Network Anomaly Detection Project Guide

1. Project Goal and Overview

The primary objective of this project was to develop and evaluate a machine learning pipeline capable of classifying network traffic as either **Normal (0)** or **Anomalous (1)**. Anomalies, in this context, include various forms of network intrusions and attacks.

The project utilized the **KDD (Knowledge Discovery and Data Mining)** dataset, a benchmark dataset for network intrusion detection, and compared three distinct modeling approaches: two supervised classifiers and one unsupervised outlier detection algorithm.

2. Data Source and Preparation

Data Source

The model was trained on the `kdd_train.csv` file, a subset of the KDD dataset.

Preprocessing Pipeline

The following steps were implemented in `data_handler.py` to prepare the raw data for modeling:

1. **Column Identification:** Identified the 41 feature columns and the `label` (target) column.
2. **Label Cleaning (Crucial Step):** The raw KDD labels often contain trailing periods (e.g., `'normal.'`). This was fixed using string cleaning operations (`.str.strip().str.replace('.', '', regex=False)`).
3. **Binarization:** The multiclass attack labels were consolidated into a binary target:
    - `'normal'` connections were mapped to **0 (Normal)**.
    - All other attack types (`neptune`, `smurf`, `portsweep`, etc.) were mapped to **1 (Anomaly)**.
4. **Feature Encoding:** The three categorical features (`protocol_type`, `service`, `flag`) were converted into a numerical format using **One-Hot Encoding** via `pd.get_dummies()`.
5. **Data Splitting:** The final dataset was split into training and testing sets (80/20 split) using `train_test_split` with **stratification** (`stratify=y`). This ensures that the test set maintains the correct proportion of Normal vs. Anomaly records, preventing biased evaluation.

3. Model Implementation and Training

Three different models were implemented in `model_trainer.py` to compare performance across different machine learning paradigms.

A. Supervised Models (Requires Labeled Data)

Model	Technique	Implementation Details
Random Forest (RF)	Ensemble/Tree-based	Used `RandomForestClassifier` from scikit-learn. Chosen for its robustness, speed, and ability to handle high-dimensional, mixed data types. Feature importance was extracted and plotted.
Multi-Layer Perceptron (MLP)	Deep Learning	Used a `Sequential` model from Keras/TensorFlow. The architecture included: Input Layer, two hidden layers (64 and 32 neurons with ReLU activation), a Dropout layer for regularization, and a final single neuron (Sigmoid activation) for binary classification.

B. Unsupervised Model (Outlier Detection)

Model	Technique	Implementation Details
Isolation Forest (IForest)	Outlier Detection	Used `IsolationForest` from scikit-learn. This model works by isolating anomalies (data points far from others) rather than learning the structure of normal data.
Contamination Factor	Hyperparameter	Set to **0.465** in `config.py`, derived from the true anomaly ratio in the training data (Anomaly count / Total count). This defines the expected proportion of outliers for the model.

4. Final Performance Results

The models were evaluated on the balanced test set (Normal: 13469, Anomaly: 11726).

Model	Accuracy	Normal (0) F1-Score	Anomaly (1) F1-Score	AUC Score	Key Observation
Random Forest	1.00	1.00	1.00	1.0000	Near-perfect separation; highly effective features.
Keras MLP	0.93	0.93	0.93	0.9456	Strong performance, validating the deep learning approach.
Isolation Forest	0.65	0.67	0.63	N/A	Performs reasonably well for an *unsupervised* model, correctly flagging outliers with no label guidance.

5. Conclusion and Next Steps

The project successfully demonstrated a robust pipeline for network anomaly detection.

- The **Random Forest Classifier** achieved the best performance, indicating that the features derived from the KDD dataset, especially after one-hot encoding the categorical fields, are highly predictive of an anomaly.

- The comparison clearly shows the superiority of **Supervised Learning** when high-quality labels are available.

Potential Future Improvements:

1. **Feature Scaling:** Implement standard scaling (`StandardScaler`) on continuous features to potentially improve MLP convergence and performance.
2. **Hyperparameter Tuning:** Use `GridSearchCV` or `RandomizedSearchCV` on the Random Forest and MLP to maximize accuracy.
3. **Dimensionality Reduction:** Explore techniques like PCA or feature selection (beyond basic importance plotting) to reduce model complexity and training time.