Insider Threat Detection

Kriti Garg, (B.Tech 2y, IIT Roorkee)

Abstract: Significant risks from insider attacks include financial losses, financial loss, intellectual property theft, and reputational harm to the firm. The large false positives and errors of current detection techniques are caused by heterogeneous data and overfitting problems. In order to improve feature extraction—which is essential for detecting insider threats—this paper presents the TabNet classifier, which is intended for tabular data. Preprocessing the data and using Tab2Img to turn it into image representations are part of the research. Moreover, GCN Conv layers are used in Graph Neural Network (GNN) models to analyze adjacency matrices and node characteristics, enhancing the network's ability to identify unusual behaviors that may be signs of insider threats.
This comprehensive approach aims to bolster cybersecurity measures against insider attacks, addressing critical vulnerabilities in organizational security.

Introduction: An insider threat is any user or individual with authorized permission to access physical or virtual corporate assets. Insider threats hold credentials that, if abused, can potentially violate the trust of the organization. Different risks might be due to careless employees, inside agents ,malicious insiders. On the technical front, Existing approaches are inadequate in accuracy and precision. They suffer from a high false positive rate.

To address these challenges, we leverage the CERT v4.2 dataset, a rich repository of system events and user activity logs from Carnegie Mellon University's CERT Coordination Center. This dataset provides crucial insights into user behaviors within corporate networks, essential for identifying anomalous activities indicative of insider threats. Our approach employs TabNet, a specialized deep learning architecture for extracting meaningful features from structured data. TabNet enables us to preprocess and transform complex, heterogeneous datasets into insightful representations. Techniques like Tab2Img further enhance our analysis by converting tabular features into image formats. For model training and classification, we utilize Graph Neural Networks (GNNs) with GCN Conv layers. These models excel in capturing intricate dependencies within networked data, thereby improving our ability to detect subtle insider threats effectively. Typically, the process involves setting up data mining and classification, as well as user profiling. Then you can create a behavioral model that continuously monitors and analyzes the operation and generates insights, prompts alerts, and performs actions that defend your network, systems, and assets.

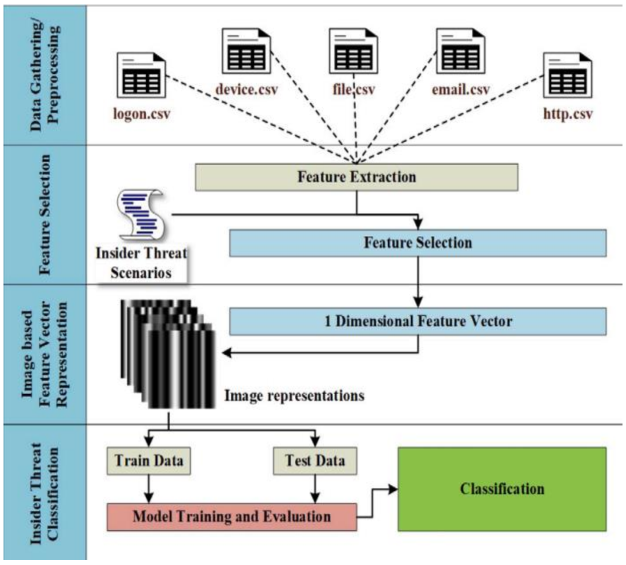
Literature Review: The analysis of insider threat detection revealed that model performance is highly dependent on the data used.Based on the available data, we have approaches varied for different types of data.
Sequential Data: This includes the *time-series data.*It uses both ML and DL techniques.
Non-Sequential Data: It comprises of the image data to identify insider threats. They are possible using Neural Networks.
On CERT r5.1, the models achieved a high success rate (75% detection) with minimal false alarms, although missing a specific insider type (scenario 2). However, for CERT r6.2 with a different organizational structure, accuracy suffered. While user-week data struggled to identify most insider threats, user-session data showed promise with good detection rates. To improve accuracy on CERT r6.2, the report suggests using Random Forest (RF) or Artificial Neural Network (ANN) models with user-day data. The two main sources are user activities , organization structure and user profile information.

