# NetSim VANETs

## Detecting sybil node attacks in VANETs using Machine Learning

09Oct2024

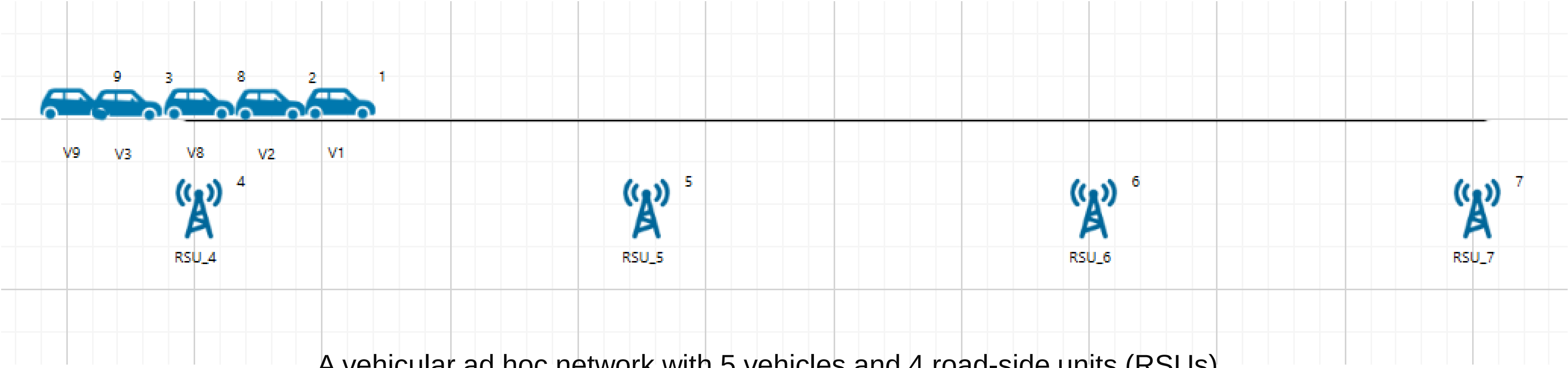
Applicable Release: NetSim v14.2 or higher
Applicable Version(s): NetSim Standard
Project download link:
The URL has the exported NetSim scenario for the examples used in this document and the python scripts to run classifiers.

# Outline

- Introduction to VANETs
- Sybil attack in VANETs
- Modeling Sybil attack in VANETs using NetSim
- Attack Scenarios with sybil nodes – Training data
  - Attack scenarios with 1,1, 1, 2, 3, 3 sybil nodes
  - Data processing
  - Feature visualization
- Attack Scenarios with sybil nodes – Test
  - Attack scenarios with 1, 1, 1, 3, 3, 3 sybil nodes
  - Data processing
  - Feature visualization
  - Classification
- Detection of sybil nodes using ML based classifiers
- Confusion Matrix: Accuracy, Precision, F1 Score, Recall
- Comparison between different classifiers: Random Forest, K-Nearest Neighbor, Decision Tree, Gradient boost.

# Introduction to VANETs

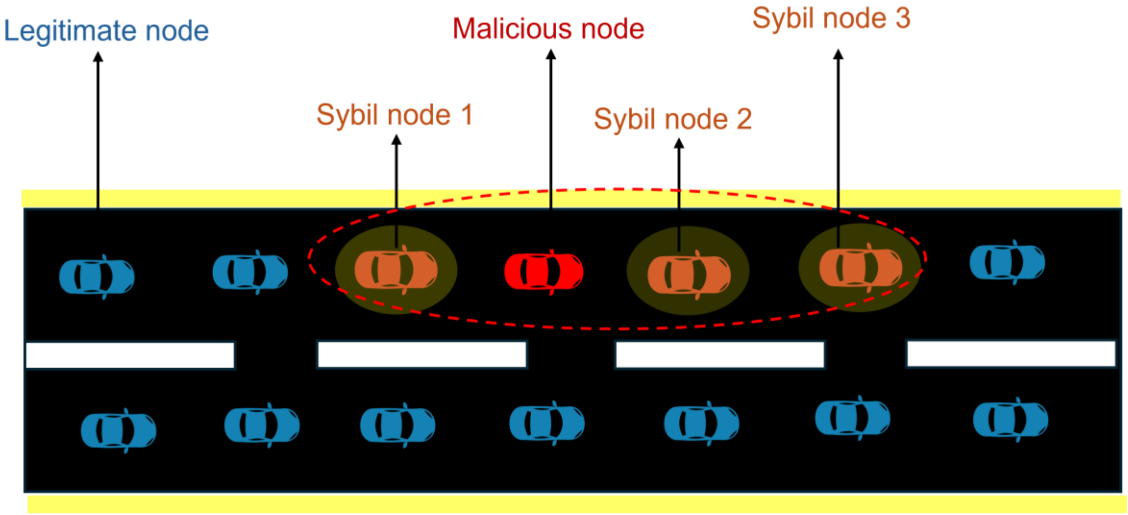
- VANETs (Vehicular Ad Hoc Networks) are crucial for Intelligent Transportation Systems (ITS) and road safety.
- Vehicles communicate with each other (V2V) and with infrastructure (V2I).
- VANETs operate in dynamic environments, with changing topologies due to vehicle speed and varying road conditions.
- The communication architecture follows the IEEE 802.11p standard for PHY and MAC layers and the IEEE 1609 standard for upper layers, enabling Wireless Access in Vehicular Environments (WAVE).



A vehicular ad hoc network with 5 vehicles and 4 road-side units (RSUs)

# Sybil attack in VANETs

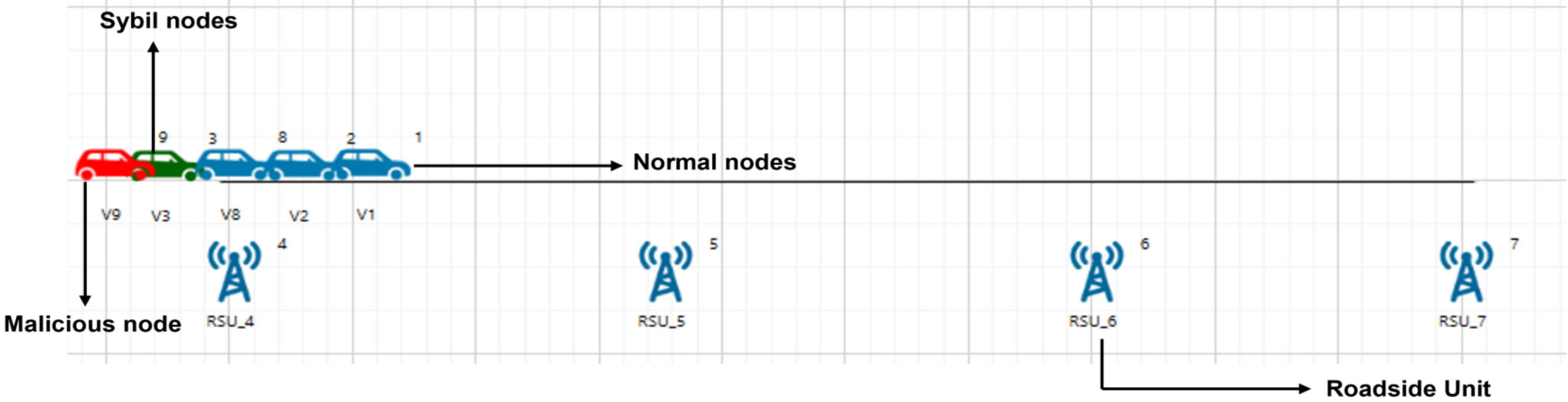
- **Definition**: A Sybil attack in VANETs occurs when a malicious vehicle creates multiple fake identities or "ghost vehicles" in the network.
- **Mechanism**: The attacker broadcasts messages from these fake identities, making it appear as if multiple vehicles are present when in reality, it's just one malicious node.
- **Purpose**: These attacks aim to manipulate traffic data, influence routing decisions, or gain an unfair advantage in safety-critical applications.
- **Impact**: Sybil attacks can lead to false traffic congestion reports, manipulated voting systems in VANETs, and compromised safety applications, potentially causing accidents or traffic disruptions.
- **Detection**: RSSI values are used to estimate the physical location of nodes in the network. In a Sybil attack, multiple fake identities originating from the same physical location should have very similar RSSI values when measured by receiving nodes. By analyzing the similarity of RSSI patterns malicious nodes can be detected



