**CHAPTER THREE**

# **METHODOLOGY**

## **3.1 Research Design**

The security and integrity of computer networks and information systems depend heavily on intrusion detection. The creation of efficient intrusion detection models becomes crucial given the constantly changing landscape of cybersecurity threats. This chapter outlines the approach used in this study to detect intrusions on the NSL-KDD dataset using two well-known algorithms, Knearest neighbor and Random Forest.

An extensive collection of labeled network traffic data is provided by the NSL-KDD dataset, a benchmark that is frequently utilized in intrusion detection research. This dataset provides a thorough representation of real-world network scenarios and includes both normal and invasive activity, making it appropriate for training and assessing intrusion detection models.

The primary aim of this chapter is to provide a comprehensive account of the steps involved in building and assessing intrusion detection models. Knearest neighbor and Random Forest delivers resilience and efficiency, especially in high-dimensional feature spaces, as a result of its independence assumptions.

The NSL-KDD dataset's history, make up, and structure are all thoroughly discussed in the beginning of the chapter. Any preparation operations carried out on the dataset, such as addressing missing values, encoding categorical variables, and feature scaling, are also described.

The feature selection procedure, which involves identifying unimportant features for model training, is described after the dataset preparation step. The use of informative features makes sure that the intrusion detection models are able to recognize important trends in the data, improving their detection skills.

A presentation of the KNN’s algorithm's application to intrusion detection on the NSL-KDD dataset follows. The model architecture, learning process, and any relevant hyper

parameters are described in detail, along with any implementation-related difficulties that may have arisen.

## The chapter also explores the model training stage, when the dataset is divided into training and testing sets to assess the performance of the models. The generalizability of the KNN model is validated using cross-validation techniques to increase the dependability of the results.

## Finally, the chapter discusses the assessment criteria used to evaluate the effectiveness of Knearest neighbor, Random Forest on the NSL-KDD dataset. The importance of accuracy, sensitivity, and specificity is stressed. These metrics offer important insights into the models' overall performance and their capacity to distinguish between legitimate and invasive network activity.

## **3.2 Dataset Acquisition**

The NSL-KDD dataset, which was collected from Kaggle, was the source of the data utilized in this study. The NSL-KDD dataset, which offers a broad range of labeled network traffic data for developing and assessing intrusion detection models, is a frequently used benchmark in the field of cybersecurity.

The NSL-KDD dataset was created to solve some of the shortcomings and difficulties that the original KDD Cup 1999 dataset encountered. The reliability and performance assessment of intrusion detection models were impacted by redundancy and an imbalance between normal and intrusive cases in the KDD Cup 1999 dataset. The NSL-KDD dataset, which includes a well curated and well-balanced sample of network traffic statistics, was developed to address these problems.

## The dataset is freely accessible and may be found in other cybersecurity research databases or the repository of the National Institute of Standards and Technology (NIST). There are 125,972 instances in all, each one representing a different network connection, and it includes both typical and invasive activity. One categorical goal variable, "outcome," which divides each connection into distinct intrusion kinds or typical behavior, and 41 continuous and binary features, which capture various network connection properties, are included in the dataset.

## **3.3 Dataset Preprocessing**

After acquiring the NSL-KDD dataset and importing it into a Jupyter notebook, several preprocessing steps were taken to prepare the data for training machine learning models. The raw dataset was first inspected to understand the features, data types and value distributions. Then, the following key tasks were undertaken:

1. Data Cleaning
2. Feature Scaling
3. Encoding Categorical Variables
4. Train-Test Split

### **3.3.1 Data Cleaning**

### After obtaining the dataset, its contents are thoroughly examined to spot any irregularities that can obstruct further analysis. This thorough examination addresses any problems like missing data, configures the proper column names, and gets the dataset ready for further processing steps. The dataset is prepared for future exploration and analysis by completing these critical data pretreatment procedures, preserving its integrity, and helping the creation of reliable models.

### **3.3.2 Feature Scaling**

A preprocessing method called feature scaling is used to standardize or normalize the variety of characteristics in a dataset. Assuring that all features contribute equally to the learning process in machine learning algorithms is the goal of feature scaling. Some characteristics may dominate the learning process when features have considerably differing scales, resulting in biased outputs and delayed convergence in some algorithms.