For handling imbalances in the dataset, we use algorithms such as SMOTEENN(Synthetic Minority Over Sampling Technique and Edited Nearest Neighbours). SMOTE is for oversampling the minority classes(it increases the number of samples) ad ENN is used to remove miss-classified instances from all the classes.In Machine Learning, they include:

(a) Class Weights: It adjust weights to optimize the output. Higher weights are assigned to minority class instances so while backpropagation, it is all rectified.
(b) Ensemble Methods: Clustering based Subset ensemble learning technique. Clusters majority class samples into a fixed number of groups, subsequently reduce the number of majority class samles. Prediction from an ensemble of classifiers decision trees, Naive Bayes and SVM.
(c) Algorithm Level Adjustments
(d)Cost Matrices: Explicitly define the misclassification cost for different classes

This enables better accuracy. Then feature extraction takes place. For *image based approaches*, where *WCNN is used*, 1D feature vector is converted into image feature vector. This is done using mat2grey feature present in MATLAB. For sequential data, *LSTMs are used along with CNNs*. Talking about ML Algorithms, among *supervised ones*, *Random Forests* give the best

results.On Data Granularity , user session data appears to give the best results. This is because it allows a system with higher malicious insider detection rate and faster response time.
RF can be employed where manpower for investigating the alarms is limited, as it gives high precision.
On the other hand, ANN gives higher malicious insider detection-rate.



[Ref:1]

There are multiple techniques to handle sequential/non-sequential data, there are multiple techniques including CNN. But with CNN, one problem was observed. Due to its pooling operations, it loses user behavioral data important for detection.

To rectify this problem, WCNN were used. WCNN are *Wavelet Convolution Networks.* They are better because they have infinitely strong and generalized convolution and pooling layer prior to losing important user information before detection. It uses both spectral and spatial analysis for the same. *WCNN* consists of *wavelets, convolution layers and pooling layers.* CNNs use spatial features while the spectral analysis uses invariant features. Thus, both spectral and spatial features are the art of a single model. They can filter and downsample image data in the frequency domain. Connect Convolutional and Pooling layers for *multi resolution analysis* .Pooling Layer is added immediately after convolution layers and average pooling provides better texture features extraction.

WCNN is based on VGG-19 Network. The WCNN takes image representations as input, passes them from a sequence of computational units that produce representations with a relevant score for classifications.

WCNN Architecture:
Based on VGG-19 Network.
Uses convolution layers with 3x3 kernels and extended stride to minimize the size of the feature maps instead of pooling layer.The global average Pooling Layer is put before fully linked layers. In a nutshell, the training model consists of generalized convolution and pooling layers. There are four decomposition levels(as wavelets for Multi-resolution analysis).

Summary of the heterogeneous log files.

Dataset CMU - CERT r4.2 Details	No. of instances
Duration for record collection	17 months
Number of Users	1,000 Users
	(930 Non-malicious, 70 Malicious)
Logon.csv	8,54,859
Email.csv	26,29,979
File.csv	4,45,581
Device.csv	4,05,380
Http.csv	2,84,34,423
Total number of events	3,27,70,222
Preprocessed Data	
Non-malicious events	71,54,815
Malicious events	7,323
Total instances	3,30,453
Non-malicious instances	3,29,487
Malicious instances	966
Total features selected	20
Balanced Classes	
Non-Malicious instances (70% Data)	2,27,455
Malicious instances (70% Data)	2,30,618
Non-Malicious instances (80% Data)	2,60,059
Malicious instances (80% Data)	2,63,756

The analysis involves processing log data from multiple CSV files into user-session, user-week, and user-day formats. Feature extraction includes frequency (e.g., emails sent) and statistical features (e.g., email size). Imbalance is handled using SMOTEENN (over-sampling minority class and removing misclassified instances) and cost-sensitive learning techniques like class weights and ensemble methods.

