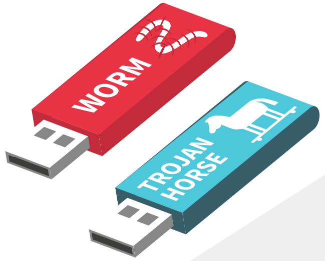
HIDM 2.0

Hybrid Intrusion Detection Model 2.0

Presented By :-

Aniket Kumar (2005288)
Pranav Pant (2005321)
Pratyush Yuvraj (2005469)

Introduction To Implementation



- Cybersecurity faces a myriad of intrusion threats, ranging attacks like Denial of Service (DoS) to threats such as malware.

- To address the multifaceted nature of intrusion threats, researchers and practitioners have explored hybrid intrusion detection systems.

- Our primary goal is to assess the performance of six different machine learning models: Random Forest, Extra Trees, Decision Tree, AdaBoost, Gradient Boosting, and Neural Network.

- The purpose is to identify each model's capacity, use various Optimization Methods, Ensemble them and explore scope and strengths in the context of intrusion detection.

Steps for Implementation

Step-01 : Import the Libraries

The code begins by importing libraries for data manipulation, machine learning tasks like pandas, scikit-learn etc.

Step-02 : Load the Data set

The pd.read_csv function from the pandas library is then used to read the dataset into a DataFrame named df.

Step-03 : Prepare the Data

The features (independent variables) are extracted into a DataFrame X, excluding the target variable, which is assigned to y.

Step-04 : Split into Train and Test

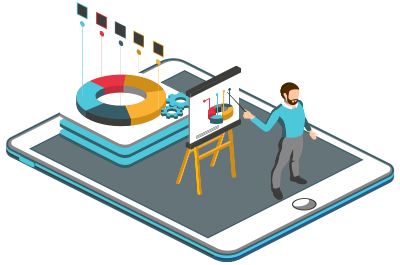
The dataset is divided into training and testing sets using the train_test_split function in 80:20 ratio

Step-05 : Creating and Training the Models

The following 6 Machine Learning Models were created and would be Trained over the 4 datasets separately and their performance will be observed over 8 evaluation Metrics one by one.

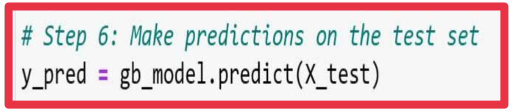
1. AdaBoost Model
2. Decision Tree
3. Random Forest
4. Gradient Boosting
5. Extremely Randomized Tree
6. Neural Networks

Further Steps for Implementation



Step-06 : Make Predictions on the Test Set

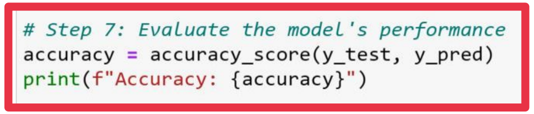
The trained model is used to predict the target variable for the test set (X_test). Predictions are stored in the variable y_pred.



```python
# Step 6: Make predictions on the test set
y_pred = gb_model.predict(X_test)
```

Step-07 : Evaluate the Model's Performance

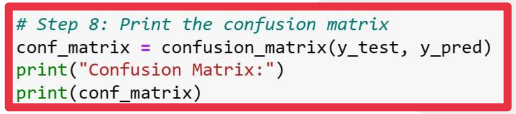
The accuracy of the model is calculated using the accuracy_score function from sklearn.metrics and printed.



```python
# Step 7: Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

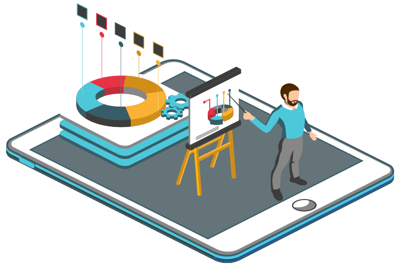
Step-08 : Print the Confusion Matrix

The confusion matrix is generated using the confusion_matrix function, providing insights into the model's classification performance.



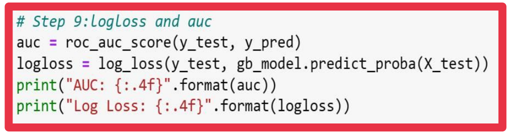
```python
# Step 8: Print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Further Steps for Implementation



