Network Intrusion Detection Report

Luv Valecha
B23CS1093

Ritik Nagar
B23EE1061

Dheeraj Kumar
B23CS1016

Shiv Jee Yadav
B23EE1095

Pratyush Chauhan
B23CM1030

Dhruv Sharma
B23EE1086

Abstract

This report outlines a structured approach to enhancing network intrusion detection using a combination of classical machine learning algorithms and strategic feature selection. Our methodology is to identify the most relevant features from the dataset using Optuna which helped us to reduce dimensionality and also improve model performance. We evaluate the effectiveness of multiple classifiers, including K-Nearest Neighbors (KNN), Logistic Regression, Support Vector Machines (SVM) with both linear and non-linear kernels, alongwith Naive Bayes and Descision tree. Each model is rigorously assessed using standard performance metrics to ensure reliability and robustness . We had implemented grid serach with cross-validation(GridSearchCV) to find best parameter for all model which significantly improved performance of the model resulted in better results. Each model is further accessed on multiple parameters such as f1 score precision recall to ensure reliability abd robustness. Through comprehensive experimentation and by trying different approaches, we identified the optimal combination of feature selection and classifier that give accurate results. Our findings demonstrate that by doing optimal feature engineering and diverse model exploration along with constant experimentation we can develop a effective network intrusion detection system that is applicable to real-world cybersecurity scenarios.. **Keywords:** predictions,accuracy,classifier

Contents

1 Introduction

This project focuses on classifying network traffic using machine learning algorithms such as Support Vector Machine, K-Nearest Neighbour, Decision Tree, and Naive Bayes. The goal is to analyze packet features and evaluate the performance of different models for accurate traffic classification. The project also includes a basic frontend and live packet capturing system to demonstrate real-time analysis.

1.1 Problem Statement

This project aims to develop a machine learning-based solution for efficient and reliable network traffic classification.

1.2 Dataset

1.2.1 Source

Network Intrusion Dataset
We used the "Network Intrusion Dataset" available on Kaggle. It contains labeled network traffic data, including features extracted from packet flows, to help train and evaluate intrusion detection models.

1.2.2 Exploratory Data Analysis (EDA)

Our dataset consists of [25000 rows $\times$ 42 columns], representing a substantial volume of data for training and evaluation. Before proceeding to model training, we performed extensive exploratory data analysis to understand the structure, distribution, and relationships within the dataset. This helped identify important patterns, detect outliers, and highlight potential feature importance.

duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent
0	tcp	ftp_data	SF	491	0	0	0	0
0	udp	other	SF	146	0	0	0	0
0	tcp	private	S0	0	0	0	0	0
0	tcp	http	SF	232	8153	0	0	0
0	tcp	http	SF	199	420	0	0	0

Table 1: Sample Network Traffic Data

Class distribution of our dataset look like this:-

```
Distribution of classes:
class
normal     13449
anomaly    11743
Class Balance: 1.15:1
```

The number distribution of numerical and categorical features:-

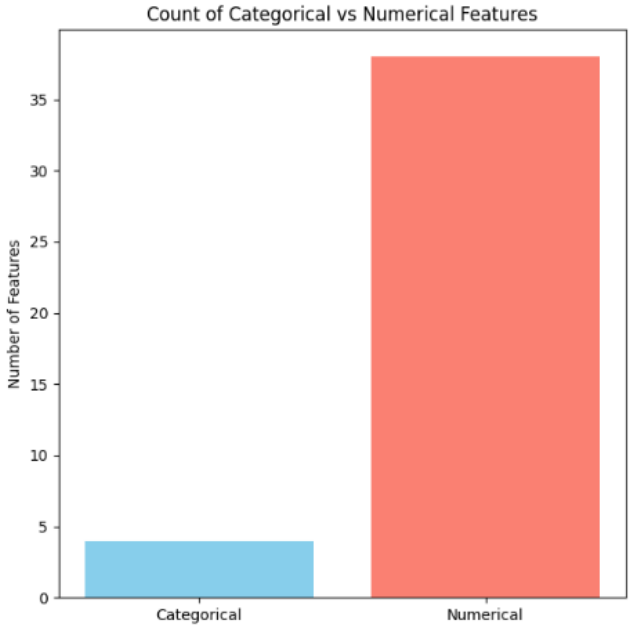

Figure 1: categorical features

We then preprocessed dataset.This is how co-relation matrix of processed dataset look like:-