Various machine learning algorithms are utilized for insider threat detection. Logistic regression estimates event probabilities using a sigmoid function and employs an lbfgs solver for efficient training. Multi-layered neural networks optimize learning with Adam optimizer and backpropagation, while random search fine-tunes hyperparameters. Random forests, composed of decision tree ensembles, enhance accuracy by training on random subsets of features and data points, with hyperparameters like tree count, features per tree, and maximum depth optimized. XGBoost sequentially builds decision trees to minimize errors, utilizing combined loss function and regularization. Hyperparameter tuning, including random search with cross-validation, maximizes algorithm performance.

Logistic Regression excels in detecting malicious insiders (high recall) but generates numerous false alarms (low precision), impacting analyst efficiency. Random Forest and Artificial Neural Networks offer improved balance, maintaining strong detection rates with fewer false positives. Notably, Random Forest achieves superior precision and lower false alarm rates compared to other methods.

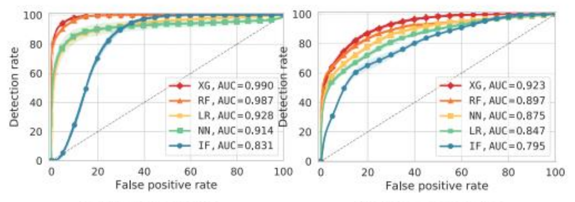
Unsupervised ensemble methods play a key role in anomaly detection for insider threat analysis. *Auto encoders (ANN)***:** These neural networks compress data for dimensionality reduction and learn useful representations during reconstruction. *Isolation Forest:* This ensemble method isolates anomalies by randomly splitting data points, making it effective for anomaly detection. *Light Weight on-line Detector of Anomalies* This combines weak anomaly detectors for different features to create a stronger overall detector. *Local Outlier Factor (LOF):* This unsupervised method identifies anomalies by comparing local data density around each point. Points with significantly lower density are considered outliers.

Accuracy: This metric reflects the overall effectiveness of the model in classifying instances correctly. It's calculated by dividing the sum of True Positives (correctly identified malicious cases) and True Negatives (correctly identified normal cases) by the total number of instances (both malicious and normal).

Precision: This metric focuses on the *positives* the model predicts and measures how many of those predicted malicious cases are actually true positives. It's calculated by dividing the number of True Positives by the total number of instances the model labeled as malicious (True Positives + False Positives).

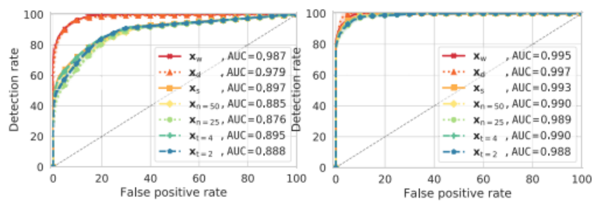
Recall: This metric emphasizes the model's ability to capture all malicious instances. It's calculated by dividing the number of True Positives by the total number of actual malicious cases in the dataset (True Positives + False Negatives)

AUC-ROC Curve (Area Under the Receiver Operating Characteristic Curve): This is a visualization metric that helps assess the model's performance across different classification thresholds. It plots the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis. A higher AUC value (closer to 1) indicates better overall classification performance. The ROC curve allows us to understand the trade-off between true positives and false positives at different thresholds.



(a) User-week data (b) User-session data

[Ref:2]



(a) Instance-based (b) User-based

[Ref:2]

Proposed Methodology: This paper has used combined deep learning and traditional machine learning approach. It starts by cleaning and preparing data from the CERT v4.2 dataset. TabNet, a deep learning model adept at handling complex tabular data, is used for feature extraction and classification. GNNs are then employed to analyze structured data and identify suspicious behavior. Feature transformation techniques like Tab2Img and incorporates traditional machine learning for comparison have been used. This is helpful in reducing false positives, ultimately enhancing insider threat detection capabilities.