Step-09 : LogLoss and AUC

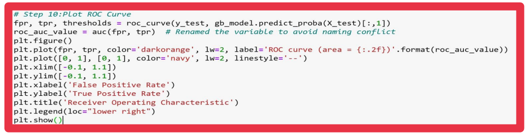
The Area Under the ROC Curve (AUC) and log loss metrics are calculated using roc_auc_score and log_loss.



```python
# Step 9:logloss and auc
auc = roc_auc_score(y_test, y_pred)
logloss = log_loss(y_test, gb_model.predict_proba(X_test))
print("AUC: {:.4f}".format(auc))
print("Log Loss: {:.4f}".format(logloss))
```

Step-10 : Plot ROC Curve

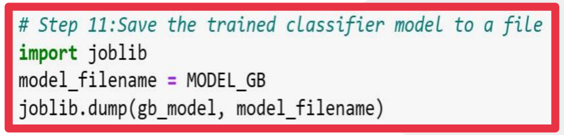
Generating the False Positive Rate (thresholds for the Receiver Operating Characteristic curve.



```python
# Step 10:Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, gb_model.predict_proba(X_test)[:,1])
roc_auc_value = auc(fpr, tpr)  # Renamed the variable to avoid naming conflict
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc_auc_value))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([-0.1, 1.1])
plt.ylim([-0.1, 1.1])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

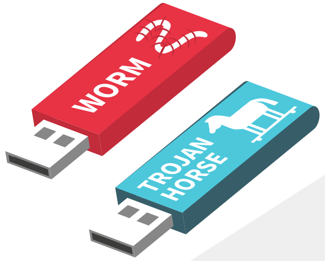
Step-11 : Save the Model to File

The trained model is serialized and saved to a file using the joblib.dump function. This step allows the model to be reused.



