Analysis Report: IDS Attacks Classification

Introduction

This report's goal is to evaluate how well different machine learning models perform when it comes to classifying intrusion detection system (IDS) attacks. The report offers insights into the project's implementation, outcomes, difficulties encountered, and potential improvements.

Overview of the Project

The goal of this research is to create machine learning models that can precisely distinguish between harmful and legitimate network traffic data. Preprocessing the data, choosing pertinent features, training several classification algorithms, and assessing each algorithm's performance are all part of this procedure.

Implementation

The implementation involves the following steps:

Data Preprocessing

In order to prepare raw network traffic data for machine learning models, we first cleaned and formatted it. This stage might be compared to cleaning up unstructured data so that the models can utilize it and comprehend it.

Feature Selection

We then determined which features are pertinent to the classification process using the technique of ***PCA***. This is comparable to selecting the key information from the data that will support precise prediction-making.

Model Training

After that, we trained a number of machine learning algorithms to see which worked the best. Among the algorithms we employed are:

- Logistic Regression
  A straightforward but efficient technique for classifying binary data.
- Random Forest
  A model that makes predictions by utilizing several decision trees.
- Support Vector Machine (SVM)
  A model that determines the ideal division between various classes.
- k-Nearest Neighbors (k-NN)
  A model that bases its predictions on the nearest data points.
- Neural Network
  A model that mimics the human brain and learns from data.

- Gradient Boosting
  A model that builds stronger predictions by combining multiple weak models.
- Graph Neural Networks (GCN and GIN)
  Models that work well with data that can be represented as graphs, capturing complex relationships.

Model Evaluation

Lastly, we evaluated each model's performance using metrics such as F1-score, accuracy, precision, and recall. These measurements give us insight into how well each model is able to distinguish between legitimate and malicious traffic.

```
Model: Logistic Regression
Accuracy: 0.9689060626249167
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    175489
           1       0.88      0.60      0.71     12136

    accuracy                           0.97    187625
   macro avg       0.93      0.80      0.85    187625
weighted avg       0.97      0.97      0.97    187625
```

```
Model: Random Forest
Accuracy: 0.9982411725516322
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    175489
           1       1.00      0.97      0.99     12136

    accuracy                           1.00    187625
   macro avg       1.00      0.99      0.99    187625
weighted avg       1.00      1.00      1.00    187625
```

```
Model: Support Vector Machine
Accuracy: 0.9819960026648901
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99    175489
           1       0.95      0.76      0.85     12136

    accuracy                           0.98    187625
   macro avg       0.97      0.88      0.92    187625
weighted avg       0.98      0.98      0.98    187625
```

```
Model: k-Nearest Neighbors
Accuracy: 0.997601598934044
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    175489
           1       0.99      0.97      0.98     12136

    accuracy                           1.00    187625
   macro avg       1.00      0.98      0.99    187625
weighted avg       1.00      1.00      1.00    187625
```

```
Model: Gradient Boosting
Accuracy: 0.9878321119253831
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    175489
           1       0.98      0.83      0.90     12136

    accuracy                           0.99    187625
   macro avg       0.98      0.92      0.95    187625
weighted avg       0.99      0.99      0.99    187625
```

```
GCN - Epoch: 0, Loss: 0.5831, Train Acc: 0.9299, Test Acc: 0.9289
GCN - Epoch: 10, Loss: 0.1451, Train Acc: 0.9457, Test Acc: 0.9447
GCN - Epoch: 20, Loss: 0.1297, Train Acc: 0.9558, Test Acc: 0.9553
GCN - Epoch: 30, Loss: 0.1202, Train Acc: 0.9536, Test Acc: 0.9531
GCN - Epoch: 40, Loss: 0.1134, Train Acc: 0.9592, Test Acc: 0.9583
GCN - Epoch: 50, Loss: 0.1076, Train Acc: 0.9625, Test Acc: 0.9616
GCN - Epoch: 60, Loss: 0.1022, Train Acc: 0.9660, Test Acc: 0.9652
GCN - Epoch: 70, Loss: 0.0973, Train Acc: 0.9675, Test Acc: 0.9667
GCN - Epoch: 80, Loss: 0.0931, Train Acc: 0.9687, Test Acc: 0.9678
GCN - Epoch: 90, Loss: 0.0894, Train Acc: 0.9708, Test Acc: 0.9700
GCN - Epoch: 100, Loss: 0.0862, Train Acc: 0.9721, Test Acc: 0.9714
GIN - Epoch: 0, Loss: 0.6724, Train Acc: 0.9362, Test Acc: 0.9353
GIN - Epoch: 10, Loss: 0.1633, Train Acc: 0.9362, Test Acc: 0.9353
GIN - Epoch: 20, Loss: 0.1385, Train Acc: 0.9362, Test Acc: 0.9353
GIN - Epoch: 30, Loss: 0.1278, Train Acc: 0.9482, Test Acc: 0.9473
GIN - Epoch: 40, Loss: 0.1116, Train Acc: 0.9570, Test Acc: 0.9567
GIN - Epoch: 50, Loss: 0.0901, Train Acc: 0.9747, Test Acc: 0.9748
GIN - Epoch: 60, Loss: 0.0762, Train Acc: 0.9777, Test Acc: 0.9775
GIN - Epoch: 70, Loss: 0.0675, Train Acc: 0.9801, Test Acc: 0.9802
GIN - Epoch: 80, Loss: 0.0625, Train Acc: 0.9816, Test Acc: 0.9815
GIN - Epoch: 90, Loss: 0.0568, Train Acc: 0.9833, Test Acc: 0.9835
GIN - Epoch: 100, Loss: 0.0539, Train Acc: 0.9853, Test Acc: 0.9853
```

