User Manual: Multi-Application Flow Classification and Analysis

---

Introduction

This guide explains how to use the provided Python script to classify network flows from an Elasticsearch dataset and analyze the performance of classification models. The script employs machine learning techniques, including Naive Bayes and Random Forest classifiers, to detect anomalies and evaluate performance across multiple applications.

---

1. Prerequisites

Requirements

1. **Elasticsearch**:
    - Ensure an Elasticsearch server is running and accessible.
    - Provide connection details (host, port, credentials, and certificate path) in the script.
2. **Dataset**:
    - An Elasticsearch index (e.g., `network_flows_fan_encoded`) containing labeled flow data.
3. **Dependencies**:
    - Install required Python packages using:

      ```
      pip install pandas numpy matplotlib seaborn elasticsearch scikit-learn tqdm joblib
      ```

---

2. How to Use the Script

Setting Up

1. Place your PCAP files in the `pcap_files` directory.
2. Ensure the ground truth file (`TRAIN.gt`) is in the `pcap_files/flows/` directory.

Execution

Run the notebook in colab or jupyter :

```
3_classification_allApp.ipynb
```

Expected Output:

- Connection confirmation to Elasticsearch.
- Data fetched and processed.
- Classification results displayed in the terminal and saved as visualizations.

Step 1: Connect to Elasticsearch

- Update the connection details:
  - `elastic_host`: Elasticsearch server address.
  - `elastic_port`: Server port.
  - `elastic_user` and `elastic_password`: Credentials.
  - `elastic_ca_path`: Path to the CA certificate for secure connections.
- The script establishes the connection and verifies it.

Step 2: Fetch Data

- The function `fetch_flows_from_elasticsearch()` retrieves data from the specified Elasticsearch index.
- Ensure the index contains columns like `label` (target) and `application_name` (application identifier).

Step 3: Preprocessing

- Data is split into training and testing sets using an 80:20 ratio.
- Application names are encoded using `LabelEncoder`.
- For some models, features are normalized to ensure compatibility (e.g., Naive Bayes).

---

3. Classification Models

3.1 Naive Bayes

- Converts features to a non-negative format compatible with the `MultinomialNB` classifier.
- **Steps**:
  - Model training on the training dataset.
  - Evaluation on the test dataset.
  - Metrics: Accuracy, Precision, Recall, F1-Score, and AUC-ROC.
  - Outputs: Confusion matrix, classification report, and ROC curve.

3.2 Random Forest

- Hyperparameter tuning to optimize model performance:
  - `n_estimators`, `max_depth`, `min_samples_split`, `min_samples_leaf`, and `max_features`.
- **Steps**:
  - Grid search for best parameters.
  - Training the optimized model.
  - Evaluation metrics as in Naive Bayes.
  - Visualization of feature importance.
- Saves the trained model using `joblib`.

---

4. Validation

4.1 Cross-Validation

- Uses 5-fold stratified cross-validation to ensure robust performance evaluation.
- Outputs:
  - Confusion matrices and ROC curves for each fold.
  - Summary of metrics (Accuracy, Precision, Recall, F1-Score, and AUC-ROC) across folds.

4.2 Error Analysis

- Analyzes misclassifications and false negatives for each application.
- Visualizations:
  - Error rates, false negative rates, and contributions to total errors by application.
  - Combined bar charts for top applications based on error rates.
- Outputs a detailed report with key statistics for applications.

---

5. Outputs

Generated Reports

1. **Performance Metrics**:
   - Classification reports with Precision, Recall, F1-Score, and AUC-ROC.
2. **Visualizations**:
   - Confusion matrices.
   - ROC curves.
   - Feature importance (Random Forest).
   - Error analysis charts.

Saved Models

- Trained models are saved as `.sav` files for future use.

Error Analysis Report

- Top 10 applications with the highest error rates, false negative rates, and contributions to overall errors.

---

6. Customization

6.1 Modifying Elasticsearch Query

- Update the query body in `fetch_flows_from_elasticsearch()` to filter or customize data retrieval.

6.2 Changing Target or Features

- Replace `label` as the target column if needed.
- Update feature selection (e.g., columns to drop or encode).

6.3 Adding New Models

- Use the structure of Naive Bayes and Random Forest to integrate additional classifiers (e.g., SVM, Gradient Boosting).

6.4 Hyperparameter Tuning

- Adjust parameter ranges in Random Forest tuning loops for finer control.

---

7. Troubleshooting

Common Issues

1. **Connection Errors**:
    - Verify Elasticsearch credentials and certificate paths.
    - Ensure the server is running and accessible.
2. **Missing Data**:
    - Confirm the Elasticsearch index contains required fields (e.g., `label`, `application_name`).
3. **Model Training Errors**:
    - Check for missing or improperly formatted columns in the dataset.

For further details or issues, refer to the inline comments within the script.