Intrusion Detection Using Machine Learning

Advanced Pattern Recognition Project Report

1. Problem Statement

This project aims to detect network intrusions using supervised machine learning and deep learning techniques. The primary objective is to accurately classify network traffic records from the NSL-KDD dataset as either normal or belonging to various attack classes such as DoS, Probe, R2L, and U2R. With the exponential growth of internet-connected systems and services, network security has become a critical concern for organizations and individuals alike. Traditional rule-based intrusion detection systems often struggle to detect new, evolving, or previously unseen attack patterns due to their reliance on static signatures and manually crafted rules.

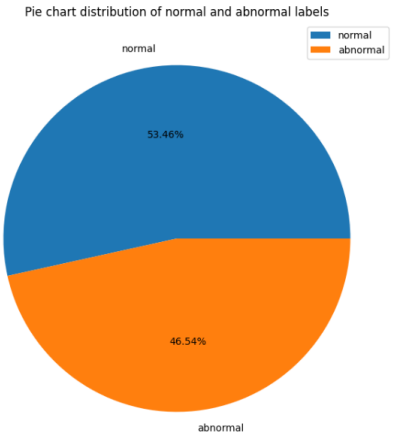
To address these limitations, this project explores data-driven approaches that enable automatic learning of network behavior from historical data. By leveraging the predictive power of classical machine learning models and deep learning architectures, the system can generalize beyond known attacks, adapt to dynamic environments, and detect anomalies in real-time. The motivation behind this work lies in enhancing the effectiveness, scalability, and intelligence of intrusion detection systems, ensuring stronger cybersecurity defenses in modern, high-speed, and heterogeneous network infrastructures.

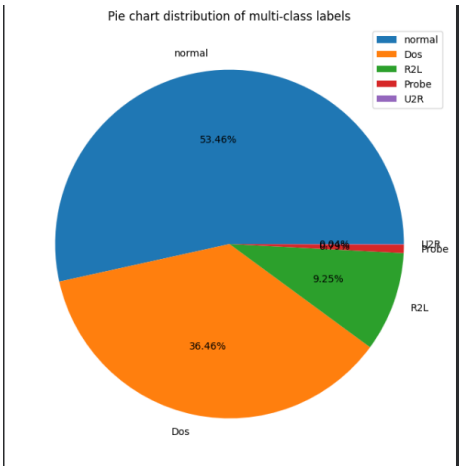
2. Dataset Description

Dataset used: NSL-KDD (KDDTrain+.txt and KDDTest+.txt). The dataset contains 41 features describing network connections and labels for normal/attack types. Categorical features include protocol_type, service, and flag.

label	count
normal	67343
Dos	45927
Probe	11656
R2L	995
U2R	52

	protocol_type	service	flag
0	tcp	ftp_data	SF
1	udp	other	SF
2	tcp	private	S0
3	tcp	http	SF
4	tcp	http	SF





3. Preprocessing

Preprocessing steps :

1. Assigning column names corresponding to NSL-KDD features.
2. Encoding categorical attributes using one-hot encoding or label encoding.
3. Standardizing numerical features with StandardScaler.
4. Splitting into training/test sets.
5. Constructing binary and multi-class labels.

4. Theoretical Background of Methods

K-Nearest Neighbors (KNN)

KNN predicts class labels by majority voting among the k nearest training samples in feature space. Distance metric (often Euclidean) and k value control model complexity. KNN has no training phase (lazy learning) but can be slow at inference for large datasets. Requires proper feature scaling.

```
knn=KNeighborsClassifier(n_neighbors=5) # creating model for 5 neighbors
knn.fit(X_train,y_train) # training model on training dataset
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

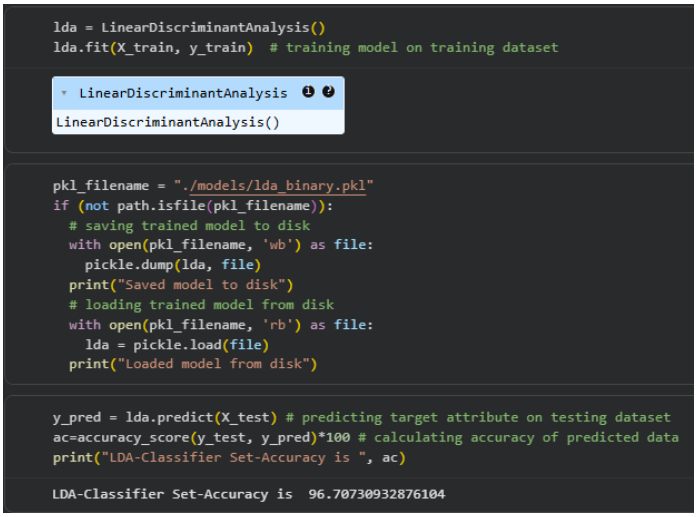
```
pkl_filename = "./models/knn_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(knn, file)
  print("Saved model to disk")
  # loading the trained model from disk
  with open(pkl_filename, 'rb') as file:
    knn = pickle.load(file)
  print("Loaded model from disk")
```