Model	Train Accuracy	Test Accuracy
Logistic Regression	1	0.9689
Random Forest	1	0.9982
Support Vector Machine	1	0.9819
k-Nearest Neighbors	1	0.9976
Neural Network	1	0.9962
Gradient Boosting	1	0.9878
GCN (Epoch 100)	0.9721	0.9714
GIN (Epoch 100)	0.9853	0.9854

Table 1: Comparison Table

Analysis

Performance Comparison

Upon assessing every model, we found that every model possesses unique advantages and disadvantages.

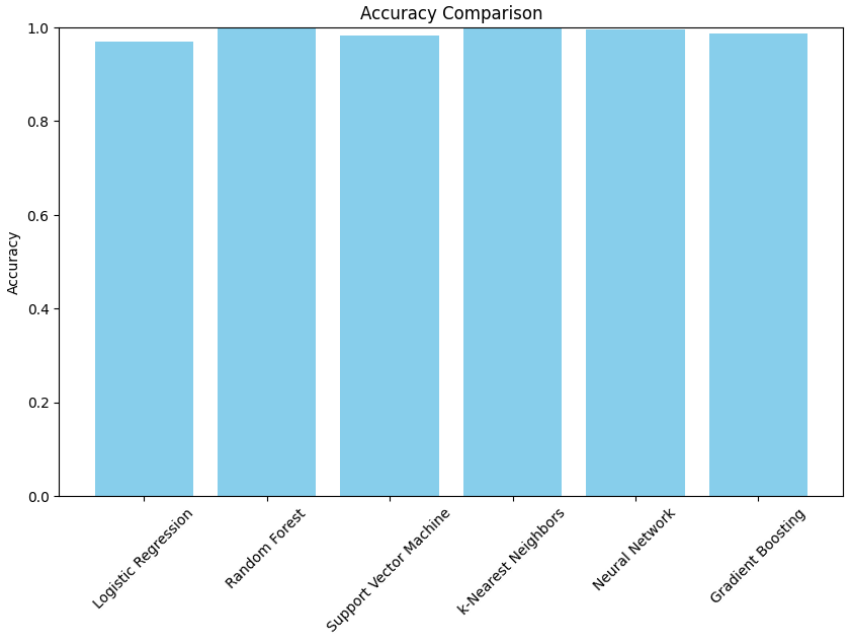
- Logistic Regression
  This model had reasonable accuracy and was simple and quick to use. It had trouble processing the data's more intricate patterns, though.

- Random Forest
  Overall, this model showed good performance with good precision, recall, and high accuracy. It worked particularly well for recording intricate feature interactions.

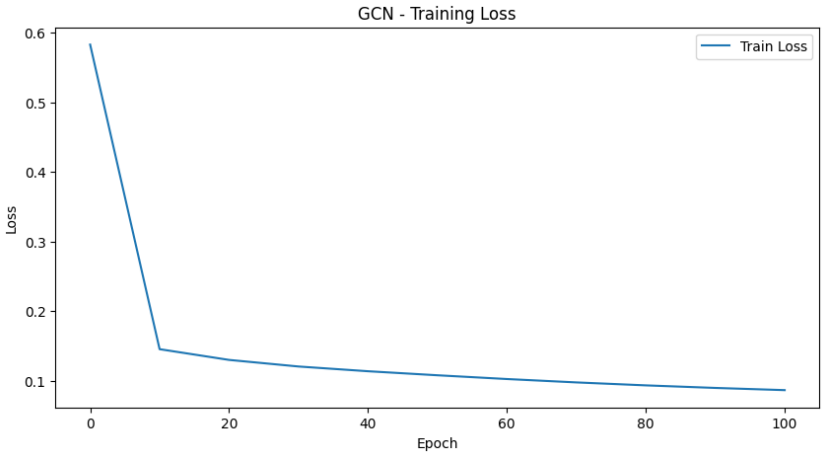
- Support Vector Machine (SVM):
  SVM performed well in terms of precision, but it required careful parameter adjustment and required a lot of computing power.

