ML-Enhanced Partial Discharge Fault Detection in Covered Conductors

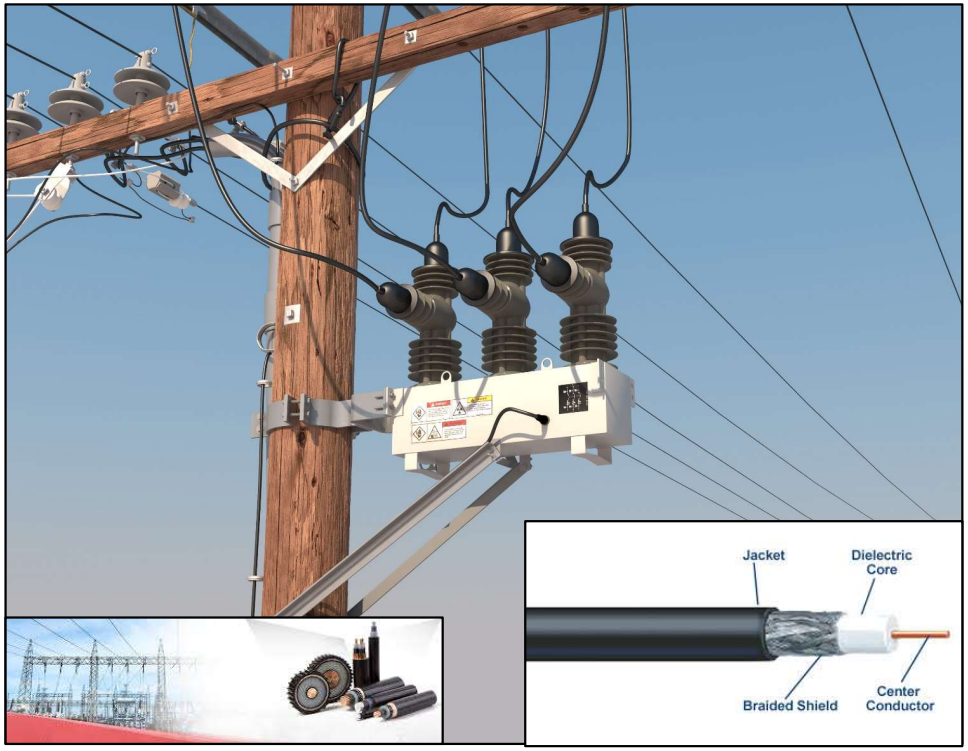
Jeffrey Egan // September 2022

Overview

- Background Information on the Project
- Quick Exploratory Data Analysis
- Primer on Partial-Discharge Faults
- Data Preparation – Signal Processing
- Feature Extraction and Features Analysis
- Machine Learning Models and Deep Learning Networks
- Model Evaluation and Performance Results
- Interesting Observations, Lessons Learned
- Ideas for Improving / Expanding upon this Project

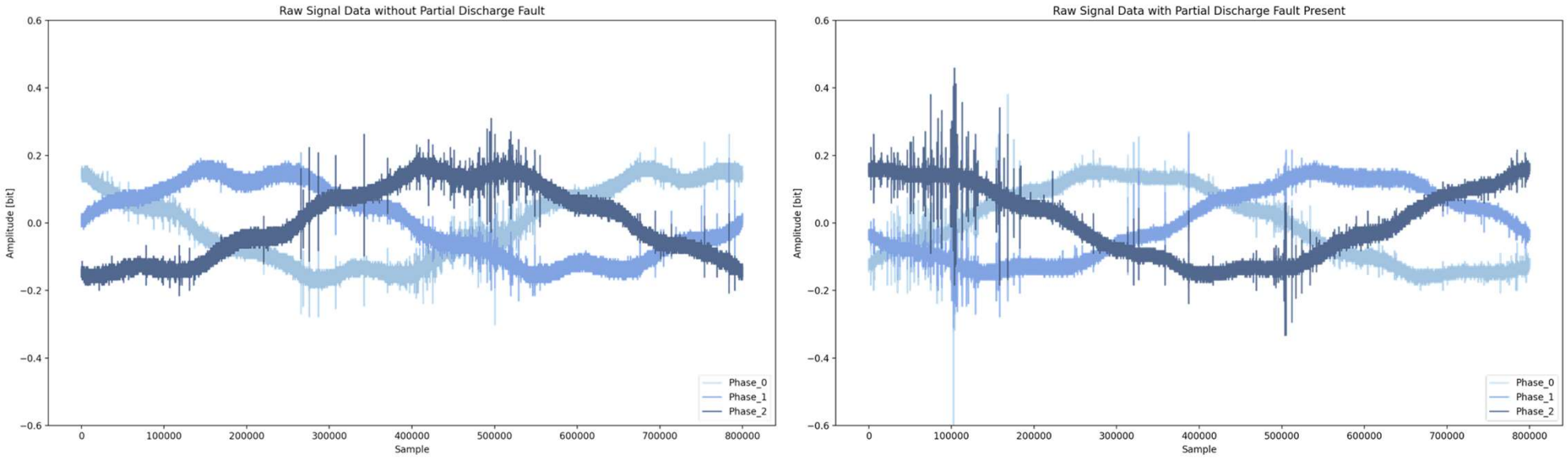
Partial Discharge Faults – Background & Motivation

- Overhead, medium-voltage power lines run for hundreds of miles. These great distances make it expensive to manually inspect for damage that doesn't immediately lead to a power outage.
- Degradation or rupturing of insulation systems can lead to a phenomenon known as partial discharge — an electrical discharge which does not bridge the electrodes between an insulation system completely.
- Partial discharges slowly damage the power line. Left unrepaired they will eventually lead to a power outage, additional damage, and possibly even start a fire.
- Meters employed to collect voltage data



Detect PD-Faults to enable proactive maintenance, avoiding service outages and reducing cost

Sample Measurements with 3 Phase Signals



Each signal (phase) contains 800,000 measurements of a power line's voltage, taken over 20 milliseconds.

For signal processing, that means each signal data set has a sample rate of 40 million samples per second.

The underlying electric grid operates at 50Hz, this means each signal covers a single complete grid cycle.

Quick EDA Reveals Highly Unbalanced Data

- All Provided Data: Labeled & Unlabeled
  - Labeled Training Set: 2,904 Measurements = 8,712 Signals (only 30% of data is labeled)
  - Unlabeled Test Set: Perform Binary Classification: 6,779 Measurements = 20,337 Signals

- Occurrence of Faults and Non-Faults in Labeled Data
  - 8187 Signals without Fault, 213 Signals with Fault (only 2.5% of labeled signals have faults!)