Sybil attack scenario in VANETs

# Sybil attack in VANETs using NetSim

- A Sybil attack in NetSim involves a malicious node (sybil attacker) creating illusory nodes, known as a Sybil nodes.
- The malicious node transmits messages as if it were two separate entities, misleading the network into believing there are two independent participants.
- To model this in NetSim, we create two nodes: V3 (the malicious node) and V9 (the Sybil node), where both nodes behave as if they are distinct entities.
- The scenario has 5 vehicles (3 real + 1 attacker + 1 sybil node) and 4 RSUs, each vehicle transmits data to all RSUs.
- Since the signal strength from the sybil attacker and the sybil node would be similar, RSSI-based classification can be used to detect malicious nodes

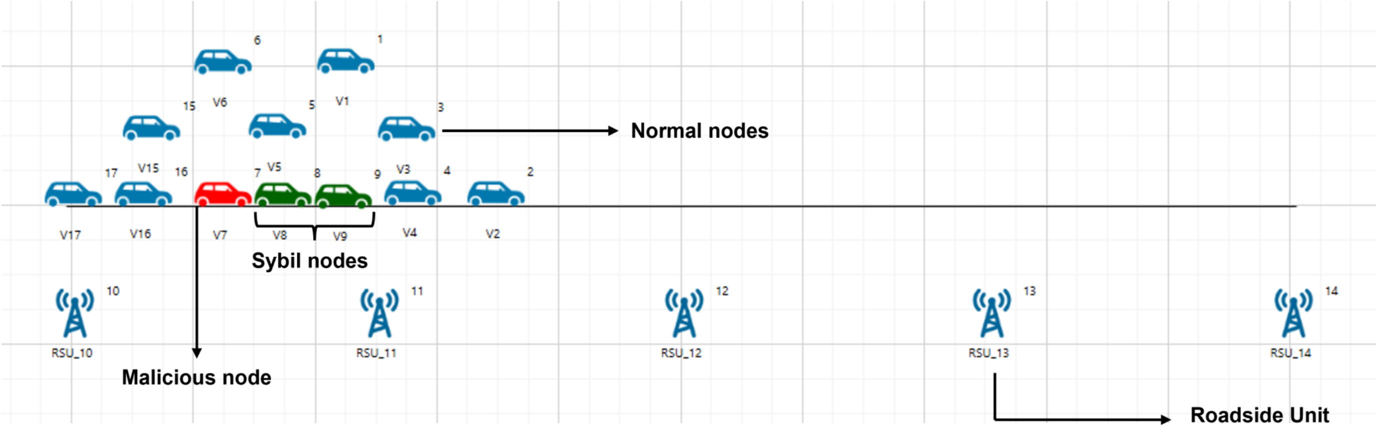


The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 1 illusion (sybil node in orange) and 3 normal vehicular (in blue)

# Training

# Attack scenarios – Training data generation

- In NetSim, we created 5 scenarios with varying total node counts (5, 7, 9, 11, 12 to 13).
- Total Node count = Normal node count + Sybil attacker count + Sybil node count
- Sybil node count in the 6 scenarios: 1, 1, 1, 2, 3, 3.
  - 1 malicious node in all cases
  - 1, 1, 1, 2, 3, 3 sybil nodes in each respective sample
- In NetSim, we enabled radio measurement log for all scenarios
- We wrote a python script to analyze the log file data and calculate the RSSI Power at each RSU from each vehicle, RSSI difference and RSSI Similarity
- Feature Extraction: RSSI Power, RSSI Difference, RSSI Similarity

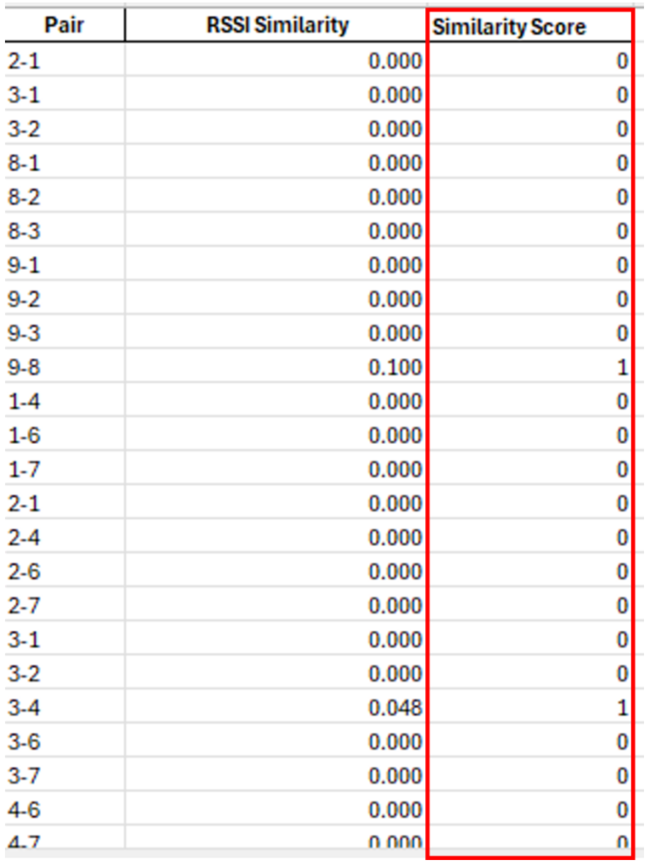


The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 2 illusion (sybil node in green) and 3 normal vehicular (in blue)

Yuan Yao et al., "Power Control Identification: A Novel Sybil Attack Detection Scheme in VANETs using RSSI," 2019, *IEEE.*

