section 6.3).

5.2.2 Convolutional Neural Networks

The architecture of a Convolutional Neural Network (CNN) can have a significant influence on its performance when optimizing hyperparameters. Unlike classic machine learning models like XGBoost, which are primarily concerned with modifying hyperparameters like learning rate and tree properties, CNNs provide an additional advantage by permitting both architectural and hyperparameter adjustments throughout the optimization process. This flexibility is critical in deep learning, where the network's depth (number of layers), kernel size in convolutional layers, activation function type, and even the inclusion of advanced methods such as dropout or batch normalization may all have a significant impact on outcomes.

Hyperparameter Optimization Strategy

The Python script in the Appendix (see Subsection D.1.1) shows CNN basic architecture. Following code (see Subsection D.1.2) is then used to create a dynamic CNN with hyperparameters optimized using Keras Tuner, a sophisticated hyperparameter tuning framework. Process of CNN tuning is depicted in Figure 5.2.

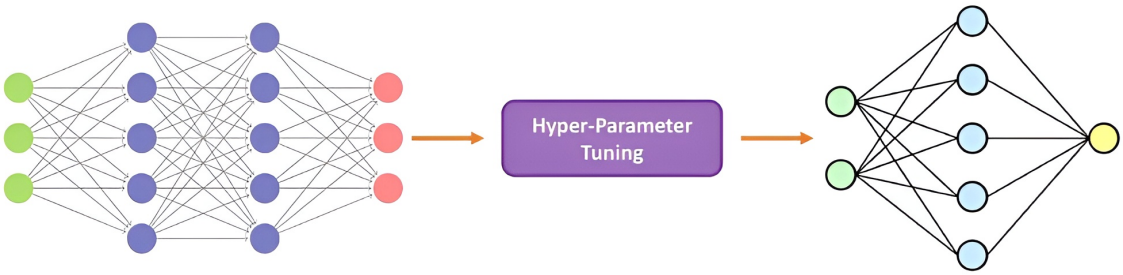


Figure 5.2: CNN achitecture tuning [63]

The output of performing tuning using Keras is shown below in the Figure 5.3.

```
Trial 3 Complete [00h 09m 28s]

val_loss: 0.06313413381576538
Best val_loss So Far: 0.01708251893520355
Total elapsed time: 00h 21m 52s

 The best number of convolutional layers is 3
The best optimizer is Adam
The best number of filters is 128
The best kernel size is 5
The best number of units is 96
The best dropout rate is 0.1
The best learning rate is 0.00023899521618106246

Epoch 1/10
3435/3435 [========================] - 4:10 4s/step - accuracy: 0.9321
Epoch 2/10
3435/3435 [========================] - 1:18 23ms/step - accuracy: 0.9620
Epoch 3/10
3435/3435 [========================] - 1:37 29ms/step - accuracy: 0.9603
Epoch 4/10
3435/3435 [========================] - 1:19 23ms/step - accuracy: 0.9608
...and so on...

Maximum Training F1 Score: 0.9786
Maximum Validation F1 Score: 0.9620
```

Figure 5.3: Model training output showing the loss, accuracy and F1 score across epochs.

In contrast to methods like Hyperopt used in simpler models, Keras Tuner allows for a wide range of tunable parameters such as the number of convolutional layers, filter

sizes, and dense layer configurations, allowing you to tailor the network architecture directly to the needs of the dataset.

In this method, the network design is not predetermined. Each attempt in the tuning process might suggest a different architecture by varying the number of layers, layer types (e.g., convolutional or pooling), and layer sequencing. This is especially helpful since it incorporates architectural discovery into the optimization process, resulting in novel designs customized for best performance on a specific job. For example, the script configures the CNN to possibly change the number of convolutional layers, use alternative activation functions, and adjust pooling and dropout rates based on validation performance indicators.

Another option is to use Hyperopt framework to dynamically adapt the architecture based on hyperparameters such as the number of layers and kernel size. This approach enables a thorough review of many configurations to determine the best structure for the specific dataset being utilized.

```
space = {
'num_layers': hp.choice('num_layers', [2, 3, 4]),
'kernel_size': hp.choice('kernel_size', [3, 5]),
'activation': hp.choice('activation', [F.relu, F.sigmoid]),
}

def objective(params):
model = Net(**params)
optimizer = Adam(model.parameters(), lr=0.001)
return train_and_evaluate(model, optimizer)

best_params = fmin(fn=objective, space=space, algo=tpe.suggest, max_evals=50)
print("Best hyperparameters:", best_params)
```

Listing 5.7: Bayesian optimization

The CNN hyperparameter tuning method found that deeper networks with smaller kernel sizes outperformed in the investigated tasks. Specifically, the inclusion of ReLU activation functions was shown to improve the learning process, most likely due to its capacity to alleviate the vanishing gradient problem in deep networks.

The following chapter will look at the particular effects of these modifications and how they affect the model's performance across different datasets. This investigation seeks to offer clear insights into how different setups affect CNN accuracy and efficiency in actual applications.

5.2.3 SVM

In this section, I will highlight the implementation of hyperparameter optimization for chosen classification techniques.

The primary hyperparameters in SVM that require meticulous tuning to enhance model performance include the regularization parameter, kernel type, and kernel coefficient (see Section 4.5).

To determine the most effective SVM kernel for our specific task, I created python script to evaluate four commonly used kernels: linear, polynomial (poly), radial basis function (RBF), and sigmoid. Simplified script is described below in Listing 5.8.

```python
# Define the kernels to test
kernels = ['linear', 'poly', 'rbf', 'sigmoid']

# Initialize dictionary to store performance metrics for each kernel
performance_metrics = {kernel: {} for kernel in kernels}

# Evaluate each kernel using SVC
for kernel in kernels:
    print(f"Evaluating {kernel} kernel...")
    svc = SVC(kernel=kernel)
    svc.fit(x_train, y_train)  # Fit the model to the scaled training data
    y_pred = svc.predict(x_test)  # Predict using the scaled test data

    # Calculate and store the performance metrics
    accuracy, precision, recall, f1, conf_matrix = ...
    performance_metrics[kernel]['accuracy'] = accuracy
    ...
    performance_metrics[kernel]['confusion_matrix'] = conf_matrix

# Determine the best kernel based on F1 score
best_k = max(performance_metrics, key=lambda k: performance_metrics[k]['f1_score'])
```

Listing 5.8: Kerner choosing strategy

Each kernel was tested using the same training and test datasets to ensure consistency in the evaluation conditions. We utilized the scikit-learn library's `SVC` class to train an SVM with each kernel and then measured the accuracy, precision, recall, F1 score, and confusion matrix of the predictions. The results of this evaluation will be presented in the following chapter (see Figure 6.8).

Based on the results from these evaluations, we identified the kernel that provided the best balance between precision and recall as indicated by the F1 score. Subsequent to selecting the most effective kernel, we engaged in a thorough hyperparameter tuning process. This involved the definition and testing of three distinct parameter grids: an optimal grid, a wide range grid, and a default grid.

The **optimal grid** was designed to focus on a narrow range of hyperparameters around values that had previously shown promise in experimental trials. This grid included parameters such as:

- `'C': [30, 37, 42, 47, 52, 58]` – centered around the value of 50, allowing us to fine-tune the regularization strength.
- `'gamma': [0.8, 0.9, 1, 1.2, 1.35, 1.5]` – adjusted around 1 to optimize the influence of individual support vectors.
- `'kernel': ['rbf', 'poly']` – incorporating both the Radial Basis Function and polynomial kernels to compare performance within focused parameters.

The **wide range grid** aimed to cover a broad spectrum of hyperparameters to ensure robustness across various data characteristics. This grid extended the exploration to include:

- `'C': [0.1, 1, 10, 50, 100, 200, 500, 1000, 2000, 5000]` – offering a comprehensive assessment from very low to very high regularization values.

- `'gamma': [0.00015, 0.0015, 0.015, 0.075, 0.35, 0.85, 3, 7, 'scale', 'auto']` – ranging from extremely low to high values, along with automatic settings.
- `'kernel': ['rbf', 'poly', 'sigmoid']` – testing the effectiveness of sigmoid alongside RBF and polynomial kernels.

Lastly, the **default grid** provided a balanced approach for initial parameter exploration, useful for preliminary analysis:

- `'C': [10, 100, 1000]`
- `'gamma': [0.01, 0.1, 1]`
- `'kernel': ['rbf']`

These grids were meticulously tested through grid search coupled with cross-validation techniques to ascertain the most effective hyperparameter set for our SVM model, tailored to the specific requirements of the data. The selection of which grid to use for SVM parameter tuning was strategically made based on the desired depth of exploration and the computational resources available.

5.3 NDF - Preprocessing

This section is an extension of the section (see Section 4.3) discussed in the preceding chapter. In this section, I will elaborate on the preprocessing of domains to a suitable format that can be utilized as input for classification models. The aim is to provide a detailed account of the methodologies employed in this study.

The feature engineering and data preparation pipeline is a crucial component of our machine learning workflow, meticulously designed to enhance the model's performance by refining the input data. This section delineates the comprehensive approach adopted, leveraging both standard libraries and advanced machine-learning techniques.

The detailed code discussed in this section is showcased in the appendix. Please see code examples in Section D.2.

5.3.1 Environment Setup

The implementation leverages a mix of standard and third-party Python libraries, establishing a rich foundation for data manipulation, analysis, and model training. Key libraries include:

- `numpy`[4] and `pandas`[5] for data handling and computations,
- `matplotlib`[6] and `seaborn`[7] for data visualization,

[4] https://numpy.org/
[5] https://pandas.pydata.org/
[6] https://matplotlib.org/
[7] https://seaborn.pydata.org/

– `scikit-learn`[8] for machine learning models and preprocessing tools,
– `xgboost`[9] for gradient boosting models,
– `pyarrow`[10] for efficient IO operations with Parquet files.
– `joblib`[11] for saving and loading models.

5.3.2 Feature Engineering CLI

The `FeatureEngineeringCLI` class encompasses the whole pipeline, from loading and preprocessing data to training and validating decision trees. It has two modes - the first mode is used for creating, scaling, and saving models for outlier boundaries and training decision trees. This mode requires two parquet files, one for benign domains and the other for malicious domains. The models are saved using joblib. The second mode is used to preprocess individual domains in real time and classify them according to the FETA project specifications. It accepts a single domain record as a numpy dataframe. Accepting and processing a single domain can be later used in real-time classification depicted in Figure 5.4.

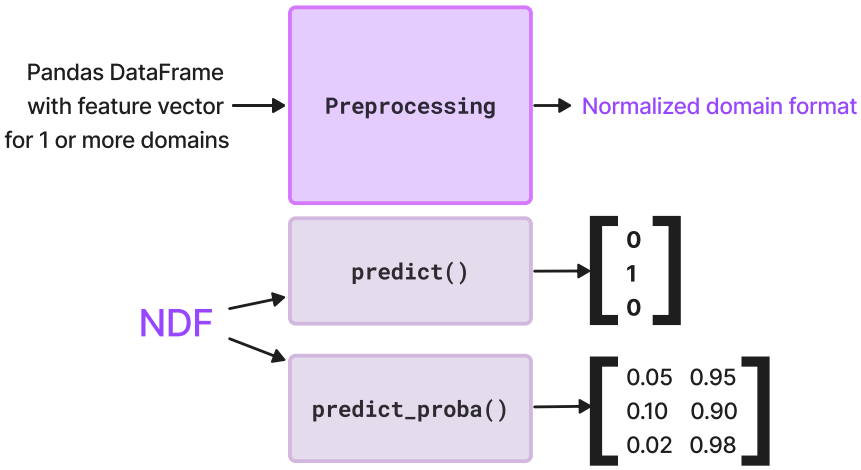


Figure 5.4: Process of class prediction for 1 domain, utilizing NDF processing

For easier orientation in code execution and data processing, I have created a custom logger, specified using the `configure_logger` method, which improves real-time monitoring and debugging by capturing crucial events throughout pipeline operation.

