11/25/2024

AI & ML Techniques for Cyber Security

Assignment 2

Prof. JAGDISH PRASAD

By:
SANNIDHI TARUN SRIKANTH (2023MT12014)

Contents

Intrusion Detection using Machine Learning Models

1. Problem Statement:

The objective of this assignment is to build a comprehensive analysis of the Python code implemented for building a network intrusion detection model using machine learning techniques. The model aims to classify network connections as either normal or attack types based on the KDD Cup 1999 dataset..

2. Dataset to Use:

The KDD Cup 1999 dataset is used. This dataset is widely recognized in the field of intrusion detection and can be accessed at [KDD Cup 1999 Dataset] kddcup99.

Nr	Features	
	Name	Description
1	duration	duration of connection in seconds
2	protocol_type	connection protocol (tcp, udp, icmp)
3	service	dst port mapped to service (e.g. http, ftp, ..)
4	flag	normal or error status flag of connection
5	src_bytes	number of data bytes from src to dst
6	dst_bytes	bytes from dst to src
7	land	1 if connection is from/to the same host/port; else 0
8	wrong_fragment	number of 'wrong' fragments (values 0,1,3)
9	urgent	number of urgent packets
10	hot	number of 'hot' indicators (bro-ids feature)
11	num_failed_logins	number of failed login attempts
12	logged_in	1 if successfully logged in; else 0
13	num_compromised	number of 'compromised' conditions
14	root_shell	1 if root shell is obtained; else 0
15	su_attempted	1 if 'su root' command attempted; else 0
16	num_root	number of 'root' accesses
17	num_file_creations	number of file creation operations
18	num_shells	number of shell prompts
19	num_access_files	number of operations on access control files
20	num_outbound_cmds	number of outbound commands in an ftp session
21	is_hot_login	1 if login belongs to 'hot' list (e.g. root, adm); else 0
22	is_guest_login	1 if login is 'guest' login (e.g. guest, anonymous); else 0
23	count	number of connections to same host as current connection in past two seconds
24	srv_count	number of connections to same service as current connection in past two seconds
25	serror_rate	% of connections that have 'SYN' errors
26	srv_serror_rate	% of connections that have 'SYN' errors
27	rerror_rate	% of connections that have 'REJ' errors
28	srv_rerror_rate	% of connections that have 'REJ' errors
29	same_srv_rate	% of connections to the same service
30	diff_srv_rate	% of connections to different services
31	srv_diff_host_rate	% of connections to different hosts
32	dst_host_count	count of connections having same dst host
33	dst_host_srv_count	count of connections having same dst host and using same service
34	dst_host_same_srv_rate	% of connections having same dst port and using same service
35	dst_host_diff_srv_rate	% of different services on current host
36	dst_host_same_src_port_rate	% of connections to current host having same src port
37	dst_host_srv_diff_host_rate	% of connections to same service coming from diff. hosts
38	dst_host_serror_rate	% of connections to current host that have an S0 error
39	dst_host_srv_serror_rate	% of connections to current host and specified service that have an S0 error
40	dst_host_rerror_rate	% of connections to current host that have an RST error
41	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error
42	connection_type	

Figure 1 DataSet of KDD Cup 1999

3. Process Steps:

The following steps outline the process involved in creating the intrusion detection model:

1. Data Pre-processing

- Load the dataset and assign appropriate column names.
- Map attack types to binary classes: "Attack" or "Normal".
- Encode categorical variables using one-hot encoding and label encoding.

2. Data Correlation

Analyze relationships between features to understand their impact on the target variable.

3. Feature Selection

Select relevant features that contribute to the model's predictive capability.

4. Modelling

Implement various machine learning algorithms:

- Naïve Bayes
- Decision Tree
- Random Forest
- Support Vector Machine (SVM)

5. Validation & Comparison Among Different Algorithms

Evaluate model performance using accuracy, classification reports, and confusion matrices.