# Data processing and feature visualization

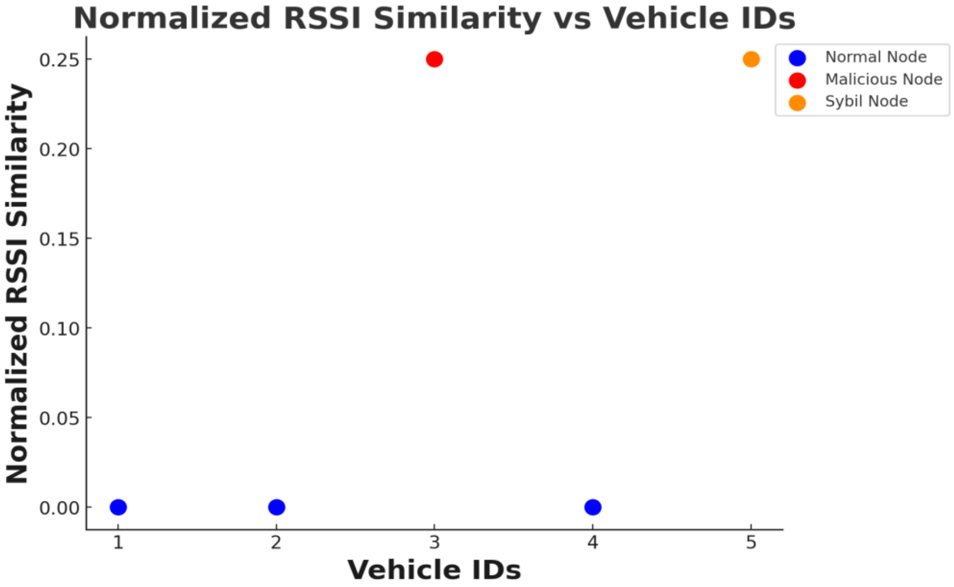
- Data extraction from IEEE radio measurement log to Excel using Python script
- Total dataset: 57 vehicles, 3 features each
- Feature normalization process:
  - Calculate the pairwise RSSI differences between vehicles for each RSU.
  - Compute the mean RSSI difference for each vehicle pair.
  - Assign a threshold to classify vehicle pairs based on RSSI differences.
  - Identify similar vehicle pairs based on the threshold.
  - Calculate the proportion of similar pairs from the total vehicle pairs.
  - Identify each unique vehicle pair.
  - Calculate the average RSSI similarity for each unique pair.
  - Label the similarity score as 0 or 1, depending on the RSSI similarity value.



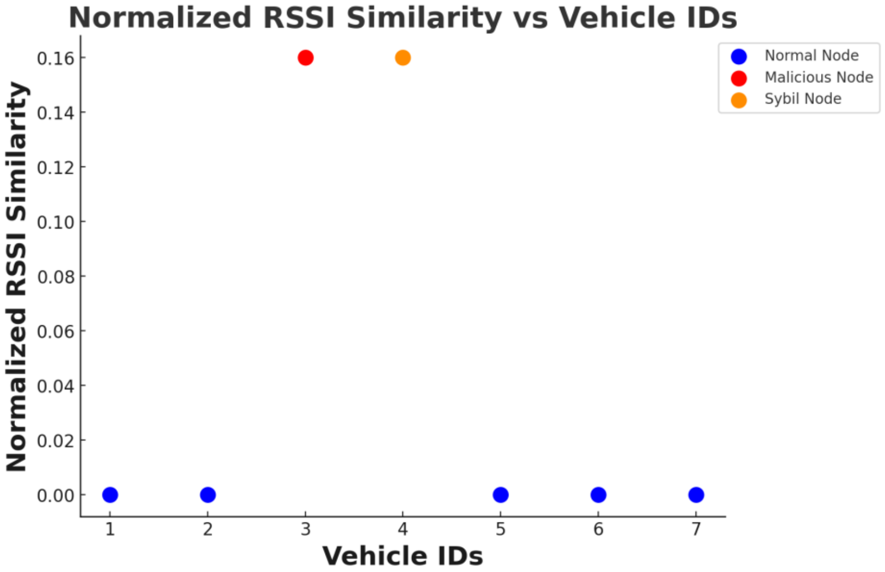
| Pair | RSSI Similarity | Similarity Score |
|---|---|---|
| 2-1 | 0.000 | 0 |
| 3-1 | 0.000 | 0 |
| 3-2 | 0.000 | 0 |
| 8-1 | 0.000 | 0 |
| 8-2 | 0.000 | 0 |
| 8-3 | 0.000 | 0 |
| 9-1 | 0.000 | 0 |
| 9-2 | 0.000 | 0 |
| 9-3 | 0.000 | 0 |
| 9-8 | 0.100 | 1 |
| 1-4 | 0.000 | 0 |
| 1-6 | 0.000 | 0 |
| 1-7 | 0.000 | 0 |
| 2-1 | 0.000 | 0 |
| 2-4 | 0.000 | 0 |
| 2-6 | 0.000 | 0 |
| 2-7 | 0.000 | 0 |
| 3-1 | 0.000 | 0 |
| 3-2 | 0.000 | 0 |
| 3-4 | 0.048 | 1 |
| 3-6 | 0.000 | 0 |
| 3-7 | 0.000 | 0 |
| 4-6 | 0.000 | 0 |
| 4-7 | 0.000 | 0 |

We label the similarity score based on features

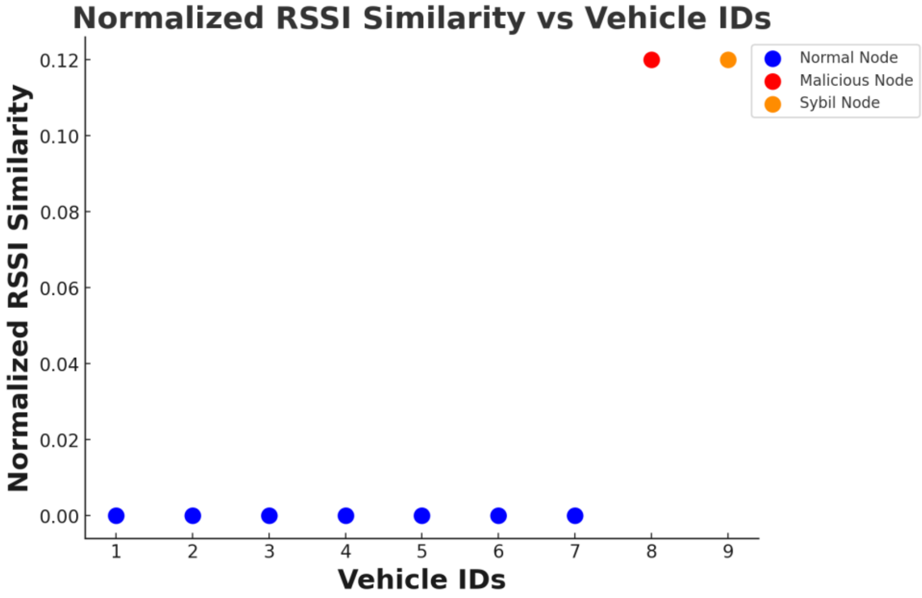
# Feature visualization: 5 ,7 and 9 vehicles