- Grouping of Signal Faults per Measurement
  - Measurements with no faults present: 2710 (93.3% of measurements exhibit no fault)
  - Measurements with fault present in only 1 phase: 19 (9.7% of cases where faults present)
  - Measurements with faults present in 2 of 3 phases: 19 (9.7% of cases where faults present)
  - Measurements with faults present in all 3 phases: 156 (80.4% of cases where faults present)

Detecting Partial Discharges is Difficult

PD Pattern Observed alongside other Noise Sources

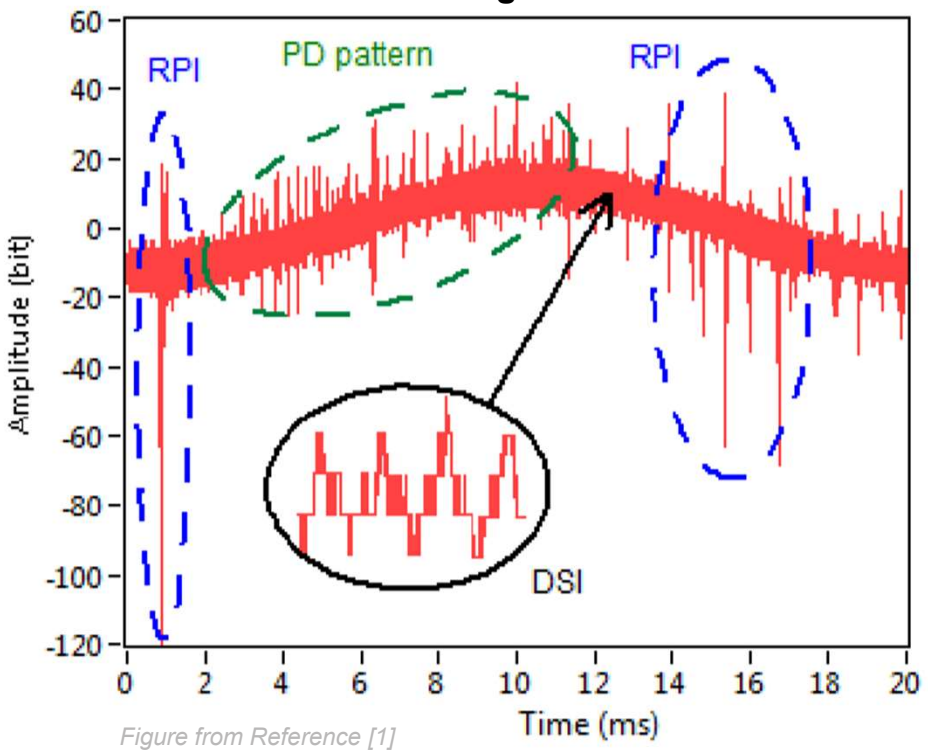


Figure from Reference [1]

Many other sources of noise could be falsely attributed to partial discharge.

- Ambient and Amplifier Noise
- PD: Partial Discharge Fault Pattern
- DSI: Discrete Spectral Interference
  - E.g. Radio Emissions, Power Electronics
- RPI: Random Pulses Interference
  - Lightning, Switching Operations, Corona

Identifying and Canceling Corona Discharge Peaks

Corona Discharge Pulse Followed by Dumped Oscillation

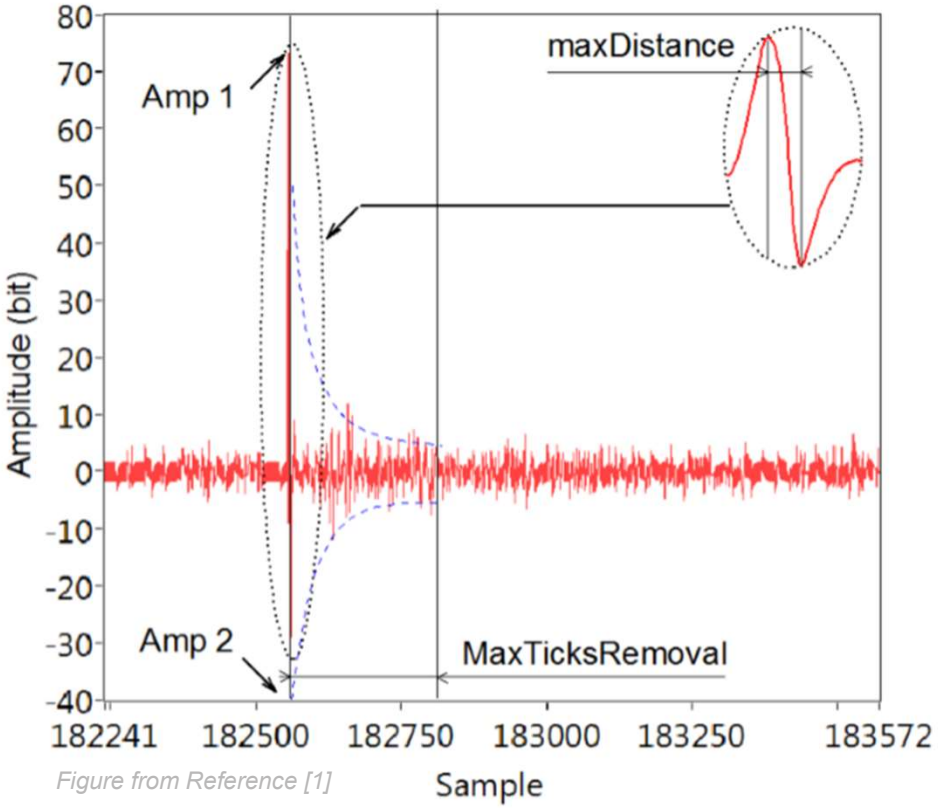


Figure from Reference [1]

- A corona discharge is an electrical discharge brought on by the ionization of a fluid such as air surrounding a conductor that is electrically charged.
- Spontaneous corona discharges occur naturally in high-voltage systems unless care is taken to limit the electric field strength.
- A corona will occur when the strength of the electric field (potential gradient) around a conductor is high enough to form a conductive region, but not high enough to cause electrical breakdown or arcing to nearby objects.

Identify and flag probable Corona discharge peaks to mitigate false positives

Focus on Portions of the Sinusoid with Rising Amplitude for Improved PD Detection