[8] https://scikit-learn.org/stable/
[9] https://xgboost.readthedocs.io/en/stable/
[10] https://arrow.apache.org/docs/index.html
[11] https://joblib.readthedocs.io/en/stable/

Data Preprocessing

Data preprocessing is a complex and multifaceted process that involves several vital steps to ensure that the data is suitable for machine learning models. The main steps involved in this process are:

1. Removal of non-training fields, ensuring focus on relevant features (see Listing D.3),
2. Application of categorical encoding Section 5.4 and scaling Listing D.5, leveraging methods like `OneHotEncoder` combined with `Decision tree` for optimal model input formatting,
3. Outlier detection and removal (see Listing D.4), employing a z-score-based approach for maintaining data integrity.

5.3.3 Operational Workflow

The `NDF` function, as presented in Listing D.6, serves as an orchestrator of a pipeline that coordinates several critical phases of a machine learning workflow. These phases include data loading, preprocessing, model training, and ultimately, dataset preparation. To ensure that the data is optimally prepared for consumption by the model, the function judiciously applies several preprocessing strategies, such as scaling and outlier removal.

The output of the function is a matrix of labels, features, and their associated values. This matrix is presented in a normalized domain format (NDF) and is notably suitable for use with convolutional neural network (CNN) classifiers. The use of the NDF format facilitates efficient and effective training and validation of these models.

An output example is presented in the Table 5.3. Dataset details highlight the structure and size of the feature set, with a total of 179 features analyzed over 372,792 observations. The feature distribution across classes is indicated by tensors reflecting the composition of the dataset, with the tensor sizes demonstrating the allocation between benign and phishing categories.

Label	**Feature_1**	**Feature_2**	**Feature_3**	...	**Feature_170**
0	0.517 850	0.527 749	0.522 712	...	Value
1	0.731 059	0.612 619	0.600 188	...	Value

Index: Label, Feature_1, ..., Feature_170
Shape: $(372,792 \times 171)$
Class Distribution: $(0 : 327,859; 1 : 44,933)$

Table 5.3: NDF Format

5.4 Encoding and Integrating Categorical Features

The decision tree model operates by assigning a specific purpose to each node in the classification process, revealing the algorithm's decision-making logic. The process

begins with the root node, which determines the most efficient feature and threshold for dividing the dataset into subgroups. Internal nodes then branch out from the root, dissecting the data based on additional characteristics and thresholds, each reflecting a decision criterion that refines the classification route. This hierarchical structure of binary decisions (see Figure 4.11) ultimately leads to the leaf nodes, representing the final categorization outcomes. Leaf nodes provide crucial information, including projected class, likelihood of class membership, sample count, and class distribution within the node. The comprehensive dissection of nodes, including decision criteria, class distributions, and purity measurements like the Gini index or entropy, highlights the decision tree's interpretability.

Upon comparing the features of benign and malignant domains, the generated decision tree feature, illustrated in the Figure 5.5, exhibits a discernible distribution. Significantly, the integrated feature, now dubbed `dtree_prob`, demonstrates higher values for malignant domains compared to benign ones.

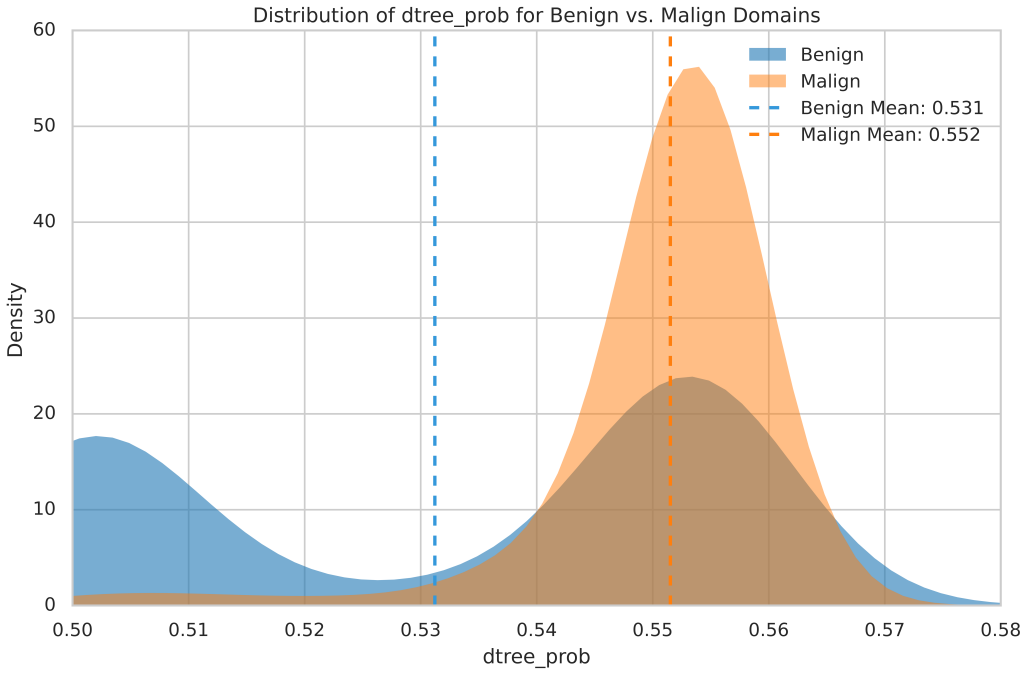


Figure 5.5: Distribution of values after using new feature

Building on the original success of the decision tree model, more research was conducted to improve its accuracy and generalizability. These new strategies included the use of ensemble techniques, which combine the strengths of numerous models to increase prediction resilience.

This multidimensional method resulted in a model that not only performed better numerically, as assessed by important criteria but also provided a sharper visual separation between classes, as seen in the Figure 5.6.

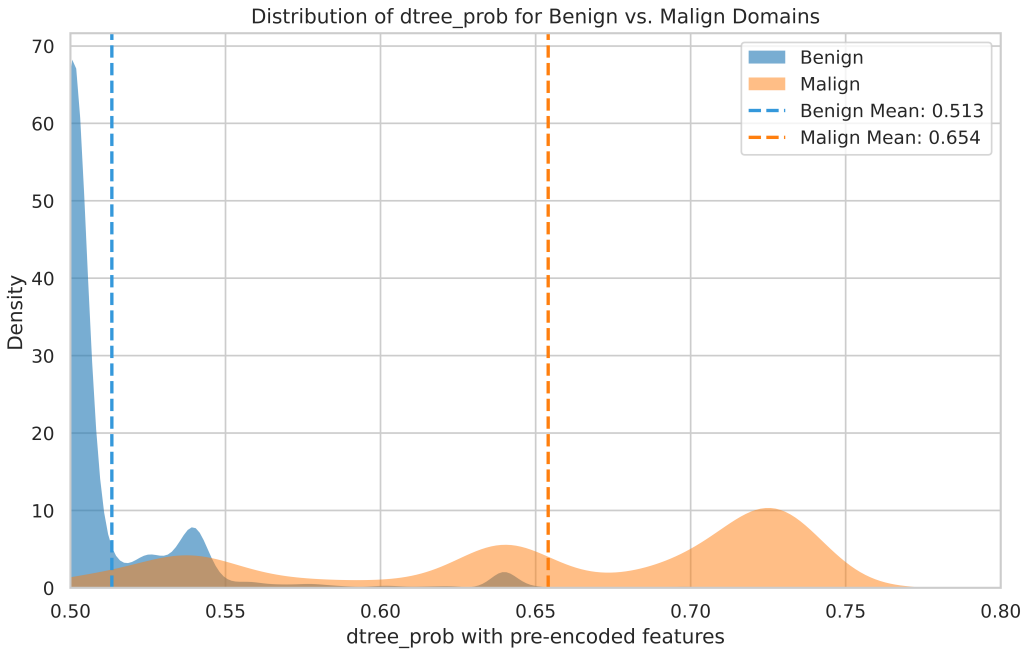


Figure 5.6: Distribution of values, using combined approach

The average dtree_prob for malignant domains stands at approximately 0.654, whereas the benign domains' average is notably lower, at around 0.513. This disparity in means indicates that the dtree_prob feature can prove useful in distinguishing between the two types of domains. This plot 5.6 implies that malignant domains are more likely to have higher dtree_prob values.

The decision tree's ability to synthesize and capture the intricate nonlinear relationships between multiple categorical features into a single dtree_prob value is of significant importance. By doing so, it has created a potent discriminative feature that has the potential to improve the accuracy of the model in detecting malignant domains. The consolidation of information into a singular feature is a strategy that serves to not only simplify the structure of a model by reducing dimensionality but also to refine the model's focus on the most indicative factors of domain malignancy.

The simplified implementation of the described logic is delineated in the subsequent Listing 5.9, and detailed code is provided in the appendix (see Listing D.7).

```
categorical_features = [...]
# Pipeline creation
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])
# Predict probabilities for both training and testing sets
probabilities_train = pipeline.predict_proba(X_train)[:, 1]
probabilities_test = pipeline.predict_proba(X_test)[:, 1]
```

Listing 5.9: Decision Tree for Categorical Features Training

The implementation initiates by ensuring the presence of the target variable, `label`, within the dataset `combined_df`, this variable represents the class or category to which each instance belongs. The process continues by defining a list of categorical features present in our datasets. These features undergo one-hot encoding, a preprocessing step that converts categorical variables into a binary matrix, essential for the Decision Tree algorithm to process.

The addition of the newly developed feature, as described in the preceding implementation (see Listing D.7), was critical in fine-tuning the model's performance measures, but with only minor improvements. This little improvement was especially noticeable in the observed increase in the F1 score Figure 5.6.

5.5 Implementing Solutions for Class Imbalance

This section focuses on practical implementations designed to address the class imbalance in prediction models. Biased models that perform poorly in predicting incidences of minority classes are frequently the result of class imbalance. In order to address this, we employ the `scale_pos_weight` option, which offers a method for assigning weights to classes in an inverse proportion to their frequency. This approach was used for XGBoost and SVM.

The computation of the `scale_pos_weight` based on the class distributions in the training dataset and its incorporation into the XGBoost classifier parameters are shown in the following Python code snippet 5.10. This method improves model impartiality and performance on unbalanced data by guaranteeing a more balanced assessment of classes throughout the training phase.

```
class_counts = y_train.unique(return_counts=True)[1]
weight_for_0 = class_counts[1].item() / class_counts[0].item()

params = {
    "max_depth": 9,
    "eta": 0.2,
    "objective": "binary:logistic",
    ...
    "grow_policy": "lossguide",
    "scale_pos_weight": weight_for_0  # Adjusting scale_pos_weight for class imbalance
}
trees = 300

model = XGBClassifier(
    **params,
    n_estimators=trees,
    eval_metric="logloss"
)
```

Listing 5.10: Scaling the Classes

In my initial attempts to eliminate class imbalance for CNN, I experimented with class weighting. However, this technique resulted in a considerable fall in the F1 score, which might be attributed to several factors. For example, weights may overcompensate for the minority class, causing the model to become too sensitive to it and produce more false positives. Furthermore, if the model is either simple or too complicated, it may fail to generalize adequately from training to test data.

The most successful technique was to use of the SMOTE oversampling method (see Subsection 3.3.3), which significantly improved the F1 score. The learning progress of this refined approach is depicted in Figure 5.7.