```python
# Step 11:Save the trained classifier model to a file
import joblib
model_filename = MODEL_GB
joblib.dump(gb_model, model_filename)
```

Introduction To Optimization

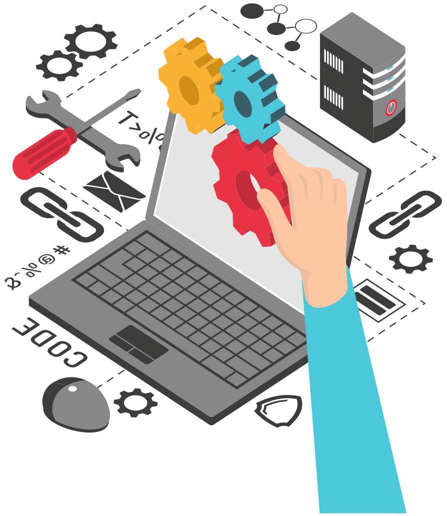


- In-depth investigation will provide insight on the complexities of model training, optimization, and ensemble strategies used to address the quirks of the NSL-KDD, SDN, and UNSW-NB15 datasets.
- We have undertaken the optimization of four models namely AdaBoost, Decision Tree, Random Forest and Extremely Randomized Tree.
- With various Optimization techniques namely Hyper Band, Grid Search CV, Gradient Based Optimization, Optuna and Bayes Optimization.
- With each of their results which includes the confusion matrix and ROC curve.Optimized result table of these 4 models and have also shown in the final model table.

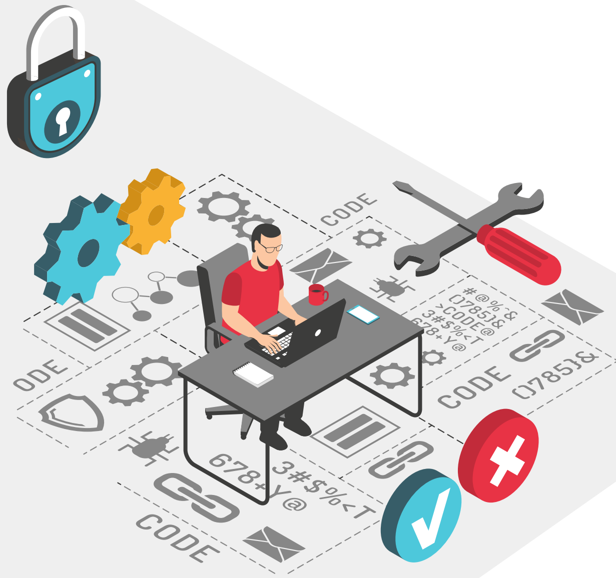
Optimization Techniques

Gradient based and Grid Search CV have only been used in Adaboost Model and Hyperband Optimization have been used in Adaboost, decision Tree, Random Forest and Extremely Randomized Tree Model.

Hyperband	Grid Search CV	Gradient Based
Hyperband is an optimization algorithm developed for hyperparameter optimization, particularly in the context of training machine learning models. The primary goal of hyperparameter optimization is to find the best set of hyperparameters for a given machine learning algorithm, leading to improved performance on a specific task.	Grid Search CV) is a hyperparameter tuning technique used to find the optimal hyperparameters for a machine learning model. Hyperparameters are parameters that are not learned during the training process but are set before training and affect the learning process.	Gradient-based optimization is used to find the minimum of a function by moving towards the direction of steepest decrease of the function. This process involves calculating the gradient of the function and updating the parameters in the direction opposite to the gradient.