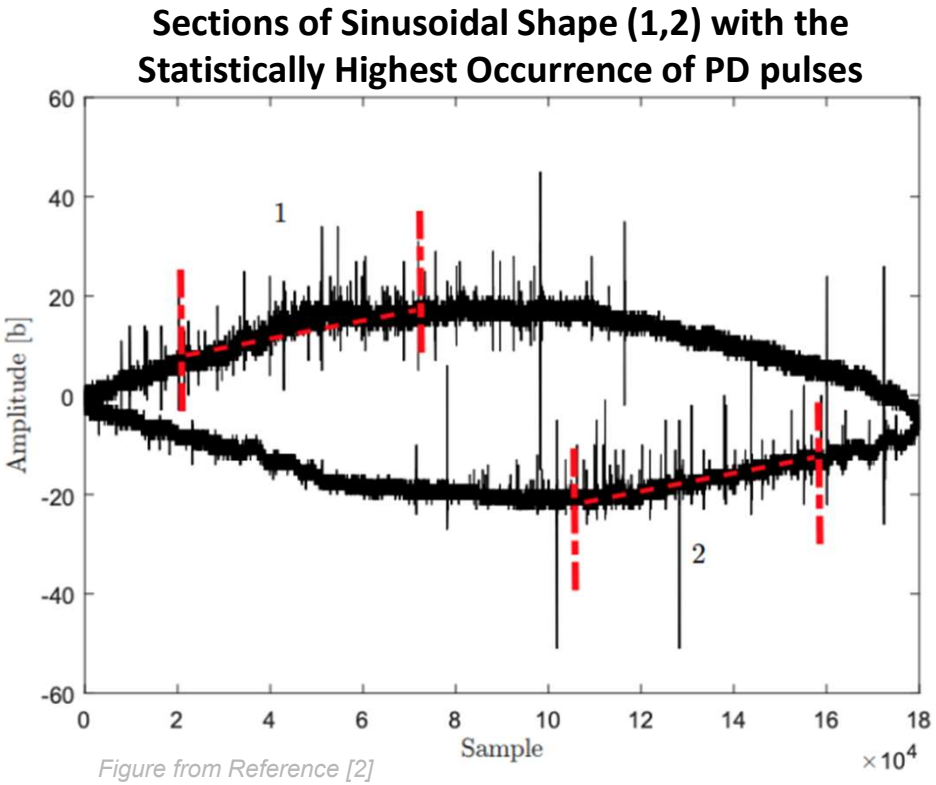
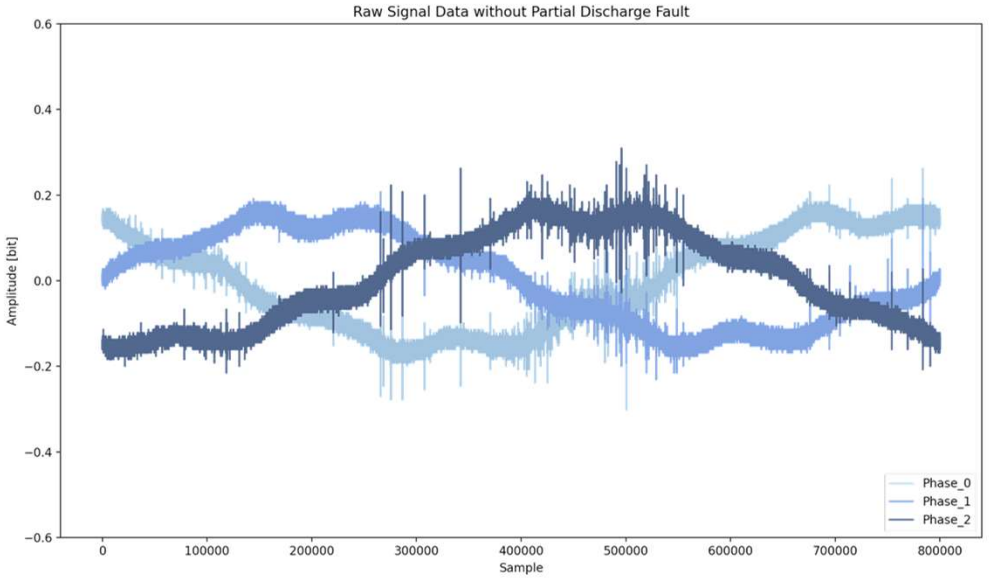
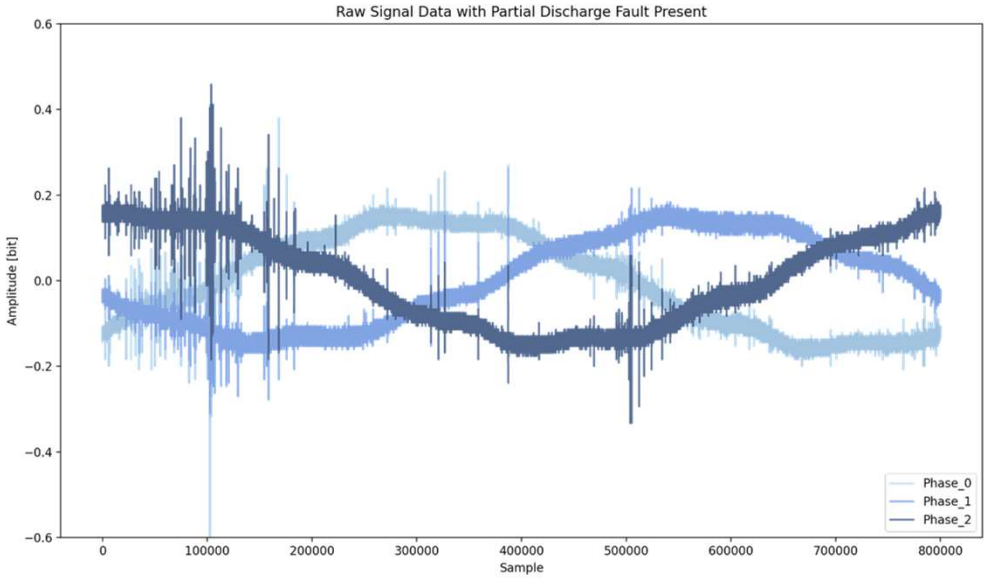


Figure from Reference [2]

- One of the most common fundamental features of the PD pattern is that it occurs on the rising amplitude edges of the sinusoidal curve.
- During feature extraction, this allows omission of half of the signal data points – increasing processing efficiency.
- Running feature extraction on the high-probable region also increases the model's ability to detect true faults and avoid false positives.

Data Preparation & Signal Processing

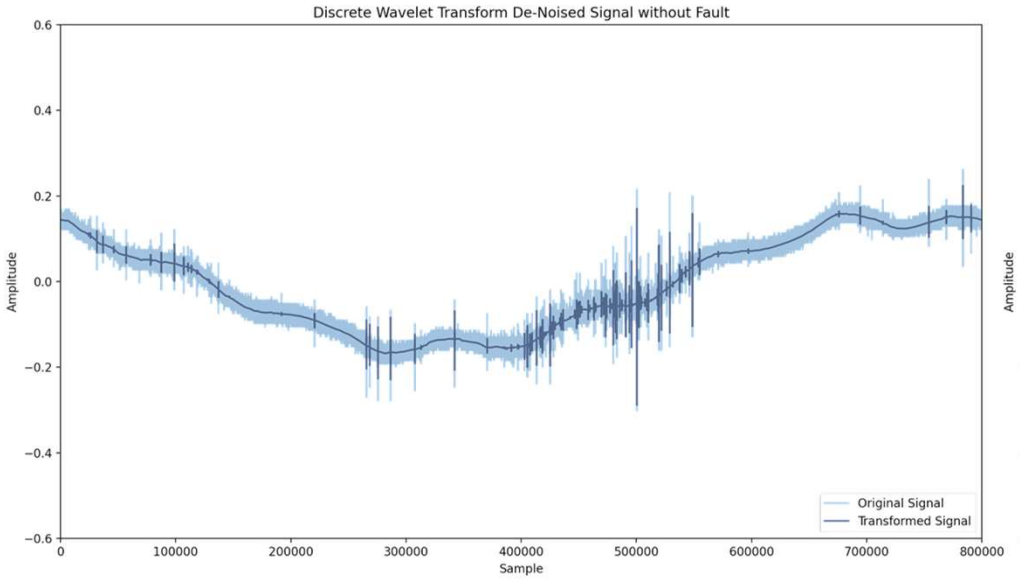


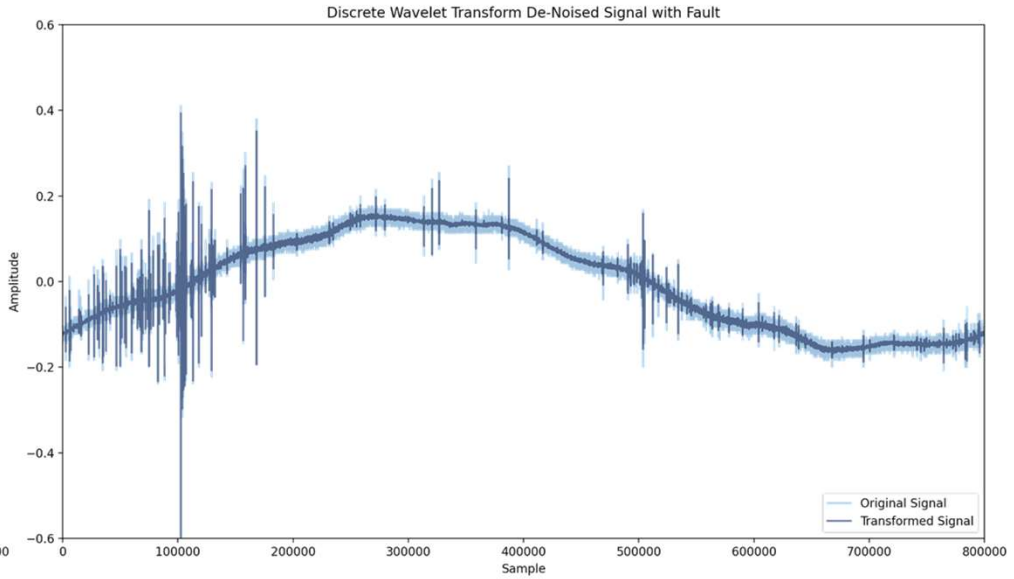


Three signals per measurement, 800k samples / signal. Signals individually labeled.

