ML-based detection of cyber attacks

Souleymane Cheikh Sidia Annika Dame Camilla Eckhardt

1. Introduction

In this paper, we explore the application of several machine learning algorithms to three cybersecurity-related datasets: the Web Page Phishing Detection dataset, the Network Intrusion dataset, and the Android Malware Detection dataset, all sourced from Kaggle. Our goal is to preprocess, modify, and optimize the data for each model to achieve the highest possible performance across different metrics.

The first step in our approach involves thorough data preprocessing. We identify and eliminate non-essential features from each dataset to ensure the remaining data is more relevant for classification tasks. This process is critical for reducing noise and improving the overall accuracy of the models. After preprocessing, we tailor the datasets to meet the specific requirements of each machine learning algorithm.

The algorithms applied in this study include Random Forest/Decision Trees, Artificial Neural Networks (ANN), and Support Vector Machines (SVM)/Regression models. For each algorithm, we explore multiple variations by adjusting parameters such as the number of hidden layers and neurons in the neural networks, activation functions, and learning rates, as well as tuning hyperparameters in tree-based models and SVMs. These tweaks aim to optimize performance and adapt each model to the unique characteristics of the datasets.

Once the models are trained and tested, we conduct a thorough analysis of the results. This includes comparing the performance of each algorithm based on accuracy, precision, recall, and other relevant metrics. We also investigate patterns in the errors and misclassifications to understand potential shortcomings in the models. By examining the correlations between the algorithms' performance and their parameter settings, we aim to determine which model performs best for each dataset and whether certain types of errors tend to cluster together.

2. Datasets

2.1. Description

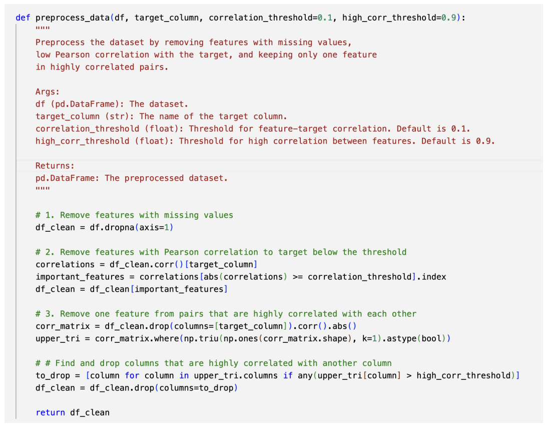
Web Page Phishing Detection Dataset:
This dataset is aimed at identifying phishing websites based on various features of the URLs. It includes 68 attributes that describe different characteristics of the URLs, such as length, the presence of IP addresses, and specific character counts. The target variable indicates whether the URL is legitimate or a phishing attempt, making it suitable for binary classification tasks in cybersecurity.

Network Intrusion Dataset:
This dataset consists of network traffic data collected to identify intrusions and malicious activities. It includes numerous features that capture different aspects of the network connections, such as packet sizes, connection duration, and various protocol types. The dataset is commonly used for developing intrusion detection systems, where the goal is to classify network traffic as normal or malicious.

Android Malware Detection Dataset
This dataset is focused on detecting malware in Android applications. It contains features derived from the APK files, such as permissions requested, API calls, and the presence of suspicious behaviors. The dataset is used for binary classification tasks, identifying whether an application is benign or malicious, helping to enhance mobile security.

2.2. Data preprocessing

In this phase, we focus on eliminating irrelevant data to enhance the efficiency and accuracy of our models. This involves discarding non-numerical data that lacks significance, as well as features that contain missing values. Additionally, we may remove features that exhibit strong correlations with one another or weak correlations with the target variable. The code provided below illustrates the initial step in our preprocessing workflow.



```python
def preprocess_data(df, target_column, correlation_threshold=0.1, high_corr_threshold=0.9):
    """
    Preprocess the dataset by removing features with missing values,
    low Pearson correlation with the target, and keeping only one feature
    in highly correlated pairs.

    Args:
    df (pd.DataFrame): The dataset.
    target_column (str): The name of the target column.
    correlation_threshold (float): Threshold for feature-target correlation. Default is 0.1.
    high_corr_threshold (float): Threshold for high correlation between features. Default is 0.9.

    Returns:
    pd.DataFrame: The preprocessed dataset.
    """

    # 1. Remove features with missing values
    df_clean = df.dropna(axis=1)

    # 2. Remove features with Pearson correlation to target below the threshold
    correlations = df_clean.corr()[target_column]
    important_features = correlations[abs(correlations) >= correlation_threshold].index
    df_clean = df_clean[important_features]

    # 3. Remove one feature from pairs that are highly correlated with each other
    corr_matrix = df_clean.drop(columns=[target_column]).corr().abs()
    upper_tri = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

    # # Find and drop columns that are highly correlated with another column
    to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > high_corr_threshold)]
    df_clean = df_clean.drop(columns=to_drop)

    return df_clean
```

Figure 1. Illustration of the Data Preprocessing Workflow

2.2.1. Web fishing dataset.

After executing the code illustrated in Figure 1, we successfully reduced the number of features from 88 to 47, resulting in a more manageable and accurate dataset.