Tabnet is used because of the following features:

Tailored for Tables**:** TabNet excels at high-dimensional user data (like activity logs) commonly used in insider threat detection.

Focuses on Important Features**:** It learns which user activities are most indicative of threats and prioritizes those during analysis.

*Interpretable Feature Selection***:** TabNet highlights the user actions that most influenced its decisions, aiding in understanding its reasoning.

*Iterative Learning***:** It refines its understanding of user behavior with each data point, continuously improving detection accuracy.

GNNs excel at insider threat detection because they can capture intricate relationships between features in user data. This is crucial for:*Modeling Normal Patterns***:** GNNs learn typical user behavior by considering connections between features, like access times and file modifications.*Identifying Anomalies:* Deviations from the learned patterns (e.g., unusual access patterns) can signal malicious intent.By analyzing data through a graph lens, GNNs provide a powerful framework for insider threat detection.

Experimental Setup:

Dataset: The dataset used is cert 4.2 dataset which is a synthetic dataset that contains both background and malicious data.

File	Description
Device.csv	Log of user's activity regarding connecting and disconnecting a thumb drive
Email.csv	Log of user's e-mail communication
File.csv	Log of user's activity regarding copying files to removable media devices
http.csv	Log of user's internet browsing history
Logon.csv	Log of user's workstation logon and logoff activity
psychometric.csv (1)	Users' psychometric scores based on big five personality traits
Ldap (1)	A set of eighteen (18) files regarding the organizations' users and their roles

(1) data not collected in our experiments.

[Ref:3]

1.Data Loading and Preparation:

Load the dataset from a CSV file using Pandas' pd.read_csv(). This dataset contains features representing user activity (X) and a target variable (y) indicating normal or malicious behavior.
We check for missing values or outliers.

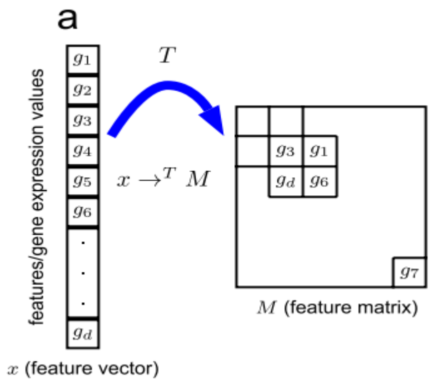
2. Feature Extraction using Tab2Img:

Tab2Img has been used to convert tabular data into images for CNN.This is done to utilize the power of Convolutional Neural Networks for non-image structured data.

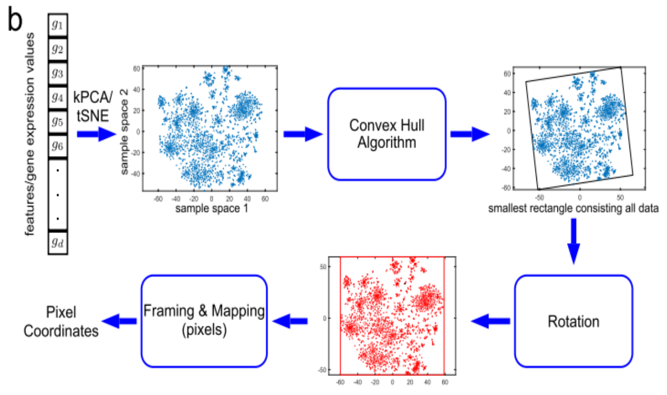
Code:

```
From sklearn.datasets import fetch_covtype

From tab2img.converter import Tab2Img

dataset=fetch_covtype()

train=dataset.data

target=dataset.target

model=Tab2Img()

images=model.fit_transform(train,target)
```

We convert the tabular data into form of images because CNN can only work on images. CNN architecture accepts a sample as an image and performs feature extraction and classification via hidden layers like convolution layers, RELU, max-pooling layers.It automatically derives features from raw elements. It finds high order statistics of image and nonlinear correlations. This improves the detection rate.