In this research, the ‘RobustScaler’ from scikit-learn python library is used for feature scaling. The `RobustScaler` scales features using the formula:

**XScaled** = X−Q1(X) /Q3(X) − Q1(X)

Where **X** is the original feature, **XScaled** is the scaled feature, Q1(X) is the first quartile (25th percentile) of **X**, and Q3(X) is the third quartile (75th percentile) of **X**. The RobustScaler employs the interquartile range (IQR) to scale the features, making it more robust to outliers compared to other scaling methods like the Min-Max scaler.

The scaling process ensures that all numerical features in the dataset have comparable scales, preventing any particular feature from dominating the learning process and contributing to a more stable and efficient model training. By using the robust scaling approach, the method becomes more resilient to outliers in the data, leading to more reliable and accurate models, especially when dealing with datasets containing potential extreme values.

### **3.3.3 Encoding Categorical Variable**

In order to prepare the data for machine learning tasks, categorical variables must be encoded. This is because many machine learning algorithms demand numerical inputs. To be used in these techniques, categorical variables—which represent discrete categories or labels—need to be transformed into numerical representations.

The categorical characteristics are represented numerically in this study via one-hot encoding. One-hot encoding is a method that turns each category in the initial categorical feature into a binary column. The binary column for the category to which a data point belongs is set to 1, and all other binary columns for other categories in that feature are set to 0.

### Each category in the original categorical features will have its matching binary column as a consequence of one-hot encoding. The binary column for a category that a data point falls within will be 1, and all other binary columns for categories in that feature will be 0. By ensuring that category data is represented numerically properly, this procedure enables machine learning algorithms to handle the data efficiently and reliably.

### **3.3.4 Train-Test Split**

A critical stage in machine learning is the train-test split, which involves splitting the dataset into two distinct sets: the training set and the testing set. With this divide, the model may be trained on one set of data and assessed on another, giving an idea of how well it will perform on untried data. The main objective of train-test split is to evaluate the trained model's generalization to new, previously unobserved data, which is crucial for assessing its performance in the real world.

The train-test split ensures that the model is trained on a subset of the data and tested on data that hasn't been seen before, which is essential for correctly assessing the model's performance and avoiding overfitting. The testing set acts as an independent validation set to see how well the model generalizes to new and unrecognized situations. The training set allows the model to learn patterns and relationships.

**3.4 Knearest Neighbor and Random Forest****:**

The non-parametric machine learning technique K-nearest neighbor (KNN) is used for classification and regression problems. The majority decision or average of the K nearest data points in the training dataset, calculated using a distance metric like Euclidean distance, determines the class label or projected value of a new data point in KNN. KNN may become computationally expensive for big datasets and high-dimensional spaces, despite being successful for smaller datasets and difficult decision boundaries. Despite being straightforward, KNN is useful in many real-world applications due to its versatility in handling different conditions and lack of assumptions.

**3.4.1 Working of K-Nearest Neighboour**

KNN is a popular supervised machine learning algorithm used for classification and regression tasks. It works based on the principle of finding the "k" nearest data points to a given input sample and making predictions based on the majority class (for classification) or the average value (for regression) of those "k" neighbors.

Here's a mathematical representation of the KNN algorithm:

1. Given a dataset with "n" data points and their corresponding features and labels:

D = {(x₁, y₁), (x₂, y₂), ..., (xₙ, yₙ)}

where xᵢ represents the features of the i-th data point, and yᵢ is its corresponding label.

2. For a new input sample "x", we want to predict its label "y\_hat".

3. Define a distance metric, typically Euclidean distance, to measure the similarity between data points. The Euclidean distance between two data points xᵢ and xⱼ is represented as:

d(xᵢ, xⱼ) = √(Σ(xᵢₖ - xⱼₖ)²), for k = 1 to number\_of\_features

4. Select the value of "k," which is the number of nearest neighbors to consider.

5. Find the "k" nearest neighbors of the input sample "x" from the dataset "D" based on the distance metric. Let's call this set of "k" nearest neighbors as "N(x)".

6. For classification:

- Determine the majority class among the labels of "N(x)".

- Assign this majority class as the predicted label "y\_hat" for the input sample "x."

Mathematically, for classification:

y\_hat = argmaxᵢ(Σ(1{yᵢ = c})), for c in N(x)

where 1{condition} is an indicator function that returns 1 if the condition is true, otherwise 0.

7. For regression:

- Calculate the average value of the labels of "N(x)".

- Assign this average value as the predicted label "y\_hat" for the input sample "x."