2.2.2. Android malware dataset.

In the analysis of the Malware Android dataset, the only feature reduction we applied was the removal of features with missing values and non-numerical data. While we initially attempted to reduce features based on correlation to the target variable, this method did not yield useful results as it led to the elimination of all features, indicating that correlation was not a reliable criterion for this dataset. After performing these transformations, we ended up with a total of 40 features remaining for further analysis.

2.2.3. Network intrusion dataset.

In the analysis of the Network Intrusion dataset, we performed the same preprocessing techniaues in addition to including handling missing values with a SimpleImputer that allows to change NaN values to the mean. and addressing non-numerical data. We retained 19 key features.

3. Methodology and Results

3.1. Machine Learning Algorithms

- **Random Forest/Decision Tree**: Decision Tree algorithms operate by recursively splitting the dataset based on feature values that maximize information gain (or minimize impurity) at each node. The process continues until all data points are classified, or another stopping condition is met (such as a maximum depth). Random Forest enhances the performance of Decision Trees by creating an ensemble of multiple trees, each trained on a random subset of the data. It aggregates the results through a majority vote (for classification) or averaging (for regression), which helps reduce overfitting and improves overall accuracy.
- **Artificial Neural Network**: An Artificial Neural Network (ANN) consists of layers of interconnected neurons or nodes. The architecture typically includes an input layer, one or more hidden layers, and an output layer. Each neuron processes inputs through a weighted sum, applies an activation function (such as ReLU, sigmoid, or tanh) to introduce non-linearity, and passes the result to the next layer. Hidden layers capture complex patterns and relationships in the data, while backpropagation is used during training to adjust the weights by minimizing the loss function. The depth and width of the network, along with the choice of activation functions and learning rate, are critical for its performance.
- **Support Vector Machine/Regression**: Support Vector Machines (SVM) operate by finding the optimal hyperplane that separates data points of different classes in a high-dimensional space. The goal is to maximize the margin between the nearest points of different classes, known as support vectors. In cases where the data is not linearly separable, SVM can apply a kernel function (e.g., linear, polynomial, radial basis function) to transform the data into a higher-dimensional space, making it easier to classify. SVM is highly effective for classification, especially in high-dimensional spaces. Support Vector Regression (SVR) adapts this principle for regression

tasks, aiming to fit a function that predicts continuous values while minimizing the prediction error within a specified tolerance.

3.2. Parameter Variations

- **Parameter Tuning**: Parameter tuning plays a critical role in optimizing the performance of machine learning models by adjusting key hyperparameters. For Random Forest and Decision Tree, this includes varying the number of trees (n estimators), the maximum depth of the trees (max depth), and the criterion used for splitting nodes (Gini or entropy). In Artificial Neural Networks (ANNs), tuning involves selecting the optimal number of hidden layers and neurons per layer, choosing the appropriate activation functions (e.g., ReLU, sigmoid), and adjusting the learning rate to control the speed of convergence. Additionally, the number of training epochs and optimization algorithms such as Adam or SGD are explored to improve model accuracy. For Support Vector Machine (SVM) and Support Vector Regression (SVR), key parameters include the regularization term C, which balances margin maximization and classification error, and the kernel type (linear, polynomial, or RBF), which transforms the input data for better separability. The kernel coefficient gamma and, for SVR, the epsilon value (which controls the margin of error) are also important factors to consider. Efficient tuning of these parameters, often through techniques like grid search, ensures that the models are both accurate and generalizable.
- **Implementation Details**: In this project, we utilized scikit-learn and pandas as the core libraries for implementing and managing our machine learning workflows. scikit-learn was used for model development, including training and evaluating the Random Forest, Decision Tree, SVM, and Linear Regression models. It provided tools for parameter tuning through grid search, cross-validation, and performance metrics, ensuring accurate and efficient modeling. On the other hand, pandas was essential for data preprocessing, such as handling missing values, normalizing features, and performing correlation analysis. Its powerful data manipulation capabilities allowed us to clean and transform the datasets effectively, making them ready for analysis and model training. Together, these libraries streamlined the entire process, from data preparation to model evaluation.