a
features/gene expression values
g_1 g_2 g_3 g_4 g_5 g_6 g_d
T
$x \rightarrow^T M$
g_3 g_1 g_d g_6 g_7
M (feature matrix)
x (feature vector)

[ref:4]


b
features/gene expression values
kPCA/ tSNE
sample space 2
sample space 1
Convex Hull Algorithm
smallest rectangle consisting all data
Rotation
Framing & Mapping (pixels)
Pixel Coordinates

[ref:4]

This technique helps us analyze features, like genes, by transforming them into a visual format. Here's how it works:

a)First, features are placed as points on a flat surface, defining their relative positions, not their actual values.
b)Then, an algorithm finds the smallest rectangular box that can hold all these feature points.
c)To make the data suitable for image analysis using Convolutional Neural Networks (CNNs), the information is rotated to fit a horizontal or vertical layout. Feature coordinates are converted into pixel locations. If multiple features land on the same pixel due to size constraints, their values are averaged before placing them on that pixel.
d)Finally, the actual feature values (like gene expression levels) are mapped to their corresponding pixel locations. If multiple features map to the same spot, their values are averaged before placement.

3. *Train-Test Split:*

The data is split into training (X_train, y_train) and validation sets (X_valid, y_valid) using train_test_split. This ensures the model is trained on a portion of the data and evaluated on another unseen portion. Here, a 67% train-33% test split is used with a random state for reproducibility.

4. *Learning Parameters:*

Learning Rate (lr=0.03): This value controls how much the model updates its internal parameters based on each training example. A higher rate can lead to faster learning but may miss the optimal solution, while a lower rate learns more cautiously but might take longer.
Optimizer (Adam): This algorithm efficiently updates the model's parameters during training. It adapts the learning rate for each parameter based on past updates, aiming to find a good balance between speed and accuracy.
Loss Function (CrossEntropyLoss): This function measures how well the model's predictions match the actual labels. It's used to guide the training process by calculating the error the model makes on each training example.

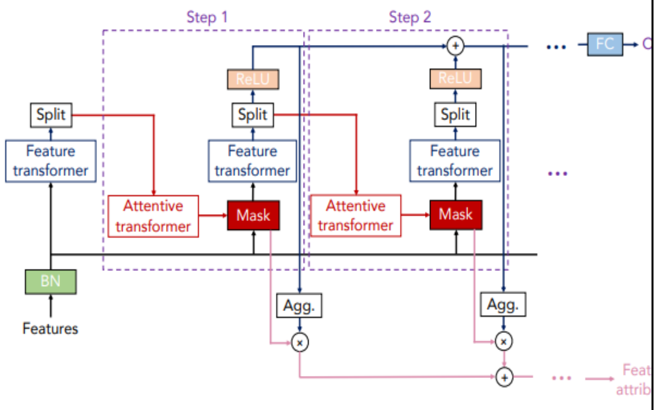
5. *Model Training:Tabnet:*

Tabnet is a deep learning architecture designed for handling complex tabular data. At each step, Tabnet analyses the features in the data using a combination of Feature Transformer, Attention Mechanism, Feature Masking.

Feature Transformer block transforms the data to prepare it for analysis. *Attention Mechanism* focusses on the most relevant features based on their past weightage in the decision process. *Feature masking* identifies and masks all the irrelevant (less important) features.

The processed data is split, with one part used for further feature selection in the next step, and the other contributing to the final model output.

So,TabNet leverages a multi-step feature selection process with attention mechanisms to identify the most critical features for accurate predictions on tabular data.