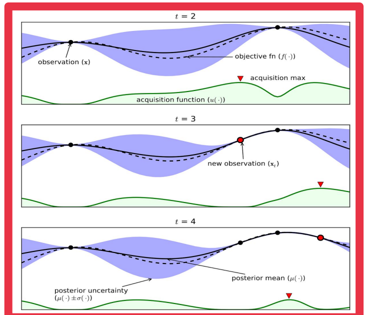
More Optimization Techniques



Optuna

Optuna is an open-source parameter optimization framework designed to automate search for hyperparameters of machine learning models.

Bayesian Optimization



Bayesian Optimization is a probabilistic model-based optimization technique that aims to find the maximum-minimum of an unknown objective function.

Note :: Optuna and Bayes were used in Decision Tree, Random Forest and Extremely Randomized Tree Models

Results After Optimization

The four fundamental models optimized models along with their algorithms’ results are given below, the best optimized model is highlighted and will be taken further in our study for this dataset.

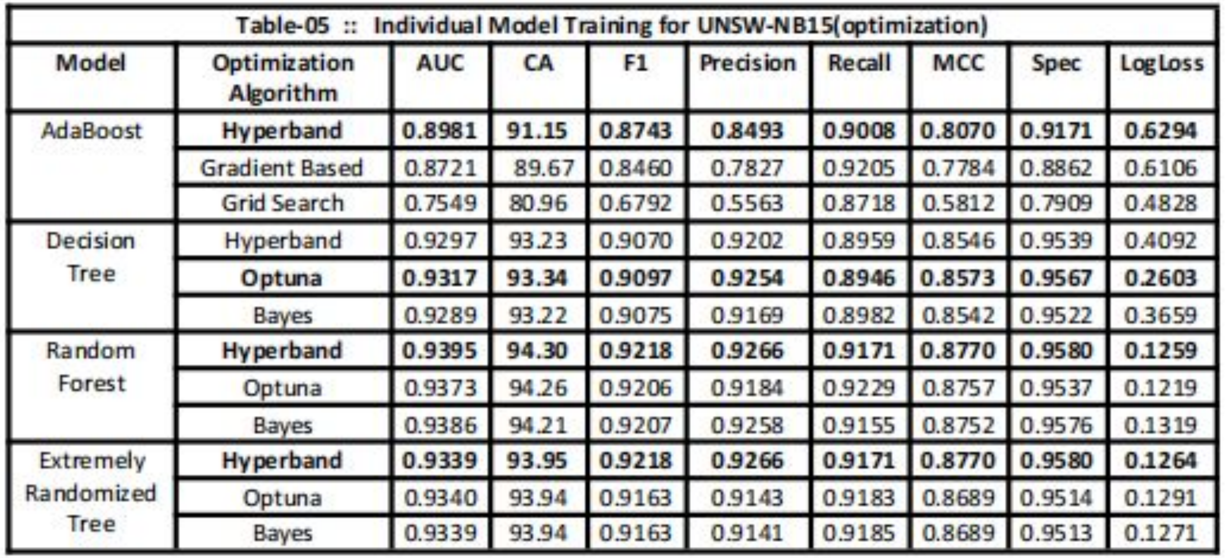
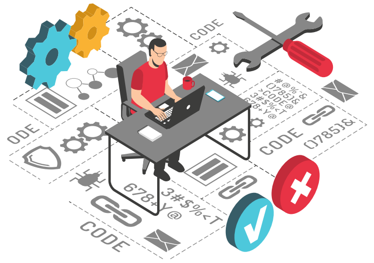


Table-05 :: Individual Model Training for UNSW-NB15(optimization)

Model	Optimization Algorithm	AUC	CA	F1	Precision	Recall	MCC	Spec	LogLoss
AdaBoost	**Hyperband**	**0.8981**	**91.15**	**0.8743**	**0.8493**	**0.9008**	**0.8070**	**0.9171**	**0.6294**
	Gradient Based	0.8721	89.67	0.8460	0.7827	0.9205	0.7784	0.8862	0.6106
	Grid Search	0.7549	80.96	0.6792	0.5563	0.8718	0.5812	0.7909	0.4828
Decision Tree	Hyperband	0.9297	93.23	0.9070	0.9202	0.8959	0.8546	0.9539	0.4092
	Optuna	**0.9317**	**93.34**	**0.9097**	**0.9254**	**0.8946**	**0.8573**	**0.9567**	**0.2603**
	Bayes	0.9289	93.22	0.9075	0.9169	0.8982	0.8542	0.9522	0.3659
Random Forest	**Hyperband**	**0.9395**	**94.30**	**0.9218**	**0.9266**	**0.9171**	**0.8770**	**0.9580**	**0.1259**
	Optuna	0.9373	94.26	0.9206	0.9184	0.9229	0.8757	0.9537	0.1219
	Bayes	0.9386	94.21	0.9207	0.9258	0.9155	0.8752	0.9576	0.1319
Extremely Randomized Tree	**Hyperband**	**0.9339**	**93.95**	**0.9218**	**0.9266**	**0.9171**	**0.8770**	**0.9580**	**0.1264**
	Optuna	0.9340	93.94	0.9163	0.9143	0.9183	0.8689	0.9514	0.1291
	Bayes	0.9339	93.94	0.9163	0.9141	0.9185	0.8689	0.9513	0.1271