3.3. Experimental Setup

After the data preprocessing outlined in the previous section, we employed **k-fold cross-validation** to rigorously assess the performance of our models. In this project, we used 10-fold cross-validation, which involves dividing the dataset into 10 subsets. The model is trained on 9 of these subsets and tested on the remaining one, with the process repeated 10 times. This approach ensures that each data point is used for both training and testing, providing a more reliable estimate of the model's accuracy.

We implemented this cross-validation method using **scikit-learn's** `cross_validate` function, which enabled us to track both the training and testing accuracy of the models. For example, in the case of our **SVM** model, we used a radial basis function (RBF) kernel and measured the accuracy over 10 folds. In addition to accuracy, other metrics such as standard deviation were captured to assess the model's consistency across different splits of the data.

This allowed us to assess the generalizability of the model and fine-tune it to achieve better performance.

3.4. Model performances

SVM/Regression

For the first dataset (web fishing) we used the Support Vector Machine (SVM) model with a radial basis function (RBF) kernel. The code utilizes 10-fold cross-validation to evaluate the model's performance on both the training and testing sets. Features from the dataset were first normalized using the `StandardScaler`, ensuring that all inputs are on the same scale, which is critical for SVM models. The `cross_validate` function was used to calculate accuracy scores across 10 different folds, and the results were visualized using a line plot, showing how train and test accuracy vary across the folds. The accuracy for training was around 91% while the test was between 83% and 87%, indicating a relatively good performance. Figure 2 shows the results. We tried next to improve the accuracy by changing the parameters C epsilon and kernel type. The results showed that the best accuracy came with these parameters: Kernel=rbf, C=10.0, Eps=0.1: Train R² (10-Fold): 95.12% Test R² (10-Fold): 85.35%

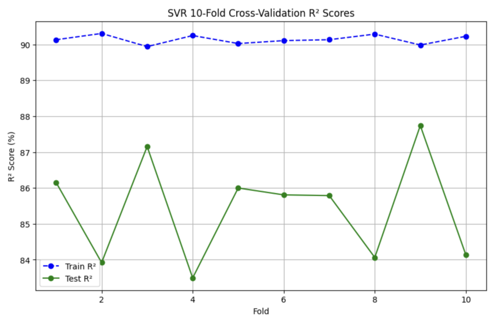


Figure 2. Illustration of SVM for web phishing

For the Android malware dataset, despite our efforts in preprocessing and reducing features, the dataset remained too large for the GPUs and CPUs of our machines to handle efficiently. To mitigate this, we decided to use only 20% of the 300k rows for our experiments. After training the model using SVM regression, the best accuracy we achieved was 30%, which suggests that this model is not well-suited for our dataset.

Using Support Vector Regression (SVR) on the third dataset yielded excellent results, particularly with the Radial Basis Function (RBF) kernel and other default parameters, including C=1.0, epsilon=0.1, and gamma='scale'. The model achieved R² scores between 92% and 93% for both the training and test datasets, indicating strong predictive performance and minimal overfitting. This demonstrates the suitability of SVR with RBF kernel for this dataset.

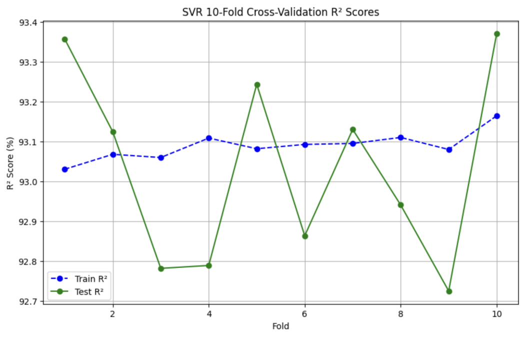


Figure 3. Illustration of SVM for Network intrusion dataset

After that we tried different variations of parameter but the first results seemed to be the most accurate.