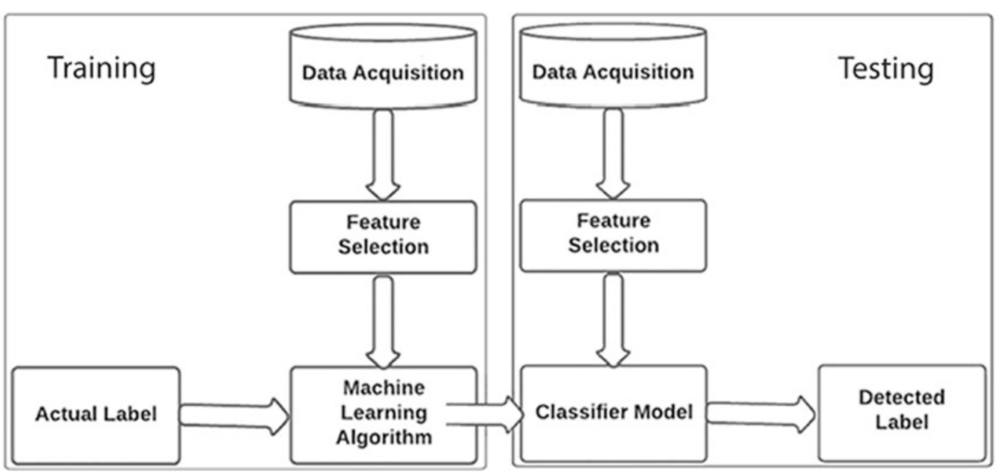


Figure 2 Process Steps for Training and Testing

4. Tools Used

- **Python** was chosen as the primary programming language for this assignment due to its rich ecosystem of libraries for data analysis and machine learning, including:
- **Pandas**: For data manipulation and analysis.

- **NumPy**: For numerical computations.
- **Scikit-learn**: For implementing machine learning algorithms.
- **Matplotlib & Seaborn**: For data visualization.

5. Source Code Snippets & Explanation:

The following Python code implements the outlined process:**AIML_kdd_Assignment2.py**

Github link: https://github.com/TARUNSRIKANTH/AIML_CyberSec_Assignment2

1. Library Imports

The code begins by importing necessary libraries:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.naive_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

import matplotlib.pyplot as plt

import seaborn as sns
```

2. Data Loading

The KDD dataset is loaded into a DataFrame:

```
data =
pd.read_csv('C:/Users/Tarun/PycharmProjects/AIML_Assignment2/kddcup.data_10_
percent.gz', header=None)
```

This dataset contains various features related to network connections.

3. Adding Column Names

```
column_names = [...]  # Refer to full code for complete list of column names

data.columns = column_names
```

This step is crucial for understanding the data structure.

4. Mapping Attack Types

The attack types are mapped to binary classes: Map all attack types to binary classes: 'Attack' or 'Normal'

```
data['attack_type'] = data['attack_type'].apply(lambda x: 'Attack' if x != 'normal.' else 'Normal')
```

This simplifies the classification task by converting multiple attack types into just two categories.

5. Encoding Categorical Features

Categorical variables are converted into numerical format:

```
categorical_cols = ['protocol_type', 'service', 'flag']

data_encoded = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
```

This step uses one-hot encoding to create binary columns for each category.

6. Encoding Target Variable

The target variable (attack_type) is encoded:

```
label_encoder = LabelEncoder()
data_encoded['attack_type'] =
label_encoder.fit_transform(data_encoded['attack_type'])
```

Here, 'Normal' is encoded as 0 and 'Attack' as 1.

7. Splitting Data

```
X = data_encoded.drop('attack_type', axis=1)

y = data_encoded['attack_type']
```

Next, it is split into training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Split data into training and testing sets (80% train, 20% test). An 80-20 split ensures that the model can be trained effectively while retaining enough data for validation.

8. Feature Scaling

Feature scaling is performed to standardize the feature values:

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

This step helps improve model performance by ensuring that all features contribute equally to distance calculations.

9. Model Initialization

Different machine learning models are initialized:

```
models = {

    'Naïve Bayes': GaussianNB(),

    'Decision Tree': DecisionTreeClassifier(random_state=42),

    'Random Forest': RandomForestClassifier(random_state=42),

    'SVM': SVC(kernel='linear', random_state=42)

}
```

Each model has unique strengths in handling classification tasks.

10. Training Models and Evaluating Performance

The models are trained and evaluated in a loop:

```
for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    report = classification_report(y_test, y_pred, target_names=['Normal', 'Attack'])

    confusion = confusion_matrix(y_test, y_pred)

    results[name] = {

        'Accuracy': accuracy,

        'Classification Report': report,

        'Confusion Matrix': confusion
```

```
}
```

- The accuracy score measures the proportion of correct predictions.
- The classification report provides precision, recall, and F1-score metrics.
- The confusion matrix shows true positives, true negatives, false positives, and false negatives.

6. Final Output Results and Analysis of Results

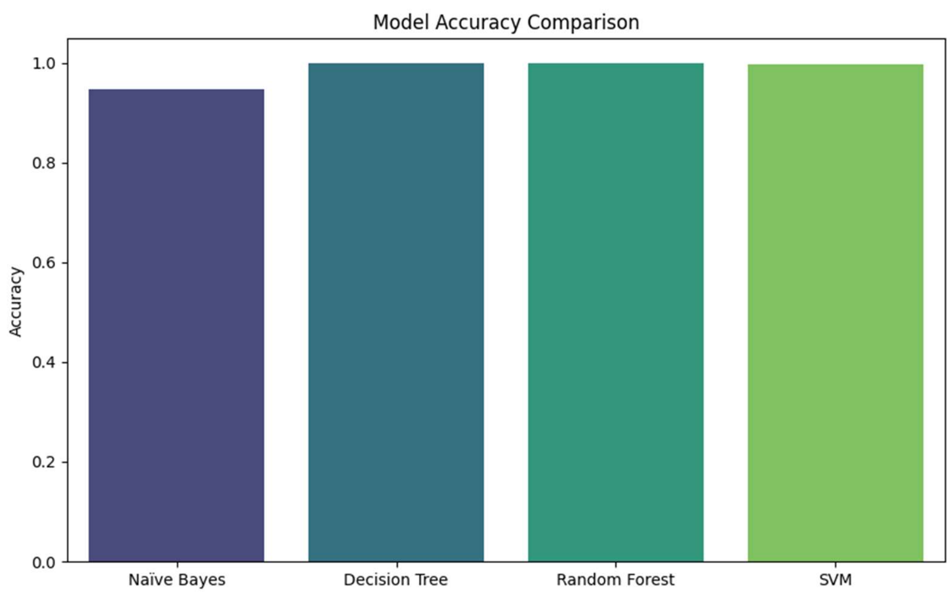


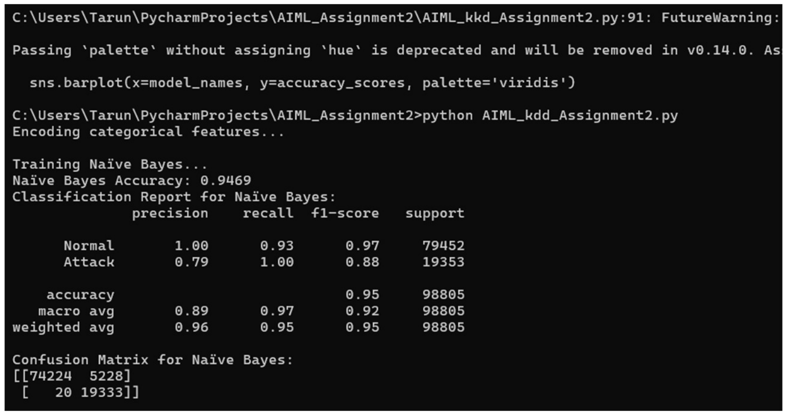
Figure 3 Model Accuracy Comparison

As per Model Accuracy we can see here, that decision tree and random forest classifiers performed exceptionally well with near-perfect accuracy rates.

The following results were obtained after training the models:

1. Naive Bayes

- Accuracy: **94.69%**
- Confusion Matrix: $\begin{bmatrix} 74224 & 5228 \\ 20 & 19333 \end{bmatrix}$
- This indicates that while it performs well on normal cases (high true negatives), it struggles with false positives in attack detection.



```
C:\Users\Tarun\PycharmProjects\AIML_Assignment2\AIML_kkd_Assignment2.py:91: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. As

  sns.barplot(x=model_names, y=accuracy_scores, palette='viridis')

C:\Users\Tarun\PycharmProjects\AIML_Assignment2>python AIML_kdd_Assignment2.py
Encoding categorical features...