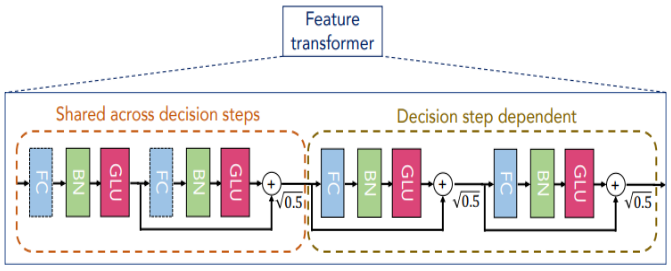


[Ref:5]

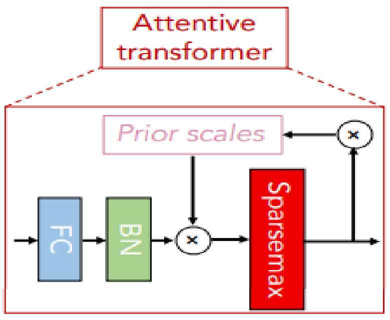
Code:

```
from pytorch_tabnet.tab_model
import TabNetClassifier
tabnet_model = TabNetClassifier()
```

```
tabnet_model.fit(
 X_train, y_train,
  eval_set=[(X_valid, y_valid)],
  eval_metric=['accuracy']
)
preds = tabnet_model.predict(X_test)
```



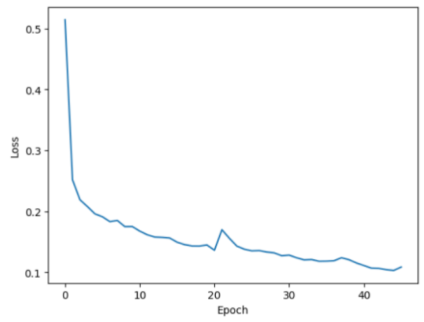
[Ref:5]



[Ref:5]

The CERT 4.2 dataset likely contains numerous features representing user activities, system attributes, or network traffic details. TabNet doesn't necessarily require extensive feature engineering. It can potentially work with the raw features, although some basic cleaning and normalization might be applied for better performance.

TabNet's strength lies in its ability to automatically select important features during the training process. This can be particularly beneficial for insider threat detection, where understanding which user activities are most indicative of malicious intent is crucial.



6.*Graph Neural Network Preparation: Preparing Node feature and adjacency matrix:*

The features are prepared as a 2D array. An adjacency matrix is created using np.eye(num_nodes), which is an identity matrix, implying that each feature is only connected to itself. Features and adjacency are converted to tensors (torch.tensor). num_nodes is set to the number of features, assuming each feature represents a node in the graph.An adjacency matrix (adjacency) is created as an identity matrix with a size of num_nodes x num_nodes. This is a simplified assumption, and in real-world scenarios, the adjacency matrix would represent the connections between nodes in the graph.Features are converted to a PyTorch tensor (features) and edge indices (edge_index) are created based on the non-zero elements of the adjacency matrix (assuming connections between all nodes in this example).

Code:

```
X = df.drop("Class_Label", axis=1).values
num_nodes = X.shape[0]
num_features = X.shape[1]
adjacency = np.eye(num_nodes)  # Assuming each node is connected to itself
features = torch.tensor(X, dtype=torch.float)
edge_index = torch.tensor(np.nonzero(adjacency), dtype=torch.long)
```

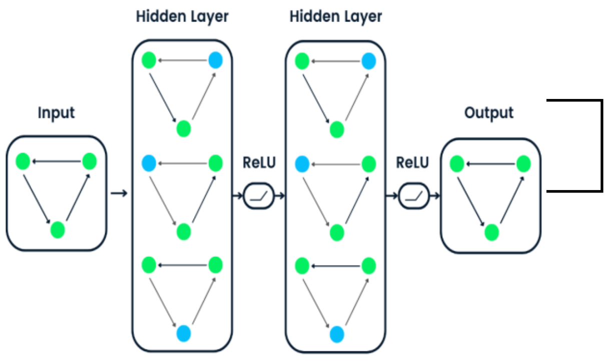
7. Defining GNN Model:

This class defines the architecture of the GNN model. init function initializes the model's layers.Two GCNConv layers from torch_geometric.nn are used. GCNConv is a specific type of convolutional layer designed for graph data, able to learn representations of nodes by considering their features and the features of their connected neighbors in the graph.Between the GCNConv layers, Dropout layers are used to prevent overfitting.