Kernel and Parameters	Train R² (10-Fold)	Test R² (10-Fold)
rbf, C=1.0, Eps=0.1	92.89% ± 0.08%	92.79% ± 0.61%
linear, C=1.0, Eps=0.1	68.22% ± 0.40%	68.01% ± 3.65%
poly, C=1.0, Eps=0.1	91.10% ± 0.09%	65.83% ± 51.90%
rbf, C=10.0, Eps=0.1	93.69% ± 0.07%	93.53% ± 0.61%

TABLE 1. Comparison of SVM Results with Different Kernels and Parameters

Decision tree

The variation in Figure 4 uses a decision tree with a maximum depth of 5, entropy as the criterion for splitting, and a best splitter. A lower depth prevents overfitting, making this model simpler and more focused on general trends rather than fine-grained distinctions. The graph shows the training and test accuracies for each of the 10 folds. The training accuracy stays consistently around 0.9 across all folds, while the test accuracy fluctuates between 0.85 and 0.95. The two lines remain close to each other, indicating a good balance between training and test performance. The model shows stable performance with a high training accuracy and minimal variance in test accuracy. The small fluctuations in the test accuracy could be due to slight variations in the data splits but are within an acceptable range. Overall, this model performs well, with no significant signs of overfitting or underfitting. It offers a solution to phishing detection with good generalization across different data folds, making it suitable for real-world applications.

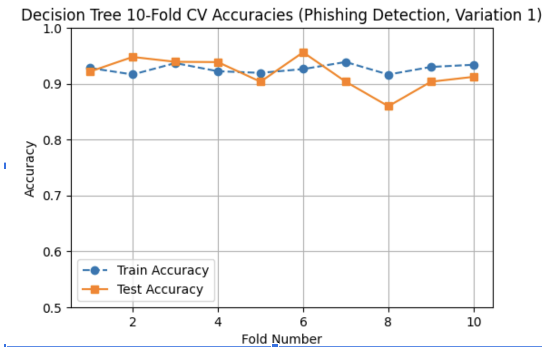


Figure 4. Illustration of Decision tree on phishing dataset (variation 1)

The variation in Figure 5 increases the maximum depth of the tree to 10 and uses the Gini index as the criterion for splitting. The deeper tree allows the model to capture more specific patterns in the data, potentially leading to improved accuracy, but with a greater risk of overfitting. The training accuracy remains close to 0.9 across all folds, similar to Variation 1, but the test accuracy shows more fluctuation, with a peak near fold 5 and a dip around fold 6. The gap between training and test accuracy widens slightly in some folds, indicating the potential for overfitting. While the model performs well, the increased variance in test accuracy suggests that it may be fitting more specific details of the training data at the expense of generalization. Although it maintains high accuracy, the test results show more sensitivity to different data splits. This variation performs comparably to Variation 1 but shows slightly more variance in test performance, which could indicate overfitting. Overall, the deeper tree captures more specific patterns, but the model's robustness may be reduced, making it slightly less reliable in predicting unseen data.

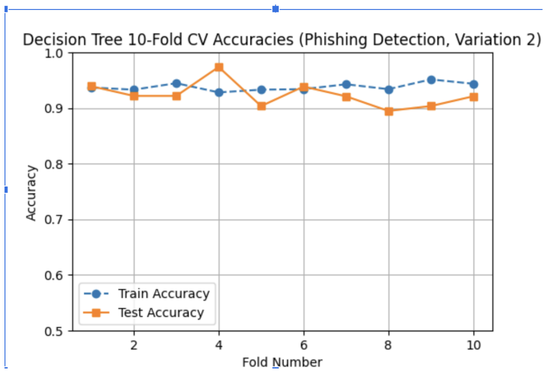


Figure 5. Illustration of Decision tree on phishing dataset (variation 2)

The final variation (Figure 6) further increases the maximum depth to 15 and introduces random splitting. The random splitting reduces bias in the splits, and the high depth allows the model to capture even more complex patterns.The training accuracy stays consistently close to 0.9, but the test accuracy fluctuates more, with occasional dips, particularly around fold 5. The gap between training and test accuracy is noticeable, with the test accuracy dropping below 0.9 in several folds. This model shows more variance in test accuracy compared to the previous variations, indicating potential overfitting. The randomness in splitting may introduce instability in performance, especially when combined with the deeper tree. In conclusion, variation 3 captures more complex patterns, but the random splitter and deeper tree result in greater fluctuation in test accuracy. Although the training performance remains strong, the test results suggest that this model may not generalize as well as the simpler variations, making it less reliable for phishing detection.