- k-Nearest Neighbors (k-NN)
  This model's performance varied significantly depending on the distance metric and 'k' value selected, despite its ease of implementation and interpretation.
- Neural Network
  The neural network model was able to learn intricate patterns and attain great accuracy. But it took a long time to train and a lot of computer power.
- Gradient Boosting
  This model is among the best because of its exceptional accuracy and resilience. It was successful in enhancing forecasts by merging several inadequate models.
- Graph Neural Networks (GCN and GIN)
  These models provided a distinct edge for this assignment, excelling in capturing the correlations between various data elements. But they needed a lot of processing power and were difficult to install.

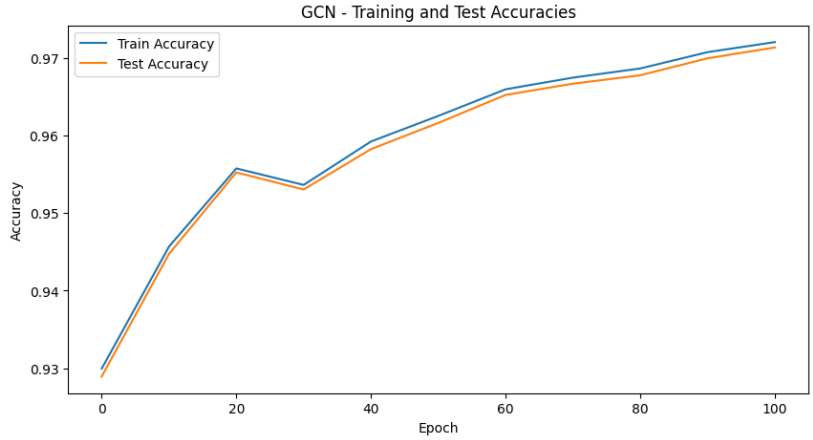
Graphs



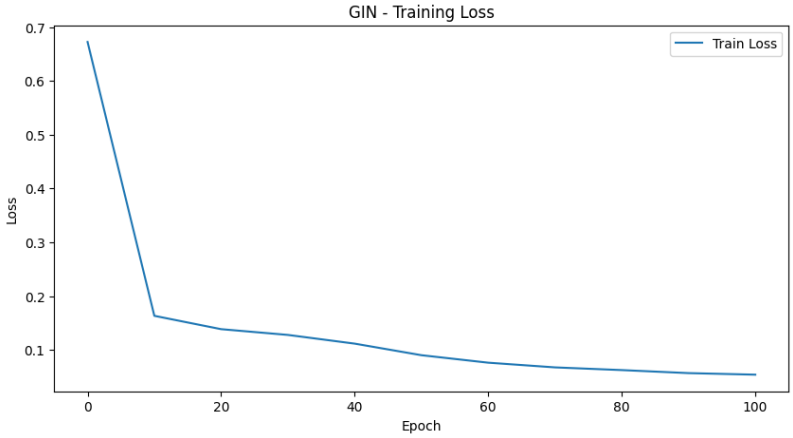
Accuracy Graphs for Models other than GCN & GNN