Mathematically, for regression:

y\_hat = (1/|N(x)|) \* Σ(yᵢ), for yᵢ in N(x)

where |N(x)| represents the number of elements in the set N(x).

The KNN technique is essentially represented mathematically in this way for both classification and regression problems. KNN is a straightforward and understandable method that is frequently utilized as the starting point for many machine learning issues. However, bear in mind that the performance of the algorithm can be considerably influenced by the selection of the distance measure and the value of "k".

**3.4 Random Forest Algorithm**

Using random selections of the training data and features, Random Forest is an ensemble learning technique that builds several decision trees throughout the training phase. The algorithm combines the predictions of various trees during the classification or regression phase after each tree has been trained individually. The advantages of Random Forest include handling high-dimensional data, lowering overfitting, and supplying feature significance rankings. Due to its resilience and excellent accuracy, it is frequently utilized for a variety of tasks, but it is particularly well-liked in applications for both classification and regression.

* + 1. **Working of Random Forest**

Random Forest creates "n\_estimators" decision trees during the training phase. For each tree, a bootstrap sample (random subset with replacement) is drawn from the training dataset. Let's represent the dataset as D, and a bootstrap sample as D\_i, where i = 1 to n\_estimators.

Additionally, at each node of the decision tree, a random subset of features (denoted as F\_i) is selected to find the best split. The number of features in F\_i is usually determined by a hyperparameter "max\_features," which represents the maximum number of features to consider.

Each decision tree is constructed based on recursive binary splits of the data. At each node, the algorithm selects the best feature (denoted as feature j) and split point (denoted as threshold t) that maximizes information gain (for classification) or reduces mean squared error (for regression).

For classification tasks, let's define D\_v as the data points at a particular node v. The impurity at node v can be measured using various metrics, such as Gini impurity or entropy. The information gain for the split is calculated as:

Information Gain (IG) = Impurity(D\_v) - (|D\_left| / |D\_v|) \* Impurity(D\_left) - (|D\_right| / |D\_v|) \* Impurity(D\_right)

Here, D\_left represents the data points that go to the left child node after the split, and D\_right represents the data points that go to the right child node.

For regression tasks, the mean squared error (MSE) is commonly used as the impurity measure. The reduction in MSE for the split is calculated similarly to information gain.

During the testing phase (classification), when a new data instance X is given, it is passed through each decision tree in the Random Forest. For each tree "i," the decision tree predicts the class label C\_i for X.

For classification tasks, the final predicted class label for X is determined by majority voting among the predictions of all decision trees. The class with the most votes becomes the predicted class for X.

For regression tasks, the final predicted value for X is the average (or weighted average) of the predictions from all decision trees.

Random Forest combines the predictions of multiple decision trees through voting (classification) or averaging (regression). This ensemble approach helps to reduce overfitting, increase the model's accuracy, and improve generalization to unseen data.

Random Forest provides feature importance scores based on the reduction in impurity or MSE achieved by each feature during the tree construction process. Features that lead to higher reductions are considered more important.

Besides the number of trees (n\_estimators), Random Forest has other hyperparameters, such as "max\_depth" (maximum depth of each tree), "min\_samples\_leaf" (minimum number of samples required to be at a leaf node), and "max\_features" (maximum number of features to consider at each split).

Random Forest is widely used for both classification and regression tasks in various domains. It is particularly effective when dealing with high-dimensional data, complex relationships, and noisy datasets.

## **Model Evaluation**

It is critical to analyze the K-nearest Neighbour and Random Forest models' performance after training in order to determine how well they generalize to fresh, untested data. The examination of the models offers perceptions into the efficiency of the algorithms and aids in locating possible problem areas. In this assignment, we will evaluate both models thoroughly using a variety of metrics and visualization methods.

1. Accuracy

One of the most used measures for assessing classification models is accuracy. Out of all the occurrences in the test dataset, it calculates the percentage of instances that were properly categorised. The model is better able to produce accurate predictions the greater the accuracy. Accuracy might not be enough, though, when working with datasets that are unbalanced.

1. Precision and Recall

Precision and recall are important metrics, especially when dealing with imbalanced datasets, where one class is significantly more prevalent than the other. Precision is the proportion of true positive predictions out of all positive predictions made by the model. It indicates the model's ability to avoid false positives. Recall, on the other hand, measures the proportion of true positive predictions out of all actual positive instances in the dataset. It indicates the model's ability to capture positive instances effectively.