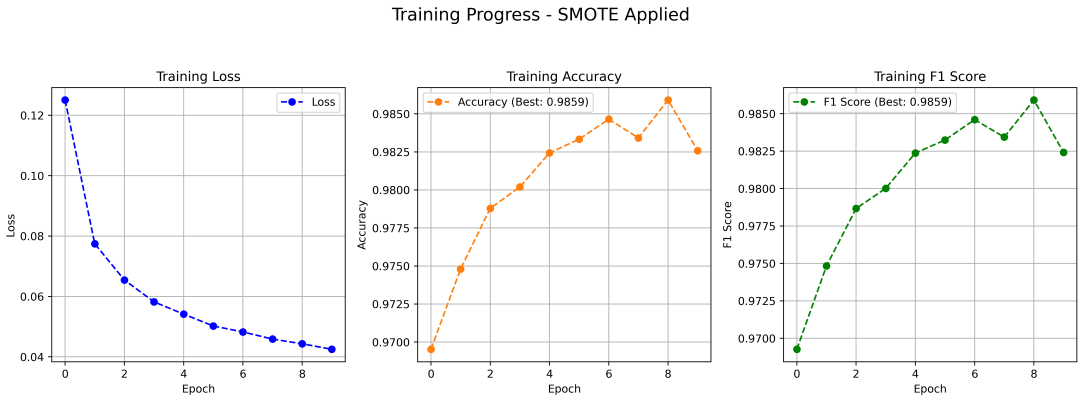


Figure 5.7: CNN evaluation incorporation SMOTE oversampling technique

Conversely, this particular approach has exhibited a notable increase (see Table 6.3) in the false positive rate (FPR), thereby raising questions about its suitability. This situation may indicate that the model is overfitting to the noise introduced by the synthetic samples.

Chapter 6

Evaluation and Examination of Chosen Optimization Techniques

This chapter analyzes the subtle effects of preprocessing techniques and dataset quality on the effectiveness of machine learning models that are intended to distinguish between benign and malign domains. This work aims to uncover the overlooked but substantial effects that model setup and data processing have on the accuracy and reliability of predicted outcomes via careful examination and comparison analysis.

6.1 Comparative Analysis of Original Datasets against Ground-truth ones

In this work, I conducted a detailed analysis to evaluate the performance of machine learning models trained on original datasets, namely "benign_2310" for benign ones, and "phishing_2307" with "phishing_2310" for phishing ones, versus those modified through a verification process, with "phishing_2311" and "benign_2312" acting as the ground-truth datasets. The major goal was to determine the influence of dataset quality on the model's ability to identify domains as benign or phishing correctly. The investigation used three different configurations of training data, each examined for precision, recall, f1-score, and total accuracy, yielding important insights into the trade-offs involved. This can be seen in the following Figure 6.1.

Original Collection	**Ground Truth**	**Falsely-labeled Domains (%)**
Cesnet data + Intersect + Thresholding + Suffix Reduction [5.1] (530k)	**benign_2312** (462k)	12.83%
benign_union_2307 (486.3k)	**cesnet3_2311** (194.5k)	60.00%
malware_2310 (88.2k)	**malware_2311** (45.6k)	48.30%
misp_2310 (68.8k)	**misp_2311** (68.4k)	0.58%

Table 6.1: Comparison of Data Collections

6.1.1 XGBoost Classifier

Configuration 1: Using Datasets „benign_2310“ and „phishing_2307“

The first configuration yielded an F1-score of 0.9530, with 758 TN and 353 FP. This setup, employing the „benign_2310“ and „phishing_2307“ datasets, exhibited high accuracy and recall, signifying a successful model performance. However, there is room for improvement, particularly in reducing FP to enhance precision.

Configuration 2: Using Datasets „benign_2312“ and „phishing_2307“

Configuration 2 demonstrated a notable enhancement, achieving an F1-score of 0.9677, alongside a reduction in FP to 212, and a slight decrease in TN to 496. By incorporating the „benign_2312“ and „phishing_2307“ datasets, this model significantly improved in detecting phishing domains. The increased precision and recall, especially in identifying phishing attempts, underscore the benefits of refining the benign dataset.

Configuration 3: Using Datasets „benign_2312“ and „phishing_2310“

The third configuration attained the highest F1-score of 0.9735, with the most significant number of TN (793) and a further reduction in FP to 201. Utilizing both „benign_2312“ and „phishing_2310“ datasets, this model configuration achieved superior overall performance. The balanced accuracy and recall across classes highlight the effectiveness of integrating validated benign and phishing datasets.

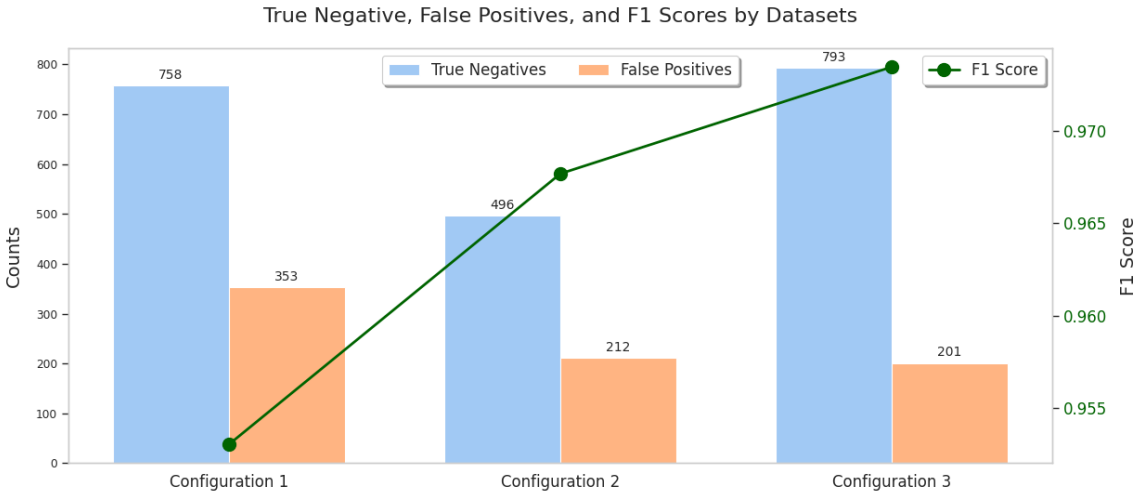


Figure 6.1: Comparison of performance using different datasets

Switching from original to verified datasets shows a clear trend of improved model performance. However, it offers trade-offs between maintaining high true negatives (important for avoiding unintentionally banning benign sites) and controlling false positives (vital for ensuring phishing domains are not neglected).

The decrease in false positives between the first and second configurations suggests that simply modifying the benign dataset can significantly increase model specificity. The somewhat higher number of false positives seen in the third configuration might be ascribed to the bigger size and potentially the increased complexity of the dataset utilized in this.

Plot 6.1 presented above clearly emphasizes the significance of dataset quality and variety in training more effective machine learning models. It is challenging to achieve a compromise between lowering false positives and preserving or enhancing true negatives (as well as overall model performance). To reduce potential risks associated with bigger and more complex datasets, careful model tuning, consideration in dataset design, and maybe the use of sophisticated approaches in data preparation and feature engineering are required.

Creating ground-truth datasets as well as reducing the number of false positive predictions may have a positive impact on the model's generalization. There are several visualization techniques for generalization illustration, I will show the improved generalization on the learning curve of the XGBoost model, Figure 6.2.

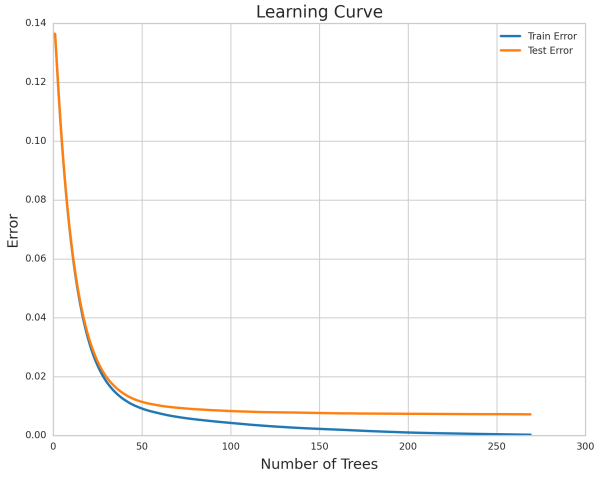


Figure 6.2: XGBoost model's learning curve on verified datasets

Both the training and testing set curves rapidly converge to a low error rate and remain close throughout the training period. This tight convergence suggests that the model is generalizing effectively, as the test error is not considerably larger than the training error. The error rates for both curves plateau at a low level, indicating that adding more trees does not result in overfitting; the model is stable. To summarize, the supplied learning curve implies good generalization. The model performs well across both the training and testing datasets, with a low error rate, showing that it has effectively learned the fundamental patterns.

6.1.2 Convolutional Neural Networks

To understand the significance of the ground-truth datasets in enhancing model performance, it's essential to scrutinize the reduction in false positives achieved through their utilization. The results of the analysis demonstrate a notable enhancement in the accuracy of models trained with these datasets, underscoring their pivotal role in refining the quality of the dataset. This finding highlights the significance of the datasets in improving the overall performance of the models.

Specifically, models trained on the original datasets without the verification process exhibited a false positive rate of approximately 0.92%. However, upon integrating the ground-truth datasets into the training process, a significant reduction in false positives was observed.

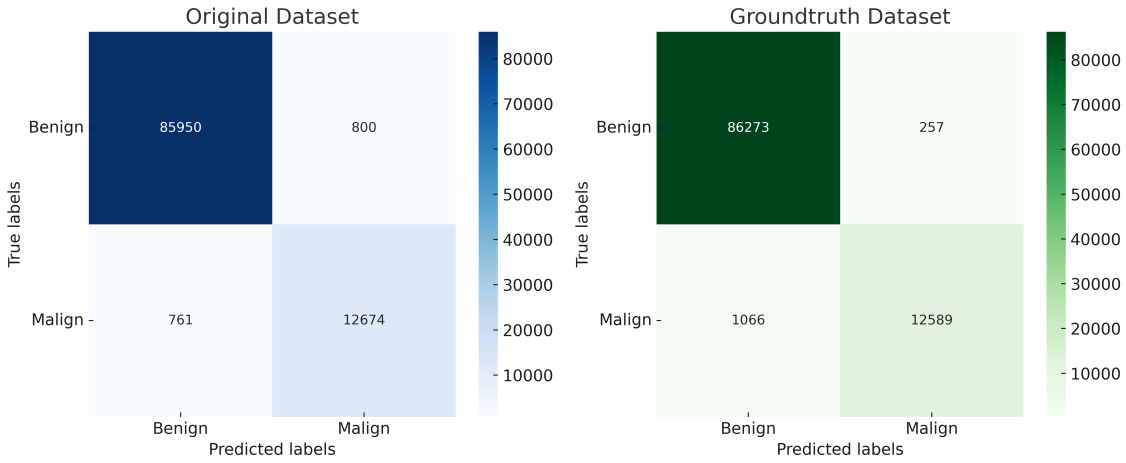


Figure 6.3: Comparison of confusion matrices, between original data and those verified using verification pipeline

The false positive rate decreased to just 0.30%, indicating a substantial 67.79% decrease in erroneous malign domain identifications among benign ones.

6.1.3 SVM

The transition from the original dataset to the ground truth dataset for training the SVM model resulted in significant improvements in model performance measures. The F1 score, which measures precision and recall, increased from 0.887 to 0.929. This indicates a 4.74 percent improvement, showing enhanced model accuracy and reliability in identifying domains as benign or malignant.

More importantly, the rate of false positives in which benign areas are wrongly labeled as malignant was also greatly decreased. The false positive rate lowered from 0.828 to 0.613 percent. This decrease of roughly 26.0% in the false positive rate is extremely important in practical applications, reducing the risk of benign domains being mistakenly blocked or labeled.

These alterations highlight remarkable improvements in the importance of thorough data collecting and collecting techniques. The significance of high-quality training data cannot be overstated, since it serves as the foundation for effective application of machine learning algorithms.

6.2 Results from Varied Preprocessing Methodologies

The new feature `dtree_prob` is a valuable addition to our dataset because it enriches the dataset with the model's assessment of how likely each instance is to belong to the positive class, based on the learned patterns from the categorical features selected (see Figure 6.6).