Normalized RSSI Similarity vs Vehicle IDs

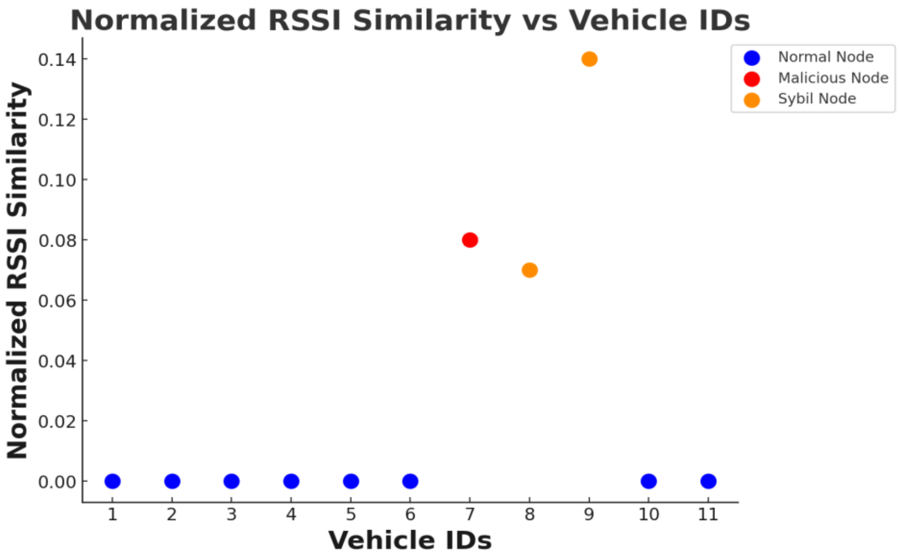


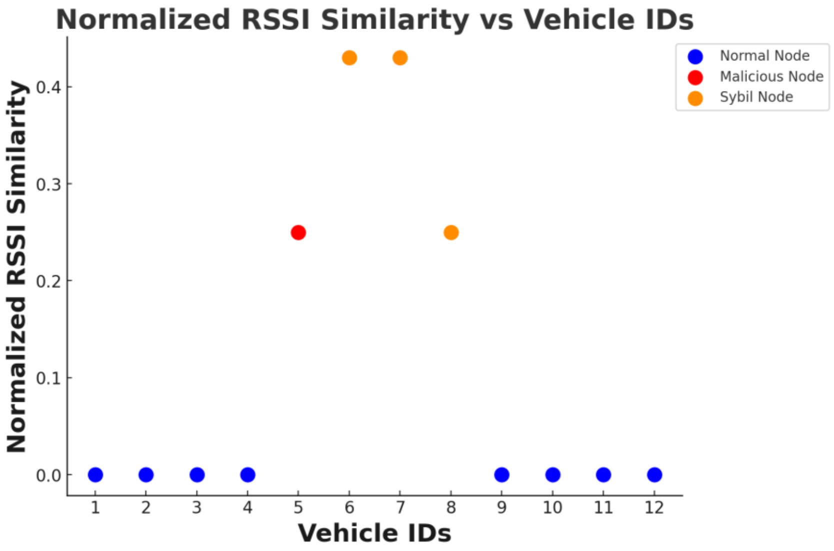
Normalized RSSI Similarity vs Vehicle IDs

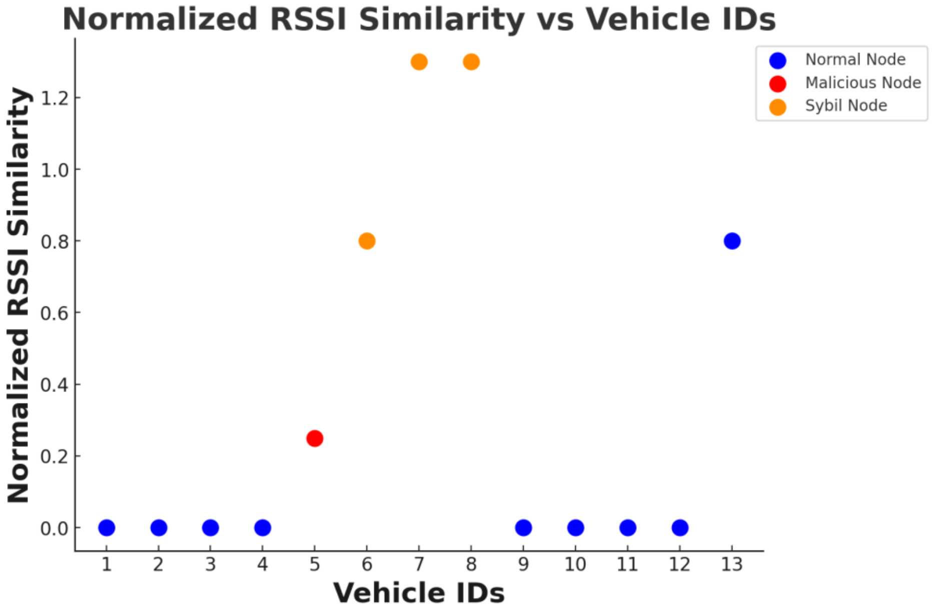


Normalized RSSI Similarity vs Vehicle IDs

# Feature visualization:11,12 and 13 vehicles







# Classifier training

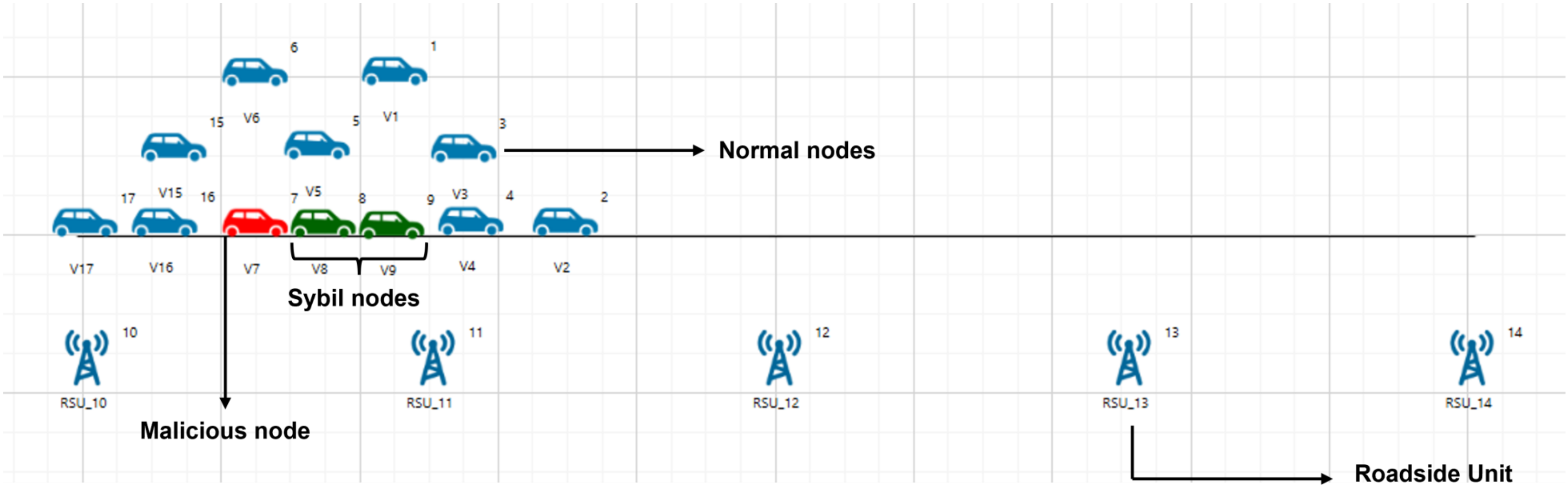
Features data was used to train the following classifiers:

- K-Nearest Neighbor
- Random Forest
- XGBoost Classifier
- Decision Tree

# Inference

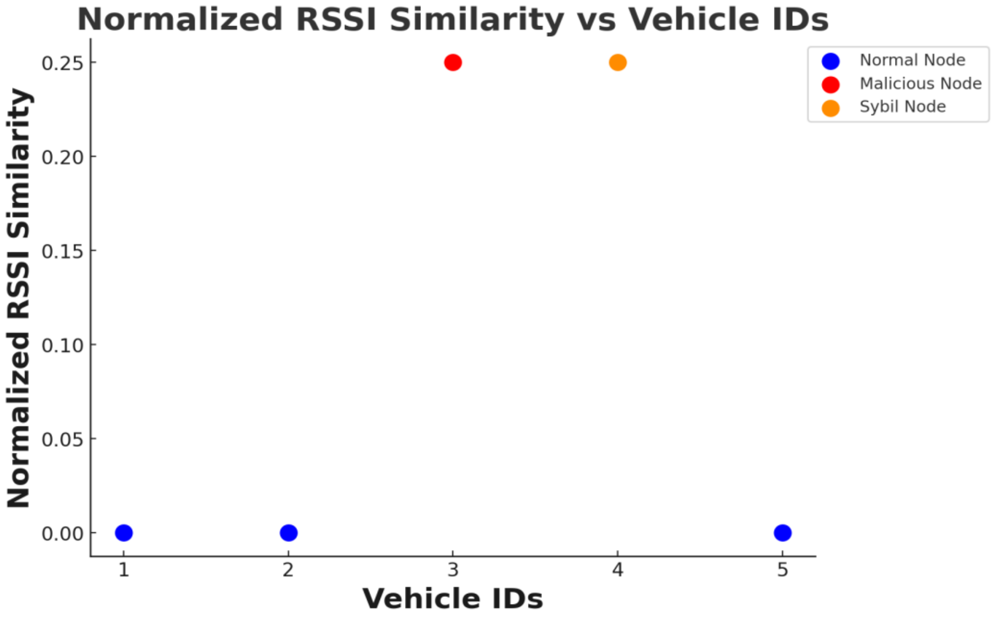
# Inference and Test Scenarios

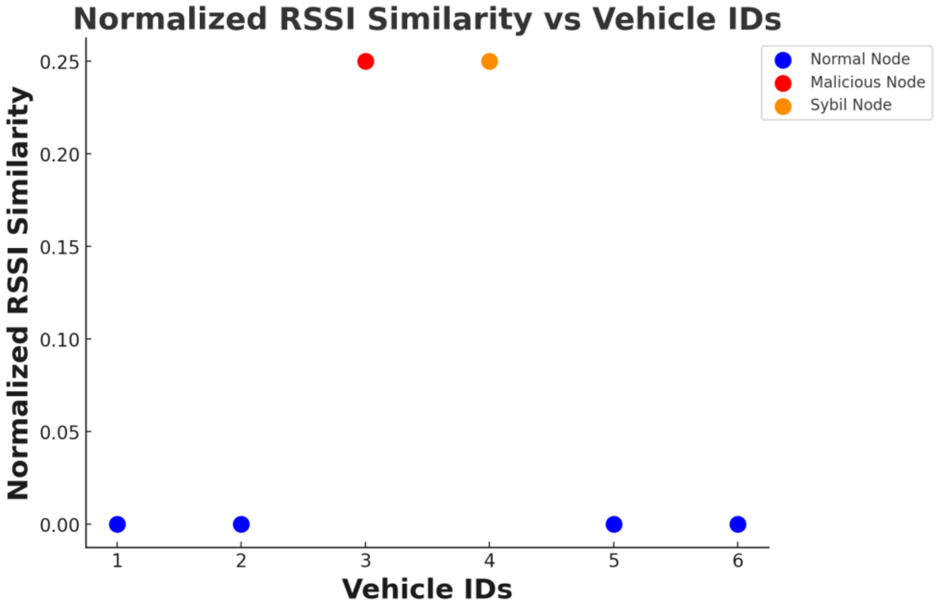
- Created 6 new scenarios with different node counts (5, 6, 9,12,12, 14).
- Sybil node count: 1, 1, 1, 3, 3, 3.
- Normalized the data using a python script as explained in slide 8.

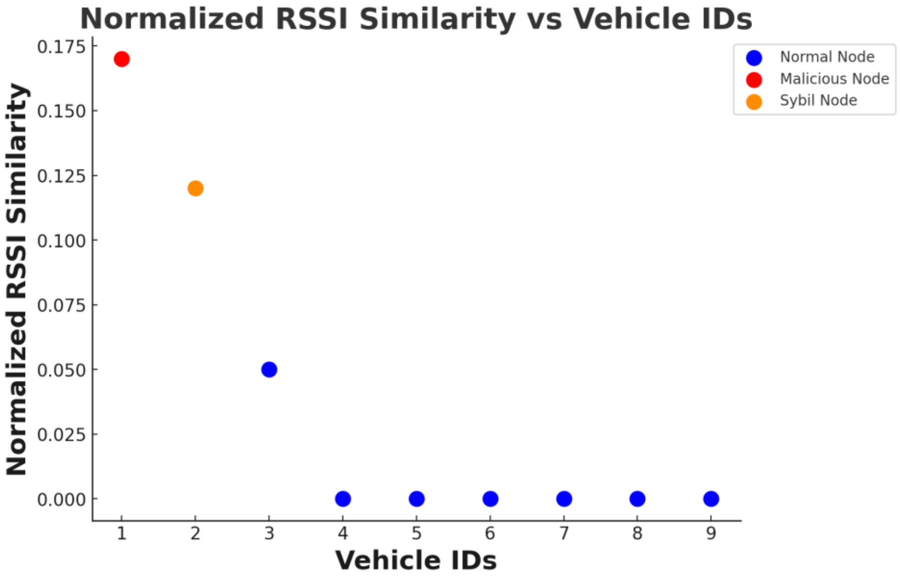


The network topology in VANETs includes 1 malicious node (sybil attacker in red) and 2 illusion (sybil node in orange) and 3 normal vehicular (in blue)