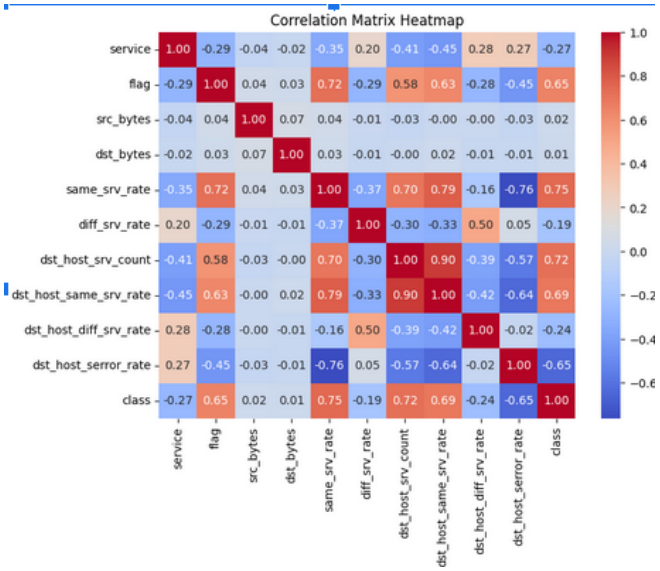

Figure 2: Co-relation matrix of processed dataset

.

1.3 Scope of the Project

This project lays the foundation for intelligent network traffic monitoring systems using machine learning. In the future, it can be extended to support deep learning models, real-time intrusion detection in enterprise networks, and automatic threat mitigation. Integration with security dashboards and deployment in large-scale environments can enhance network defense mechanisms.

2 Methodology

For network intrusion detection, we employed a machine learning-based approach. The raw data was first preprocessed through multiple steps (outlined below), followed by training and evaluation using a range of classification models.

2.1 Data Preprocessing

The preprocessing involves:

1. **Categorical Encoding:** Categorical columns such as `protocol_type`, `service`, `flag`, and `class` are encoded using `LabelEncoder`, and the encoders are saved for inference.
2. **Train-Test Split:** An 80-20 split is performed to create training and testing datasets.
3. **Missing Value Removal:** Columns with over 50% missing values are dropped.
4. **Low-Variance Removal:** Columns with a single unique value are removed.
5. **Sparse Feature Removal:** Columns containing more than 90% zero values are eliminated.
6. **Feature Selection:** To reduce dimensionality and improve model performance, we applied feature selection based on mutual information. Specifically, we used the `SelectKBest` method from the `sklearn.feature_selection` module, combined with `mutual_info_classif` as the scoring function. A total of $k = 10$ top features were selected, representing the most informative variables with respect to the target class. This selection process was implemented in a custom utility function `GetKbestfeatures`, which fits the selector on the training data and transforms both the training and testing datasets to retain only the top-k features. This step helps eliminate redundant and irrelevant features, ensuring that the models are trained on the most relevant subset of data.
7. **Train-Test Split:** An 80-20 split is performed to create training and testing datasets.
8. **Saving Data:** The processed datasets are saved as `train_data.csv` and `test_data.csv`.

2.2 Models Used

We experimented and test with a wide range of supervised learning algorithms, that are listed below

- Logistic Regression, Linear SVM, Non-linear SVM
- Decision Tree, Random Forest, K-Nearest Neighbors (KNN)
- Bernoulli Naive Bayes, Gradient Boosting, AdaBoost, XGBoost
- A Voting Classifier that aggregates results from multiple models

Each model is then trained on the preprocessed dataset.