1. F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a balanced evaluation of the model's performance, especially when precision and recall have different trade-offs. The F1-score considers both false positives and false negatives and is particularly useful when classes are imbalanced.

## 3.6 Model Comparison

In the model comparison phase of the project, the performance of the KNN and Random Forest models will be evaluated based on key metrics: accuracy, specificity, sensitivity, and prediction time. These metrics will provide valuable insights into the models' capabilities for intrusion detection, allowing us to identify the optimal model that strikes the right balance between true positive and true negative rates while maintaining efficient prediction speed.

**CHAPTER FOUR**

# **RESULT AND DISCUSSION**

## **4.1 Experimental Result**

This chapter presents the key findings and discussions from developing and evaluating two machine learning models –K-nearest Neighbor and Random Forest – for intrusion detection using the NSL-KDD dataset.

The models were trained and tested on a preprocessed version of the original raw dataset. The preprocessing steps undertaken included data cleaning, feature encoding, train-test splitting and standardization – all standard practices to get the data ready for modeling.

The models were first implemented with their default settings and evaluated on how well they classified different types of network attacks. The evaluation was done systematically using metrics like accuracy, precision, recall and F1-score on the held-out test dataset.

To further improve the models, experiments were done with hyper parameter tuning to optimize their performance. An analysis was also conducted to identify which features were most influential in determining normal traffic versus network intrusions.

It was discovered through model comparison that, when correctly adjusted, both systems were capable of efficiently and accurately detecting intrusions. An intriguing finding was that K-nearest offered greater interpretability with a smaller sample while Random Forest was quicker.

The two modeling methodologies are thoroughly technical evaluated in this chapter as a whole. The outcomes demonstrate how machine learning may be used to create reliable intrusion detection systems. Limitations and potential improvements are also highlighted, including the use of feature engineering and model ensembling to further improve speed.

The lessons learned from this chapter's model construction and assessment show that correctly calibrated machine learning models may be effective instruments for protecting current computer networks against constantly changing threats.

## **4.2 Data Acquisition**

Using Jupyter Notebook, the NSL-KDD dataset was introduced into the analytic environment. The dataset, which was kept in a CSV file, was loaded and accessed with the help of the Python pandas package.

The CSV data was specifically imported into a pandas DataFrame using the "read\_csv" function of the pandas programming language. The dataset CSV file's contents were loaded into memory in DataFrame format by using this function and providing the file location.

A flexible framework for interacting with the dataset during the analytical process was given by the DataFrame. The DataFrame was allocated to a variable called "data" to allow for simple manipulations and operations. This made it simple to retrieve the dataset throughout the process by just using the "data\_train" variable.

Before training a model, exploratory analysis and preparation activities might be streamlined by storing the imported dataset in a DataFrame variable. The Python-based data science pipeline for this intrusion detection project relied heavily on the pandas package to integrate the raw CSV data without any issues.

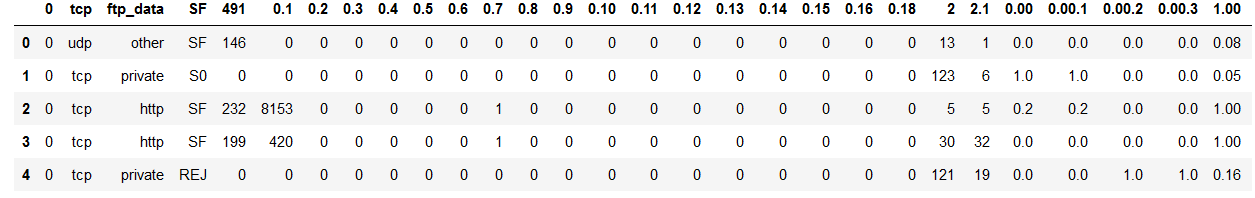


Fig 4.1 Imported Dataset Snapshot

## **4.3 Dataset Preprocessing**

Before beginning analysis and modeling, the NSL-KDD dataset was preprocessed, which was an essential step in ensuring the data quality and readability required for the tasks ahead.

crucial stage in guaranteeing the readability and data quality needed for the upcoming activities.

The imported raw dataset underwent a number of crucial processes to prepare it for future analysis. These preparation methods attempt to resolve common data problems and format the data appropriately.