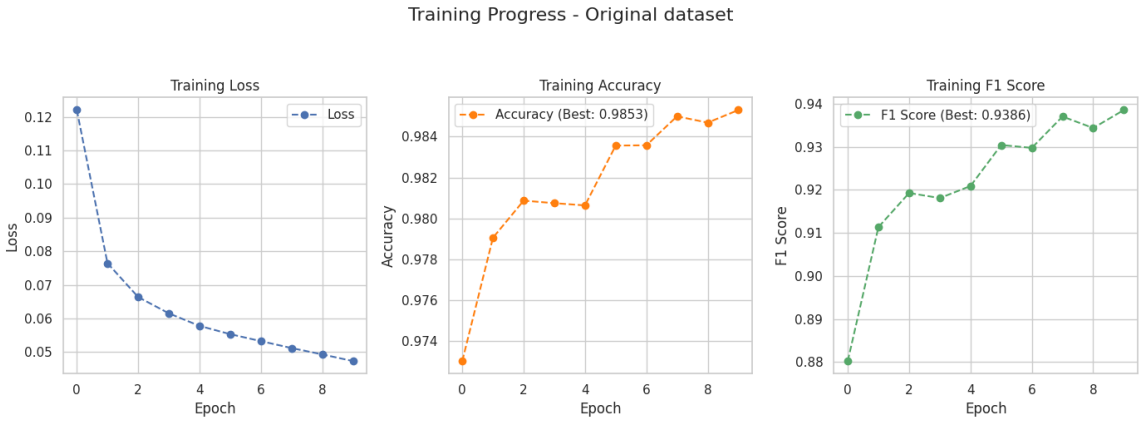


Figure 6.4: Training on Original Data

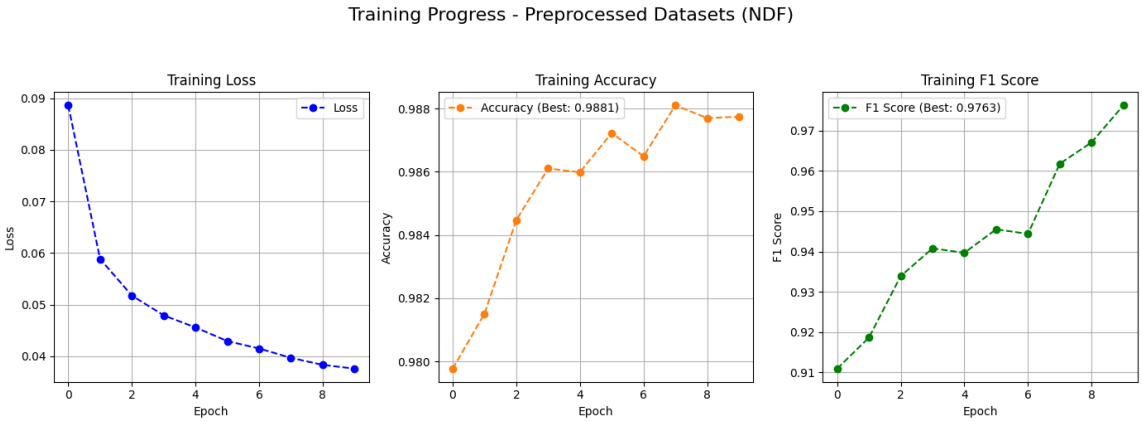


Figure 6.5: Training on NDF Data

An increase in the F1 score Figure 6.4 implies that the model accomplished a more balanced trade-off between accuracy and recall, implying an improvement in the model's capacity to categorize cases with more reliability properly. Furthermore, the minor increase in accuracy emphasizes the feature's role in improving the model's overall prediction skills.

The considerable increase in CNN performance depicted in Figure 6.5, demonstrates the significant impact that using a matrix format, together with the extensive preprocessing steps discussed earlier, has on machine learning models. These preprocessing techniques provide the models with more informative inputs, allowing for more precise decision-making processes. As a result, the conclusions are not only more precise, but also better reflect the distribution of the underlying data. This emphasizes the important role of rigorous preprocessing in improving the effectiveness of machine learning algorithms.

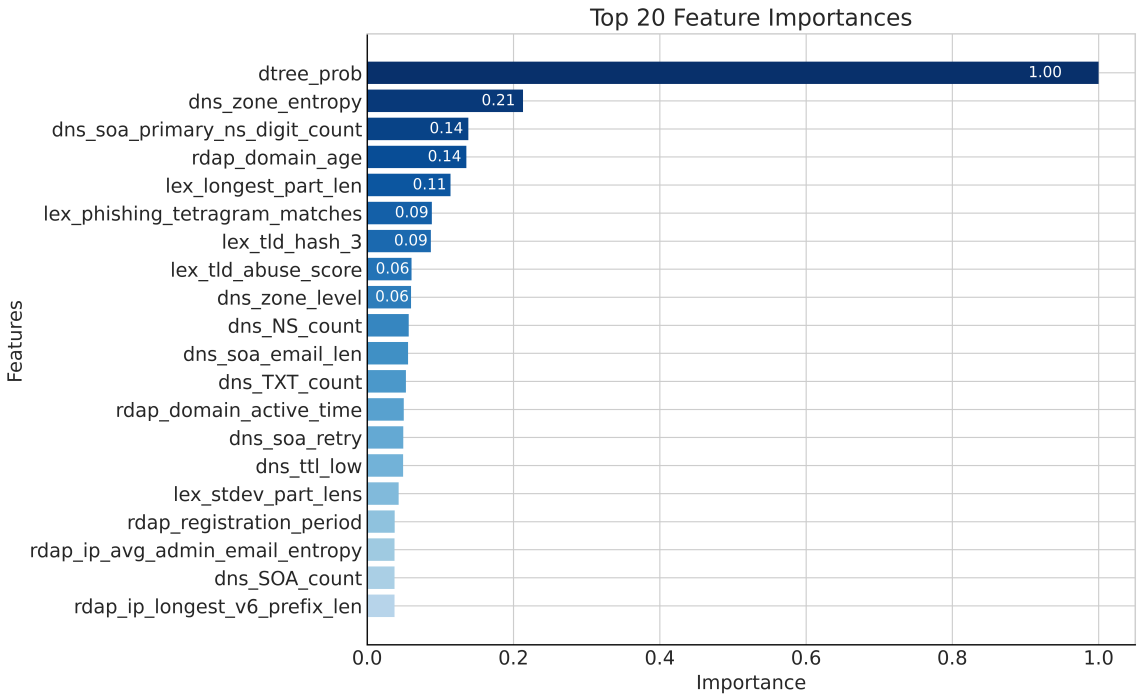


Figure 6.6: Most important features based on XGboost importances

6.3 Comparison of Hyperparameter Tuning Methods

Following the optimization processes (see Section 5.2), the performance of the model under each method was assessed using several metrics, including accuracy, precision, recall, F1 score, and AUC. The results for XGBoost are presented in the Table 6.2.

Optimization Method	Accuracy	Precision	Recall	F1 Score	AUC
Random Search	0.9847	0.9793	0.9089	**0.9428**	0.9956
Bayesian Opt.	0.9850	0.9774	0.9650	**0.9540**	0.9955
Grid Search	0.9831	0.9751	0.9715	**0.9593**	0.9946

Table 6.2: Performance metrics obtained from each optimization approach

The exploration of parameter optimization methods revealed distinct trade-offs between computational efficiency and optimization performance. Random Search, despite its simplicity, provided competitive results with significantly less computational time compared to Grid Search. This method requires less time due to its stochastic nature, sampling parameter values from a distribution without requiring exhaustive exploration of all combinations.

Bayesian Optimization with the Hyperopt library showcased an improved balance between efficiency and performance. It strategically navigates the parameter space by building a probabilistic model to predict areas with promising results, thereby reducing unnecessary computations. This method struck a good balance, offering a slight improvement in metrics such as F1 score and AUC over Random Search, albeit at a higher computational cost.

Grid Search, known for its thoroughness, exhaustively searched through the predefined parameter grid. While it guarantees that the optimal parameter combination within the grid is found.

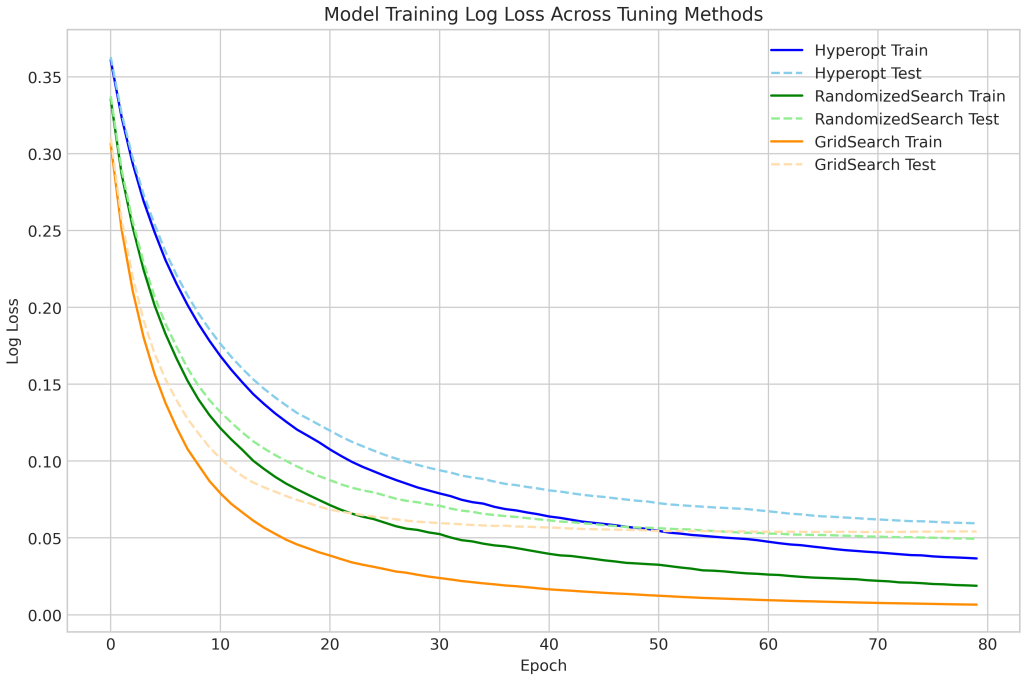


Figure 6.7: Log loss across different approaches used in parameters choosing for XGBoost

To summarize, the Figure 6.7 presented above illustrates the efficacy of each tuning method for XGBoost over the training period. Hyperopt and RandomizedSearch exhibit faster initial improvements, while GridSearch shows more stability between the training and test loss, which might indicate a better generalization performance. For the SVM the tuning process's main part consists of choosing the right kernel type, that is capable of accurately capturing the complex structure of the decision boundaries in data. This choice significantly influences the SVM's effectiveness in handling high-dimensional spaces and non-linear relationships among data points, directly affecting both its classification accuracy and generalization to new data (see Figure 6.8).

The evaluation of three distinct Support Vector Machine (SVM) kernels, namely linear, polynomial (poly), and radial basis function (rbf), is presented in Figure 6.8. This evaluation involves the assessment of metrics such as Accuracy, Precision, Recall, F1 Score, and False Positive Rate (FPR). The performance of each kernel is presented independently, with the use of different colors to depict varying metrics.

With a competitively high F1 Score, the radial basis function (rbf) kernel exhibits a good balance between accuracy and precision. Among the three, the rbf kernel notably obtains the lowest FPR, which is important for applications where lowering the occurrence of false positives is important. For cases requiring strict error tolerances, the rbf kernel is seen to be the best option due to its relatively high F1 Score and lowest FPR.