2.3 Evaluation

Models are evaluated based on standard classification metrics which include accuracy, precision, recall, F1 score, and confusion matrix. These evaluations are tested on both training and test datasets to assess generalization and performance consistency.

3 Experiments and Results

We trained and evaluated machine learning models: Support Vector Machine, K-Nearest Neighbour, Decision Tree, Naive Bayes and Logistic Regression. The dataset was split into training and testing sets, and standard metrics like accuracy, precision, recall, and F1-score were used for evaluation. Results from each model are compared to identify the most effective approach for network traffic classification.

3.1 Experiments

Each model was trained using various hyperparameter configurations to optimize performance. Grid search and manual tuning were applied to parameters such as kernel type in SVM, number of neighbors in KNN, max depth in Decision Tree, priors in Naive Bayes, and regularization strength in Logistic Regression. Cross-validation was used to ensure reliable evaluation and avoid overfitting.

3.1.1 Support Vector Machine

we had applied both lienar and non-linear svm model and these are our finding:-

LinearSVM Through Linear SVM, we got the best accuracy of 89.86%. We tried `GridSearchCV` to find the best parameter and ultimately settled for a lambda parameter of 0.01, which gave the best outcome. Here are the parameters we tried:

Parameter	**Values tried**		
learning rate	0.001	0.01	0.1
lambda parameter	0.001	0.01	0.1
iterations	500	1000	2000

Table 2: Grid search parameters used for Linear SVM.

And here are the best parameter with their accuracy:

```
learning_rate: 0.01
lambda_param: 0.1
n_iterations: 500
Accuracy: 0.89859099902758484
```

3.1.2 Non-LinearSVM

Through Non-Linear SVM, we got the best accuracy of 99.05%. We tried `GridSearchCV` to find the best parameter and ultimately settled for a parameter value Gamma: 0.001,C: 100.0, which gave the best outcome. The parameters tried are as follows:

gamma values	0.001	0.01	0.1	1.0
C values	0.1	1.0	10.0	100.0

Table 3: Grid search parameters used for Linear SVM.

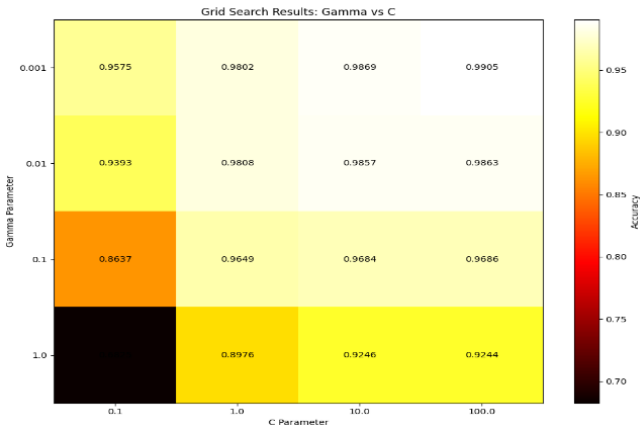

Figure 3: Grid search results

The best parameters with their accuracy:

```
Best parameters:
Gamma: 0.001
C: 100.0
Accuracy: 0.9905
```

3.1.3 Kth Nearest Neigbour

Through KNN, we got the best accuracy of 99.36%.We tried different values of k and identified the best parameter. The k vs accuracy curve is as follows:

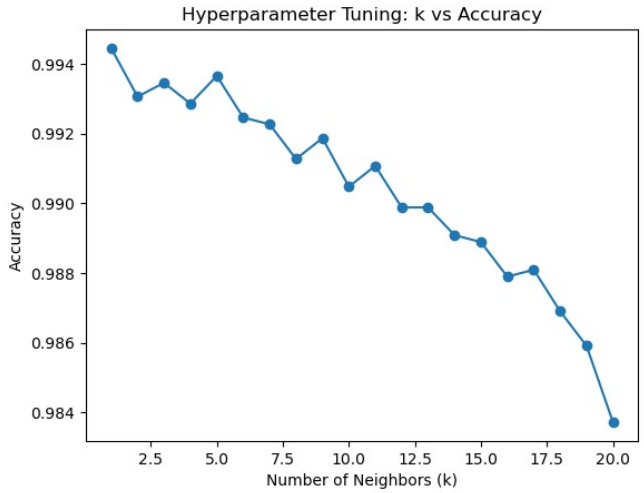

Figure 4: K vs Accuracy

Although k = 1 gives the highest accuracy, it is more sensitive to noise and can lead to overfitting. Hence, we selected k = 5 for better generalization and model stability.

The best parameters with their accuracy:

```
Best parameters:
k: 5
Accuracy: 0.9936
```

3.1.4 Decision Tree

Through Decision Tree, we got the best accuracy of 99.44%. We tried `GridSearchCV` to find the best parameters. The parameters tried are as follows:

max_depth	10	15	18	20
min_samples_leaf	1	4	8	10
min_information_gain	0.01	0.02	0.05	0.1

Table 4: Grid search parameters used for Decision Tree

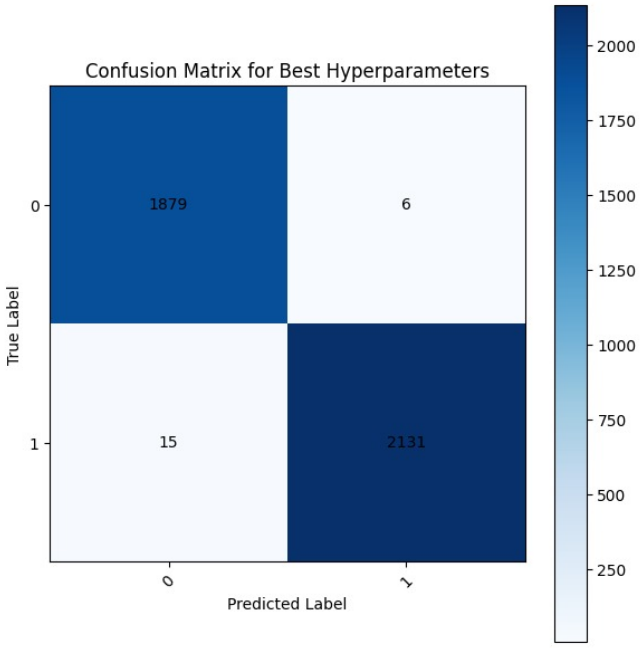

Figure 5: Grid search results

The best parameters with their accuracy:

```
max_depth: 18
min_samples_leaf: 1
min_information_gain: 0.01
Accuracy: 0.9944
```

3.1.5 Bernoulli Naive Bayes

Through Bernoulli Naive Bayes, we achieved the best accuracy of **92.72%**. We performed a manual grid search to tune the smoothing parameter α for optimal performance. The table below summarizes the parameters we tried:

Alpha	0.001	0.01	0.1	1.0	10.0

Table 5: Grid search parameters used for Bernoulli Naive Bayes.

The best parameter and corresponding accuracy are summarized below:

```
Best parameters:
alpha: 0.001
Accuracy: 0.9272
```

3.1.6 Random Forest

Through Random Forest, we got the best accuracy of 99.66%. We tried `GridSearchCV` to find the best parameter.The parameters tried are as follows:

n_learners	100	250	500
max_depth	5	10	15
min_samples_leaf	1	5	10
min_information_gain	0.01	0.05	0.1

Table 6: Grid search parameters used for Random Forest

The best parameters with their accuracy:

```
Best parameters:
n_learners: 100
max_depth: 15
min_samples_leaf: 1
min_information_gain: 0.01
Accuracy: 0.9966
```