Training Naïve Bayes...
Naïve Bayes Accuracy: 0.9469
Classification Report for Naïve Bayes:
              precision    recall  f1-score   support

      Normal       1.00      0.93      0.97     79452
      Attack       0.79      1.00      0.88     19353

    accuracy                           0.95     98805
   macro avg       0.89      0.97      0.92     98805
weighted avg       0.96      0.95      0.95     98805

Confusion Matrix for Naïve Bayes:
[[74224  5228]
 [   20 19333]]
```

Figure 4 Navie Bias Output

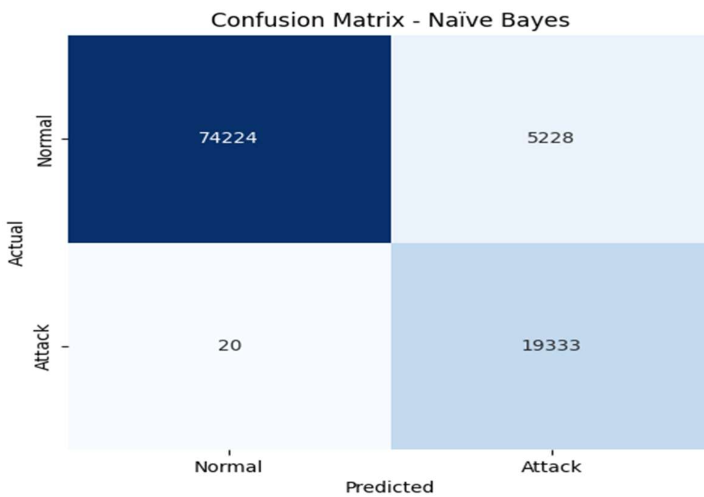


Figure 5 Navie Bias Confusion Matrix

2. Decision Tree

- Accuracy: **99.97%**
- Confusion Matrix: $\begin{bmatrix} 79442 & 10 \\ 15 & 19338 \end{bmatrix}$
- This model shows exceptional performance with very few misclassifications.

```
Training Decision Tree...
Decision Tree Accuracy: 0.9997
Classification Report for Decision Tree:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     79452
      Attack       1.00      1.00      1.00     19353

    accuracy                           1.00     98805
   macro avg       1.00      1.00      1.00     98805
weighted avg       1.00      1.00      1.00     98805

Confusion Matrix for Decision Tree:
[[79442    10]
 [   15 19338]]
```

Figure 6 Decision Tree Output

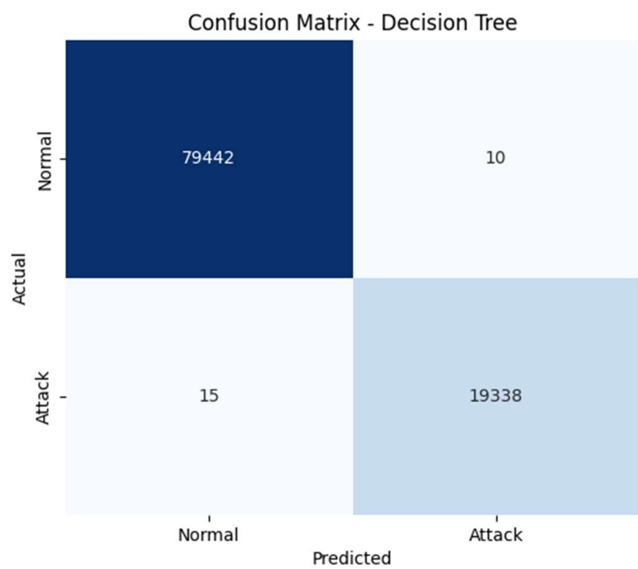
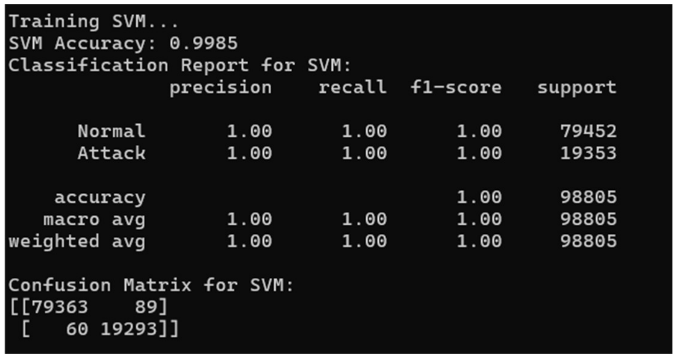


Figure 7 Decision Tree Confusion Matrix

3. Random Forest

- Accuracy: **99.97%**
- Confusion Matrix: $\begin{bmatrix} 79436 & 16 \\ 9 & 19334 \end{bmatrix}$
- Similar to the decision tree, it demonstrates high accuracy with minimal errors.