Further Results after Optimization



After optimization, the new and better optimized models along with other models are given below and these are the final models which we used to make our hybrid model for the UNSW-NB15.

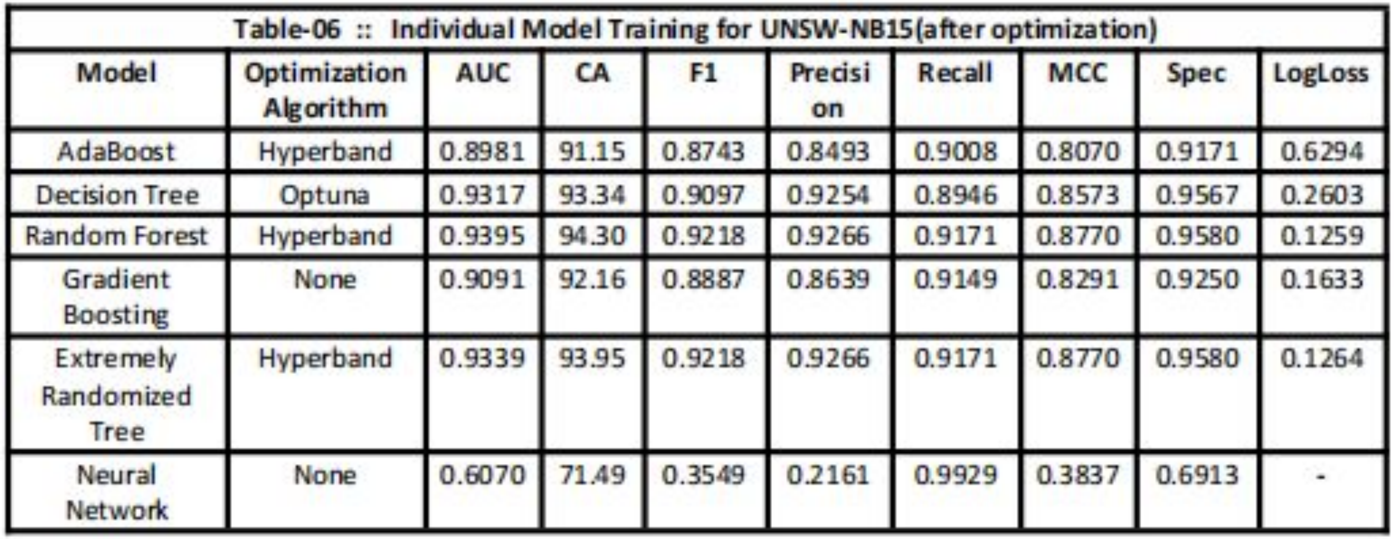
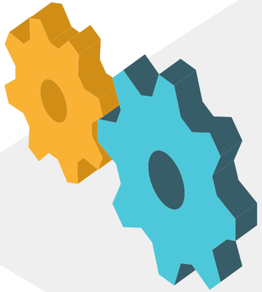
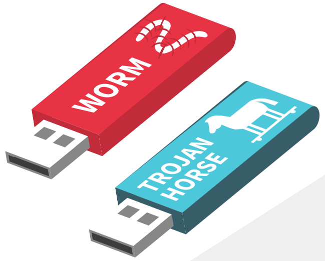


Table-06 :: Individual Model Training for UNSW-NB15(after optimization)

Model	Optimization Algorithm	AUC	CA	F1	Precision	Recall	MCC	Spec	LogLoss
AdaBoost	Hyperband	0.8981	91.15	0.8743	0.8493	0.9008	0.8070	0.9171	0.6294
Decision Tree	Optuna	0.9317	93.34	0.9097	0.9254	0.8946	0.8573	0.9567	0.2603
Random Forest	Hyperband	0.9395	94.30	0.9218	0.9266	0.9171	0.8770	0.9580	0.1259
Gradient Boosting	None	0.9091	92.16	0.8887	0.8639	0.9149	0.8291	0.9250	0.1633
Extremely Randomized Tree	Hyperband	0.9339	93.95	0.9218	0.9266	0.9171	0.8770	0.9580	0.1264
Neural Network	None	0.6070	71.49	0.3549	0.2161	0.9929	0.3837	0.6913	-



Introduction To Ensembling

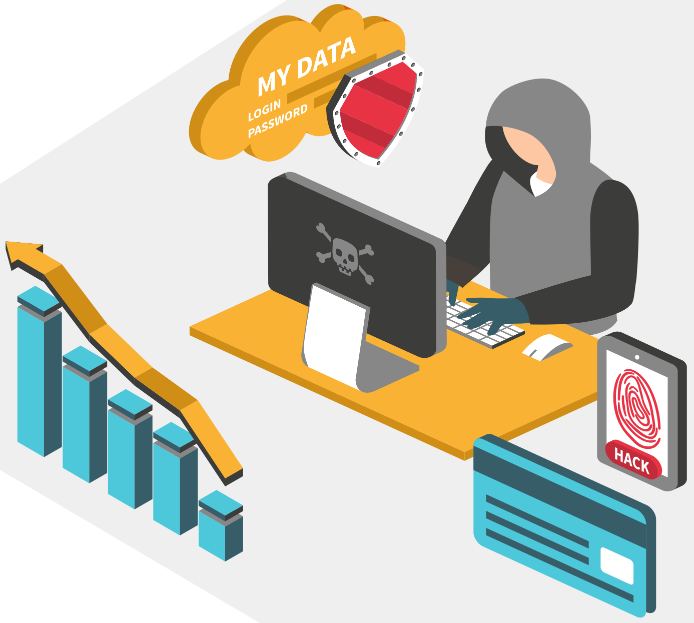


- Ensembling is a machine learning technique that involves combining the predictions from multiple models to improve overall performance.

- The idea is that by leveraging the strengths of diverse models, an ensemble can often achieve better generalization and robustness than individual models.

- Ensembling can be applied to a variety of machine learning models, ranging from simple models like decision trees to complex deep learning models.

- The choice of ensembling method depends on the characteristics of the data and the problem at hand. The result is a set of final predictions representing the consensus of the ensemble.



Ensembling Techniques

Hard Voting Method

Type of ensemble method used in classification tasks, where multiple individual models independently make predictions, and the final prediction is determined by a majority vote.

Each model in the ensemble "votes" for a class, and the class with majority of votes is final predicted class.