3.1.7 Logistic Regression

Through Logistic Regression, we got the best accuracy of 88.75%. We used GridSearchCV to find the best parameters and ultimately settled for parameter values C: 10.0, max_iter: 100, which gave the best outcome. The parameters tried are as follows:

C values	0.01	0.1	1	10 100
max_iter values	100	300	500	1000

Table 7: Grid search parameters used for Logistic Regression

The best parameters with their accuracy:

```
Best parameters:
C: 10.0
max_iter: 100
Accuracy: 0.8875
```

3.2 Results

Model	Accuracy	Precision	Recall	F1 Score
KNN	99.36%	99.50%	99.50%	99.50%
Random Forest	99.50%	99.50%	99.50%	99.50%
AdaBoost	96.59%	96.59%	96.59%	96.59%
Decision Tree	99.44%	99.44%	99.44%	99.44%
Linear SVM	89.86%	85.29%	97.72%	91.13%
Non-Linear SVM	97.14%	95.24%	99.59%	97.39%
XGBoost	99.66%	99.66%	99.66%	99.66%
BernoulliNB	92.72%	92.72%	92.72%	92.71%
Logistic Regression	88.71%	88.72%	88.71%	88.70%
Gradient Boosting	99.31%	99.31%	99.31%	99.31%
ANN	95.87%	95.87%	95.87%	95.87%

Table 8: Performance of different models

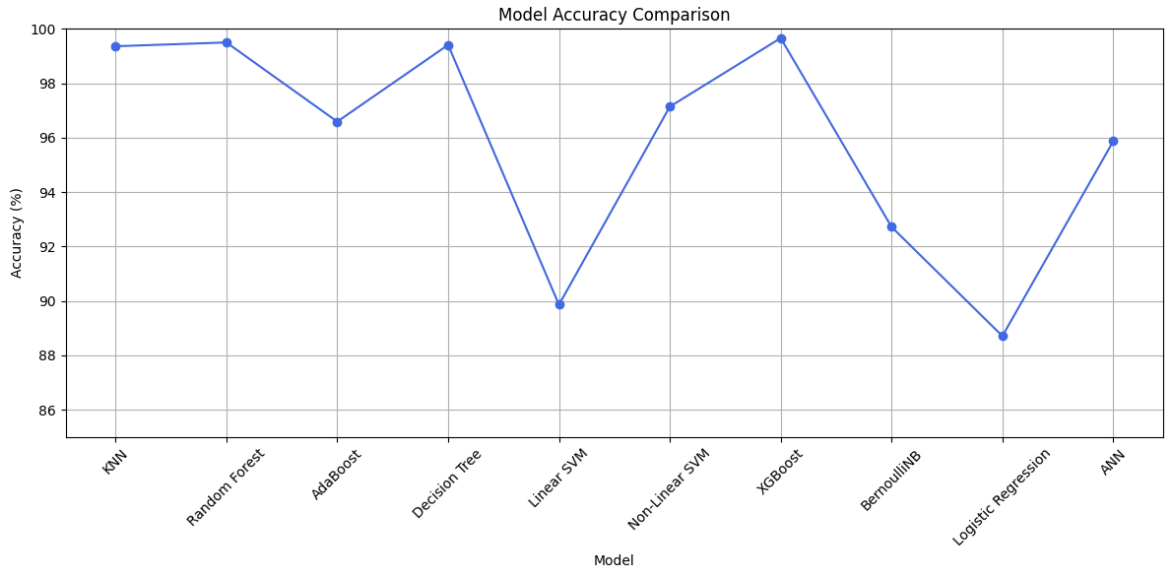

Figure 6: Grid search results

4 Additional Work

4.1 Deployment on Google Cloud

The backend of our machine learning project was developed using the **Flask** framework to create a lightweight and efficient RESTful API. This API serves as the interface between the machine learning model and the client-side application, handling requests, processing data, and returning predictions. For deployment, the Flask API was hosted on **Google Cloud Platform (GCP)**, leveraging its scalability, reliability, and ease of integration with other cloud services. This deployment setup ensures that our solution is accessible, secure, and capable of handling real-time inference requests effectively.