Raw Signal → Denoising → Fit Sinusoid → Probable Region → De-Trend → Trimming → Feature Extraction

Denoising – Remove DSI & Ambient Noise with DWT

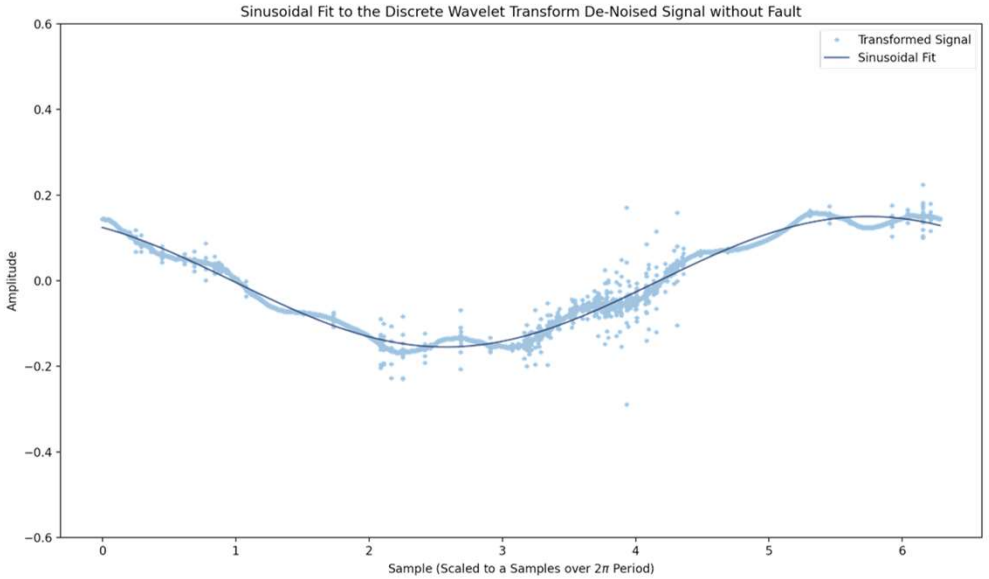


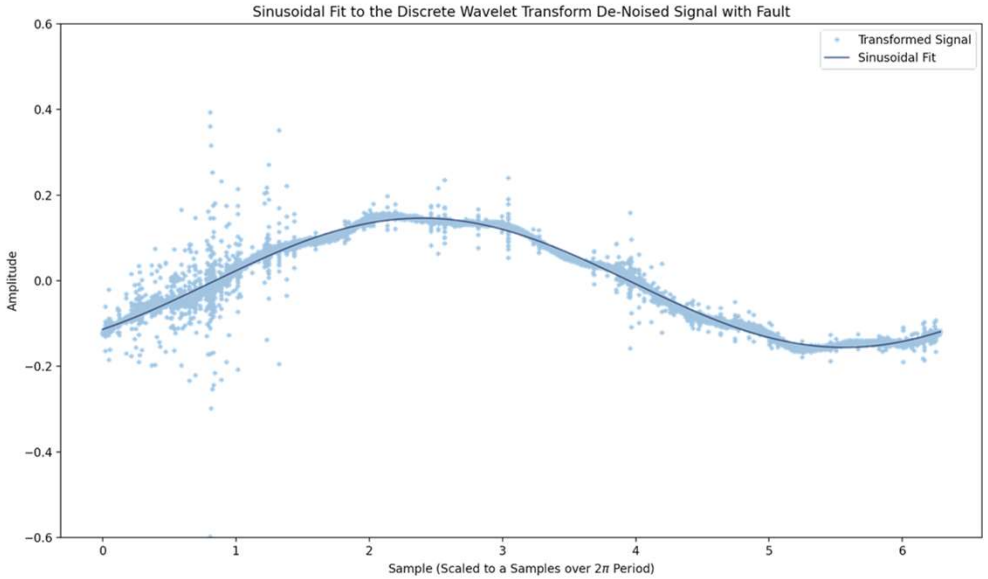


Discrete Wavelet Transform to Remove Spectral Interference and Ambient Noise – Daubechies 4 Wavelet (db4)

Raw Signal → **Denoising** → Fit Sinusoid → Probable Region → De-Trend → Trimming → Feature Extraction

Fit a Sinusoidal Function to the DWT Signal

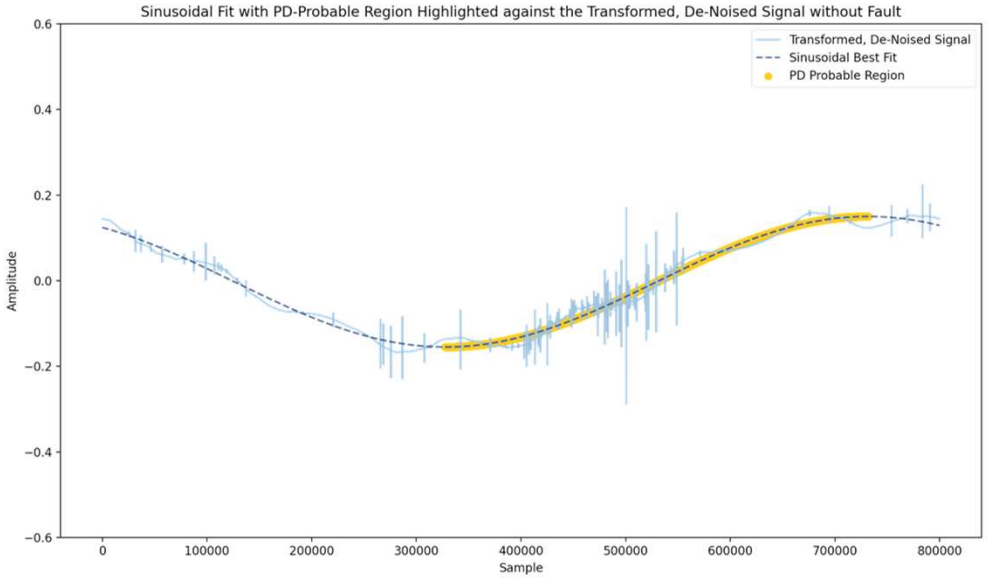


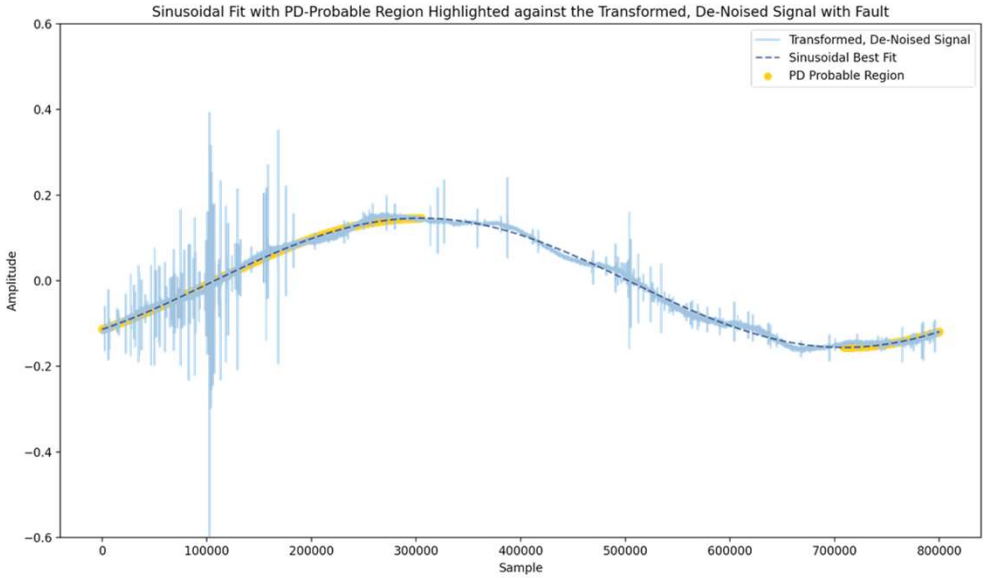


Fit a 50Hz sinusoid to the de-noised signal with a least squares approach

Raw Signal → Denoising → **Fit Sinusoid** → Probable Region → De-Trend → Trimming → Feature Extraction