GNNs: Each node is paired with an embedding. This embedding establishes the node's location in the data space. Graph Neural Networks are topologies of neural networks that operate on graphs.

A GNN architecture's primary goal is to learn an embedding that contains information about its neighborhood. We may use this embedding to tackle a variety of issues, including node labeling, node and edge prediction, and so on.

In other words, Graph Neural Networks are a subclass of Deep Learning techniques that are specifically built to do inference on graph-based data. They are applied to graphs and are capable of performing prediction tasks at the node, edge, and graph levels.



Code:

Model Initialization:

```
model_gnn= GNNModel(num_features=num_nodes, num_classes=2)
```

Train function:

```
def train(model, features, edge_index, labels, optimizer):
    model.train()
    optimizer.zero_grad()
    output = model(features, edge_index)
```

loss = criterion(output, labels)

loss.backward()optimizer.step()

return loss.item()

8.*GNN Model Forward pass:*

This function defines how the data flows through the GNN model. Features are passed through the first GCNConv layer with ReLU activation and dropout for regularization. The process is repeated for the second GCNConv layer.Finally, a log softmax function is applied to obtain the model's output representing class probabilities ("Malicious" or "Non-Malicious").

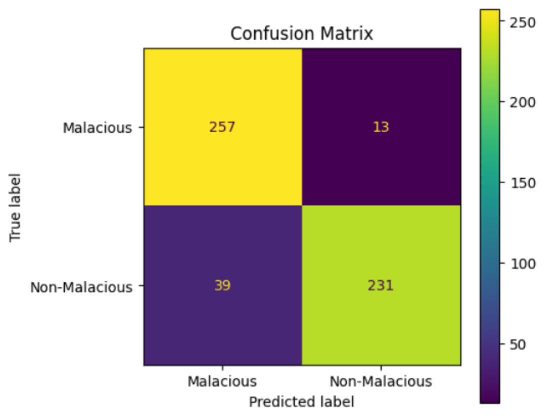
9. Model Training:

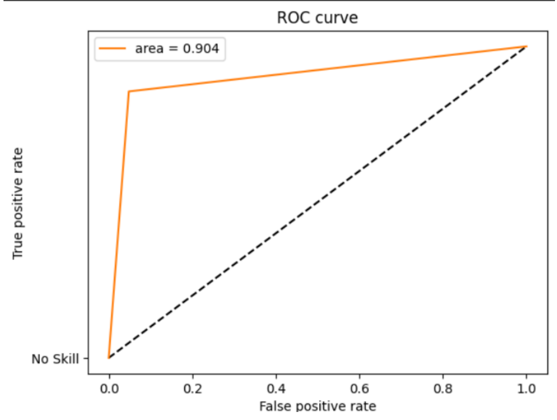
model_gnn is instantiated with the defined number of features and classes.
An Adam optimizer (*optimizer_gnn*) and *CrossEntropyLoss* criterion are used for *training. train* performs a single training epoch. test performs model evaluation on a given dataset: The code iterates for a specified number of epochs (e.g., 300) performing training and evaluation on each epoch.

10. Evaluation
Validation During Training: After each training epoch, evaluate the model on both the training and validation sets. Calculate and record the training loss, training accuracy, and validation accuracy.
Final Evaluation on Test Set: Load the test dataset and convert it into PyTorch tensors. Evaluate the trained model on the test set to determine its accuracy. Plot a confusion matrix to visualize the classification performance.
Performance Metrics: Calculate key performance metrics including accuracy, precision, recall, F1 score, true positive rate (sensitivity), true negative rate (specificity), false positive rate, and false negative rate. Use these metrics to comprehensively evaluate the model's performance.