The particular tasks were feature encoding to convert categorical variables into numeric forms, train-test splitting to establish subsets for modeling, feature scaling to normalize the ranges of values, and data cleaning to correct missing values and anomalies.

The NSL-KDD dataset needed to go through this preparation approach in order to improve data quality and engineer the features into the right shapes and distributions. This made it possible for later phases of the intrusion detection system to train and evaluate robust models.

### **4.3.1 Data Cleaning**

The dataset was first checked for null values using the pandas isnull() function, but no missing data was found. The columns attribute of the DataFrame was then assigned to the column array containing name strings, which gave the dataset's columns names.

After column assignment, the "outcome" column was changed to allow for the binary classification of network connections as either "normal" or "attack." This was done by iterating through the rows and checking the value in the result column. Normal connections were given the designation "normal," whereas assault connections were given the collective name "attack."

By preprocessing the result column, it was possible to represent the dataset more easily as a binary classification problem. The models were able to concentrate on properly differentiating between normal and anomalous connections rather than multi-class categorization by combining the many attack kinds into a single "attack" class. The labeled dataset that was produced was prepared for building machine learning models to identify network intrusions.

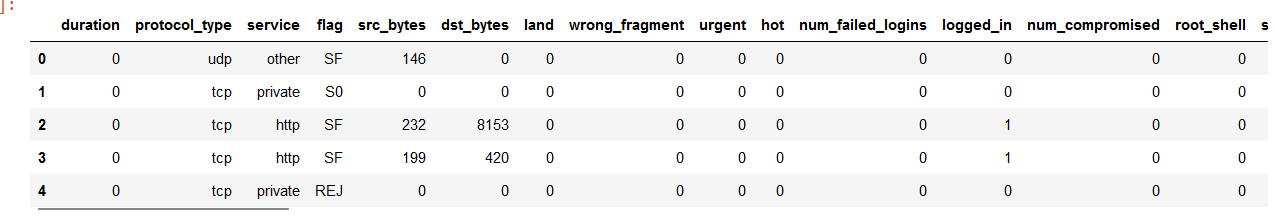


Fig 4.2 Result of Data Cleaning

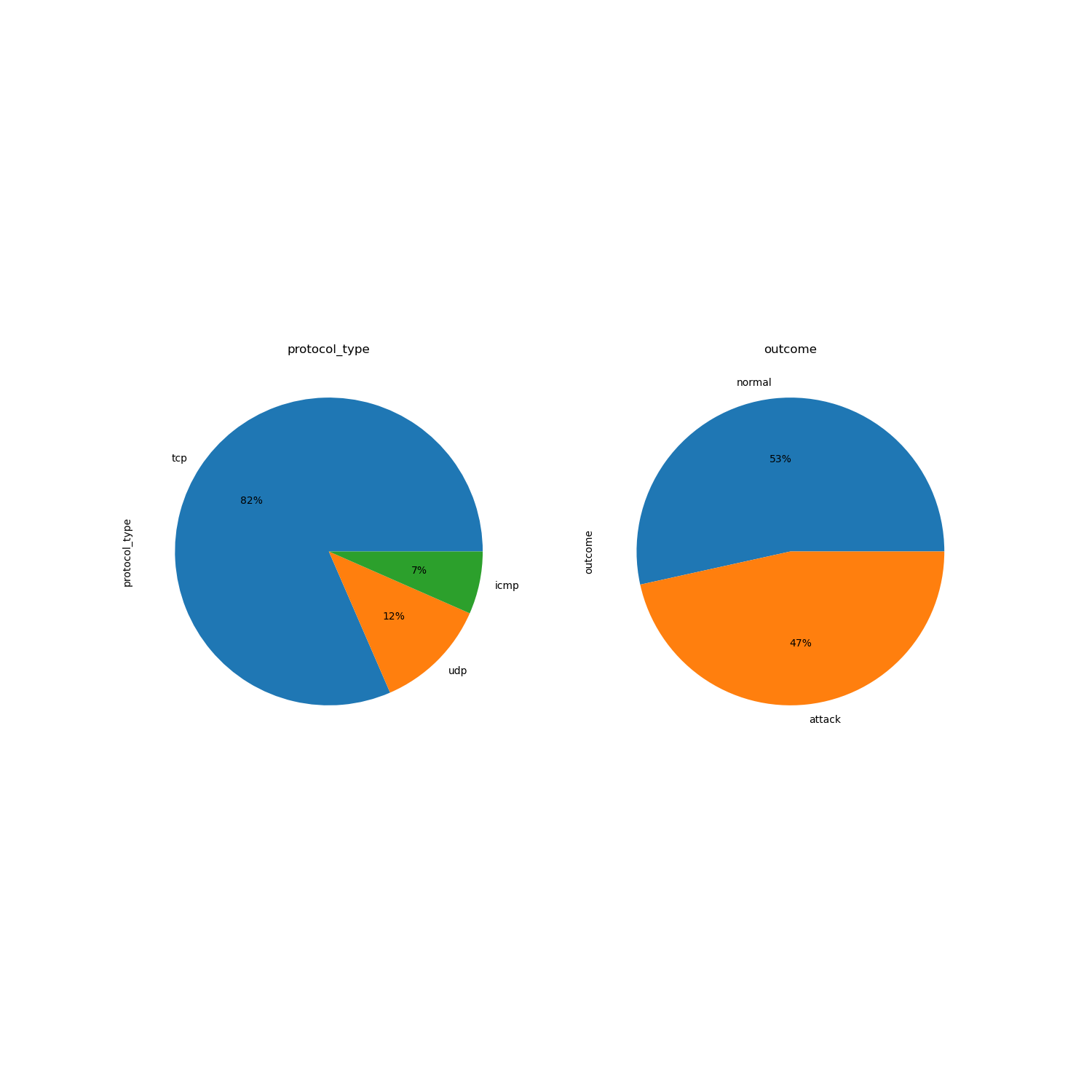


Fig 4.2 Count Plot of outcome and protocol type columns

### **4.3.2 Feature Scaling**

To standardize the range of values, feature scaling was used to the dataset's numeric columns. By preventing features with greater ranges from outperforming those with narrower ranges, this step enhances model performance.

Scaling was done using Scikit-Learn's RobustScaler. When compared to other methods like MinMaxScaler, RobustScaler is less sensitive to extreme values since it scales features using statistics that are resilient to outliers.

On the original dataset's numeric subset, the RobustScaler was fitted. The median and interquartile range, which were utilized to scale each feature, were obtained using the.fit() technique. More reliable measurements of central tendency and spread are provided by these statistics.