Use Rising Amplitude to Identify the PD-Probable Region

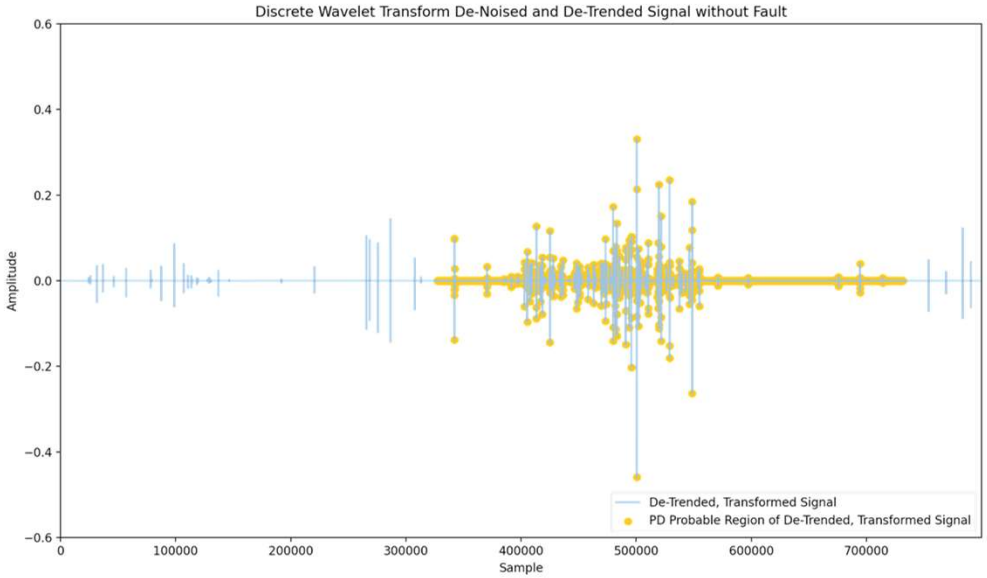


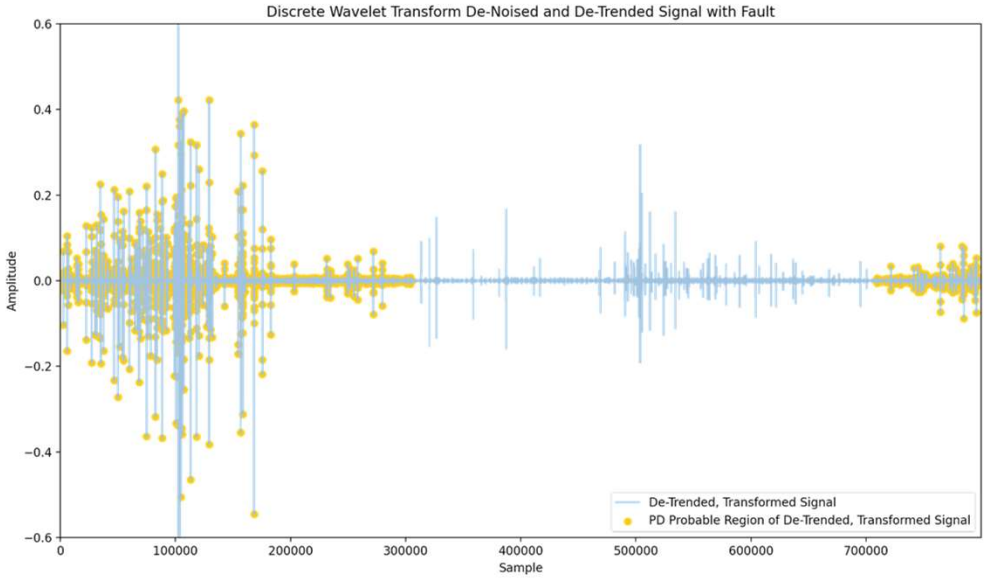


Use first derivative to index the rising amplitude of the sinusoid and highlight the PD-probable region of the signal

Raw Signal → Denoising → Fit Sinusoid → **Probable Region** → De-Trend → Trimming → Feature Extraction