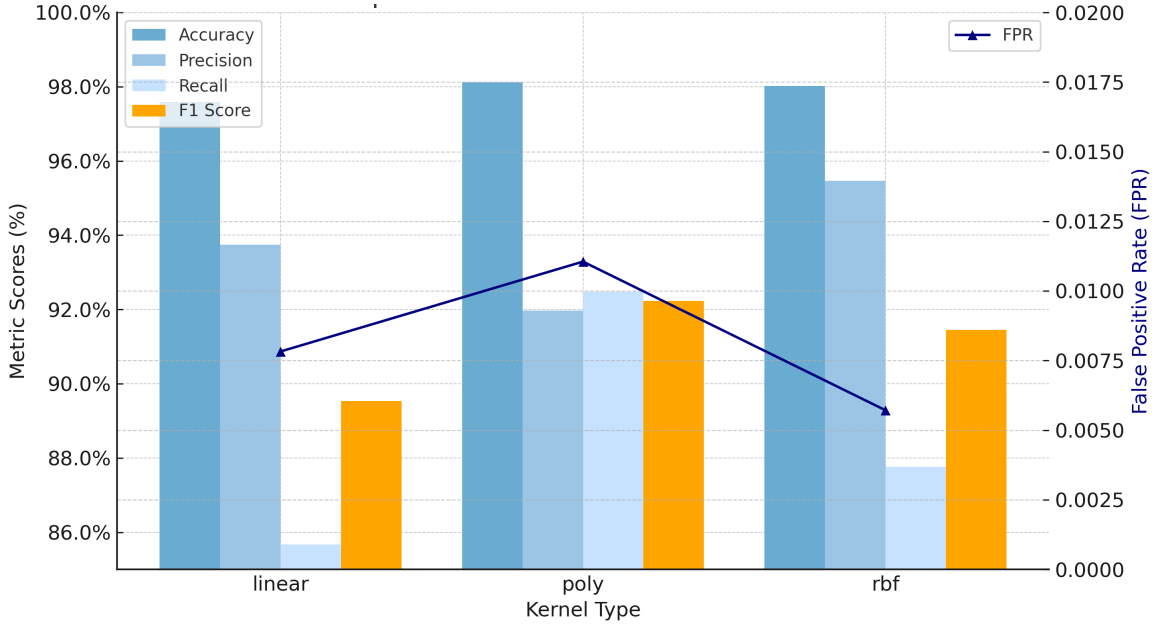


Figure 6.8: Evaluation of different kernel types used in SVM, combining also NDF preprocessing and ground-truth datasets

6.4 Evaluation and Comparison with Original Models

Four optimization approaches were analyzed: Ground-truth dataset utilization, NDF preprocessing techniques, Parameter selection strategies, and Imbalance handling methods. These approaches were systematically applied to each classifier to evaluate performance improvements. The results from this comprehensive analysis, presented in Table 6.3, reveal the distinct strengths and limitations of each model.

	SVM		CNN		XGBoost	
Optimization	F1	FPR[%]	F1	FPR[%]	F1	FPR[%]
Original Dataset	0.887	0.828	0.9158	0.847	0.9530	0.457
Ground-truth Dataset	0.929	0.613	0.9386	0.300 6.3	0.9735 6.1	0.199
NDF-preprocessing	0.902	0.695	0.9763 6.5	0.576	0.9719	0.350
Parameter Selection	0.897	0.301	0.9620 5.3	0.690	0.9593 6.2	0.419
Imbalance Handling	0.897	0.301	0.9859 5.7	1.400	0.9694 5.10	0.188
Combined Approach	0.959 6.8	0.395 6.8	0.9926 6.9	0.896	0.9787	0.062

Table 6.3: Comparison of Optimization Approaches Across SVM, CNN, and XGBoost Classifiers

6.4.1 Results for Support Vector Machine

The `SVM` faced challenges, particularly in high-dimensional space handling. The results of this classification model were notably lower compared to other approaches.

Due to the intensive and time-consuming nature of its tuning and training operations, which might take more than 20 hours for a full dataset, the SVM was trained on a very small portion of the original data. This was primarily because allocating significant resources to evaluate each optimization independently was impossible. Despite these constraints, even this limited tuning showed some improvements when the model was trained and evaluated on the entire dataset.

The SVM's class imbalance was effectively resolved during parameter selection by including `„class_weight": „balanced"` into the configuration. This change had the best outcomes in addressing class disparities. The combined approach shows a marked improvement in performance, achieving an F1 score of **0.959** and an FPR of **0.395%**. This outcome indicates that with sufficient tuning and optimization, `SVM` can still perform effectively, though it may not be the most efficient model for our scenario.

6.4.2 Results for XGBoost

`XGBoost` excelled in minimizing false positives, achieving the lowest FPR of **0.062%** with the combined optimization approach. This performance is crucial in reducing the rate of false alarms, a significant achievement for practical applications. The high F1 score of **0.9735**, observed with the ground-truth dataset optimization, further attests to the robustness and reliability of `XGBoost` in this domain.

6.4.3 Results for Convolutional Neural Networks

The `CNN` model demonstrated exceptional capabilities, achieving an F1 score as high as **0.9926** with the combined optimization approach, results are depicted in Figure 6.9. This result not only highlights the model's accuracy but also its efficiency in operational scenarios where speed is critical.

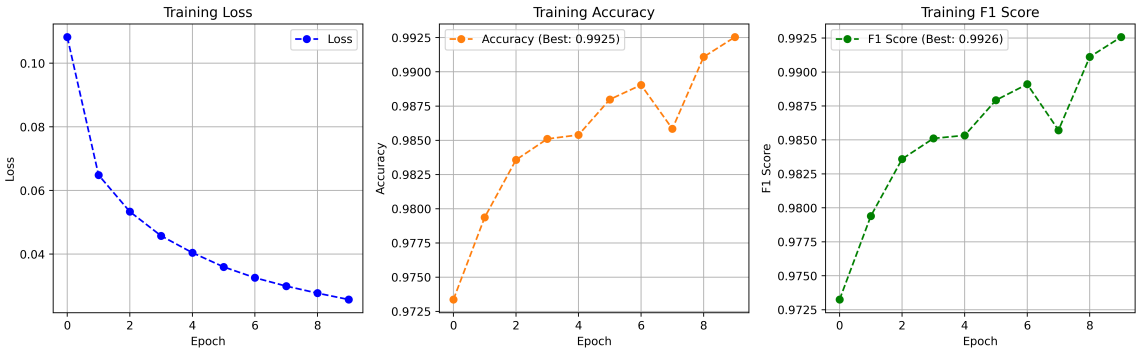


Figure 6.9: Combination of all optimization approaches for CNN

Unfortunately, taking a combined strategy, particularly in terms of imbalanced data management, had a negative impact on the final FPR value. The best performance was achieved utilizing NDF format under the ground-truth dataset optimization, resulting in the best FPR recorded at **0.300%**. This can bee seen in Figure 6.10.

In addition to highlighting the differences between classification techniques, it is important to also focus on the benefits of each optimization approach. This comprehensive approach will enable a more thorough understanding of the various methods and their potential applications. By examining the advantages of each technique, we can gain insights into their strengths and limitations, which can inform our decision-making processes.

Through the use of a ground-truth dataset, the False Positive Rate (FPR) was significantly reduced across all models. `XGBoost` and `CNN` saw reductions to 0.199% and 0.300%, respectively, showcasing the power of accurate and well-labeled data to enhance model specificity.

NDF-preprocessing was particularly advantageous for the `CNN`, which leverages the matrix output format, leading to a significant boost in its F1 score to 0.9763. Optimal parameter selection and fine-tuning were essential in achieving improved F1 scores for both `XGBoost` and `CNN`. This preprocessing method, though effective in improving especially `CNN`'s performance, had mixed effects on other models, sometimes leading to an increase in computational demands without proportionate gains in F1 scores.

Depending on the model and method employed, addressing data imbalances produced a variety of effects. Addressing data imbalances using SMOTE and other techniques improved the F1 score of `CNN`, although they caused overfitting in simpler models. Class weighting, on the other hand, worked well with `XGBoost` and `SVM`, supporting or enhancing performance without the disadvantages of other methods. Inappropriate imbalance handling methods led to a significant drop in F1 scores for `XGBoost` and `SVM`, highlighting the significance of choosing the suitable strategy depending on the characteristics of the model and the complexity of the dataset.

Ultimately, the combined approach, integrating all optimization techniques, yielded the most substantial improvements, with `CNN` achieving an F1 score of 0.9926 and `XGBoost` reducing its FPR to an impressive 0.062%, underscoring the compounded benefits of using multiple optimization strategies concurrently. Overall results are depicted in Figure 6.10 demonstrating remarkable improvement in classification performance.

Figure 6.10: Comparison of results

Chapter 7

Conclusion

Throughout my research, I conducted a thorough study of classification models and approaches aimed at identifying malicious domains. This involved reviewing existing models and comparing their respective strengths and weaknesses, as well as identifying areas for improvement. I managed to create a pipeline for creating high-quality datasets and proposed strategies to reduce false positives, exploring hyperparameter tuning and data preprocessing in the process.

I have successfully advanced the optimization of classification models for the detection of malicious domains, with notable success across several machine learning architectures including Convolutional Neural Networks (CNN), XGBoost, and Support Vector Machines (SVM). The CNN model performed exceptionally well, with an F1 score of 0.9926 with a remarkably low false positive rate (FPR) of 0.003.

Although the Convolutional Neural Network (CNN) models outperformed the XGBoost and Support Vector Machine (SVM) models, these two have also demonstrated significant improvements in their ability to classify benign and malignant domains. As part of the FETA project, I have designed and implemented a verification pipeline that has generated several ground truth datasets. These datasets have become a valuable resource for ongoing research projects, providing a reliable basis for both model training and validation.

Looking ahead, there are numerous ways to further enhance our solutions. Exploring the use of Generative Adversarial Networks (GANs) to create synthetic training data could significantly improve the models' ability to generalize. Additionally, incorporating more deep-learning models into our arsenal could lead to new insights and potentially more effective detection techniques.

In conclusion, this thesis significantly contributes to the field of cybersecurity by improving malicious domain detection capabilities and lays a solid foundation in the field of cybersecurity. The models will continue to be successful against the changing landscape of cyber risks if the ground truth datasets are updated regularly with the most recent information on cyber threats. I consider my thesis to be a significant contribution to the detection of malicious domains, and I am committed to continuing this research in the upcoming months. I intend to implement even more strategies I have proposed, and conduct further experiments to advance the field even further.

Bibliography

[1] *Basic Introduction to Convolutional Neural Network in Deep Learning* [online]. Dec 2023. Available at: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/.

[2] *Ground Truth in Machine Learning: Process & Key Challenges* [online]. Dec 2023. Available at: https://datagen.tech/guides/data-training/ground-truth/.

[3] *An Intro to Hyper-parameter Optimization using Grid Search and Random Search* [online]. Dec 2023. Available at: https://medium.com/@cjl2fv/an-intro-to-hyper-parameter-optimization-using-grid-search-and-random-search-d73b9834ca0a.

[4] *Quarto Cheat Sheet (Previously Known as RMarkdown)* [DataCamp]. 2023. Accessed: 30 November 2023. Available at: https://www.datacamp.com/cheat-sheet/quarto-cheat-sheet-previously-known-as-r-markdown.

[5] Abad, S., Gholamy, H. and Aslani, M. Classification of Malicious URLs Using Machine Learning. *Sensors* [online]. 2023. Accessed: 26 November 2023. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10537824/.

[6] Aljofey, A., Jiang, Q., Rasool, A. and Chen, H. *An effective detection approach for phishing websites using URL and HTML features* [online]. Dec 2022. DOI: 10.1038/s41598-022-10841-5. Available at: https://www.nature.com/articles/s41598-022-10841-5.

[7] Aponyi, A. *Domain Classification with Natural Language Processing* [online]. 2021. Available at: https://www.taus.net/resources/blog/domain-classification-with-natural-language-processing.

[8] Ayoub, I., Lenders, M. S., Ampeau, B., Balakrichenan, S., Khawam, K. et al. *Understanding IoT Domain Names: Analysis and Classification Using Machine Learning*. 2024. DOI: 10.48550/arXiv.2404.15068.

