**Automated ESA Model Training: A User Guide**

**1. Overview**

This guide provides instructions on how to use the suite of Python scripts designed to automate the training of fault detection models based on Electrical Signature Analysis (ESA). These scripts take raw harmonics data from a machine, process it, build a specific fault detection model or configuration, and log all results to MLflow for versioning and traceability.

The primary goal is to enable operations or reliability personnel to onboard new equipment and generate production-ready fault detection configurations with a single command, without needing deep machine learning expertise.

[Image of a machine learning workflow diagram](https://encrypted-tbn3.gstatic.com/licensed-image?q=tbn:ANd9GcSAlJD_sEN7q43xesqWJTAvYgXvhdNhMT_hMD-AAPeLNdfWHR-8qHL8efXFXmlSMwwT-BxxrNMUUB4IsEy9WqDIoxmTGE3xQ0R2f5PDlNuttMXHOsI)

The general workflow for each script is:

1. **Ingest Data**: Load harmonics data from a specified CSV file.
2. **Feature Engineering**: Calculate specific metrics relevant to a particular fault (e.g., phase imbalance, sideband energy, harmonic profiles).
3. **Model/Config Generation**: Train a model (like an SVM) or derive statistical thresholds (like mean + 3σ).
4. **Artifact Creation**: Save the model (.pkl) and/or configuration (.yaml) files.
5. **Logging**: Log all parameters, metrics, and artifacts to an MLflow run for reproducibility.

**2. Initial Setup (One-Time)**

Before running any scripts, ensure your environment is set up correctly.

**Directory Structure**

Your project must follow this structure for the scripts to locate the data files correctly:

your\_project\_folder/

├── data/

│ └── iotts.harmonics\_257.csv <-- Place all equipment CSV data here

├── bearing\_model\_training.py

├── gearbox\_model\_training.py

├── winding\_model\_training.py

├── rotor\_model\_training.py

└── mlflow\_base.py

**Environment Variables**

The scripts connect to MLflow using environment variables. Ensure the following are set in your terminal session or a .env file:

* MLFLOW\_TRACKING\_URI: The URL of your MLflow tracking server.
* MLFLOW\_TRACKING\_USERNAME: Your MLflow username.
* MLFLOW\_TRACKING\_PASSWORD: Your MLflow password.

**3. The Training Scripts**

Below are the details for each of the four fault detection scripts. To use them, open your terminal, navigate to your\_project\_folder/, and run the desired command.

**a. Bearing Fault Detection**

* **Script**: bearing\_model\_training.py
* **Purpose**: Detects bearing faults by calculating stateless spectral features (kurtosis, crest factor, etc.) and fault band energies (BPFO/BPFI). It generates a YAML configuration file with feature weights and normalization stats.
* **Method**: Weighted combination of normalized spectral features.
* **Key Parameters**: Requires bearing geometry details for accurate fault frequency calculation.
* **How to Run**:

python bearing\_model\_training.py --tenant-id "<tenant\_id>" --machine-id "<machine\_id>" --dataset-filename "<your\_data.csv>" --shaft-rpm <rpm> --ball-diameter <diameter\_mm> --pitch-diameter <diameter\_mm> --num-elements <count>

**b. Gearbox Fault Detection**

* **Script**: gearbox\_model\_training.py
* **Purpose**: Identifies gearbox faults by creating "harmonic profiles" (low, intermediate, high, and optionally odd/even parity orders). It calculates normalization stats for these profiles and derives a fault score threshold.
* **Method**: Harmonic profiling with an unlabeled evaluation to set a decision threshold.
* **Key Parameters**: The core inputs are the machine IDs and dataset. Algorithm behavior can be tuned with flags like --use-parity-profiles.
* **How to Run**:

python gearbox\_model\_training.py --tenant-id "<tenant\_id>" --machine-id "<machine\_id>" --dataset-filename "<your\_data.csv>" --use-parity-profiles

**c. Winding Fault Detection**

* **Script**: winding\_model\_training.py
* **Purpose**: Detects potential winding issues by measuring the phase imbalance across current and voltage harmonics. This is the simplest model, generating a YAML file with statistical thresholds.
* **Method**: Calculates mean and standard deviation of phase imbalance, setting a mean + 3σ threshold.
* **Key Parameters**: Only requires machine IDs and the dataset.
* **How to Run**:

python winding\_model\_training.py --tenant-id "<tenant\_id>" --machine-id "<machine\_id>" --dataset-filename "<your\_data.csv>"

**d. Rotor Fault Detection**

* **Script**: rotor\_model\_training.py
* **Purpose**: Trains a machine learning model to detect rotor faults. It uses sideband harmonics as features, reduces their dimensionality with PCA, and trains a One-Class SVM to identify anomalies.
* **Method**: PCA for dimensionality reduction + One-Class SVM for anomaly detection.
* **Key Parameters**: In addition to IDs, you can tune the number of PCA components (--n-components) and SVM parameters (--svm-nu).
* **How to Run**:

python rotor\_model\_training.py --tenant-id "<tenant\_id>" --machine-id "<machine\_id>" --dataset-filename "<your\_data.csv>" --n-components 5 --svm-nu 0.05

**4. General Workflow & Next Steps**

1. **Prepare Data**: Place the harmonics CSV for the new equipment into the data/ folder.
2. **Choose Script**: Select the appropriate script based on the fault you want to model (e.g., gearbox\_model\_training.py for a gearbox).
3. **Run Command**: Execute the script from your terminal, replacing placeholder values (e.g., <tenant\_id>) with the actual information for the new equipment.
4. **Verify in MLflow**: After the script finishes, log into your MLflow instance. Find the experiment (e.g., gearbox\_fault\_monitoring/<tenant\_id>/<machine\_id>) and review the latest run. Check the logged parameters, metrics, and ensure the artifact(s) (.yaml and/or .pkl) were created.
5. **Deploy**: The generated artifacts can now be used by the model serving repository for real-time inference on the new equipment.