

Configuration Manual

Practicum Part 2
Msc Cyber Security

Anusha Varghese
Student ID: 23217693

School of Computing
National College of Ireland

Supervisor: Mr. Vikas Sahni

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Anusha Varghese
Student ID: 23217693
Programme: Msc Cyber Security **Year:** 2024 - 2025
Module: Practicum Part 2
Lecturer: Mr. Vikas Sahni
Submission Due Date: 12 December 2024
Project Title: Enhancing Vehicle Security: Intrusion Detection Using Machine Learning

Word Count: 744

Page Count: 7

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Anusha Varghese

Date: 12 December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anusha Varghese
Student ID: 23217693

1 Introduction

This manual defines the framework for the intrusion detection system based on machine learning. It describes its purpose and scope, including high false positives, real-time processing, and scalability across different vehicle models.

2 Hardware Specifications

- Processor: Intel Core i7.
- Memory (RAM): 8GB minimum.
- Storage: 512 GB minimum.
- Peripherals: Input/output devices for monitoring and testing.

3 Software Specifications

- Operating System: Windows 11.
- Programming Language: Python 3.8.10.
- Libraries: pandas, numpy, scikit-learn, matplotlib, imbalanced-learn.
- Development Environment: Jupyter Notebook.

4 System Requirements

4.1 Functional Requirements

- Support for heterogeneous automotive datasets.
- Ability to handle imbalanced datasets using advanced resampling techniques.
- Feature engineering to enhance interpretability and accuracy.
- Deployment of machine learning algorithms with high classification performance.
- Output detailed metrics like confusion matrices and evaluation scores.

4.2 Non-Functional Requirements

- Real-time detection capability.
- Scalability for large datasets.
- Robustness to noisy or incomplete inputs.
- Modular design for ease of maintenance.

5 Data Preparation

5.1 Data Collection

```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import ast
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import SelectKBest, mutual_info_classif
```

In [2]:

```
# Load Datasets
attack_free_df = pd.read_csv('Attack_free_dataset.csv')
dos_attack_df = pd.read_csv('DoS_attack_dataset.csv')

print("DoS Attack Data:\n", dos_attack_df.head())
print("Attack-Free Data:\n", attack_free_df.head())
```

Figure 1: Importing Libraries and Dataset

- Two datasets: attack-free and DoS-attack datasets with over 2.8 million combined entries.
- Key features:
 - Timestamp: Floating-point representation of the time.
 - Identifier: Unique network message identifiers.
 - RTR and DLC: Indicators of data transmission characteristics.
 - Data: Hexadecimal strings encapsulating payloads.

5.2 Preprocessing

```
# Data Cleaning
# For DoS Attack Dataset, we handle missing values by filling them with default values.
dos_attack_df['Data'] = dos_attack_df['Data'].fillna(['00', '00', '00', '00', '00', '00', '00', '00'])

print("Missing Values in DoS Attack Dataset after filling:", dos_attack_df['Data'].isnull().sum())
```

Missing Values in DoS Attack Dataset after filling: 0

Figure 2: Missing Value Removal

- Missing values replaced with a default placeholder payload (['00'] * 8).
- Payload data transformed from hexadecimal to numerical lists.
- Removed insignificant columns like RTR to reduce the computational burden.

5.3 Data Balancing

```
: # Balance the dataset using SMOTE (oversampling minority class)
print("Class distribution before SMOTE:\n", y.value_counts())

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Checking class imbalance after applying SMOTE
print("Class distribution after SMOTE:")
print(y_resampled.value_counts())

Class distribution before SMOTE:
label
0    2268519
1     656579
Name: count, dtype: int64
Class distribution after SMOTE:
label
0    2268519
1    2268519
Name: count, dtype: int64
```

Figure 3: Data Balancing

- Used Synthetic Minority Over-sampling Technique (SMOTE) to handle the class imbalance.

6 Feature Engineering

6.1 Extracted Features

```
# Feature Extraction from 'Data_processed' (min, max, mean, std deviation of byte values)
def extract_features(df):
    df['min_data'] = df['Data_processed'].apply(min)
    df['max_data'] = df['Data_processed'].apply(max)
    df['mean_data'] = df['Data_processed'].apply(lambda x: sum(x) / len(x))
    df['std_data'] = df['Data_processed'].apply(lambda x: pd.Series(x).std())
    return df

# Apply feature extraction to both datasets
attack_free_df = extract_features(attack_free_df)
dos_attack_df = extract_features(dos_attack_df)
```

Figure 4: Feature Extraction

- Min Value: Minimum byte value in the payload.
- Max Value: Maximum byte value in the payload.
- Mean Value: Average byte value.
- Standard Deviation: Variance in byte values.

6.2 Feature Selection

```
# Create Labels
# Label the attack-free dataset as 0 and DoS attack dataset as 1
attack_free_df['label'] = 0
dos_attack_df['label'] = 1

# Combine Both Datasets
combined_df = pd.concat([attack_free_df, dos_attack_df])

# Define Features and Labels
# Select the feature columns for training
feature_columns = ['min_data', 'max_data', 'mean_data', 'std_data']
X = combined_df[feature_columns]
y = combined_df['label']

# Feature selection using SelectKBest
selector = SelectKBest(score_func=mutual_info_classif, k='all')
X_selected = selector.fit_transform(X, y)

# Show the selected features and their scores
feature_scores = selector.scores_
print("Feature Scores:\n", list(zip(feature_columns, feature_scores)))
```

Feature Scores:
[('min_data', 0.0001142296457965486), ('max_data', 0.21371457013783934), ('mean_data', 0.20357265719018858), ('std_data', 0.29770359448595785)]

Figure 5: Best Feature Selection

- Utilized SelectKBest with mutual information to rank features.

7 Model Development

7.1 Machine Learning Models

```
# Model Training - Random Forest Classifier
rf_clf = RandomForestClassifier(random_state=42, class_weight='balanced')
rf_clf.fit(X_train_scaled, y_train)

# Model Training - Voting Classifier (Random Forest + Logistic Regression)
log_clf = LogisticRegression(max_iter=1000, random_state=42, class_weight='balanced')
voting_clf = VotingClassifier(estimators=[('rf', rf_clf), ('lr', log_clf)], voting='hard')
voting_clf.fit(X_train_scaled, y_train)
```

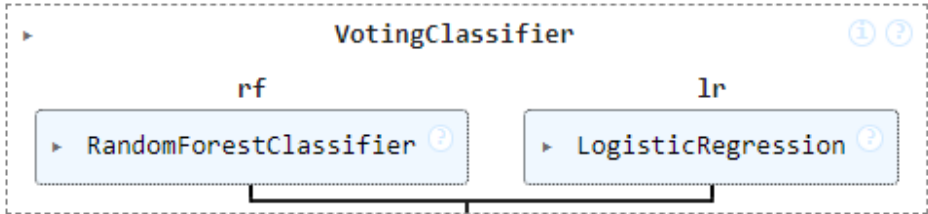


Figure 6: ML Models and Training

- Random Forest Classifier: Ensemble method for robust classification.
- Voting Classifier: Combines Random Forest and Logistic Regression for higher accuracy.

7.2 Training Workflow

- Splitting the dataset into 70% training and 30% testing sets.
- Normalizing numerical features using StandardScaler.

8 Evaluation Metrics

```
from sklearn.metrics import confusion_matrix

# Model Evaluation for Random Forest
rf_predictions = rf_clf.predict(X_test_scaled)
rf_accuracy = accuracy_score(y_test, rf_predictions)
rf_precision = precision_score(y_test, rf_predictions)
rf_recall = recall_score(y_test, rf_predictions)
rf_f1 = f1_score(y_test, rf_predictions)
rf_conf_matrix = confusion_matrix(y_test, rf_predictions)

# Model Evaluation for Voting Classifier
voting_predictions = voting_clf.predict(X_test_scaled)
voting_accuracy = accuracy_score(y_test, voting_predictions)
voting_precision = precision_score(y_test, voting_predictions)
voting_recall = recall_score(y_test, voting_predictions)
voting_f1 = f1_score(y_test, voting_predictions)
voting_conf_matrix = confusion_matrix(y_test, voting_predictions)

# Print Results
print("Random Forest Results:")
print(f"Accuracy: {rf_accuracy}")
print(f"Precision: {rf_precision}")
print(f"Recall: {rf_recall}")
print(f"F1-Score: {rf_f1}")
print(f"Confusion Matrix:\n {rf_conf_matrix}")

print("\nVoting Classifier Results:")
print(f"Accuracy: {voting_accuracy}")
print(f"Precision: {voting_precision}")
print(f"Recall: {voting_recall}")
print(f"F1-Score: {voting_f1}")
print(f"Confusion Matrix:\n {voting_conf_matrix}")
```

Figure 7: Model Performance Metrics

- Accuracy: Ratio of correctly classified instances.
- Precision: True positive predictions out of total predicted positives.
- Recall: True positive predictions out of actual positives.
- F1-Score: Harmonic mean of precision and recall.
- Confusion matrices for detailed classification insights.

9 Implementation Pipeline

9.1 Steps

- Data Ingestion: Load and normalize data.
- Preprocessing: Handle missing values and transform data.
- Feature Engineering: Extract statistical attributes.
- Model Training: Apply Random Forest and Voting Classifier.
- Evaluation: Measure performance using metrics and confusion matrices.

9.2 Code Modularization

Functions:

- `load_data`: Load datasets.
- `process_data_column`: Convert payloads to numeric.
- `extract_features`: Derive statistical features.
- `train_models`: Train classifiers and optimize parameters.
- `evaluate_models`: Calculate and visualize metrics.

10 Results and Discussion

10.1 Random Forest Classifier

- Accuracy: 89.61%
- Precision: 91.24%
- Recall: 87.64%
- F1-Score: 89.41%

10.2 Voting Classifier

- Accuracy: 76.16%
- Precision: 88.84%
- Recall: 59.86%
- F1-Score: 71.53%

10.3 Limitations

- Feature scalability remains constrained by handcrafted engineering.
- Computational efficiency challenges in real-time applications.

11 Future Directions

- Automated feature extraction using deep learning techniques like CNN.
- Broaden attack scope to include spoofing and replay attacks.
- Optimize models for deployment in resource-constrained environments using techniques like quantization or FPGA acceleration.

References

- Makarfi, A.U., Rabie, K.M., Kaiwartya, O., Li, X. and Kharel, R., 2020, May. Physical layer security in vehicular networks with reconfigurable intelligent surfaces. In 2020 IEEE 91st vehicular technology conference (VTC2020-Spring) (pp. 1-6). IEEE.
- Moulahi, T., Zidi, S., Alabdulatif, A. and Atiquzzaman, M., 2021. Comparative performance evaluation of intrusion detection based on machine learning in in-vehicle controller area network bus. IEEE Access, 9, pp.99595-99605.
- Mourad, A., Tout, H., Wahab, O.A., Otrouk, H. and Dbouk, T., 2020. Ad hoc vehicular fog enabling cooperative low-latency intrusion detection. IEEE Internet of Things Journal, 8(2), pp.829-843.
- Narasimhan, H., Ravi, V. and Mohammad, N., 2021. Unsupervised deep learning approach for in-vehicle intrusion detection system. IEEE Consumer Electronics Magazine, 12(1), pp.103-108.
- Pascale, F., Adinolfi, E.A., Coppola, S. and Santonicola, E., 2021. Cybersecurity in automotive: An intrusion detection system in connected vehicles. Electronics, 10(15), p.1765.

Wang, K., Zhang, A., Sun, H. and Wang, B., 2022. Analysis of recent deep-learning-based intrusion detection methods for in-vehicle network. *IEEE Transactions on Intelligent Transportation Systems*, 24(2), pp.1843-1854.

Yang, Y., Duan, Z. and Tehranipoor, M., 2020. Identify a spoofing attack on an in-vehicle CAN bus based on the deep features of an ECU fingerprint signal. *Smart Cities*, 3(1), pp.17-30.