[9] Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*. 2012, vol. 13, Feb, p. 281–305.

[10] Bhandari, P. *How to Find Outliers | 4 Ways with Examples & Explanation* [online]. Scribbr, 21. Jun 2023. Accessed: 2023-11-10. Available at: https://www.scribbr.com/statistics/outliers/.

[11] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[12] Bonthu, H. *Detecting and Treating Outliers | How to Handle Outliers* [online]. 2021. Last updated on September 28th, 2023. Available at: https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/.

[13] Braei, M. *Master thesis - Anomaly Detection* [ResearchGate]. Dissertation. Accessed: 26 November 2023. Available at: https://www.researchgate.net/publication/342898418_Master_thesis.

[14] Bugaj, M., Wrobel, K. and Iwaniec, J. Model Explainability using SHAP Values for LightGBM Predictions. In: *2021 IEEE XVIIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. 2021, p. 102–106. DOI: 10.1109/MEMSTECH53091.2021.9468078.

[15] Buitin, A. *5 Anomaly Detection Algorithms to Know* [Built In]. 2023. Accessed: 26 November 2023. Available at: https://builtin.com/machine-learning/anomaly-detection-algorithms.

[16] ByJim. 5 Ways to Find Outliers in Your Data. [online]. 2018. Accessed: 2023-11-26. Available at: https://statisticsbyjim.com/basics/outliers/.

[17] CFA, FRM, and Actuarial Exams Study Notes. *Overfitting and Methods of Addressing It* [online]. 2022. Available at: https://analystprep.com/study-notes/cfa-level-2/quantitative-method/overfitting-methods-addressing/.

[18] Chica, M. Authentication of bee pollen grains in bright-field microscopy by combining one-class classification techniques and image processing. *Microscopy research and technique*. november 2012, vol. 75, p. 1475–85. DOI: 10.1002/jemt.22091.

[19] Das, A. *Oversampling to Remove Class Imbalance Using SMOTE*. 2019. Available at: https://medium.com/@asheshdas.ds/oversampling-to-remove-class-imbalance-using-smote-94d5648e7d35.

[20] DataCamp. *Top Techniques to Handle Missing Values Every Data Scientist Should Know* [online]. 2023. Accessed: 2023-11-06. Available at: https://www.datacamp.com/tutorial/techniques-to-handle-missing-data-values.

[21] Deri, L., Martinelli, M., Sartiano, D., Serrecchia, M., Sideri, L. et al. Implementing Web Classification for TLDs. In: *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. 2015, vol. 1, p. 85–88. DOI: 10.1109/WI-IAT.2015.112.

[22] Faroughi, A., Morichetta, A., Vassio, L., Figueiredo, F., Mellia, M. et al. Towards website domain name classification using graph based semi-supervised learning. *Computer Networks*. 2021, vol. 188, p. 107865. DOI: https://doi.org/10.1016/j.comnet.2021.107865. Available at: https://www.sciencedirect.com/science/article/pii/S1389128621000384.

[23] Faroughi, A., Morichetta, A., Vassio, L., Figueiredo, F., Mellia, M. et al. Towards website domain name classification using graph based semi-supervised learning. *Computer Networks*. Elsevier BV. april 2021, vol. 188, p. 107865. DOI: 10.1016/j.comnet.2021.107865. ISSN 1389-1286. Available at: http://dx.doi.org/10.1016/j.comnet.2021.107865.

[24] Flach, P. A. and Lachiche, N. Naive Bayesian Classification of Structured Data. *Machine Learning*. 2004, vol. 57, p. 233–269. DOI: 10.1023/B:MACH.0000039778.69032.ab. Available at: https://doi.org/10.1023/B:MACH.0000039778.69032.ab.

[25] Garg, A., Trivedi, N., Lu, J., Eirinaki, M., Yu, B. et al. An evaluation of machine learning methods for domain name classification. In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, p. 4577–4585. DOI: 10.1109/BigData50022.2020.9377787.

[26] GeeksforGeeks. *XGBoost* [geeksforgeeks]. 2023. Available at: https://www.geeksforgeeks.org/xgboost/.

[27] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning* [online]. MIT Press, 2016. http://www.deeplearningbook.org.

[28] Guyon, I. and Elisseeff, A. An introduction to variable and feature selection. *Journal of Machine Learning Research* [online]. 2003, vol. 3, Mar, p. 1157–1182. Available at: https://www.jmlr.org/papers/volume3/guyon03a/guyon03a.pdf.

[29] Hand, D. and Till, R. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*. 2001, vol. 45, p. 171–186. DOI: https://doi.org/10.1023/A:1010920819831.

[30] Hastie, T., Tibshirani, R. and Friedman, J. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009. ISBN 978-0-387-84857-0.

[31] Holbert, C. Outlier Identification Using Mahalanobis Distance | Charles Holbert. [online]. 2023. Accessed: 2023-11-26. Available at: https://www.cfholbert.com/blog/outlier_mahalanobis_distance/.

[32] Horak, A. and Polisensky, J. *dr-collector: Domain Research Collector* [online]. 2022. Accessed: 2023-11-26. Available at: https://github.com/OviOvocny/dr-collector.

[33] Horák, A. *Detekce škodlivých domén na základě externích zdrojů dat*. 2023. [cit. 2023-12-02]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Supervisor Ing. Radek Hranický, P.

[34] Hosmer Jr, D. W., Lemeshow, S. and Sturdivant, R. X. *Applied logistic regression*. John Wiley & Sons, 2013.

[35] IBM. *What is Logistic regression? | IBM* [IBM]. Available at: https://www.ibm.com/topics/logistic-regression.

[36] In, B. Mahalanobis Distance & Multivariate Outlier Detection in R. [online]. 2023. Accessed: 2023-11-26. Available at: https://builtin.com/data-science/mahalanobis-distance.

[37] Jain, A. *Support Vector Machine (SVM) Classifiers and Kernels* [Medium]. 2020. Available at: https://medium.com/@apurvjain37/support-vector-machine-s-v-m-classifiers-and-kernels-9e13176c9396.

[38] James, G., Witten, D., Hastie, T. and Tibshirani, R. An Introduction to Statistical Learning. *Springer*. 2013.

[39] Jiang, Y., Wang, D. and Xu, D. DeepDom: Predicting Protein Domain Boundary from Sequence Alone Using Stacked Bidirectional LSTM. *Pacific Symposium on Biocomputing* [online]. 2019. Accessed: 26 November 2023. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6417825/.

[40] Karkala, D. *Data Analysis and Anomaly Detection in Buildings using Sensor Data* [[Publishing Platform]]. 2023. Accessed: 26 November 2023. Available at: https://www.deepakkarkala.com/docs/articles/machine_learning/anomaly_detection/thesis.pdf.

[41] Keita, Z. *Classification in machine learning: A guide for beginners* [DataCamp]. 2022. Accessed: 26 November 2023. Available at: https://www.datacamp.com/blog/classification-machine-learning.

[42] Kelleher, J. D., Mac Namee, B. and D'Arcy, A. *Fundamentals of Machine Learning for Predictive Data Analytics, second edition*. 2020. ISBN 9780262361101.

[43] Kocinec, P. *Classification and Blocking of DGA Domain Names*. Dec 2023. Available at: https://is.muni.cz/th/fnnl4/outlier_detection.zip?info.

[44] Kohavi, R. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection*. Stanford University, 1995.

[45] Korstanje, J. *All You Need to Know About the F1 Score in Machine Learning* [online]. 2021. Accessed on September 13, 2023. Available at: https://towardsdatascience.com/the-f1-score-bec2bbc38aa6.

[46] Kumar, P., Bhatnagar, R., Gaur, K. and Bhatnagar, A. Classification of Imbalanced Data: Review of Methods and Applications. *IOP Conf. Series: Materials Science and Engineering*. 2021, vol. 1099, p. 012077. DOI: 10.1088/1757-899X/1099/1/012077.

[47] Kundu, R. *F1 Score in Machine Learning: Intro & Calculation* [online]. 2022. Accessed on October 12, 2023. Available at: https://www.v7labs.com/blog/f1-score-guide.

[48] Lee, S., Lee, C., Mun, K. G. and Kim, D. Decision Tree Algorithm Considering Distances Between Classes. *IEEE Access*. 2022, vol. 10, p. 69750–69756. DOI: 10.1109/ACCESS.2022.3187172.

[49] Liu, Z., Zhang, Y., Chen, Y., Fan, X. and Dong, C. Detection of Algorithmically Generated Domain Names Using the Recurrent Convolutional Neural Network with Spatial Pyramid Pooling. *Entropy (Basel)*. Multidisciplinary Digital Publishing Institute. 2020, vol. 22, no. 9, p. 1058. DOI: 10.3390/e22091058.

[50] Madhugiri, D. *Exploratory Data Analysis (EDA): Types, tools, process* [KnowledgeHut]. 2023. Accessed: 26 November 2023. Available at: https://www.knowledgehut.com/blog/data-science/eda-data-science.

[51] Mantovani, R. *Use of meta-learning for hyperparameter tuning of classification problems* [ResearchGate]. Dissertation. Available at: https://www.researchgate.net/publication/346044422_Use_of_meta-learning_for_hyperparameter_tuning_of_classification_problems.

[52] MathWorks. *Compare Deep Learning Models Using ROC Curves* [online]. 2023. Available at: https://www.mathworks.com/help/deeplearning/ug/compare-deep-learning-models-using-ROC-curves.html.

[53] Mehta, A. Why is Logistic Regression Called „Regression“ if it is a Classification Algorithm? *Medium* [plainenglish]. 2020. Available at: https://ai.plainenglish.io/why-is-logistic-regression-called-regression-if-it-is-a-classification-algorithm-9c2a166e7b74.

[54] Mg, T., M S, K. and Nair, S. Domain Classification and Tagging of Research Papers using Hybrid Keyphrase Extraction Method. In: [online]. June 2017. Accessed: 30 November 2023.

[55] Muraleedharan, V. *What is Linear Discriminant Analysis (LDA)*. Jun 2021. Available at: https://vivekmuraleedharan73.medium.com/what-is-linear-discriminant-analysis-lda-7e33ff59020a.

[56] Name, A. F. How to Handle Missing Data. *Towards Data Science* [online]. 2018. Accessed: 2023-11-26. Available at: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.

[57] Narkhede, S. Understanding AUC - roc curve. *Medium*. Towards Data Science. 2021. Available at: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[58] Patel, H. *What is Feature Engineering — Importance, Tools and Techniques for Machine Learning* [online]. 2021. Accessed on October 12, 2023. Available at: https://towardsdatascience.com/what-is-feature-engineering-importance-tools-and-techniques-for-machine-learning-2080b0269f10.

[59] PhD, E. G. *Exploring Dataset Splitting Strategies: Techniques and Implementations in Python with Synthetic Data* [online]. Nov 2023. Available at: https://medium.com/@evertongomede/exploring-dataset-splitting-strategies-techniques-and-implementations-in-python-with-synthetic-515ddacb5c1c.

[60] Picard, R. R. and Berk, K. N. Data Splitting. *The American Statistician*. Taylor & Francis. 1990, vol. 44, no. 2, p. 140–147. DOI: 10.1080/00031305.1990.10475704. Available at: https://www.tandfonline.com/doi/abs/10.1080/00031305.1990.10475704.

[61] Polišenský, J. *Vyhodnocování rizik internetových domén*. 2022. [cit. 2023-12-02]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Supervisor Mgr. Ing. Pavel Očenášek, P. Available at: https://www.fit.vut.cz/study/thesis/25126/.

[62] Ponraj, A. 5 Most important Data Pre-Processing Techniques – Impute missing data – Part II. *DevSkrol* [online]. Jan 2022. Accessed: 2023-11-06. Available at: https://devskrol.com/2022/01/25/imputation-of-missing-data/.