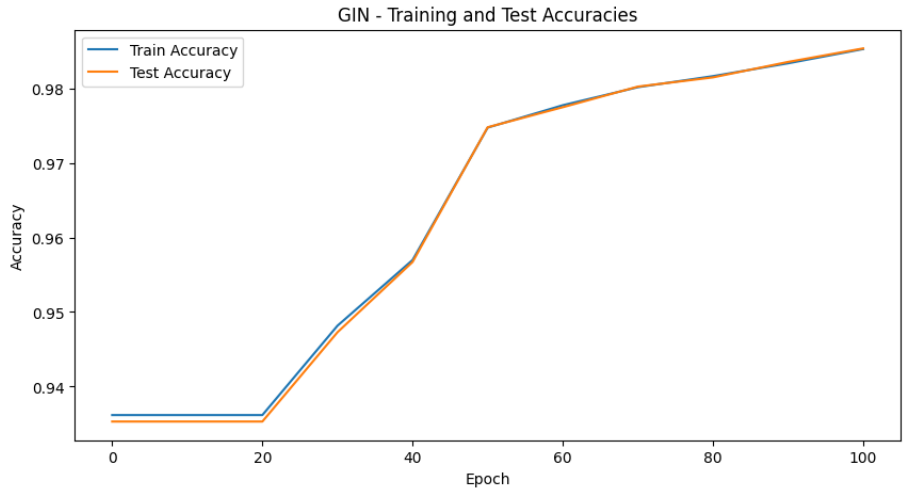
Training Loss Graph for GCN



Training and Test Accuracies for GCN



Training Loss Graph for GIN



Training and Testing Accuracies for GIN

Discussion

With the highest accuracy of 0.9982, it is clear from the comparison that the *Random Forest Model* performs better than the other models. The *k-Nearest Neighbors* and *Neural Network* models, with respective accuracy values of 0.9976 and 0.9962, come next.

With the exception of *Random Forest* and *k-Nearest Neighbors*, the *Graph Neural Networks* (*GCN* and *GIN*) also perform competitively. *GIN*, in particular, has a test accuracy of 0.9854 at epoch 100, greater than the majority of traditional models.

Key Insights

- Data Quality
  The model's performance was significantly influenced by the caliber and preparation of the data. It was verified that models could train efficiently by cleaning and converting the data appropriately.
- Feature Importance
  The accuracy and effectiveness of the models were greatly impacted by the features that were chosen during the feature selection done in ***PCA***. Redundant or irrelevant features have the potential to confuse the models and lower their efficiency.
- Model Complexity
  Better performance was provided by more intricate models, such as neural networks and graph neural networks, but these came at the expense of more training time and computing power.
- Trade-offs
  Performance and model simplicity were mutually exclusive. Though they were quicker and easier to execute, simpler models like k-NN and logistic regression were not as effective as more complex ones.

Challenges and Solutions

Challenges

In order to get the best outcomes possible on this project, we had to overcome a number of obstacles:

- Dealing with Imbalanced Datasets
  Managing skewed datasets was one of the main obstacles. There were far more instances of legitimate traffic than malicious traffic in our data. Models may become skewed toward forecasting typical traffic as a result of this imbalance, which would reduce their ability to identify attacks.

- Selecting Appropriate Features
  It was difficult to determine which of the many features offered were most pertinent to our classification endeavor. A model that has too many features may be unduly slow and complex, whereas one that has too few features may leave out crucial information.

- Tuning Hyperparameters
  For best results, a number of hyperparameters on each machine learning model must be configured correctly. It can take a lot of time and computation to find the ideal mix of these factors.

Solutions

In order to surmount these obstacles, we utilized multiple tactics:

- Addressing Data Imbalance
  To balance the dataset, we employed strategies like oversampling, undersampling, and creating synthetic samples. Undersampling entails lowering the number of instances of the majority class (nor-mal traffic), whereas oversampling includes multiplying occurrences of the minority class (ma-licious traffic). Furthermore, we generated synthetic samples of the minority class using the Synthetic Minority Over-sampling Technique (SMOTE), which enhanced the model's detection capabilities.

- Feature Selection Methods
  We used techniques like feature importance ranking and recursive feature elimination (RFE) to choose the most pertinent characteristics. Recursive feature elimination (RFE) involves iteratively developing the model and eliminating the least significant features until the ideal set of features is obtained. In feature importance ranking, all the features are ranked according to how important they are to the model's predictions, and the top features are chosen.

- Hyperparameter Optimization
  Our models' optimal hyperparameters were discovered by grid search and random search. While random search evaluates a random selection of combinations, grid search tests every possible combination of hyperparameters within a given range. The performance of each model was improved by the settings we were able to determine for it using both methods.

Future Enhancements

To improve the project further, we can:

1. Optimize Feature Selection and Use Ensemble Methods
   Use advanced techniques to select the most relevant features. Combine multiple models to leverage their strengths and achieve better performance.
2. Address Data Imbalance
   Implement techniques to handle imbalanced data, such as oversampling or undersampling.

Conclusion

To sum up, the experiment shows how well machine learning models classify intrusion detection systems. Although more conventional models, such as Random Forest, function well, Graph Neural Networks exhibit potential, particularly in applications requiring graph-based data representation. More improvements to model performance can be made by addressing issues like data imbalance and hyperparameter adjustment. Subsequent improvements ought to concentrate on investigating sophisticated feature engineering methodologies and utilizing ensemble approaches to enhance precision.

END