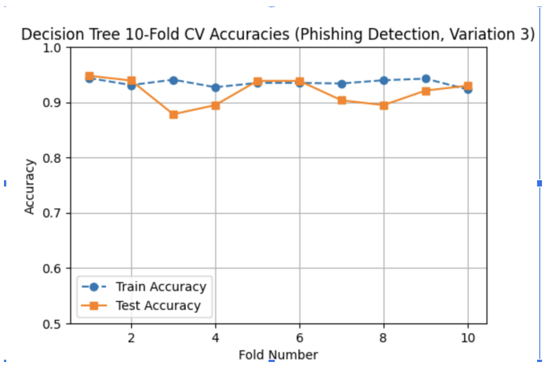


Figure 6. Illustration of Decision tree on phishing dataset (variation 3)

For the android malware detection we see the results below.

For the first variation the training and test accuracies are closely aligned, both remaining steady around 0.6 across all 10 folds. There is minimal fluctuation between the folds, and both lines stay very close to one another. While the model is stable, the overall accuracy is low, indicating that a tree with a depth of 5 struggles to capture the necessary complexity of the malware dataset. The lack of significant variance between training and test accuracy suggests no overfitting but also limited learning capacity. This variation underperforms in malware detection, with low overall accuracy across the board. Overall, the simple tree structure is likely insufficient for capturing this dataset, and the model does not generalize well to unseen data, suggesting the need for a more complex approach.

As for the second variation the training and test accuracies remain close, as in Variation 1, but there is slightly more fluctuation. Both accuracies hover around 0.6 to 0.65, showing minimal improvement over the previous variation. Despite the deeper tree, the overall accuracy remains low, with no significant increase in performance. The final variation (maximum depth 15) is similar to Variation 2, the training and test accuracies hover around 0.6 to 0.7.

In our last dataset, the network intrusion dataset, we implemented a Random Forest classifier with hyperparameters set to max depth=15, min samples split=10, and min samples leaf=5. This configuration

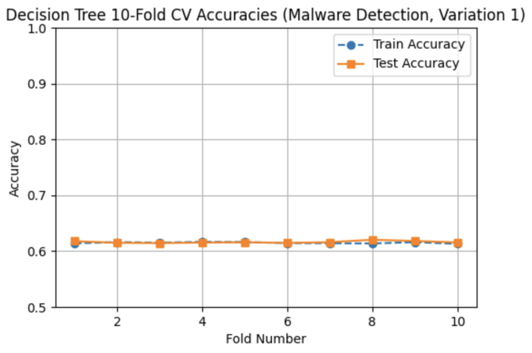


Figure 7. Illustration of Decision tree on Android Malware Dataset (variation 1)

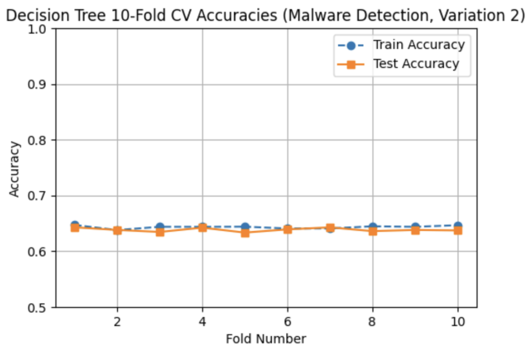


Figure 8. Illustration of Decision tree on Android Malware Dataset (variation 2)

yielded an impressive 100% training accuracy and a remarkable 99% testing accuracy. The near-perfect performance on both the training and test datasets indicates that the Random Forest model is highly effective for this specific application. The ability of the model to generalize well, maintaining such high accuracy on unseen data, suggests that it can reliably identify and classify network intrusion patterns, making it a robust choice for tackling security challenges in this domain.

Artificial Neural network

The results, as shown in graph below (Figure 9), utilizes a neural network architecture consisting of two hidden layers with 64 and 32 neurons, respectively, employing the ReLU activation function. The output layer utilizes a softmax activation function to handle multi-class classification. This architecture allows the model to learn complex patterns in the data while maintaining a manageable complexity to prevent overfitting.

In this variation (Figure 10), We implemented an Artificial Neural Network to analyze the phishing

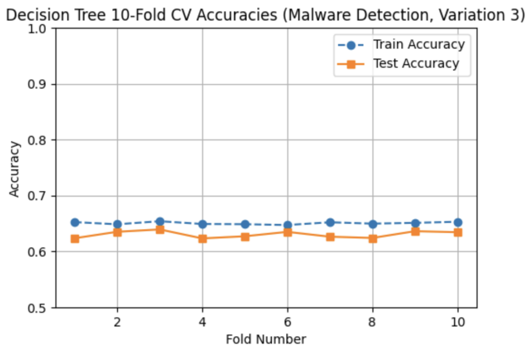


Figure 9. Illustration of Decision tree on Android Malware Dataset (variation 3)