```
Training SVM...
SVM Accuracy: 0.9985
Classification Report for SVM:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     79452
      Attack       1.00      1.00      1.00     19353

    accuracy                           1.00     98805
   macro avg       1.00      1.00      1.00     98805
weighted avg       1.00      1.00      1.00     98805

Confusion Matrix for SVM:
[[79363    89]
 [   60 19293]]
```

Figure 8 Random Forest Output

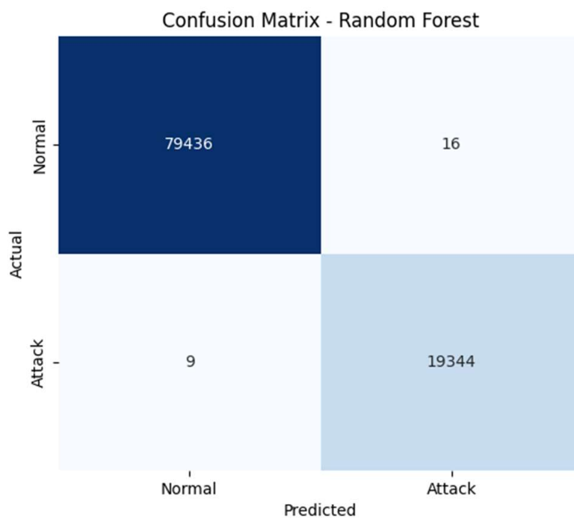


Figure 9 Random Forest Confusion Matrix

4. Support Vector Machine (SVM)

- Accuracy: **99.85%**
- Confusion Matrix: $\begin{bmatrix} 79363 & 89 \\ 60 & 19293 \end{bmatrix}$
- The SVM also performs well but has slightly more false positives compared to the decision tree and random forest models.

```
Training SVM...
SVM Accuracy: 0.9985
Classification Report for SVM:
              precision    recall  f1-score   support

      Normal       1.00      1.00      1.00     79452
      Attack       1.00      1.00      1.00     19353

    accuracy                           1.00     98805
   macro avg       1.00      1.00      1.00     98805
weighted avg       1.00      1.00      1.00     98805

Confusion Matrix for SVM:
[[79363    89]
 [   60 19293]]
```

Figure 10 SVM Output

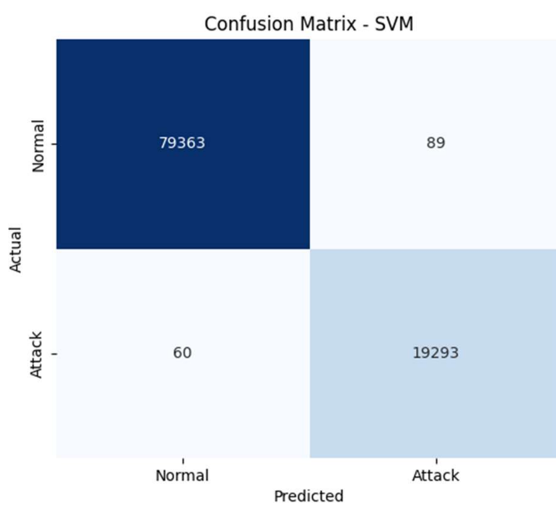


Figure 11 SVM Confusion Matrix

7. Visualizations

1. Model Accuracy Comparison

A bar plot visualizes the accuracy of each model:

```
plt.figure(figsize=(10, 6))

sns.barplot(x=model_names, y=accuracy_scores, palette='viridis')

plt.title('Model Accuracy Comparison')

plt.ylabel('Accuracy')

plt.show()
```

2. Confusion Matrix Visualization

Confusion matrices for each model are plotted using heatmaps to provide a clear view of prediction outcomes:

```
def plot_confusion_matrix(conf_matrix, model_name):
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
        xticklabels=['Normal', 'Attack'], yticklabels=['Normal', 'Attack'])
plt.title(f'Confusion Matrix - {model_name}')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

8. Conclusion

The analysis says that all models achieved high accuracy rates with the KDD Cup dataset for intrusion detection tasks. The decision tree and random forest classifiers performed exceptionally well with near-perfect accuracy rates. In contrast, the Naive Bayes classifier showed lower performance due to its assumptions about feature independence which may not hold true in this context.Overall, this code serves as a robust framework for developing an intrusion detection system using machine learning techniques effectively tailored to handle various network traffic scenarios.

References

Geeksforgeeks → intrusion-detection-system-using-machine-learning-algorithms