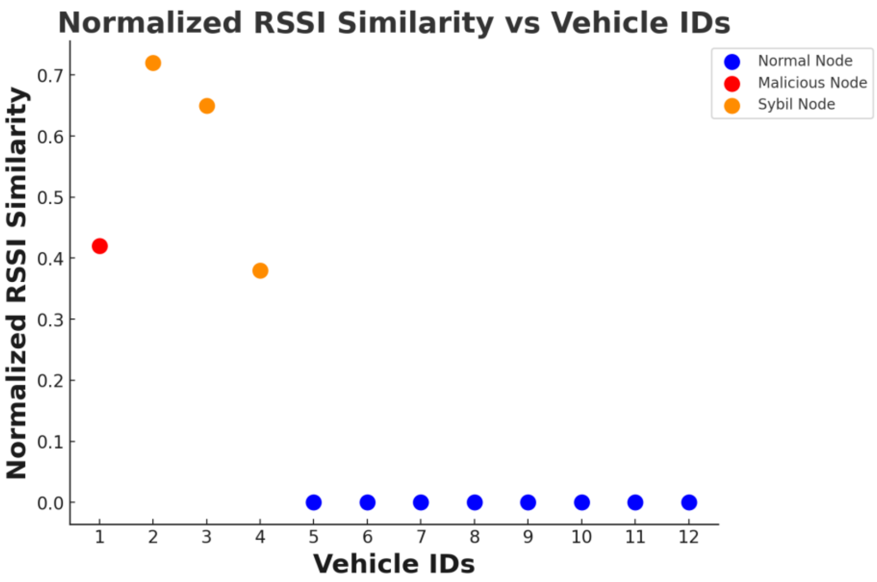
# Feature visualization: 5, 6 and 9 vehicles

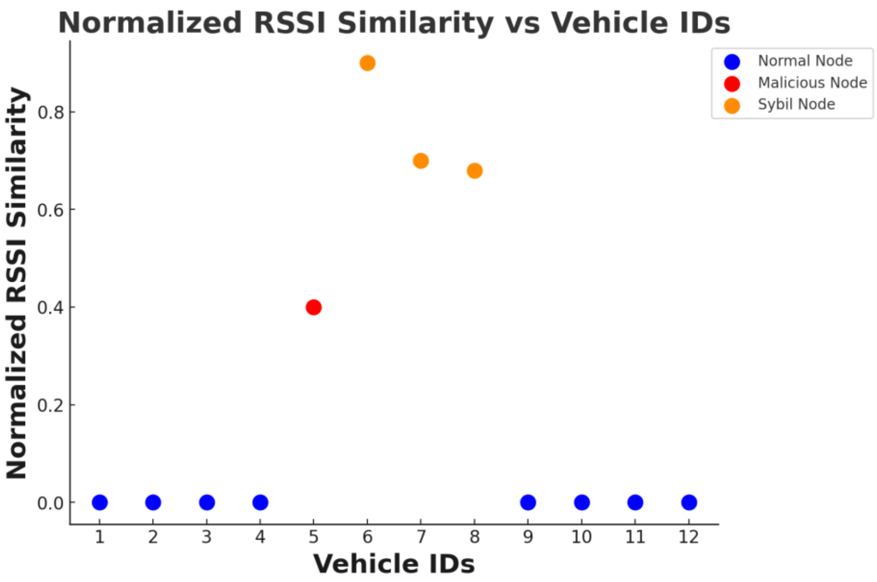

Normalized RSSI Similarity vs Vehicle IDs

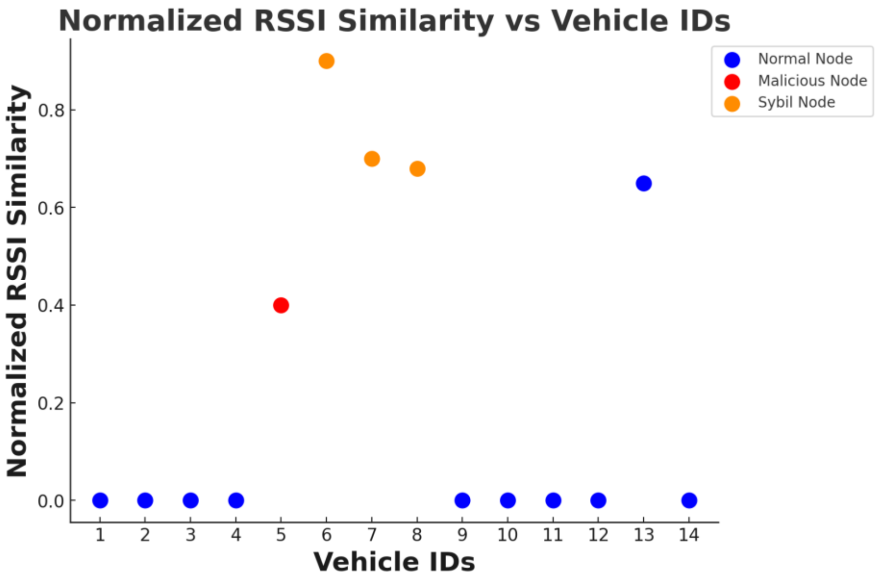

Normalized RSSI Similarity vs Vehicle IDs


Normalized RSSI Similarity vs Vehicle IDs

# Feature visualization: 12, 12 and 14 vehicles


Normalized RSSI Similarity vs Vehicle IDs


Normalized RSSI Similarity vs Vehicle IDs

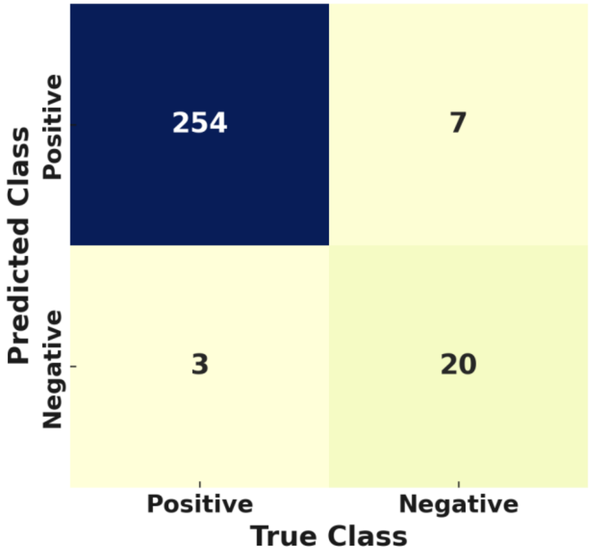

Normalized RSSI Similarity vs Vehicle IDs

# Confusion matrix

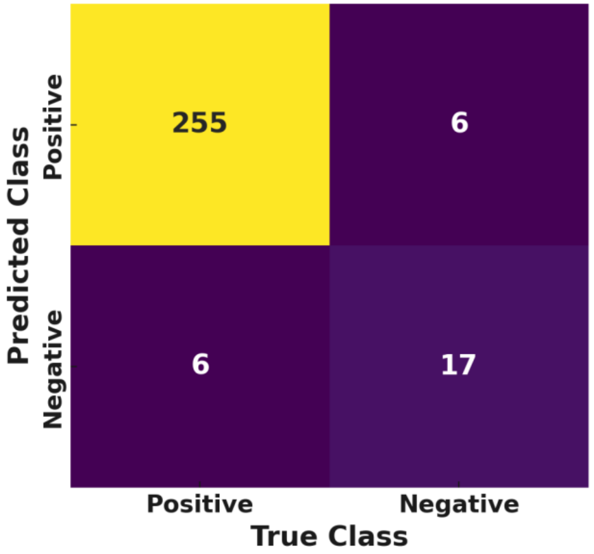
- Confusion matrix summarizes the performance of a machine learning model on a set of test data.
    - Displays the number of accurate and inaccurate instances based on the model’s predictions.
    - Used to measure the performance of classification models
- Confusion matrix components:
    - True Positive (TP): Predicted as positive, and it actually is positive.
    - True Negative (TN): Predicted as negative, and it actually is negative.
    - False Positive (FP): Predicted as positive, but it is actually negative.
    - False Negative (FN): Predicted as negative, but it is actually positive
- Performance metrics:
    - Accuracy: The overall correct predictions (TP + TN) divided by the total number of instances.
    - Precision: The number of true positives divided by the total number of predicted positives (TP + FP).
    - Recall: The number of true positives divided by the total number of actual positives (TP + FN).
    - F1 Score: The harmonic mean of precision and recall, providing a balance between the two.