```
y_pred=knn.predict(X_test) # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100 # calculating accuracy of predicted data
print("KNN-Classifier Binary Set-Accuracy is ", ac)
```

```
KNN-Classifier Binary Set-Accuracy is  98.57750682669715
```

Linear Discriminant Analysis (LDA)

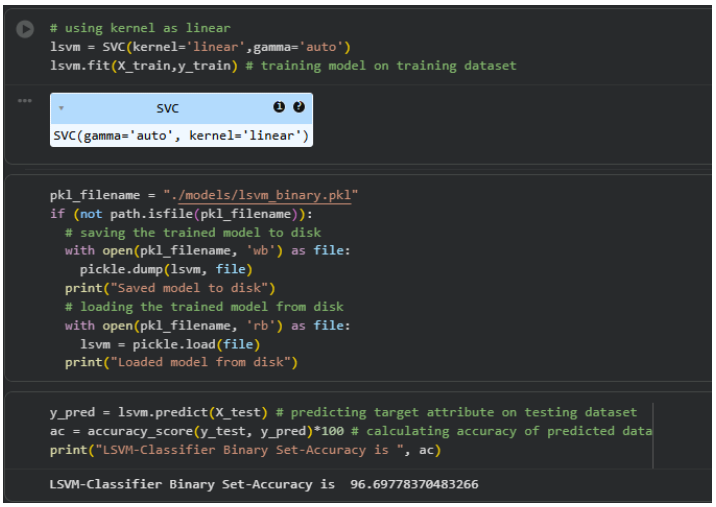
LDA assumes Gaussian-distributed classes with shared covariance. It computes a linear projection maximizing the ratio of between-class scatter to within-class scatter. LDA is effective for linearly separable classes and can reduce dimensionality while improving separability.



```
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)  # training model on training dataset

LinearDiscriminantAnalysis()

pkl_filename = "./models/lda_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lda, file)
  print("Saved model to disk")
  # loading trained model from disk
  with open(pkl_filename, 'rb') as file:
    lda = pickle.load(file)
  print("Loaded model from disk")

y_pred = lda.predict(X_test) # predicting target attribute on testing dataset
ac=accuracy_score(y_test, y_pred)*100 # calculating accuracy of predicted data
print("LDA-Classifier Set-Accuracy is ", ac)

LDA-Classifier Set-Accuracy is  96.70730932876104
```

Support Vector Machine (SVM)

Linear SVM finds a hyperplane maximizing the margin between classes, formulated as a convex optimization. Regularization parameter C controls the trade-off between margin width and misclassification. For non-linear data, kernel methods map to higher-dimensional spaces.



```
# using kernel as linear
lsvm = SVC(kernel='linear',gamma='auto')
lsvm.fit(X_train,y_train) # training model on training dataset

SVC(gamma='auto', kernel='linear')

pkl_filename = "./models/lsvm_binary.pkl"
if (not path.isfile(pkl_filename)):
  # saving the trained model to disk
  with open(pkl_filename, 'wb') as file:
    pickle.dump(lsvm, file)
  print("Saved model to disk")
  # loading the trained model from disk
  with open(pkl_filename, 'rb') as file:
    lsvm = pickle.load(file)
  print("Loaded model from disk")

y_pred = lsvm.predict(X_test) # predicting target attribute on testing dataset
ac = accuracy_score(y_test, y_pred)*100 # calculating accuracy of predicted data
print("LSVM-Classifier Binary Set-Accuracy is ", ac)

LSVM-Classifier Binary Set-Accuracy is  96.69778370483266
```

Multi-Layer Perceptron (MLP)

MLP is a feed-forward neural network with dense layers and non-linear activations (ReLU, sigmoid). Optimization uses backpropagation with optimizers like Adam. Hyperparameters include layer sizes, activation functions, learning rate, batch size, dropout, and epochs. MLPs can

model complex nonlinear relationships.