Soft Voting Method

Instead of each model in the ensemble casting a "hard" vote for a specific class, they provide a probability distribution over the classes.

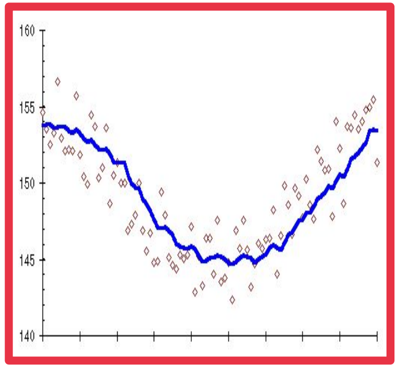
The final prediction is then determined by averaging the predicted probabilities from all models and choosing the class with the highest probability.

Dynamic Ensemble Method

Dynamic ensemble methods adaptively adjust the composition of the ensemble during the learning process based on the performance and diversity of individual models.

Dynamically incorporate or remove models to improve overall performance or address specific challenge.

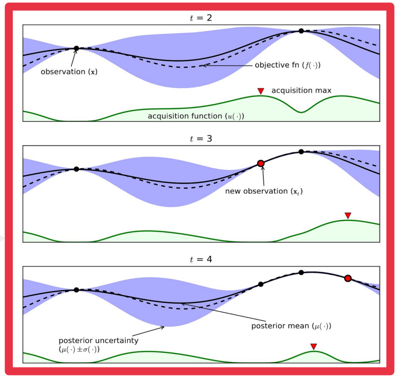
More Ensembling Techniques



Weighted Average

Type of average where different elements in the set have different weights, indicating their relative importance.

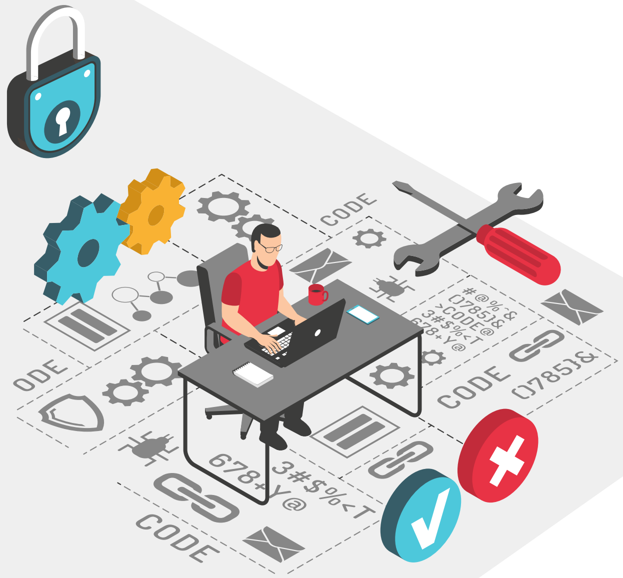
A weighted average involves assigning weights to each model's prediction based on some criteria and then combining the predictions accordingly.



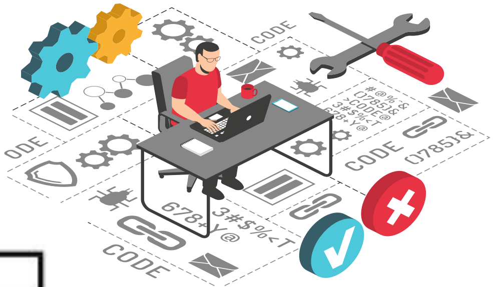
Bayesian Average

A technique used to combine predictions from multiple models by taking into account the uncertainty associated with each model's prediction.

It's based on Bayesian probability theory and provides a principled way to handle model uncertainty.



Final Results and Conclusion



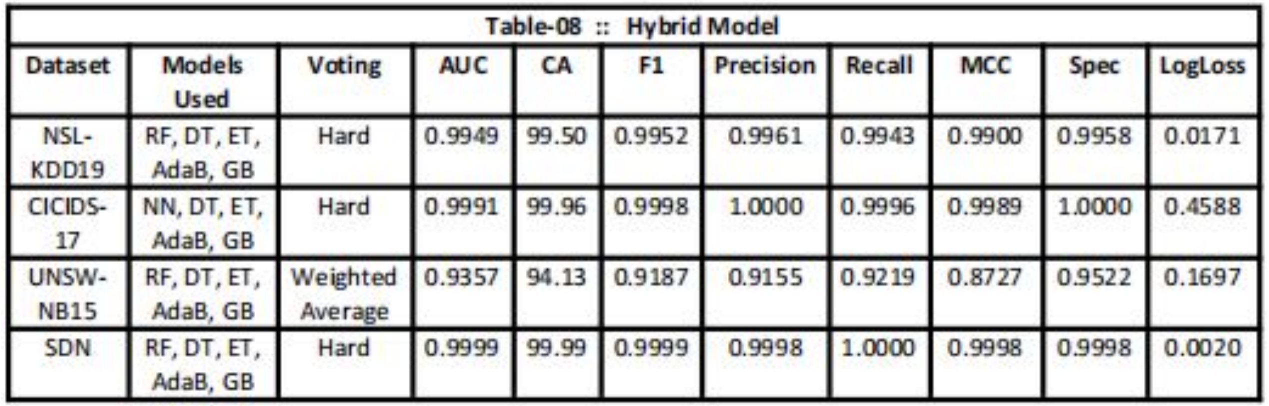
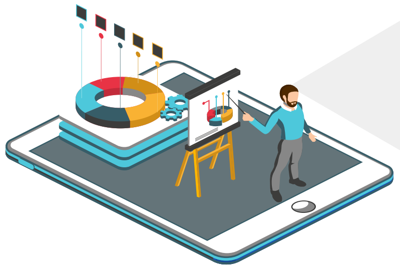


Table-08 :: Hybrid Model

Dataset	Models Used	Voting	AUC	CA	F1	Precision	Recall	MCC	Spec	LogLoss
NSL-KDD19	RF, DT, ET, AdaB, GB	Hard	0.9949	99.50	0.9952	0.9961	0.9943	0.9900	0.9958	0.0171
CICIDS-17	NN, DT, ET, AdaB, GB	Hard	0.9991	99.96	0.9998	1.0000	0.9996	0.9989	1.0000	0.4588
UNSW-NB15	RF, DT, ET, AdaB, GB	Weighted Average	0.9357	94.13	0.9187	0.9155	0.9219	0.8727	0.9522	0.1697
SDN	RF, DT, ET, AdaB, GB	Hard	0.9999	99.99	0.9999	0.9998	1.0000	0.9998	0.9998	0.0020