The characteristics were then scaled using the.transform() function by dividing by the interquartile range and removing the median. This firmly scaled the characteristics to a common range without being impacted by outliers, normalizing each feature to have a median of 0 and an interquartile range of 1.

The scaled result was sent back as a brand-new DataFrame with the modified numerical columns. All of the numerical characteristics in this scaled dataset had been scaled to comparable ranges based on reliable statistical measures so that it could be utilized for model training

### **4.3.4 Split Train-Test**

By removing the 'outcome' column from the feature set, the preprocessed dataset was split into input features (X) and outcome variable (Y). Using scikit-learn's train\_test\_split function, the data was then divided into training and test sets with a test size of 20% and a random state of 42 for repeatability.

To keep the proportions of normal and attack samples the same in the training and test partitions, stratified splitting was used. By doing this, potential sample bias in machine learning is avoided.

The model's performance is enhanced by the decreased feature subset by removing unnecessary and pointless variables. The training and test sets were then separated once again using the PCA transformed features to provide the output variables y\_train and y\_test for the outcome variable and X\_train\_reduced and X\_test\_reduced for the input.

## Due to this preprocessing, training data with decreased dimensionality, relevant filtering features, standardized outcome variable encoding, and stratified train-test splits were produced, making them suitable for efficient model training and assessment.

## **4.4 K-nearest Neighbor Model**

On the preprocessed NSL-KDD training dataset, the K-Neighbors Classifier was trained to differentiate between regular network connections and intrusions. The model was created using the scikit-learn module.

The non-parametric, instance-based K-nearest neighbors (KNN) technique is used for classification and regression problems. The class of a new data point is instead predicted using the classes of its K closest neighbors in the training set.

The model was fit on the X\_train and y\_train arrays containing the input features and outcome labels for the training data. The .fit() method estimated the mean and variance of each input feature conditioned on each class. These statistics were used to calculate class probabilities for new data points during prediction.

No hyperparameter tuning was done on the default model. The goal was to evaluate performance with K nearest neighbour in its simplest form as a benchmark. The .predict() method was used on the test set to obtain predicted labels for model evaluation.