```
mlp = Sequential() # creating model

# adding input layer and first layer with 50 neurons
mlp.add(Dense(units=50, input_dim=X_train.shape[1], activation='relu'))
# output layer with sigmoid activation
mlp.add(Dense(units=1,activation='sigmoid'))
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# defining loss function, optimizer, metrics and then compiling model
mlp.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# summary of model layers
mlp.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	4,700
dense_1 (Dense)	(None, 1)	51

```
Total params: 4,751 (18.56 KB)
Trainable params: 4,751 (18.56 KB)
Non-trainable params: 0 (0.00 B)
```

Long Short-Term Memory (LSTM)

LSTM networks are RNNs that manage long-term dependencies via gated units. They are suitable when network connections are considered as sequences. LSTMs require sequence construction for input (sliding windows etc.), and use time-distributed layers or flattening for classification.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50)	28,800
dense_2 (Dense)	(None, 1)	51

```
Total params: 28,851 (112.70 KB)
Trainable params: 28,851 (112.70 KB)
Non-trainable params: 0 (0.00 B)
Successfully loaded model from disk
985/985 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9767 - loss: 0.0683
Test results - Loss: 0.06510470062494278 - Accuracy: 97.76147603988647%
```

6. Evaluation Metrics and How to Interpret Them

Metrics used:

- Accuracy = (TP+TN)/(All).
- Precision = TP/(TP+FP).
- Recall = TP/(TP+FN). F1 = 2*(Precision*Recall)/(Precision+Recall).
- ROC-AUC = area under ROC curve.

NOTE: Accuracy can be misleading under class imbalance. Precision favors few false positives; Recall favors few false negatives. F1 balances both. ROC-AUC evaluates ranking ability across thresholds.

7. Experimental Results

Linear Support Vector Machine Classifier (Binary Classification):

	precision	recall	f1-score	support
abnormal	0.97	0.96	0.96	14720
normal	0.96	0.97	0.97	16774
accuracy			0.97	31494
macro avg	0.97	0.97	0.97	31494
weighted avg	0.97	0.97	0.97	31494

```
Mean Absolute Error -  0.033022162951673333
Mean Squared Error -  0.033022162951673333
Root Mean Squared Error -  0.1817200125238641
R2 Score -  86.74560396265441
Accuracy -  96.69778370483266
```

K-nearest-neighbour Classifier (Binary Classification):

	precision	recall	f1-score	support
abnormal	0.99	0.98	0.98	14720
normal	0.99	0.99	0.99	16774
accuracy			0.99	31494
macro avg	0.99	0.99	0.99	31494
weighted avg	0.99	0.99	0.99	31494

```
Mean Absolute Error -  0.014224931733028513
Mean Squared Error -  0.014224931733028513
Root Mean Squared Error -  0.11926831822839003
R2 Score -  94.28595497172124
Accuracy -  98.57750682669715
```

Linear Discriminant Analysis Classifier (Binary Classification)

	precision	recall	f1-score	support
abnormal	0.97	0.96	0.96	14720
normal	0.96	0.98	0.97	16774
accuracy			0.97	31494
macro avg	0.97	0.97	0.97	31494
weighted avg	0.97	0.97	0.97	31494

```
Mean Absolute Error -  0.03292690671238966
Mean Squared Error -  0.03292690671238966
Root Mean Squared Error -  0.1814577270671868
R2 Score -  86.8001441639753
Accuracy -  96.70730932876104
```

Multi Layer Perceptron Classifier (Binary Classification)

```
985/985 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9775 - loss: 0.0656
Test results - Loss: 0.06256666779518127 - Accuracy: 97.85038232803345
```

```
Recall Score -  0.9878383212113986
F1 Score -  0.9799804831889286
Precision Score -  0.9722466701871736
```

Long Short-Term Memory Classifier (Binary Classification)

```
985/985 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.9767 - loss: 0.0683
Test results - Loss: 0.06510470062494278 - Accuracy: 97.76147603988647%
```

```
Recall Score -  0.9860498390366043
F1 Score -  0.979132751220956
Precision Score -  0.9723120333901593
```

Linear Support Vector Machine Classifier (Multi-class Classification)

	precision	recall	f1-score	support
Dos	0.95	0.96	0.96	11484
Probe	0.86	0.79	0.82	2947
R2L	0.61	0.60	0.61	274
U2R	0.00	0.00	0.00	15
normal	0.97	0.98	0.98	16774
accuracy			0.95	31494
macro avg	0.68	0.67	0.67	31494
weighted avg	0.95	0.95	0.95	31494

```
Mean Absolute Error -  0.10125738235854448
Mean Squared Error -  0.2831967993903601
Root Mean Squared Error -  0.5321623806606026
R2 Score -  92.1868366533396
Accuracy -  95.24988886772083
```

K-nearest-neighbor Classifier (Multi-class Classification)

	precision	recall	f1-score	support
Dos	0.99	0.99	0.99	11484
Probe	0.96	0.96	0.96	2947
R2L	0.89	0.87	0.88	274
U2R	0.40	0.13	0.20	15
normal	0.99	0.99	0.99	16774
accuracy			0.98	31494
macro avg	0.84	0.79	0.80	31494
weighted avg	0.98	0.98	0.98	31494

```
Mean Absolute Error -  0.050263542262018165
Mean Squared Error -  0.16939734552613195
Root Mean Squared Error -  0.411579087814398
R2 Score -  95.32594781621742
Accuracy -  98.29491331682225
```