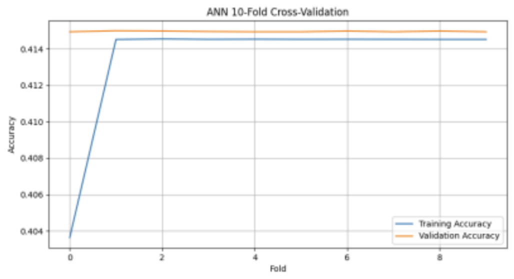


Figure 10. Illustration of Artificial Neural Network for Android Malware (variation 1)

dataset. The model consists of a single hidden layer with 64 neurons and employs the ReLU activation function to introduce non-linearity. The output layer utilizes a sigmoid activation function for binary classification, allowing the model to predict whether a URL is phishing or legitimate. This architecture was chosen to balance complexity and performance, aiming to capture the underlying patterns in the data effectively. Additionally, We increased the number of epochs to 50, providing the model with more opportunities to learn from the training data. This change enhances the model's ability to generalize to unseen data, though the results may vary based on the inherent characteristics of the dataset.

The results, as shown in Figure 11, use a neural network architecture with three hidden layers containing 128, 64, and 32 neurons, respectively. I decided to use the Leaky ReLU activation function in the hidden layers because it helps with the vanishing gradient problem, allowing the model to learn better during training. The output layer still uses the softmax activation function for multi-class classification. With this

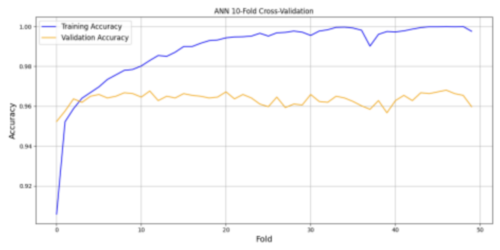


Figure 11. Illustration of Artificial Neural Network for Web phishing (variation 1)

new setup, the model has more capacity to learn complex patterns in the data since it can dig deeper with the extra hidden layer. We wanted to give the model a better chance to pick up on the different features in the dataset while keeping an eye on overfitting. Overall, this variation aims to improve the model's performance without getting too complicated.

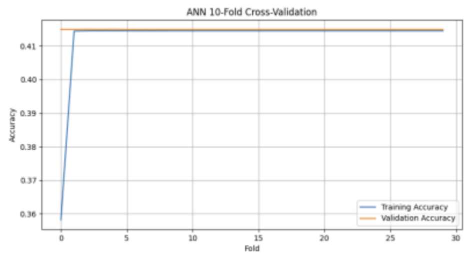


Figure 12. Illustration of Artificial Neural Network for Android Malware (variation 2)

On this next variation for Android Malware We changed the amount of neurons again and only had one hidden layer. Additionally, We increased the number of epochs significantly, moving from 10 to 50 epochs. This adjustment was made to give the model more time to learn the patterns in the data, allowing for a more thorough training process. By extending the training duration, We hoped to enhance the model's ability to generalize to unseen data without compromising its performance. You can see the figures for Android malware are all very similar with just slight differences but really not that much of a difference. Reducing the number of hidden layers and neurons simplifies the model. While this can help prevent overfitting, it may also limit the model's capacity to capture more intricate patterns in the data. If the dataset's complexity is high, a simpler model may not perform significantly better than a more complex one.

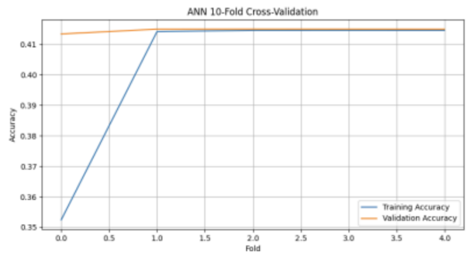


Figure 13. Illustration of Artificial Neural Network for Android Malware (variation 3)

For the next variation, We decreased the number of folds to simplify the cross-validation process, opting for just 10 folds. This change aims to reduce the computational load while still allowing for a robust evaluation of the model's performance. With fewer folds, each fold has a larger training set, which can help the model learn more effectively from the data. However, this adjustment might also lead to slightly higher variability in the accuracy results, as the model will be trained on fewer unique samples for validation in each fold. Since the model maintains its performance, it suggests that reducing the folds does not significantly hinder its learning capabilities, making the training process more efficient without sacrificing accuracy.