The meticulous evaluation of the hybrid model showcased promising advancements in intrusion detection capabilities. The ensemble model demonstrated improved performance metrics, surpassing the individual models in terms of accuracy, precision, recall, and F1 score. This achievement substantiates the efficacy of the hybrid approach in enhancing the overall detection accuracy and reducing false positives.

FUTURE SCOPE



Optimization of Computational Efficiency

Future research can focus on optimizing the computational efficiency of the hybrid ensemble models and Algorithms.

Integration with Deep Learning Architectures

Explore the potential benefits of incorporating deep neural networks to capture intricate patterns in network traffic data.

Scalability in Big Data Analytics

Model can efficiently scale to handle massive volumes of network data, making it applicable and effective in large-scale data environments.

Real-Time Threat Intelligence

Enhance the system's ability to adapt to the latest threats by incorporating up-to-the-minute information about malicious entities.

REFERENCES

[1] HIDM : A Hybrid Intrusion Detection Model for Cloud Based Systems by Lalit Kumar Vashishtha, Akhil Pratap Singh and Kakali Chatterjee, September 2022

[2] A Detail Analysis on Intrusion Detection Datasets by Dr Santosh Kumar Sahu, Sanjay Kumar Jena and Saurav Ranjan Sarangi, February 2014

[3] Hybrid Intrusion Detection System using Machine Learning by Amar Meryem and Bouabid EL Ouahidi, May 2020

[4] A Hybrid Intrusion Detection System based on ABC-AFS Algorithm for Misuse and Anomaly Detection by Vajiheh Hajisalem and Shahram Babaie, February 2018

[5] Improving the Performance of Machine Learning-Based Network Intrusion Detection Systems on the UNSW-NB15 Dataset by Soulaiman Moualla, Khaldoun Khorzom and Assef Jafar, June 2021

THANK YOU !!

Presented By :–

Aniket Kumar	(2005288)
Pranav Pant	(2005321)
Pratyush Yuvraj	(2005469)