Results
The model trained is evaluated using Precision, Recall, Accuracy Score, ROC AUC Curve.

Tabnet Classifier:

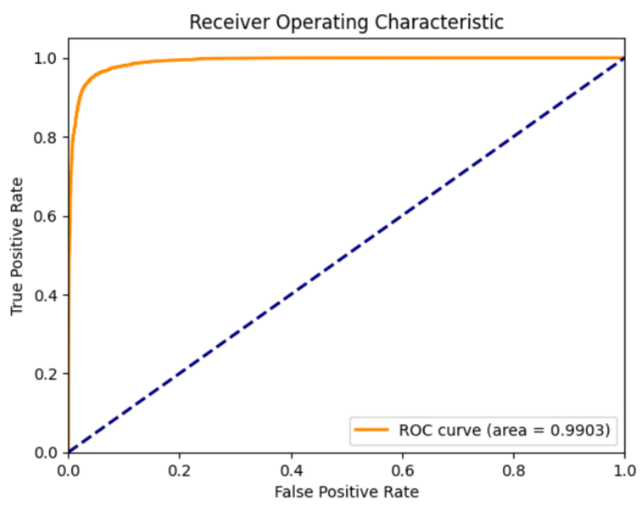




	precision	recall	f1-score	support
0.0	0.87	0.95	0.91	270
1.0	0.95	0.86	0.90	270
accuracy			0.90	540
macro avg	0.91	0.90	0.90	540
weighted avg	0.91	0.90	0.90	540

GNN Classifier:

GNN Accuracy: 0.9539
Precision: 0.9539
Recall: 0.9539
F1 Score: 0.9539
Confusion Matrix:
[[7385 375]
[349 7601]]



References:

1. Krunal Randive ∗ , R. Mohan, Ambairam Muthu Sivakrishna https://www.sciencedirect.com/science/article/abs/pii/S2214212623000194 2.
2. Duc C. Le , *Student Member*, *IEEE*, Nur Zincir-Heywood, *Member*, *IEEE*, and Malcolm I. Heywood , *Senior Member*, *IEEE* *https://ieeexplore.ieee.org/document/8962316*
3. Andreas Nicolaou, Stavros Shiaeles, Nick Savage https://www.researchgate.net/publication/343134193_Mitigating_Insider_Threats_Using_Bio-Inspired_Models
4. 4. Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A. Boroevich & Tatsuhiko Tsunoda https://www.nature.com/articles/s41598-019-47765-6

5. Sercan O. Arik, Tomas Pfister https://arxiv.org/abs/1908.07442v5
6. Duc C. Le; Nur Zincir-Heywood https://ieeexplore.ieee.org/document/9399116
7. Duc C. Le; A. Nur Zincir-Heywood https://ieeexplore.ieee.org/abstract/document/8717892
8. Duc C. Le; A. Nur Zincir-Heywood https://ieeexplore.ieee.org/document/8424659

9. Shuhan Yuan [a], Xintao Wu [b]

[b]https://www.sciencedirect.com/science/article/abs/pii/S0167404821000456

10. 10.Yichen Wei, Kam-Pui Chow, Siu-Ming Yiuhttps://www.sciencedirect.com/science/article/pii/S266628172100024X
11. 11.Sercan O. Arik, Tomas Pfisterhttps://arxiv.org/abs/1908.07442
12. Alessandro Sperduti and Antonina Staritahttps://ieeexplore.ieee.org/abstract/document/572108
13. Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S. Yu, Bo Lihttps://arxiv.org/pdf/1812.10528
14. Junchao Xiao; Lin Yang; Fuli Zhong; Xiaolei Wang; Hongbo Chen; Dongyang L https://ieeexplore.ieee.org/document/9953125
15. Shuhan Yuan Utah State University Logan, UT, USA Xintao Wu https://arxiv.org/pdf/2005.12433