Detrend Signal to Remove Sinusoidal Element

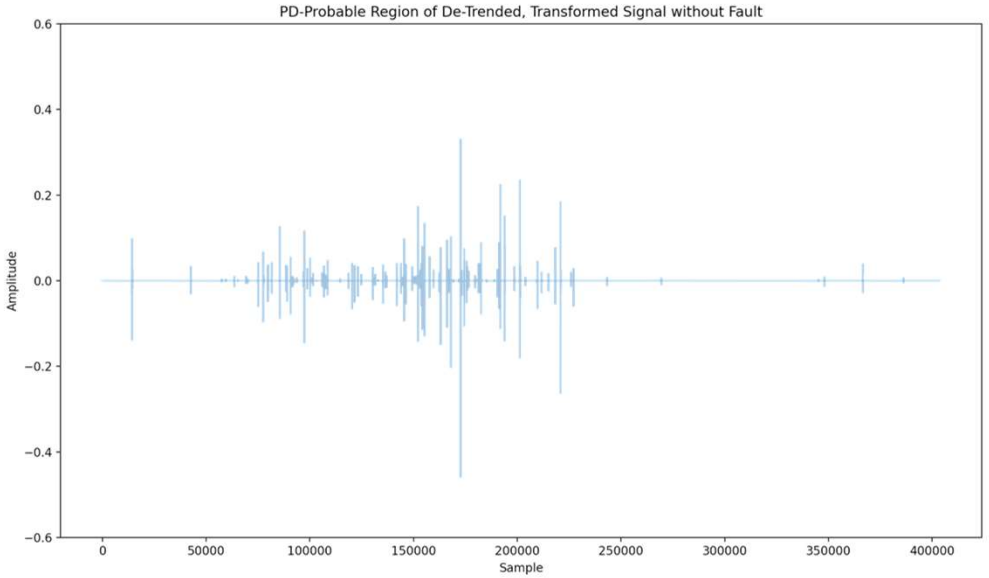


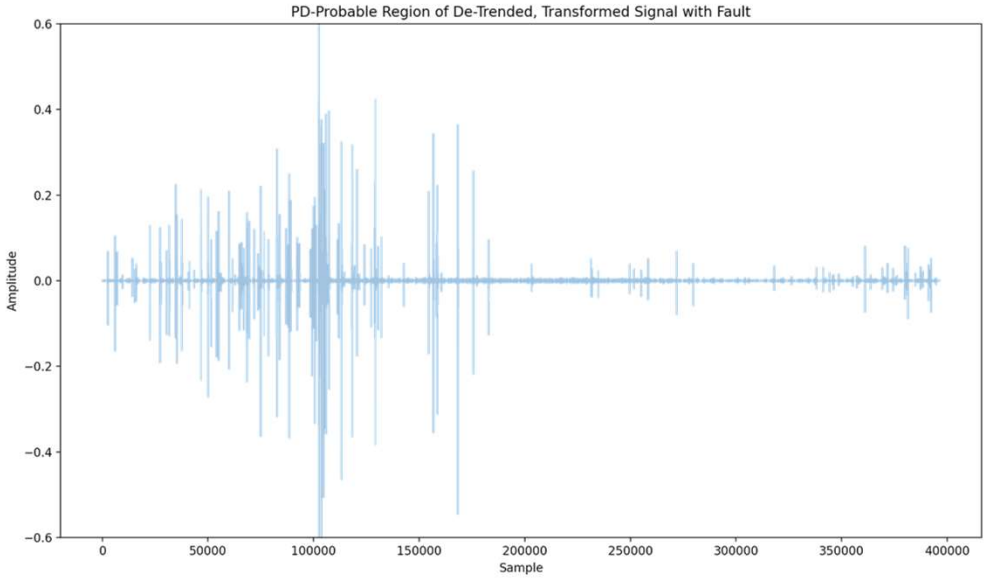


Use an Nth Discrete Difference to De-Trend the Function – First Order was Sufficient

Raw Signal → Denoising → Fit Sinusoid → Probable Region → **De-Trend** → Trimming → Feature Extraction

Trim Signal to Only Look at the PD-Probable Region





Discard the non-probable region, and trim the signal from 800k to 400k samples

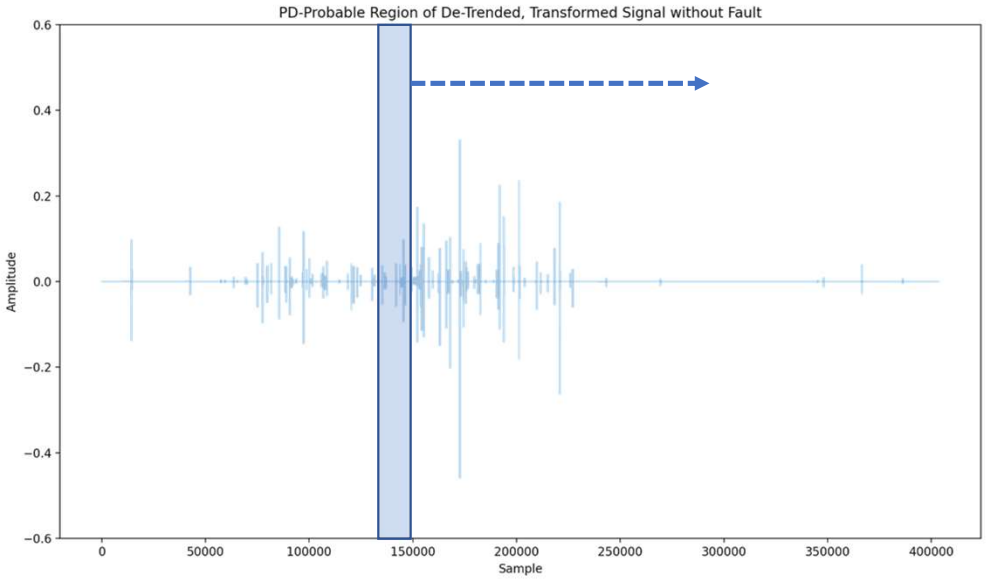
Raw Signal → Denoising → Fit Sinusoid → Probable Region → De-Trend → **Trimming** → Feature Extraction

Perform Feature Extraction on Signal

Process the PD-Probable Region of the De-Trended, De-Noised Signal and Build a Feature Matrix to be used as Input to Machine Learning and Deep Learning Models.

Extraction Yields 36 Features

- Mean
- Standard Deviation
- Variance
- Mean + Std. Deviation
- Mean - Std. Deviation
- Max Range
- Root Mean Square
- Entropy
- Number Crossings (mean & zero)
- Percentiles [0, 1, 25, 50, 75, 99, 100]
- Relative Percentiles (pctl. - mean)
- Peak Counts (True, Cancelled)
- Peak Statistics (mean/max height, width, etc)



Slice the signal into 400 parts (each 1000 samples in length)
Calculate and determine features per slice
Final Feature Array Shape: [8275, 36, 400]

False Peaks Cancellation

- From reviewing background literature on the subject [2], it was made clear that a statistically significant portion of the false positives in prior research were driven by false peaks.
- False peaks…
  - Reach much higher amplitude than the PD-pattern
  - Were often followed by another large peak of opposite polarity, creating a nearly symmetric pair
  - Have an identifiable “pulse trains” in the time domain.