[63] Ramesh, S. A Guide to an Efficient Way to Build Neural Network Architectures - Part II: Hyper-parameter. *Medium* [online]. 2021. Accessed: 19 April 2024. Available at: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7.

[64] Rathor, S. and Jadon, R. S. Domain Classification of Textual Conversation Using Machine Learning Approach. In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2018, p. 1–7. DOI: 10.1109/ICCCNT.2018.8494197.

[65] Ravi, V., Kp, S., Poornachandran, P., Soman, A. and Elhoseny, M. Improved DGA Domain Names Detection and Categorization Using Deep Learning Architectures with Classical Machine Learning Algorithms. In:. June 2019, p. 161–192. ISBN 978-3-030-16836-0.

[66] Reitermanova, Z. et al. Data splitting. In: Matfyzpress Prague. *WDS*. 2010, vol. 10, p. 31–36. ISBN 978-80-7378-139-2.

[67] Rendyk. *Bayesian optimization: BAYES_OPT or hyperopt* [online]. Analytics Vidhya, 2021. Accessed: 10 May 2024. Available at: https://www.analyticsvidhya.com/blog/2021/05/bayesian-optimization-bayes_opt-or-hyperopt/.

[68] Saritas, M. M. and Yasar, A. Performance Analysis of ANN and Naive Bayes Classification Algorithm for Data Classification. *International Journal of Intelligent Systems and Applications in Engineering* [online]. 2023. Available at: https://ijisae.org/index.php/IJISAE/article/view/934.

[69] Scribbr. How to Find Outliers | 4 Ways with Examples & Explanation. [online]. 2021. Accessed: 2023-11-26. Available at: https://www.scribbr.com/statistics/outliers/.

[70] Sims, Z., Strgar, L., Thirumalaisamy, D., Heussner, R., Thibault, G. et al. SEG: Segmentation Evaluation in absence of Ground truth labels. *BioRxiv*. Feb 2023, p. 2023.02.23.529809. DOI: 10.1101/2023.02.23.529809.

[71] Singh, A. *Hyperparameter tuning of neural networks using Keras Tuner* [online]. Analytics Vidhya, 2021. Accessed: 10 May 2024. Available at: https://www.analyticsvidhya.com/blog/2021/08/hyperparameter-tuning-of-neural-networks-using-keras-tuner/.

[72] Su, M.-Y. and Su, K.-L. BERT-based Approaches to Identifying Malicious URLs. *Sensors (Basel, Switzerland)* [online]. 2023. Accessed: 26 November 2023. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10610561/.

[73] Tharwat, A. Parameter investigation of support vector machine classifier with kernel functions. *Knowledge and Information Systems*. Springer. 2019, vol. 61, p. 1269–1302. DOI: https://doi.org/10.1007/s10115-019-01335-4.

[74] The365Team. *Introduction to Decision Trees: Why Should You Use Them?* [online]. 2023. Available at: https://365datascience.com/tutorials/machine-learning-tutorials/decision-trees/.

[75] Torroledo, I., Camacho, L. D. and Bahnsen, A. C. Hunting malicious TLS certificates with deep neural networks. In: *Proceedings of the 11th ACM workshop on Artificial Intelligence and Security*. 2018, p. 64–73.

[76] Vinayakumar, R., Soman, K., Poornachandran, P. and Sachin Kumar, S. Evaluating deep learning approaches to characterize and classify the DGAs at scale. *Journal of Intelligent & Fuzzy Systems*. IOS Press. 2018, vol. 34, no. 3, p. 1265–1276.

[77] Ying, X. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*. IOP Publishing. feb 2019, vol. 1168, no. 2, p. 022022. DOI: 10.1088/1742-6596/1168/2/022022. Available at: https://dx.doi.org/10.1088/1742-6596/1168/2/022022.

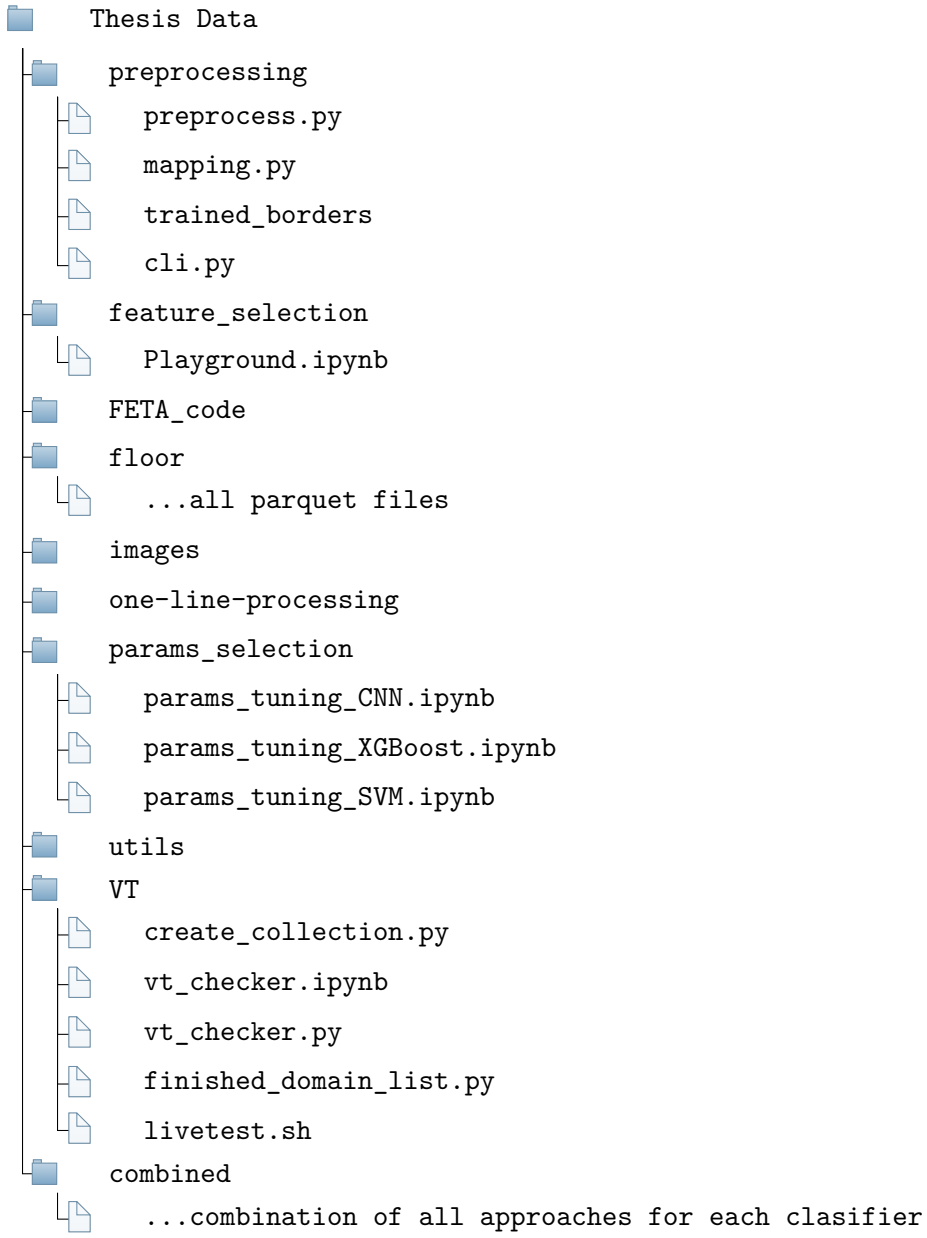
[78] Zagatti, F. R., Silva, L. C., Dos Santos Silva, L. N., Sette, B. S., Caseli, H. d. M. et al. MetaPrep: Data preparation pipelines recommendation via meta-learning. In: *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2021, p. 1197–1202. DOI: 10.1109/ICMLA52953.2021.00194.

[79] Zats, M. *The art of feature engineering: Techniques for creating better machine learning models in Python* [online]. 2023. Accessed on October 12, 2023. Available at: https://python.plainenglish.io/the-art-of-feature-engineering-techniques-for-creating-better-machine-learning-models-in-python-acd9a2b78f1e.

[80] Zhao, C., Zhang, Y., Zang, T., Chen, Y. and Wang, Y. A Multi-feature-based Approach to Malicious Domain Name Identification from DNS Traffic. In: *2020 27th International Conference on Telecommunications (ICT)*. 2020, p. 1–5. DOI: 10.1109/ICT49546.2020.9239506.

Appendix A

Contents of the Storage Medium



The media that is provided contains a number of folders and files that are essential to the thesis project. These include tools, feature selection notebooks, preprocessing scripts, some of the code from the FETA project, and also `.pdf` and `.tex` versions of my thesis.

Essential scripts such as `preprocess.py`, which shows how to use the Normalized Domain Format (NDF) from utilities, and `trained_borders`, which includes saved joblib models for outliers, scaling, and categorical feature training, are included in the `preprocessing` folder. This allows the script to be used to classify domains one by one in real-time later on.

Interactive preprocessing is implemented in `cli.py`. `Playground.ipynb`, a Jupyter notebook used for improved feature testing, is located in the `feature_selection` directory.

Extra folders consist of `FETA_code` containing additional project code, `floor` containing all parquet files, and `images` archiving visuals used in the master's thesis.

For the purpose of fine-tuning parameters for different models, such as CNN, XGBoost, and SVM, use the `params_selection` folder. Utility scripts are organized under `utils`, enhancing various data processing aspects.

The `VT` folder contains scripts and notebooks associated with VirusTotal, emphasizing functionality like data gathering verification and MongoDB collection creation.

Finally, the `combined` folder contains 3 files, 1 for each classifier. There is:

- `xgboost_combined.ipynb`
- `SVM_combined.ipynb`
- `CNN_combined.ipynb`

Each of these files presents a combination of all implemented optimization approaches.

Appendix B

Manual

The manual, which provides a comprehensive guide for installing and running the software created for this master's thesis, is available in the `README.md` file. The project utilizes Poetry for dependency management and packaging to ensure a consistent development environment.

B.1 Software Requirements

The software is developed in Python and uses Poetry as a dependency management tool. Before proceeding with the installation, ensure that the following requirements are met:

- Python 3.10 or higher
- Poetry for Python

B.2 Installing Poetry

Poetry is a tool for dependency management and packaging in Python, allowing for reproducible builds and straightforward dependency resolution. Once Poetry is installed, you can set up the project by following these steps:

1. Clone the project repository from its source. Replace `<url-to-repository>` with the actual URL of the repository:

2. Navigate into the project directory:

   ```
   cd xpoucp01_masters_thesis_code
   ```

3. Install all dependencies managed by Poetry:

   ```
   poetry install
   ```

   This command reads the `pyproject.toml` file and installs all necessary Python packages into a virtual environment specifically for this project.

B.3 Running the Software

To run the software, ensure that you are within the project's virtual environment. If not already activated, you can activate the Poetry-managed virtual environment by using:

```
poetry shell
```

After activating the virtual environment, you can run the software according to the project's documentation or as follows:

```
python3 <path/to/script.py>
```

B.3.1 Preprocess CLI

File called `cli.py` is the only file with additional arguments. It can be run as:

```
python3 cli.py -eda --model <cnn|xgboost|adaboost|svm> <--scaling>
```

Ussage example:

```
python3 cli.py -eda --model cnn --scaling
```

B.4 Additional Notes

- The `pyproject.toml` file in the project directory includes all the necessary configurations and dependencies for the project.
- Always ensure that you are operating within the virtual environment to avoid conflicts with other Python projects or system-wide packages.
- For detailed documentation on Poetry, visit the official Poetry documentation website.
- If you somehow run into an missing dependency error, you can easily add it to the `pyproject.toml` file, run: `poetry add <package-name>`.