# Confusion Matrix: Accuracy, Precision, F1 Score, Recall

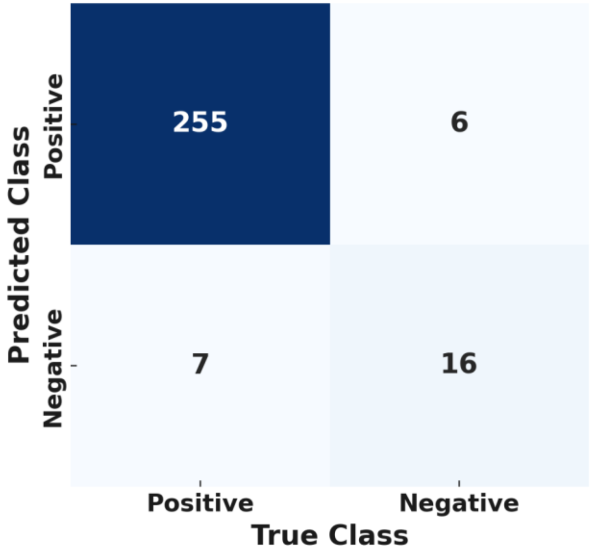
Confusion Matrix for Random Forest Classifier



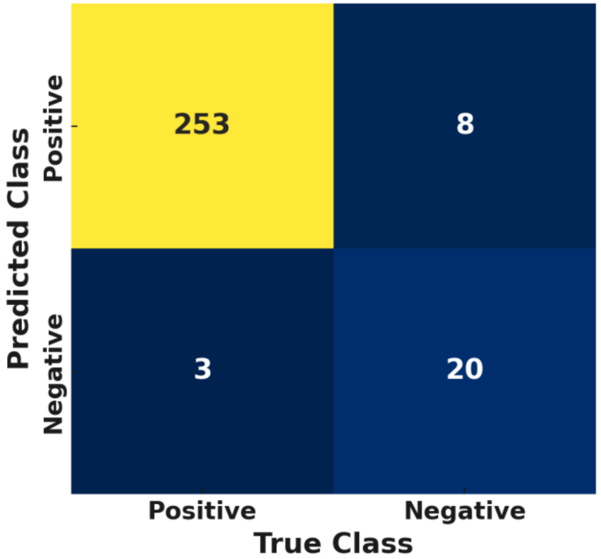
Confusion Matrix for Gradient Boost Classifier



Confusion Matrix for K-Nearest Neighbour Classifier



Confusion Matrix for Decision Tree Classifier



# Comparison and future work

| Classifier | True Positives | True Negatives | False Positives | False Negatives | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|---|---|
| Random Forest | 254 | 20 | 7 | 3 | 0.96 | 0.74 | 0.87 | 0.80 |
| KNN | 255 | 16 | 6 | 7 | 0.95 | 0.73 | 0.70 | 0.71 |
| xgBoost | 255 | 17 | 6 | 6 | 0.97 | 0.80 | 0.87 | 0.83 |
| Decision Tree | 255 | 20 | 8 | 3 | 0.96 | 0.74 | 0.74 | 0.74 |

Key Observations

- Accuracy: All classifiers perform well, with accuracy ranging from 95% to 97%.
- Precision: Varies from 73% (KNN) to 80% (xgBoost),
- Recall: Varies from 70% to 87% suggesting good but not perfect detection of Sybil nodes.
- F1 Scores: Range from 0.71 to 0.83; a reasonable performance; xgBoost has the best score.

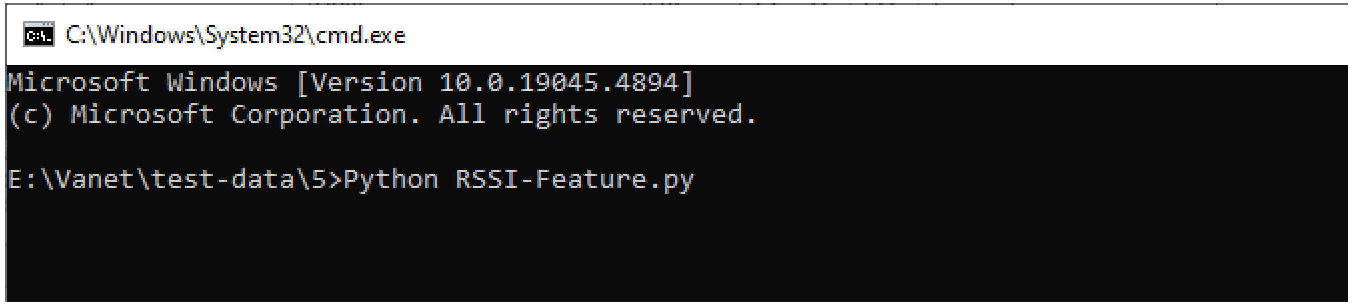
Future Work

- Test with larger networks to assess scalability.
- Experiment with ensemble methods to potentially improve overall performance.
- Impact of different VANET scenarios (e.g., urban vs. highway) on classifier performance.
- Explore ways to improve precision

# Appendix: How-to-Guide

# How to classify data?

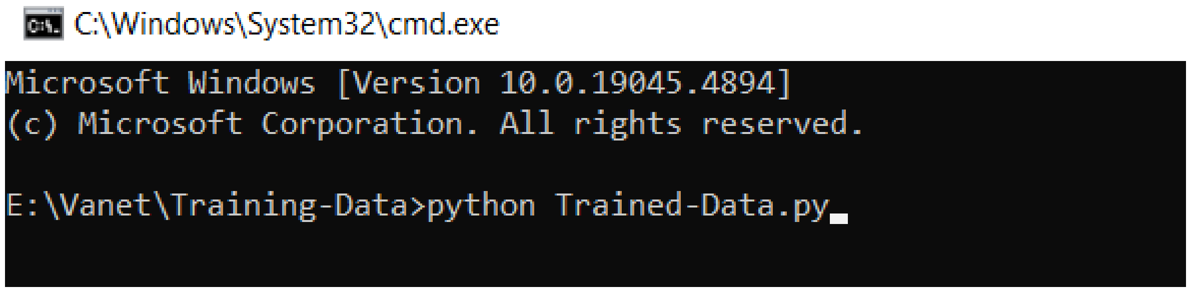
- To generate an excel file containing  3 feature messages for each vehicle, follow these steps,
- Modify the file paths in the python script according your setup.
- Open the command prompt.
- Navigate the folder containing python script.
- Run the “RSSI-Feature.py” script to process the radio measurement file and generate excel file.
- You can place the python script anywhere, as long as the file paths are correctly set to locate the necessary data files, including the test and training data scenarios that contain the radio measurement file.
- After running the script, it will generate the RSSI-features.csv file in the folder contains radio measurement log.



```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

E:\Vanet\test-data\5>Python RSSI-Feature.py
```