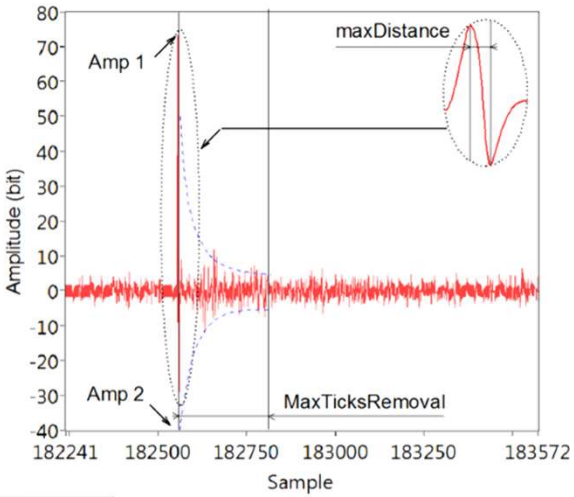
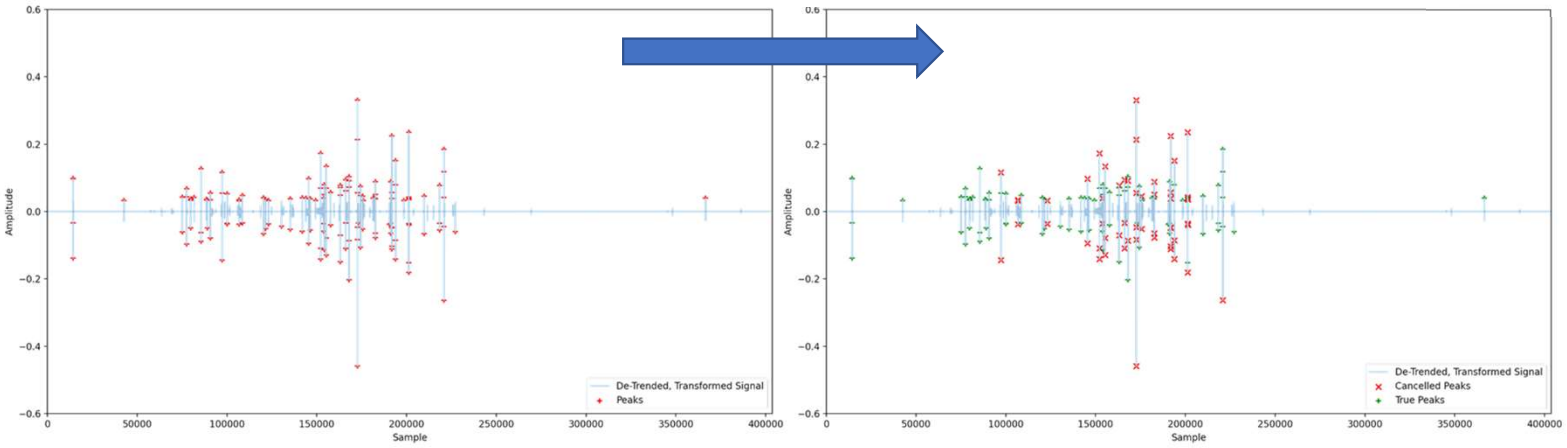
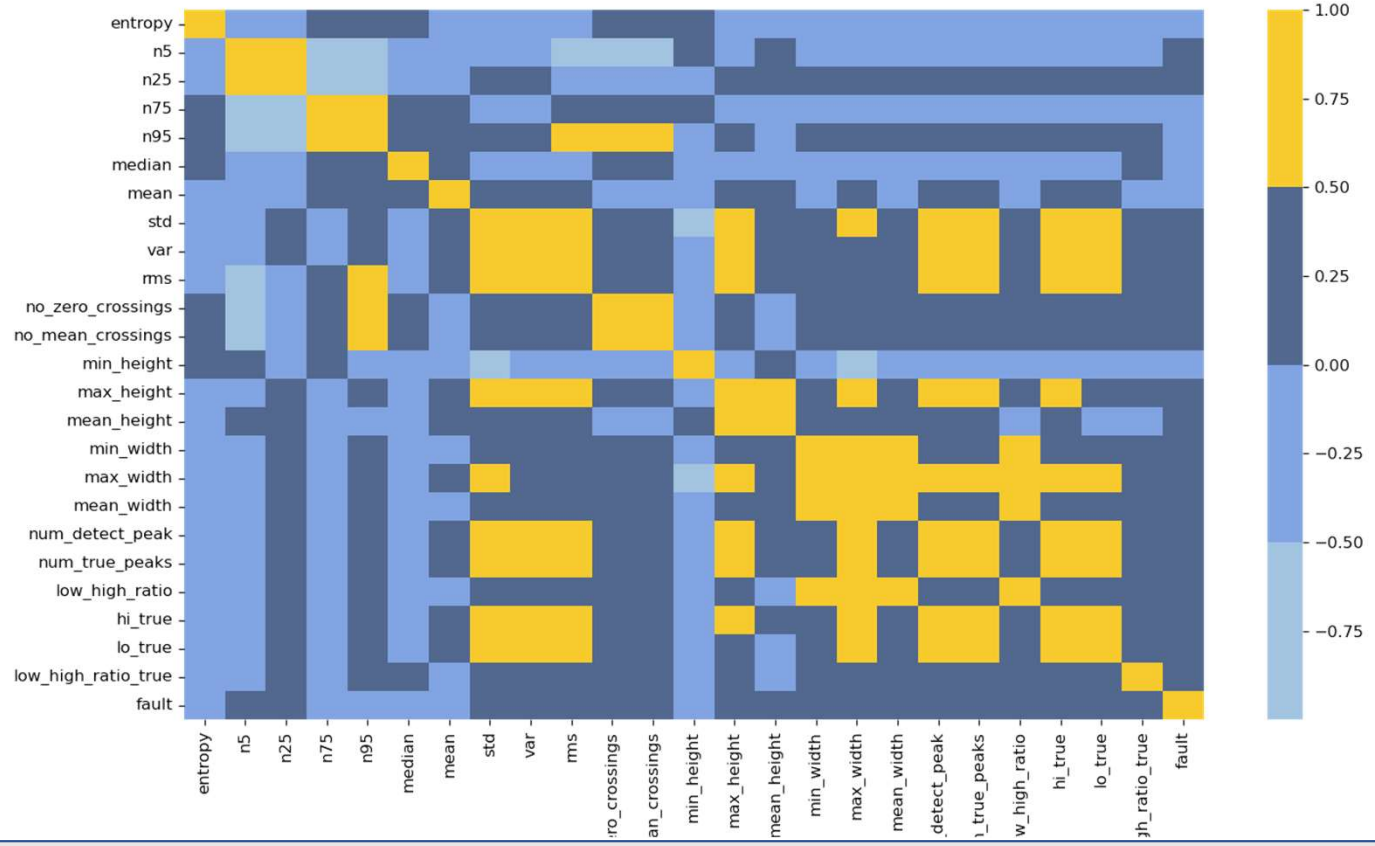


Figure from Reference [1]

Signal without Fault, Before and After False Peak Cancellation

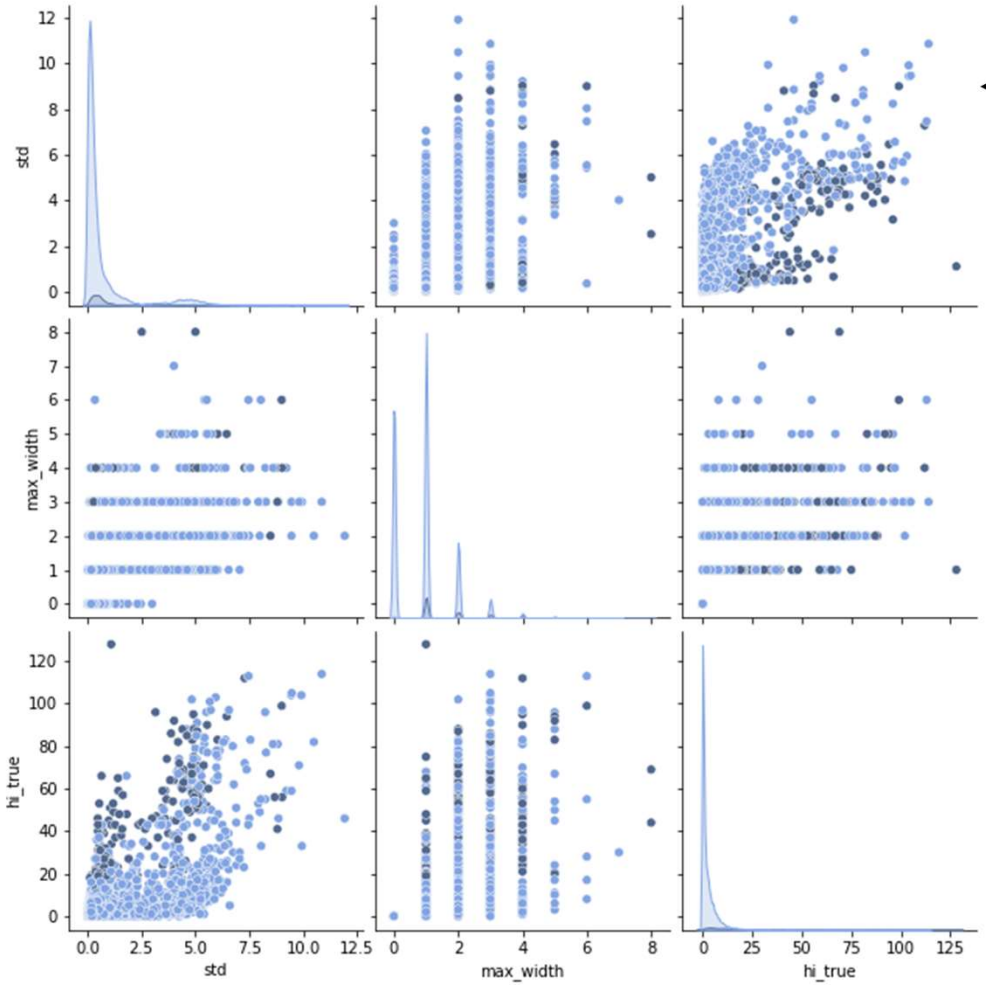


Feature Exploration – Correlation Heat Map



Useful for a quick sanity check on extracted features. May also be used to identify redundant features.