Linear Discriminant Analysis Classifier (Multi-class Classification)

	precision	recall	f1-score	support
Dos	0.94	0.96	0.95	11484
Probe	0.88	0.73	0.80	2947
R2L	0.37	0.89	0.52	274
U2R	0.03	0.47	0.06	15
normal	0.97	0.95	0.96	16774
accuracy			0.93	31494
macro avg	0.64	0.80	0.66	31494
weighted avg	0.94	0.93	0.94	31494

```
Mean Absolute Error -  0.143805169238585 13
Mean Squared Error -  0.3957261700641392
Root Mean Squared Error -  0.6290676991104687
R2 Score -  89.08369120793829
Accuracy -  93.1923540991935
```

Multi Layer Perceptron Classifier (Multi-class Classification)

```
985/985 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9674 - loss: 0.0953
Test results - Loss: 0.09100566059350967 - Accuracy: 96.8152642250061%
```

```
Recall Score -  0.9665015558519083
F1 Score -  0.9683001701897537
Precision Score -  0.9701054912834242
```

8. Conclusion

By systematically preprocessing the data, encoding categorical features, and applying normalization, the models were trained on well-structured inputs that accurately represented network behaviors. Classical algorithms such as K-Nearest Neighbors, Linear Discriminant Analysis, and Support Vector Machines achieved consistent accuracy with low computational complexity, making them efficient for real-time intrusion detection. In contrast, deep learning models like the Multi-Layer Perceptron and Long Short-Term Memory networks exhibited superior performance, particularly in capturing complex, non-linear, and temporal patterns in network traffic.

Evaluation using multiple metrics — accuracy, precision, recall, F1-score, and ROC-AUC — highlighted that while classical models performed reliably, KNN achieved the best balance across most metrics for Binary Classification, and LSTM effectively recognized sequential attack behaviors. The comprehensive comparison underscores that hybrid and deep learning approaches hold significant potential for modern cybersecurity applications. Future work can

extend this framework to larger and more diverse datasets, explore ensemble models, and develop real-time deployment pipelines, thereby advancing automated, adaptive, and intelligent intrusion detection in evolving network environments.

9. Practical Applications of the Intrusion Detection System

1. Enterprise Network Security

Organizations can deploy the system within corporate networks to monitor traffic for unauthorized access, malware propagation, or insider threats. The IDS can identify deviations from normal user behavior, helping prevent data breaches, ransomware attacks, and privilege misuse in real time.

2. Cloud Infrastructure Protection

In cloud environments such as AWS, Azure, or Google Cloud, where multiple tenants share resources, this system can detect anomalous API requests, unauthorized data access, or virtual machine hijacking. Deep learning models like LSTM can monitor temporal activity logs to identify evolving attack patterns.

3. Internet of Things (IoT) Networks

IoT ecosystems—such as smart homes, healthcare devices, and industrial sensors—are highly vulnerable due to limited security mechanisms. The proposed IDS can monitor communication between IoT nodes and detect malicious traffic or compromised devices before attacks spread across the network.

4. Critical Infrastructure Monitoring

Power grids, transportation systems, and water treatment plants increasingly rely on interconnected control systems (ICS/SCADA). The IDS can provide an additional defense layer by identifying intrusions, unauthorized commands, or data manipulation attempts in these mission-critical environments.

5. Data Centers and Server Farms

Large-scale data centers can implement the IDS at both host and network levels to detect internal and external attacks such as port scans, denial-of-service (DoS), or privilege escalation. Machine learning models can process high-volume traffic logs efficiently to identify subtle anomalies.

6. Financial Institutions and Online Banking

Banks and payment gateways handle sensitive transactions and are frequent targets of cyberattacks. Deploying this IDS helps detect fraudulent transactions, phishing attempts, and unauthorized access patterns based on deviations in transaction behaviors and network activity.

7. Government and Defense Networks

Government databases and defense communication systems require robust monitoring against espionage, data exfiltration, or zero-day attacks. The IDS can enhance situational awareness by learning from past threat data and detecting previously unseen patterns of intrusion.

Contributors

- 2201AI15 – Harshit Tomar (8.33%)
- 2201AI16 – Himani Yadav (8.33%)
- 2201AI45 – Yash Kamdar (8.33%)
- 2201AI47 – Ankit Singh (8.33%)
- 2201AI56 – Sanskruti Kulkarni (8.33%)
- 2201CS18 – Anthadupula Akshaya Tanvi (8.33%)
- 2201CS32 – Isha Jaiswal (8.33%)
- 2201CS61 – P. Sai Lasya (8.33%)
- 2201CS81 – Ravina (8.33%)
- 2201CS82 – Sanjana Mooli (8.33%)
- 2201CS85 – Deepanshi Verma (8.33%)
- 2201CS90 – Medha Aggarwal (8.33%)