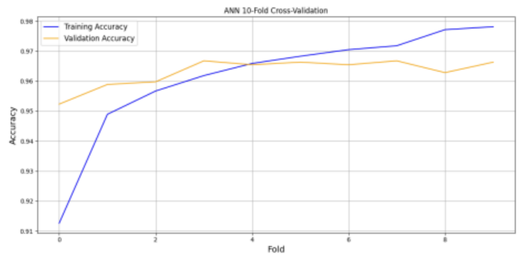


Figure 14. Illustration of Artificial Neural Network for Web phishing (variation 2)

In the next variation of the artificial neural network (ANN), We introduced dropout layers to the model architecture to combat overfitting, a common challenge in deep learning. Dropout randomly sets a fraction of input units to zero during training, which prevents the model from becoming overly reliant on any single feature and promotes more robust learning. Specifically, I increased the dropout rate to 0.4, effectively removing 40% of the neurons in the layer during each training.

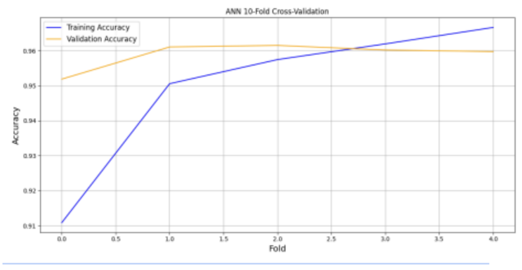


Figure 15. Illustration of Artificial Neural Network for Web phishing (variation 3)

3.5. Error Analysis

3.5.1. Web phishing dataset . In order to figure out why we only got 85% accuracy for the SVM model, we tried to visualize the errors made by the model on the test dataset. After training the SVM model, we made predictions on the test set and calculated the residuals, which represent the absolute differences between the true and predicted values. By analyzing these residuals, we could identify which samples the model struggled with. We further refined our analysis by defining a threshold for high-error samples; specifically, we categorized samples with residuals above the 80th percentile as high-error samples. This categorization helped us focus on the instances where the model's predictions were significantly off, providing insights into where the SVM might be failing.

To visualize the relationship between the features and the errors, we applied Principal Component Analysis (PCA) to reduce the dimensionality of the test dataset. PCA allows us to project high-dimensional data into a two-dimensional space while retaining as much variance as possible. We transformed the scaled test features into two principal components, which enabled us to create a scatter plot for easier interpretation.

In the resulting scatter plot, we highlighted the high-error samples in red and the normal samples in blue. This visual representation provided a clear overview of how the high-error samples were distributed in relation to the overall dataset. By examining the plot, we could intuitively assess whether the high-error samples formed distinct clusters or were scattered throughout the feature space. Such insights could guide further investigations into potential patterns or characteristics associated with the errors, ultimately leading us to improve the model's performance.

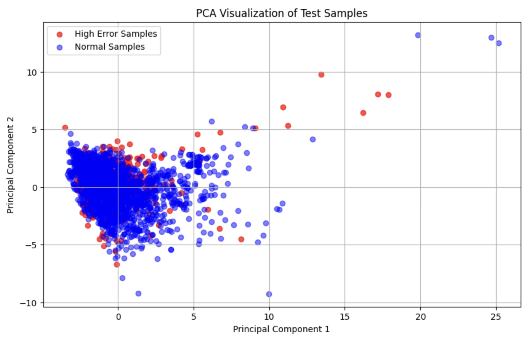


Figure 16. Illustration of errors in web phishing dataset classification

Some of the high error points are outliers on the top right corner but not a single clustering method could combine all the errors together.

3.5.2. Network intrusion dataset. for this dataset we got a 93% accuracy with SVM so just like the previous section we plotted the errors to visaulize and see if any patters standout.

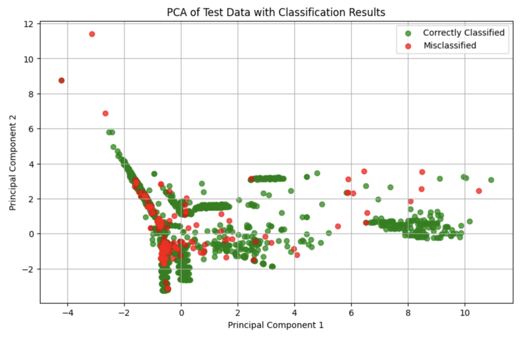


Figure 17. Illustration of errors in Network intrusion dataset classification

While outlier elimination might have contributed to a slight increase in accuracy—perhaps by 1%—the overall accuracy was already high, and removing these outliers would not have made a dramatic difference. This indicates that the model was quite robust even with the presence of a few noisy data points, and more advanced unsupervised techniques like clustering for anomaly detection wouldn't have brought substantial improvements in this case.