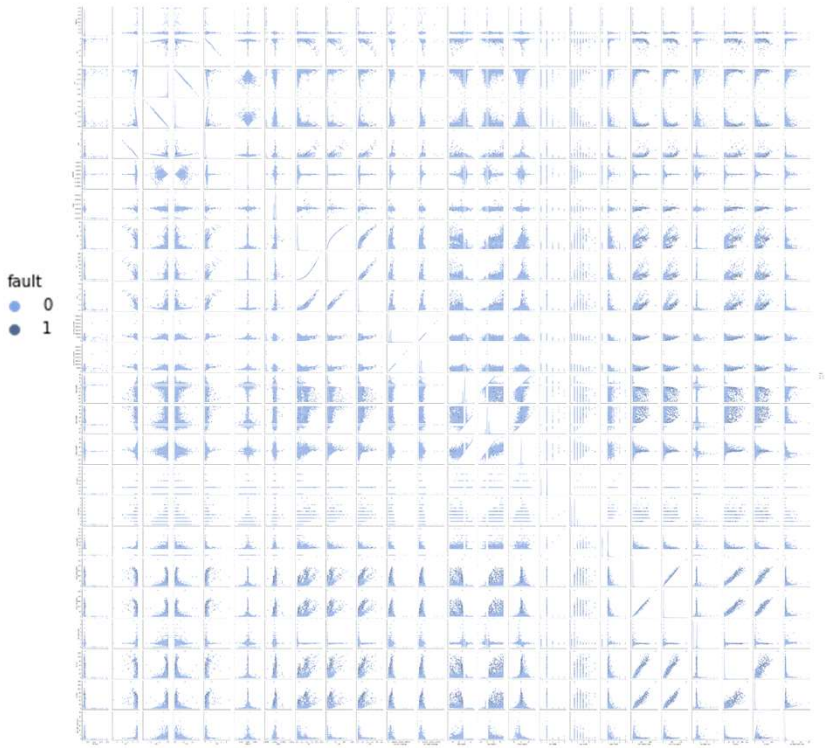
Feature Exploration – Pair Grids & PDF Estimates



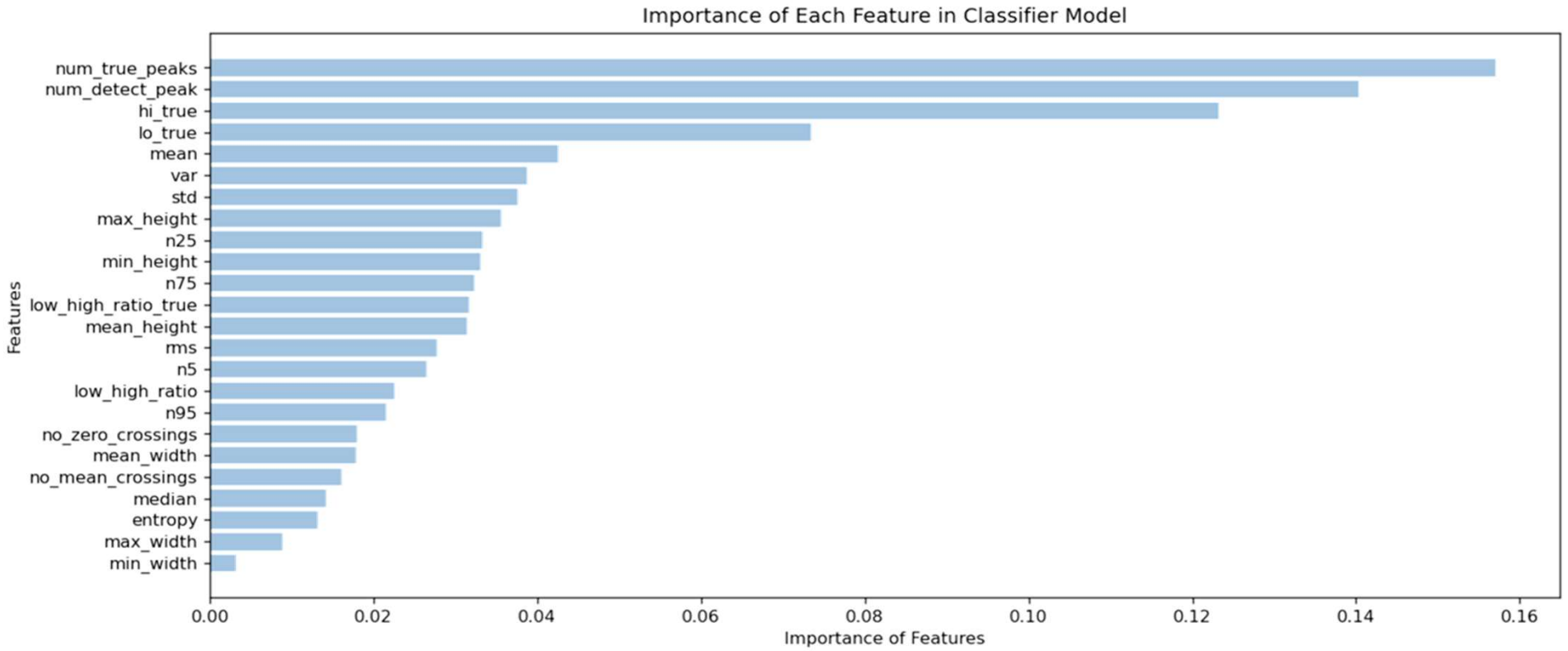
3x3 Pair Grid to Show Detail

Scatters and KDE (probability density estimations)

Full Pair Grid



Not All Features are Equally Influential



Later in the process, some features with minimal influence may be omitted to reduce the size of a model

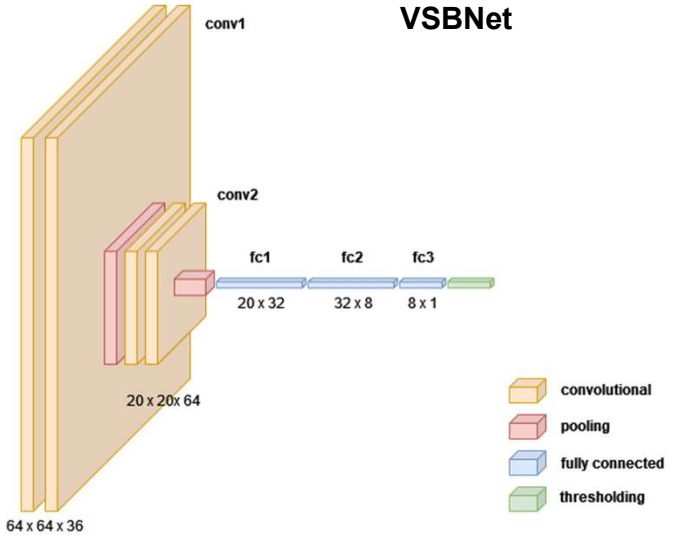
Machine Learning Approaches

- k-Nearest Neighbors (k-NN)
  - Monte Carlo over k parameter
- Support Vector Machines (SVM)
  - Radial Basis Function (rbf) kernel and Monte Carlo over gamma parameter.
- Random Forest Decision Trees (RF)
  - Regarded for handling unbalanced data when tuned properly.
- Light Gradient Boosting Framework
  - Powerful, tree-based learning algorithm that grows vertically (leaf-wise)
  - Sensitive to overfitting on small data sets (<10,000 data points)
  - In binary classification, model returns probability that a sample belongs to the positive class.

Deep Learning Approaches

- Recurrent Neural Network (LSTM)
  - Signal length (400k) too excessive to feed directly without further processing (e.g. truncating, compressive sampling, or some encoder-decoder architecture).
  - They can be adapted for classification but prior experience with LSTM models used for time series prediction

- Convolutional Neural Network (CNN)
  - Effective at pattern recognition and anomaly detection
  - