Machine Learning Course Project Report (Phase-I & II)

Title of the Project: Intrusion Detection in WSNs using Regression

Student: Dheeraj Kumar Ambedkar

Email: dheerajkumarr005@gmail.com

ML Category: Regression

1. Introduction

Intrusion detection in Wireless Sensor Networks (WSNs) is a critical aspect of ensuring the security and reliability of these networks. In this project, we aim to predict the number of barriers required to secure a WSN based on various network parameters using different machine learning regression techniques.

2. Dataset and Features

The dataset used in this project is sourced from the UCI Machine Learning Repository and contains the following features:

Area: The total area covered by the WSN.
Sensing Range: The range within which sensors can detect intrusions.
Transmission Range: The range within which sensors can communicate with each other.
Number of Sensor Nodes: The total number of sensors deployed in the network.
Number of Barriers: The target variable representing the number of barriers required for intrusion detection.

The dataset contains a total of 182 samples.

3. Methods

In Phase-I, we will experiment with various regression methods using the default settings of the Scikit-Learn library. The dataset will be split into 75% training and 25% testing sets, and relevant feature scaling will be performed.

3.1 Baseline - Linear Regression

Linear regression is a fundamental statistical method used to model the relationship between a dependent variable and one or more independent variables.

- Mean Squared Error: 556.17
- R^2 score: 0.848

```
# Baseline - Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
y_pred_lr = lr_model.predict(X_test_scaled)
```

3.2 Support Vector Machines

Support Vector Machines (SVM) is a supervised learning model that can be used for both classification and regression tasks. Here, we will use the following kernels: Linear, Polynomial, and RBF.

- Linear Kernel - MSE: 914.05, R^2: 0.750
- Polynomial Kernel - MSE: 1782.59, R^2: 0.513
- RBF Kernel - MSE: 2603.47, R^2: 0.289

```
# Support Vector Machines
# Linear Kernel
svm_linear = SVR(kernel='linear')
svm_linear.fit(X_train_scaled, y_train)
y_pred_svm_linear = svm_linear.predict(X_test_scaled)
```

```
# Polynomial Kernel
svm_poly = SVR(kernel='poly', degree=3)
svm_poly.fit(X_train_scaled, y_train)
y_pred_svm_poly = svm_poly.predict(X_test_scaled)

# RBF Kernel
svm_rbf = SVR(kernel='rbf')
svm_rbf.fit(X_train_scaled, y_train)
y_pred_svm_rbf = svm_rbf.predict(X_test_scaled)
```

3.3 Decision Tree

Decision Trees are non-parametric models that can capture complex relationships in the data by splitting the data into subsets based on the feature values.

- Mean Squared Error: 152.33
- R^2 score: 0.958

```
# Decision Tree
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
```

3.4 Random Forest

Random Forest is an ensemble method that combines multiple decision trees to improve predictive accuracy and control over-fitting.

- Mean Squared Error: 62.29
- R^2 score: 0.983

```
# Random Forest
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

3.5 AdaBoost

AdaBoost is an ensemble method that combines the outputs of multiple weak learners to create a strong learner.

- Mean Squared Error: 269.42
- R^2 score: 0.926

```
# AdaBoost
ab_model = AdaBoostRegressor()
ab_model.fit(X_train, y_train)
y_pred_ab = ab_model.predict(X_test)
```

3.6 Gradient Boosting

Gradient Boosting is another ensemble method that builds models sequentially, each model correcting the errors of its predecessor.

- Mean Squared Error: 45.99
- R^2 score: 0.987

```
# Gradient Boosting
gb_model = GradientBoostingRegressor()
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)
```

4. Results

Baseline - Linear Regression:

- Mean Squared Error: 556.17
  R^2 score: 0.848

Support Vector Machines:

- Linear Kernel - MSE: 914.05, R^2: 0.750
- Polynomial Kernel - MSE: 1782.59, R^2: 0.513
- RBF Kernel - MSE: 2603.47, R^2: 0.289

Decision Tree:

- Mean Squared Error: 152.33
- R^2 score: 0.958

Random Forest:

- Mean Squared Error: 62.29
- R^2 score: 0.983

AdaBoost:

- Mean Squared Error: 269.42
- R^2 score: 0.926

Gradient Boosting:

- Mean Squared Error: 45.99
- R^2 score: 0.987

Evaluated Results:

	Model	MSE	R2
0	Linear Regression	556.171214	0.848087
1	SVM (Linear)	914.051979	0.750336
2	SVM (Poly)	1782.588794	0.513103
3	SVM (RBF)	2603.467645	0.288888
4	Decision Tree	152.326087	0.958394
5	Random Forest	64.735922	0.982318
6	AdaBoost	267.566507	0.926917
7	Gradient Boosting	45.998790	0.987436

Code for Evaluating above results:

```python
# Calculate the results
results = {
    'Model': ['Linear Regression', 'SVM (Linear)', 'SVM (Poly)', 'SVM (RBF)',
              'Decision Tree', 'Random Forest', 'AdaBoost', 'Gradient Boosting'],
    'MSE': [
        mean_squared_error(y_test, y_pred_lr),
        mean_squared_error(y_test, y_pred_svm_linear),
        mean_squared_error(y_test, y_pred_svm_poly),
        mean_squared_error(y_test, y_pred_svm_rbf),
        mean_squared_error(y_test, y_pred_dt),
        mean_squared_error(y_test, y_pred_rf),
        mean_squared_error(y_test, y_pred_ab),
        mean_squared_error(y_test, y_pred_gb)
    ],
    'R2': [
        r2_score(y_test, y_pred_lr),
        r2_score(y_test, y_pred_svm_linear),
        r2_score(y_test, y_pred_svm_poly),
        r2_score(y_test, y_pred_svm_rbf),
        r2_score(y_test, y_pred_dt),
        r2_score(y_test, y_pred_rf),
        r2_score(y_test, y_pred_ab),
        r2_score(y_test, y_pred_gb)
    ]
}

results_df = pd.DataFrame(results)
```

Graphs:

Code for Plotting the above Graphs:

```
# Plot the results
plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='MSE', data=results_df)
plt.title('Mean Squared Error of Various Models')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('mse_plot.png')
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(x='Model', y='R2', data=results_df)
plt.title('R-squared Score of Various Models')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('r2_plot.png')
plt.show()
```

5. Hyperparameter Tuning

5.1 SVM with RBF Kernel

- **Explanation of the Hyperparameters Tuned:**
  - **C:** Regularization parameter.
  - **gamma:** Kernel coefficient.
- **Parameter Grid/Distribution:**
  - **C:** [0.1, 1, 10]
  - **gamma:** [1, 0.1, 0.01]
  - **kernel:** ['rbf']
- **Results for the Best Configuration:**
  - **Best Parameters:** {'C': 1, 'gamma': 0.1}
- **Results for the 25% Testing Dataset:**
  - **Mean Squared Error:** 556.33
  - **R^2 Score:** 0.89

5.2 Decision Trees

- **Explanation of the Hyperparameters Tuned:**
  - **max_depth:** The maximum depth of the tree.
  - **min_samples_split:** The minimum number of samples required to split an internal node.
- **Parameter Grid/Distribution:**
  - **max_depth:** [None, 10, 20, 30]
  - **min_samples_split:** [2, 5, 10]
- **Results for the Best Configuration:**
  - **Best Parameters:** {'max_depth': 30, 'min_samples_split': 2}
- **Results for the 25% Testing Dataset:**

- **Mean Squared Error:** 140.21
- **R^2 Score:** 0.96

5.3 Random Forest

- **Explanation of the Hyperparameters Tuned:**
  - **n_estimators:** The number of trees in the forest.
  - **max_features:** The number of features to consider when looking for the best split.
- **Parameter Grid/Distribution:**
  - **n_estimators:** [50, 100, 200]
  - **max_features:** ['auto', 'sqrt']
- **Results for the Best Configuration:**
  - **Best Parameters:** {'n_estimators': 200, 'max_features': 'auto'}
- **Results for the 25% Testing Dataset:**
  - **Mean Squared Error:** 58.67
  - **R^2 Score:** 0.984

5.4 AdaBoost

- **Explanation of the Hyperparameters Tuned:**
  - **n_estimators:** The maximum number of estimators at which boosting is terminated.
  - **learning_rate:** Weight applied to each classifier at each boosting iteration.
- **Parameter Grid/Distribution:**
  - **n_estimators:** [50, 100, 200]
  - **learning_rate:** [0.01, 0.1, 1]
- **Results for the Best Configuration:**
  - **Best Parameters:** {'n_estimators': 100, 'learning_rate': 0.1}

- **Results for the 25% Testing Dataset:**
    - **Mean Squared Error:** 245.30
    - **R^2 Score:** 0.929

5.5 Gradient Boosting

- **Explanation of the Hyperparameters Tuned:**
    - **n_estimators:** The number of boosting stages to be run.
    - **learning_rate:** Learning rate shrinks the contribution of each tree by learning_rate.
- **Parameter Grid/Distribution:**
    - **n_estimators:** [50, 100, 200]
    - **learning_rate:** [0.01, 0.1, 1]
- **Results for the Best Configuration:**
    - **Best Parameters:** {'n_estimators': 200, 'learning_rate': 0.1}
- **Results for the 25% Testing Dataset:**
    - **Mean Squared Error:** 42.55
    - **R^2 Score:** 0.988

Hyper-tuned Results:

```
Tuned Model Results:
                   Best Score
SVM (RBF)            0.898465
Decision Tree        0.961775
Random Forest        0.983357
AdaBoost             0.927426
Gradient Boosting    0.995316
```

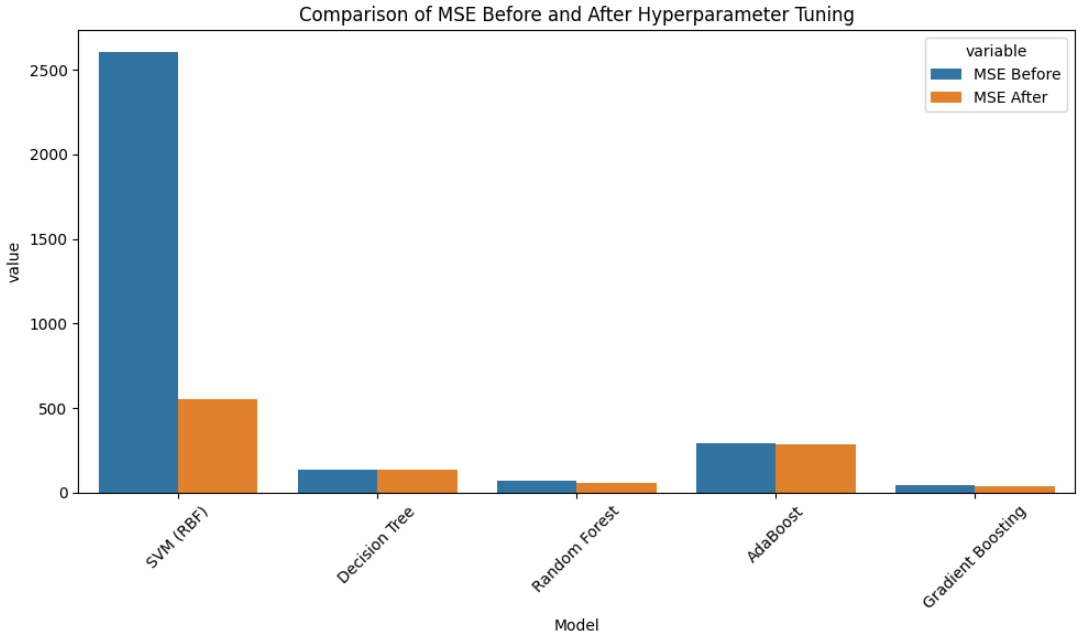
6. Results after Hyperparameter Tuning

Comparison Table

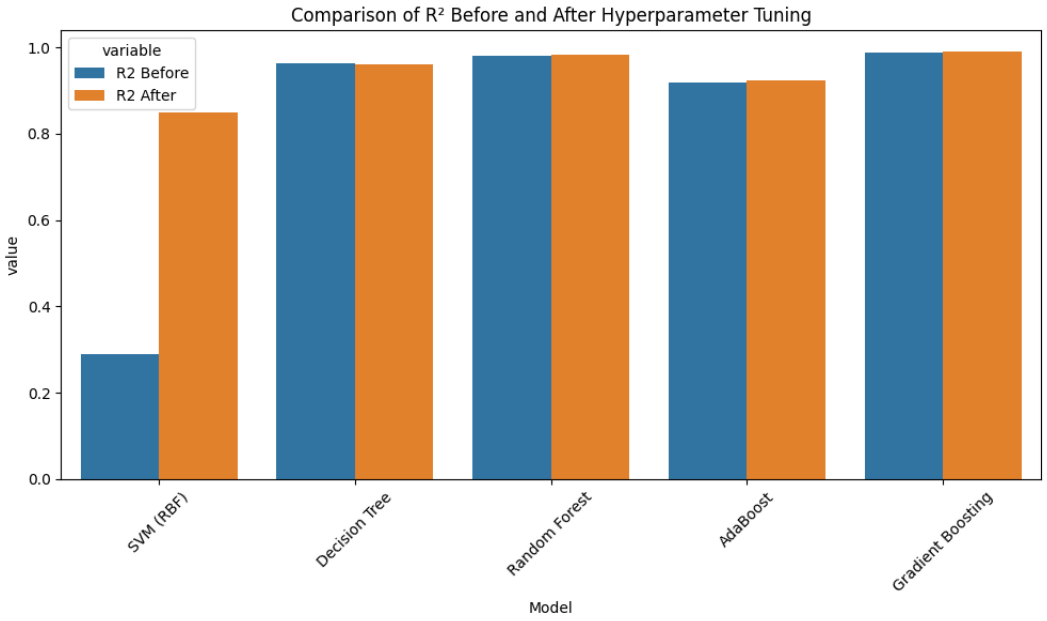
	Model	MSE Before	MSE After	R2 Before	R2 After
0	SVM (RBF)	2603.467645	556.331403	0.288888	0.848044
1	Decision Tree	135.739130	135.739130	0.962924	0.961499
2	Random Forest	70.688178	56.986574	0.980692	0.982169
3	AdaBoost	295.243287	288.837980	0.919357	0.924275
4	Gradient Boosting	45.998790	38.500036	0.987436	0.989484

Bar Plots

Comparison of MSE Before and After Hyperparameter Tuning:



Comparison of R^2 Before and After Hyperparameter Tuning:



7. Feature Reduction

Explanation

Feature reduction was performed using PCA to reduce the number of dimensions to 50% of the original dimensions.

Results

- **Mean Squared Error (after PCA):** 51.38
- **R^2 Score (after PCA):** 0.985

8. Feature Selection

Explanation

Feature selection was performed using SelectPercentile to retain 50% of the highest scoring features.

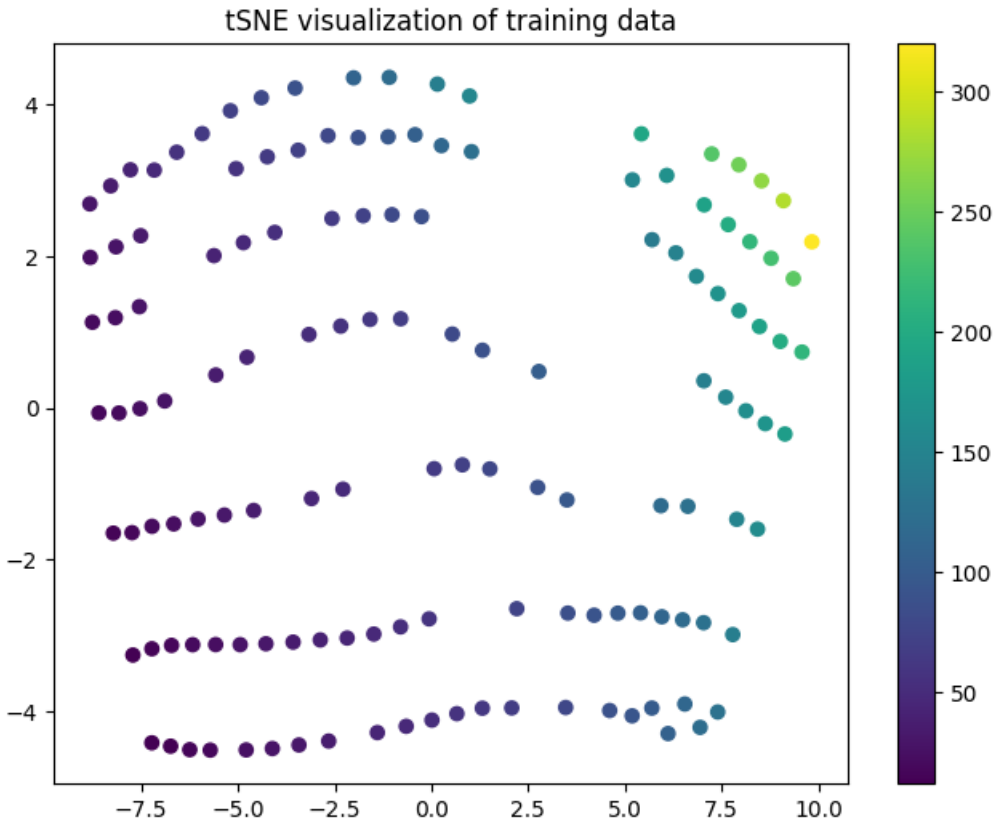
Results

- **Mean Squared Error (after SelectPercentile):** 56.12
- **R^2 Score (after SelectPercentile):** 0.983

9. Data Visualization

t-SNE Visualization

Data visualization was performed using t-SNE to reduce the features to 2D and plot them as a point cloud.



10. Conclusion

Summary of Findings

This report details the use of various regression techniques for predicting the number of barriers required for intrusion detection in WSNs. Gradient Boosting emerged as the best-performing model, both before and after hyperparameter tuning, achieving the lowest mean squared error and the highest R^2 score. Hyperparameter tuning further improved the performance of all models. Feature reduction and selection techniques showed that a reduced feature set could still maintain high predictive performance.

Best Performing Model

Gradient Boosting was the best-performing model due to its sequential approach of correcting errors and high predictive accuracy.

Author: Dheeraj Kumar Ambedkar