# How to Normalize the data?

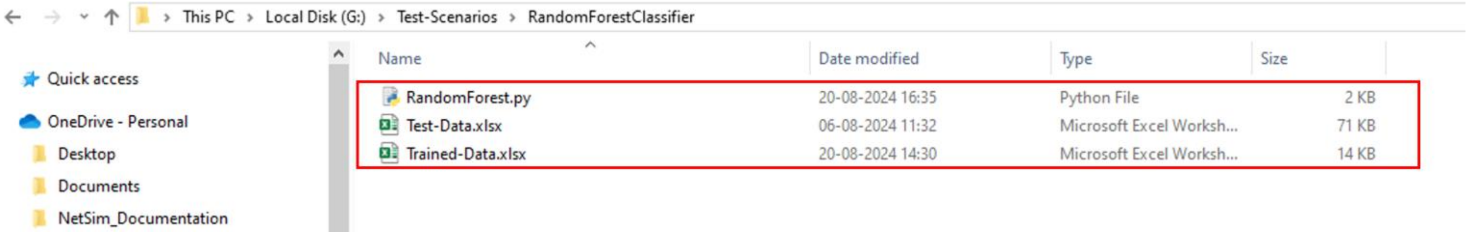
- To generate an excel file containing the normalization of 3 feature messages of all the test and training  scenarios, follow these steps:
- Modify the file paths and input files required for merging the data into an Excel file in the Python script according to your setup.
- Open the command prompt.
- Run Trained.py separately to obtain all features in a new Excel file. Manually provide the similarity score.
- Run Test-Data.py to extract all features into a single Excel file.



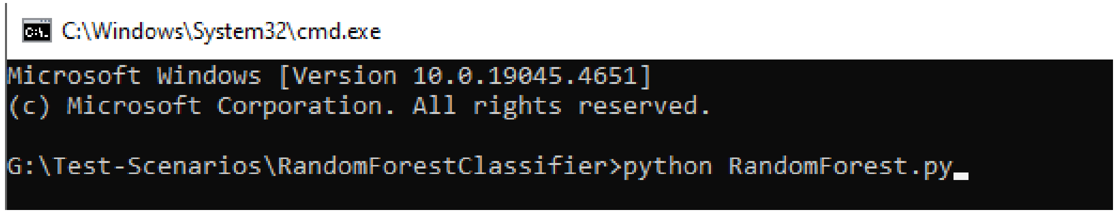
```
Microsoft Windows [Version 10.0.19045.4894]
(c) Microsoft Corporation. All rights reserved.

E:\Vanet\Training-Data>python Trained-Data.py_
```

# How to run the classifiers

- After normalizing the data, name it 'Test-Data’.
- Run each classifier against the trained data by providing the file locations for both the Trained-Data and Test data.
- Place the Test-Data , and Training-data in the same folder.
- Modify the path in the python script of each classifier according to your setup.
- Open the command prompt and run the script as shown below


This PC › Local Disk (G:) › Test-Scenarios › RandomForestClassifier
Quick access
OneDrive - Personal
Desktop
Documents
NetSim_Documentation

| Name | Date modified | Type | Size |
|---|---|---|---|
| RandomForest.py | 20-08-2024 16:35 | Python File | 2 KB |
| Test-Data.xlsx | 06-08-2024 11:32 | Microsoft Excel Worksh... | 71 KB |
| Trained-Data.xlsx | 20-08-2024 14:30 | Microsoft Excel Worksh... | 14 KB |

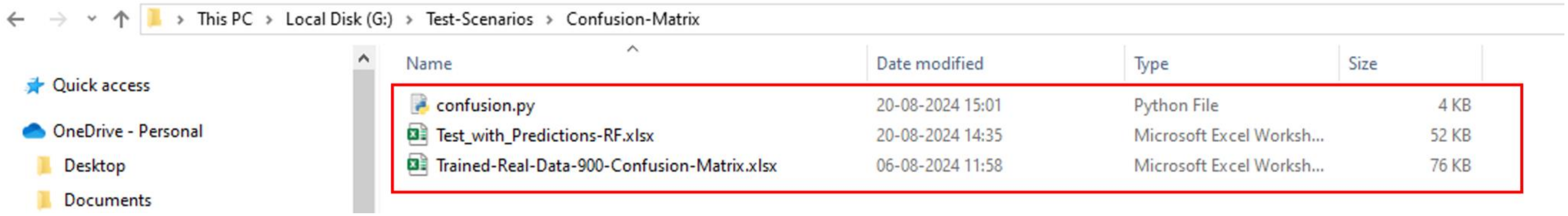


```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

G:\Test-Scenarios\RandomForestClassifier>python RandomForest.py_
```

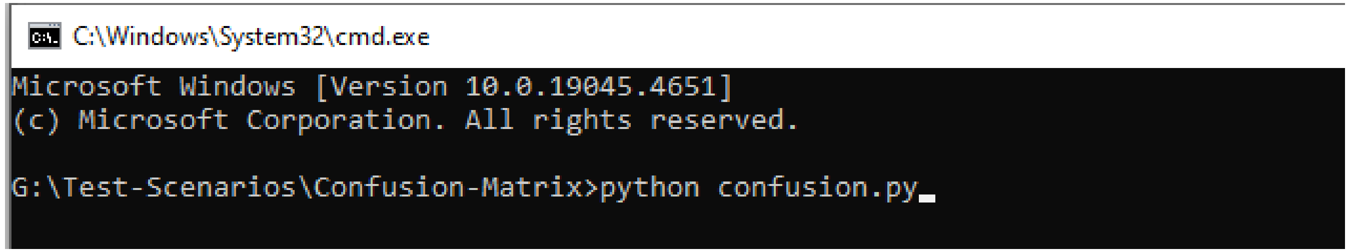
The Python script generates four sets of predicted labels and uses this data to create confusion matrices.

# How to get the confusion matrix?

- After obtaining the predicted labels from the classifiers,
- Use these predicted label excel files along with the real training data to run the python script and generate the confusion matrix.
- Place the predicted label excel files, and the real training data in the same folder.
- Modify the paths in the python script of Confusion-Matrix according to your setup.
- Open the command prompt and run the script as shown below.
- Each classifier's data generates a confusion matrix for that classifier


This PC › Local Disk (G:) › Test-Scenarios › Confusion-Matrix
Quick access
OneDrive - Personal
Desktop
Documents

| Name | Date modified | Type | Size |
|---|---|---|---|
| confusion.py | 20-08-2024 15:01 | Python File | 4 KB |
| Test_with_Predictions-RF.xlsx | 20-08-2024 14:35 | Microsoft Excel Worksh... | 52 KB |
| Trained-Real-Data-900-Confusion-Matrix.xlsx | 06-08-2024 11:58 | Microsoft Excel Worksh... | 76 KB |



```
Microsoft Windows [Version 10.0.19045.4651]
(c) Microsoft Corporation. All rights reserved.

G:\Test-Scenarios\Confusion-Matrix>python confusion.py
```