4.2 Frontend

The frontend of our network intrusion detection system is developed using **React.js**, offering a responsive and intuitive user interface. It allows users to interact seamlessly with the system, whether uploading input data,

triggering predictions, or visualizing model outputs. The frontend communicates with the backend via **RESTful APIs**, ensuring efficient and real-time data exchange with the ensemble machine learning model.

Deployed on **Vercel**, the frontend provides fast, scalable access and delivers a smooth user experience across devices.

The web application is divided into the following key modules:

- **Single Packet Prediction:** Enables manual input of individual packet features such as service type, flag, source and destination bytes, and various traffic rates.
- **CSV Upload:** Facilitates bulk predictions by allowing users to upload a CSV file containing multiple packet entries.
- **Auto Capture:** Acts as a live packet analyzer, where users can enter a server URL to capture and analyze real-time traffic.

In addition to these modules, the application includes:

- **History Section:** Maintains a log of all previous predictions, storing the data in the browser's local storage.
- **About Section:** Provides a comprehensive overview of the project, its objectives, and its functionality.

The interface is designed to be clean, organized, and user-friendly, promoting accessibility and ease of use.

4.3 Live Packet Capturing

To support real-time network analysis, the application integrates a **Live Packet Capturing** feature within the frontend. This tool allows users to detect and classify real-time network packets incoming on their local server. To use this feature, users must set up the backend server on their desktop locally and run it.

Upon initiating capture via the *"Start Capture"* button, the system begins monitoring and analyzing live network traffic in real-time and predicting it.

The interface also provides convenient access to essential setup files, including `packetcapture.py` and `requirements.txt`, along with detailed step-by-step installation instructions for Windows, macOS, and Linux platforms. This ensures cross-platform compatibility and simplifies the configuration process, enabling users to quickly establish a capture environment for live analysis and intrusion detection.

5 Summary

In this project, we developed a comprehensive machine learning-based Network Intrusion Detection System (NIDS). Our approach involved systematic data preprocessing, including handling missing values, encoding categorical features, and applying feature selection.

We implemented and evaluated a wide range of classification models—ranging from traditional algorithms like Logistic Regression, Naive Bayes, Decision Tree, and K-Nearest Neighbors, to advanced ensemble methods like Random Forest, AdaBoost, XGBoost, and Voting Classifiers. Hyperparameter optimization was performed using GridSearchCV and cross-validation, ensuring optimal model performance and robustness.

Among all the models tested, XGBoost and Random Forest achieved the highest accuracy (99.66%), followed closely by KNN and Decision Tree, demonstrating the effectiveness of ensemble and distance-based approaches in detecting intrusions. Each model was assessed using accuracy, precision, recall, and F1-score to ensure a well-rounded evaluation.

Beyond model development, we also created a user-friendly frontend interface using React.js. The web application supports single packet prediction, CSV uploads for bulk classification, and a real-time live packet capturing module. The frontend is deployed via Vercel and integrates seamlessly with the backend models through RESTful APIs.

A Contribution of each member

1. Luv valecha(B23CS1093): Implemented Random Forest, AdaBoost Classifier, Voting Classifier and Frontend.
2. Pratyush Chauhan(B23CM1030): Implemented Bernoulli Naive Bayes and Gradient Boosting Classifier. Prepared Spotlight video.

3. Dheeraj Kumar(B23CS1016): Implemented AdaBoost,Xgboost, deployed backend on Google Cloud Platform
4. Dhruv Sharma(B23EE1086): Implemented LinearSVM and Non-LinearSVM. Created Project Report.
5. Ritik Nagar(B23EE1061): Implemented KNN,Logistic regression, Created Mid Report and Frontend.
6. Shiv Jee yadav(B23EE1095): Implemented Decision Tree, ANN, Created Project Page and Frontend