Network Intrusion Detection

1. Project Overview

This projects implements a simple yet effective machine learning model for detecting potential malicious intrusions using the KDD99 Cup dataset.

I've applied Logistic Regression on 15 different network traffic parameters which is then used to classify whether a record is 'normal' or an 'attack'.

2. Dataset

Official site: KDD Cup 1999 Data

Training data (CSV): https://drive.google.com/file/d/1Bi5MJwwMJUOfPojXo174xBRV9NeGA_Qp/view?usp=sharing

Official test set: https://drive.google.com/file/d/1QRinffdxl-aolgiokm6-XJElKGJB1ZSH/view?usp=sharing

PS: If you're opening this in Excel, don't save the file since the file exceeds Excels row limit and that would lead to deletion of around 3 million rows lol.

3. Code

```
#Author: Samuel Thomas
#Very simple network intrusion detection prediction using LogisticRegression

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv("kdd_full_numeric.csv")

features = [
    'duration', 'src_bytes', 'dst_bytes', 'count', 'srv_count',
    'serror_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate',
```

```
    'same_srv_rate', 'diff_srv_rate', 'dst_host_count',
    'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'is_guest_login'
]

X = df[features].astype(float).values
y = (df['label'] != 'normal').astype(int).values

print("X shape:", X.shape)
print("y shape:", y.shape)
print("attack ratio:", y.mean())

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state = 42, stratify=y)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report

logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train, y_train)

y_pred = logreg.predict(X_test)

acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print("Accuracy:", acc)
print("Confusion matrix:\n", cm)
print("\nClassification report:\n", classification_report(y_test,
    y_pred))


df_corrected = pd.read_csv("corrected_full_42cols.csv")

X_corr = df_corrected[features].astype(float).values
y_corr = (df_corrected["label"] != "normal").astype(int).values

X_corr_scaled = sc.transform(X_corr)
y_corr_pred = logreg.predict(X_corr_scaled)
```

```
59.
60.acc_corr = accuracy_score(y_corr, y_corr_pred)
61.cm_corr = confusion_matrix(y_corr, y_corr_pred)
62.
63.print("\n=== OFFICIAL TEST SET ===")
64.print("Accuracy:", acc_corr)
65.print("Confusion matrix:\n", cm_corr)
66.print("\nClassification report:\n", classification_report(y_corr,
   y_corr_pred))
```

4. Evaluation Results

Dataset	**Accuracy**	**ROC-AUC**
Test set from the Training set(20%)	0.9939	0.9959
Official test set	0.811	0.926

5. Limitations

1. Data age : The dataset is now more than 20 years old and modern cyber attacks like APTs and DDoS variants differ substantially from the attacks present in the dataset.

2. Different attack types were merged into a single ‘attack’ class. Model cannot distinguish between different attacks.

3. As seen in the table above, the accuracy drops from 99% to 81%. The reason why the accuracy is only around 80% is because the official test set contains new threats that weren’t in the training set to see how the model would perform when faced with unknown threats.

6. Final thoughts

This model was something I thought of making a while back since I wanted to do something related to my field(Networks). This is just a starting point, I’ll definitely be

back with a better model with much better accuracy and maybe even the ability to track threats live. It also shows the potential serviceability of ML in this field.

This model proves that even a simple , well-trained model can contribute to network defence and provides a good foundation to build upon.