3.5.3. Android Malware. The Random forest/Decision tree classifier for this one didnt perform greatly and was only around 67% accuracy as per section 3.4. We decided to also do PCA then plot all test points to see if any patterns stand out.

We see that all the errors can be clustered together in the middle line which suggests a clustering could help eliminate the errors in the dataset and that there is a pattern of data that is not well suited for the model.

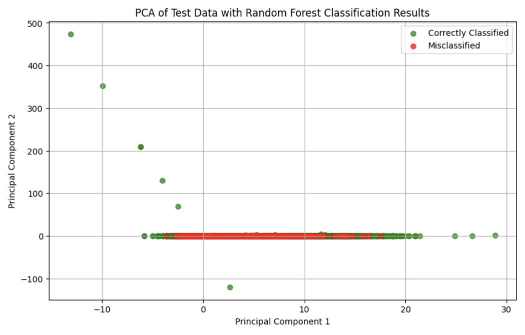


Figure 18. Illustration of errors in Malware dataset classification

4. Discussion

- **Best Performing Model**: The Support Vector Machine (SVM) performed well for 2 out of the 3 datasets evaluated, specifically excelling in the Phishing and Network intrusion datasets. In contrast, the Random Forest and decision trees model demonstrated strong performance primarily on the Network Intrusion Detection System (IDS) dataset. Meanwhile, the Artificial Neural Network (ANN) exhibited good performances except for the android malware dataset.
- **Limitations**: Despite the promising results, several limitations were identified in this study. A significant issue was the size of the datasets, which resulted in extended training times for the models. This limitation can hinder experimentation and the ability to fine-tune hyperparameters effectively. Furthermore, the Android malware dataset appeared to be poorly suited for all models, as the features and class distribution may not have aligned well with the assumptions of the algorithms used.
- **Parameter Impact**: The impact of varying parameters on model performance was significant across the different algorithms. For the SVM, tuning hyperparameters such as the kernel type and regularization parameter influenced its ability to find optimal decision boundaries, directly affecting classification accuracy. In the case of Random Forest, adjustments to parameters like the number of trees and maximum tree depth were found to impact both overfitting and the model's generalization ability. Proper parameter tuning for each model was essential to maximize performance, highlighting the importance of experimentation in developing robust predictive models.

5. Conclusion

In conclusion, the results of our experiments provide a clear comparison of how different machine learning models performed across the phishing detection, Android malware detection, and network intrusion datasets. For the phishing detection dataset, the SVM model with an RBF kernel was the top performer, achieving R^2 scores between 92% and 93% on both training and test sets. This indicates that the SVM model effectively balanced accuracy and generalization, making it well-suited for phishing detection without significant overfitting. For the network intrusion dataset, the Random Forest/Decision Tree model with a maximum depth of 5 demonstrated excellent results between 99 and 100%.

The ANN performed exceptionally well on the phishing dataset likely due to the numerical nature of the data and the ability of ANNs to capture complex patterns and relationships. However, the ANN struggled with the Android malware dataset. This dataset contained more complex and diverse features, including categorical and textual data, which made it challenging for the ANN to learn useful patterns effectively. The pre-processing steps used may not have been sufficient to transform the data into a format that the neural network could utilize optimally. Additionally, the structure and depth of the network might not have been well-suited for this particular dataset, leading to lower performance compared to the other datasets. Overall, while the ANN demonstrated strong capabilities with numerical datasets, adapting its architecture and preprocessing strategies for datasets with mixed data types or more complex structures remains crucial for enhancing its performance across different scenarios.

Across all datasets, our results found that deeper tree models, such as those used in variations 2 and 3 of the phishing detection dataset, led to increased accuracy on the training data but led to more fluctuation and potential overfitting in test accuracy. This suggests that simpler models might be more reliable for phishing detection. Meanwhile, for the Android malware dataset, increasing the tree depth did not yield significant gains, implying that the complexity of the model might have surpassed the needs of the dataset. In future analyses, employing more sophisticated techniques like parameter optimization and larger datasets could help address these limitations and provide even better generalization, particularly in datasets prone to overfitting like malware detection. Overall, the results illustrate the diverse performance of different models across datasets, with each model showing strengths that could be leveraged in specific scenarios.