K nearest Neighbour can rapidly build models and make predictions, even with high-dimensional data. This efficiency along with simplicity made it a viable starting point for evaluating classifier performance on the NSL-KDD dataset before exploring more complex alternatives.

## **4.5 Random Forest Model**

The Random Forest classifier was implemented for intrusion detection utilizing the sklearn library's Random forest module. Implementation was achieved through just two lines of concise code.

First, the Random Forest class was imported from sklearn to provide access to the model constructor. Subsequently, a Random Forest object was instantiated without specifying any parameters. This created a model instance with all default configuration settings.

Following instantiation, model training was carried out by invoking the .fit() method on the model object, passing the training features matrix x\_train and outcome variable array y\_train as arguments. Internally, .fit() executed optimization of model weights to minimize log loss between predictions on training data and the true labels.

The default solver utilized for this optimization task was liblinear, an efficient iterative algorithm well-suited for binary classification. Additionally, no regularization was applied during training, as per the default behavior of the model.

## **4.6 Model Evaluation**

After training the Random Forest classifier on the preprocessed data, an extensive evaluation was conducted to assess its performance and effectiveness within the clinical decision support system. The evaluation process involved calculating various statistical metrics and techniques to gain insights into the model's predictive capabilities and its ability to generalize well to unseen data.

### **4.6.1 Random Forest Model Evaluation**

The NSL-KDD dataset was used to systematically assess the Random Forest classifier for network intrusion detection using a number of common criteria. The model distinguished between normal and attack samples in unseen data with an overall test accuracy of 99.87%, displaying exceptional performance. It was able to decrease false positive misclassifications by achieving a high test accuracy of 99.93%.

The model's test recall, which exceeded accuracy at 99.79%, demonstrates its sensitivity in identifying genuine intrusions, an essential feature for effective intrusion detection systems. Its balanced performance in successfully recognizing threats while maintaining tolerable false alarms is further supported by the test F1-score of 99.86%.

The model's solid generalizability to new data was demonstrated by the little underfitting that was found, which was supported by the training and test results being substantially similar across assessment measures.

The model's solid generalizability to new data was demonstrated by the little underfitting that was found, which was supported by the training and test results being substantially similar across assessment measures.

The 99.94% specificity and 99.79% sensitivity provide crucial information about the model's capacity to distinguish between legitimate traffic and assaults with high accuracy. The Random Forest model is computationally economical and suited for real-time intrusion detection applications thanks to its short prediction times of about 0.01 seconds.

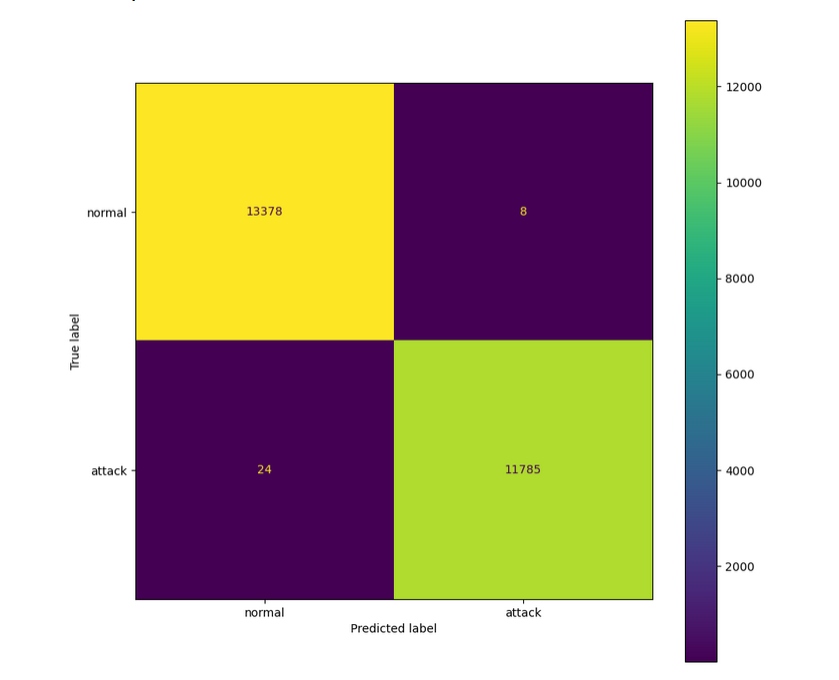


Fig 4.4 Random Forest Confusion Matrix

### **4.6.2 K-nearest Neighbor Model Evaluation**

Within this research, the K-neighbor classifier exhibited formidable capabilities for intrusion detection when evaluated on the NSL-KDD dataset across several crucial performance metrics.