Appendix C

Feature Vector Overview

The feature vector for phishing detection is comprehensive, encompassing 169 distinct features. A substantial portion of these features are innovative, and developed by the FETA group, reflecting insights gained through our data analysis. The remaining features are derived from prior research. We have categorized these features into six distinct groups based on their nature and origin:

- **Lexical Features (48):** Derived from the textual analysis of domain names, these features include aspects like the overall length of the domain name, whether the domain starts with a digit, the frequency of phishing-related words, and the entropy measures of various domain parts like subdomains.
- **DNS-based Features (40):** These include quantitative counts of various DNS record types such as A, MX, and NS records, TTL values, and the integrity of DNSSEC as evaluated by DNSSEC scoring metrics, which help in determining the authenticity and reliability of the domain records.
- **IP-based Features (9):** Focused on the properties of IP addresses linked to the domain, these features capture the count of distinct IP addresses, the entropy of IP address distributions across different prefixes, and metrics like the average round-trip time, which can indicate the response efficiency of the server.
- **RDAP-based Features (24):** Originating from RDAP data, these features encompass registration-related information such as the domain's age, the period since the last registration update, and textual characteristics of the domain's registrar and registrant, providing insights into the legitimacy of the domain ownership.
- **TLS-based Features (34):** Extracted from analyzing TLS handshakes and certificate chains, these include the length of certificate validity, the specific TLS version negotiated during the handshake, and the presence of various security policies, which are critical for understanding the security posture of the domain.
- **Geolocation Features (16):** These features evaluate the geographic distribution of servers related to the domain, assessing metrics such as the number of different countries involved, the variability in geographic coordinates like latitudes and longitudes, and unique identifiers for regional combinations, aiding in the identification of patterns typical to phishing operations.

Appendix D

Code examples

D.1 Parameters selection

D.1.1 CNN - base architecture

```
class Net(nn.Module):
    def init(self, num_layers=2, kernel_size=3, activation=F.relu, side_size=28):
        super(Net, self).init()

        self.num_layers = num_layers
        self.kernel_size = kernel_size
        self.activation = activation
        self.side_size = side_size
            self.layers = []
            in_channels = 1
            out_channels = 32

            for i in range(num_layers):
                self.layers.append(nn.Conv2d(in_channels, out_channels,
                kernel_size=kernel_size, stride=1, padding=kernel_size//2))
                self.layers.append(self.activation())
                in_channels = out_channels
                out_channels *= 2

            self.features = nn.Sequential(*self.layers)
            # Initialize these to None; will be created on first forward pass
            self.fc1 = None
            self.fc2 = None

    def forward(self, x):
        x = self.features(x)
        # On first pass, initialize fully connected layers based on feature map size
        if self.fc1 is None:
            n_size = x.data.view(x.size(0), -1).size(1)
            self.fc1 = nn.Linear(n_size, 128).to(x.device)
            self.fc2 = nn.Linear(128, 2).to(x.device)
        x = torch.flatten(x, start_dim=1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Listing D.1: CNN architecture initialization

D.1.2 CNN - Keras tuning

```
X = np.array(dataset['features'])
y = np.array(dataset['labels'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

def build_model(hp):
    model = Sequential()
    for i in range(hp.Int('num_conv_layers', 1, 3)):
        model.add(Conv1D(
            filters=hp.Int(f'filters_{i}', min_value=32, max_value=256, step=32),
            kernel_size=hp.Choice(f'kernel_size_{i}', values=[3, 5, 7]),
            activation=hp.Choice(f'activation_{i}', values=['relu', 'tanh']),
            padding='same',  # Maintain dimensionality
            input_shape=(X_train.shape[1], 1) if i == 0 else None))

        if hp.Boolean(f'batch_norm_{i}'):
            model.add(BatchNormalization())

        model.add(MaxPooling1D(pool_size=2))
        if hp.Boolean(f'dropout_{i}'):
            model.add(
                Dropout(rate=hp.Float(f'dropout_rate_{i}',
                min_value=0.0,
                max_value=0.5,
                step=0.1)))

    model.add(Flatten())
    model.add(Dense(
        units=hp.Int('dense_units', min_value=32, max_value=256, step=32),
        activation=hp.Choice('dense_activation', values=['relu', 'tanh'])))

    model.add(Dropout(
        rate=hp.Float('final_dropout',
        min_value=0.0,
        max_value=0.5,
        step=0.1)))
    model.add(Dense(1, activation='sigmoid'))

    optimizer_choice = hp.Choice('optimizer', ['Adam', 'SGD', 'RMSprop'])
    learning_rate = hp.Float(
                    f'lr_{optimizer_choice}',
                    min_value=1e-4,
                    max_value=1e-2,
                    sampling='LOG')
    optimizer = getattr(optimizers, optimizer_choice)(learning_rate=learning_rate)

    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Set up the tuner focusing on validation loss minimization
tuner = RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=20,
    executions_per_trial=1,
```

```
    directory='cnn_keras_tuning',
    project_name='cnn_tuning'
)

# Run the hyperparameter search
tuner.search(X_train, y_train, epochs=20, validation_data=(X_test, y_test))

# Get the optimal hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]

# Build the model with the optimal hyperparameters and train it on the data
model = tuner.hypermodel.build(best_hps)
history = model.fit(X_train, y_train, epochs=50, validation_data=(X_test, y_test))
```

Listing D.2: Dynamic CNN architecture initialization in TensorFlow

D.2 NDF Preprocessing

D.2.1 Data Loading and Processing Steps Overview

```
def perform_eda(self, model=None, apply_scaling=False) -> None:
    # Load the data
    if self.one_line_processing:
        # Loading single record
        self.single_record_df = input_data
    else:
        # Input_data is a dictionary with benign and malign dataset paths
        self.benign_path = input_data.get('benign')
        self.malign_path = input_data.get('malign')
        self.single_record_df = None

    # Categorical Encoding
    (See 5.4)

    # Handling missing values and outliers
    features, labels = self.remove_outliers(features, labels, std_multiplier=8)

    # Apply scaling if requested
    if apply_scaling:
        features = self.apply_scaling(features, scaler_type)

    # Additional preprocessing steps...
    modified_data = pa.Table.from_pandas(features)
    output_path = os.path.join(self.DEFAULT_INPUT_DIR, 'modified_dataset.parquet')
    feature_names = features.columns

    # Modified combined dataset saved
    pq.write_table(modified_data, output_path)
    return torch.tensor(features.values).float(),
           torch.tensor(labels.values).float(),
           feature_names
```

Listing D.3: Data Loading and Preprocessing

D.2.2 Removing Outliers

```
def remove_outliers(self, features, labels, std_multiplier=8):
    if not self.one_line_processing:
        # Directly update self.outliers with new thresholds
        self.outliers = {column: (mean_val - std_multiplier * std_val,
                                  mean_val + std_multiplier * std_val)
            for column in features.select_dtypes(include=[np.number]).columns
            for mean_val, std_val in [(features[column].mean(),
                                       features[column].std())]}
        self.save_borders()
    else:
        self.load_borders()

    if not self.outliers:
        raise ValueError("...")

    # Apply loaded or newly computed thresholds to remove outliers
    for column, (lower_bound, upper_bound) in self.outliers.items():
        if column in features.columns:
            outlier_condition = (features[column] < lower_bound) |
                                (features[column] > upper_bound)
            outlier_indices = features[outlier_condition].index

            # Remove outliers from features and labels
            features.drop(outlier_indices, inplace=True)
            labels.drop(outlier_indices, inplace=True)

            if removed_count > 0:
                # Log if any rows were removed

    self.logger.info("Completed outlier removal.")
    return features, labels
```

Listing D.4: Removing Outliers

D.2.3 Data Scaling

```python
def apply_scaling(self, df: pd.DataFrame, scaler_type: str = 'StandardScaler'):
    numeric_df = df.select_dtypes(include=[np.number])

    # Branch logic based on one_line_processing
    if not self.one_line_processing:
        if scaler_type == 'StandardScaler':
            self.scaler = StandardScaler()
            scaled_data = self.scaler.fit_transform(numeric_df)
        elif scaler_type == 'MinMaxScaler':
            self.scaler = MinMaxScaler()
            scaled_data = self.scaler.fit_transform(numeric_df)
        elif scaler_type == 'RobustScaler':
            self.scaler = RobustScaler()
            scaled_data = self.scaler.fit_transform(numeric_df)
        elif scaler_type == 'MinMaxScaler + Sigmoid':
            self.scaler = MinMaxScaler()
            scaled_data = self.scaler.fit_transform(numeric_df)
            # Apply sigmoid scaling
            scaled_data = 1 / (1 + np.exp(-scaled_data))
        else:
            raise ValueError(f"Unsupported scaler type: {scaler_type}")
        self.save_borders()  # Save the scaler for future use
    else:
        self.load_borders()  # Load the previously saved scaler
        scaled_data = self.scaler.transform(numeric_df)

    scaled_df = pd.DataFrame(scaled_data,
                             columns=numeric_df.columns,
                             index=df.index)

    # Combine scaled numeric columns with non-numeric data
    for col in df.columns:
        if col not in numeric_df.columns:
            scaled_df[col] = df[col]

    return scaled_df
```

Listing D.5: Applying Scaling

D.2.4 NDF Function - output

```
def NDF(model: str, scaling: bool, input_data, one_line_processing: bool):
    fe_cli = FeatureEngineeringCLI(
                input_data=input_data,
                one_line_processing=one_line_processing)

    features, labels, feature_names = fe_cli.perform_eda(model, scaling)

    dataset = {
        'name': dataset_name,
        'features': features,
        'labels': labels,
        'dimension': features.shape[1],
        'feature_names': feature_names,
        'one_line_processing': one_line_processing  # Include this flag in your dataset
    dictionary
    }

    with open(file_path, 'wb') as file:
        pickle.dump(dataset, file, protocol=pickle.HIGHEST_PROTOCOL)

    if not one_line_processing:
        x_train, x_test, y_train, y_test = train_test_split(
            dataset['features'],
            dataset['labels'],
            test_size=0.2,
            random_state=42
        )
        display_dataset_subset(x_train, y_train, dataset['name'], dataset['dimension'])
    else:
        # For single record processing
        x_train, y_train = dataset['features'], dataset['labels']
        display_dataset_subset(x_train,
                               y_train,
                               dataset['name'],
                               dataset['dimension'],
                               subset_size=1)

    return dataset
```

Listing D.6: Creating NDF format

D.2.5 Categorical Features Encoding

```
# Check for label column
if 'label' not in combined_df.columns: raise ValueError("...")

labels = combined_df['label']
categorical_features = ['geo_continent_hash', 'geo_countries_hash',
                        'rdap_registrar_name_hash', 'tls_root_authority_hash',
                        'tls_leaf_authority_hash']

# Preprocessing for categorical features
preprocessor = ColumnTransformer(transformers=[
                                ('cat', OneHotEncoder(handle_unknown='ignore'),
                                categorical_features)
                                ], remainder='passthrough')

# Splitting dataset
X_train, X_test, y_train, y_test = train_test_split(
                                        combined_df[categorical_features],
                                        labels,
                                        test_size=0.2,
                                        random_state=42,
                                        stratify=labels
                                    )

# Pipeline creation
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', DecisionTreeClassifier(random_state=42))
])

# Predict probabilities for both training and testing sets
probabilities_train = pipeline.predict_proba(X_train)[:, 1]  # Probability of class 1
probabilities_test = pipeline.predict_proba(X_test)[:, 1]

# Assign probabilities to their respective rows in combined_df
combined_df.loc[X_train.index, 'dtree_prob'] = probabilities_train
combined_df.loc[X_test.index, 'dtree_prob'] = probabilities_test

# Training the model
pipeline.fit(X_train, y_train)

# Model evaluation
train_accuracy = pipeline.score(X_train, y_train)
test_accuracy = pipeline.score(X_test, y_test)
scores = cross_val_score(pipeline, combined_df[categorical_features], labels, cv=5)
```

Listing D.7: Decision Tree for Categorical Features Training