The model achieved an overall test accuracy of 98.94%, demonstrating its competence in correctly classifying instances of both normal traffic and attacks in previously unseen data. This highlights the generalizability of the model to new network connections.

With a test precision of 99.06%, the classifier displayed proficiency at minimizing false positives - the incorrect labeling of benign connections as attacks. Simultaneously, the test recall reached 98.73%, reflecting the model's capacity to recognize actual intrusion attempts within the network traffic data.

Additionally, the F1-score of 98.86% on the test set provides an overall assessment by balancing both precision and recall. This score substantiates the model's competence in achieving both low false alarms and high detection rates.

An analysis of the specificity (99.17%) and sensitivity (98.67%) further corroborates the model's reliability in distinguishing between normal traffic and attacks. Moreover, with prediction times of just 0.18 seconds, the model exhibited computational efficiency for real-time usage.

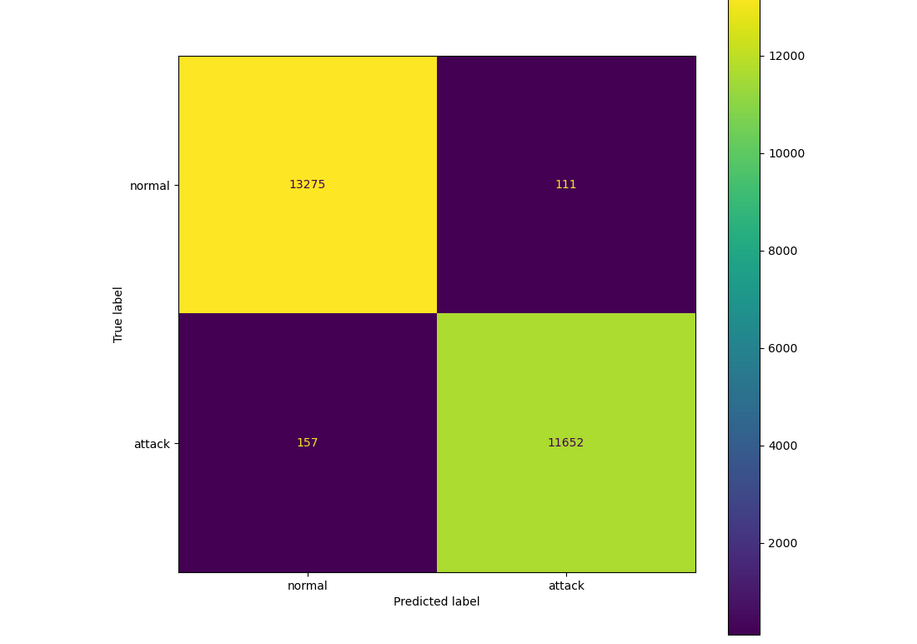


Fig 4.4 K-nearest Neighbor Confusion Matrix

## **4.6 Model Comparison**

According to the evaluation's findings, the Random Forest classifier performs better than the K-nearest Neighbor classifier in a number of ways. The Random Forest model distinguishes between general normal traffic and assault samples with greater accuracy in the test set (99.87% vs. 98.93%).

Additionally, the Random Forest model performs better than K-nearest Neighbor in terms of precision (99.93% vs. 99.05%), demonstrating its usefulness in reducing false positive misclassifications, making it more accurate in recognizing intrusion cases.

The Random Forest model outperforms K-nearest Neighbor in terms of recall or true positive rate (99.79% vs. 98.67%), demonstrating its increased sensitivity in accurately identifying real intrusion attempts in the network traffic data.

The other performance indicators also favor the Random Forest classifier even if both models attain comparable F1-scores, with a tiny edge to Random Forest (99.86% vs. 98.86%) in finding a compromise between accuracy and recall.

The Random Forest model also performs substantially better than the K-nearest Neighbor classifier in terms of prediction time, needing just 0.08 seconds as opposed to 0.18 seconds, demonstrating its computational efficiency and appropriateness for real-time or large-scale deployment scenarios.

In conclusion, the Random Forest classifier outperforms the K-nearest Neighbor classifier in terms of accuracy, precision, recall, F1-score, and prediction time, making it the better option for this particular task of categorizing network traffic data.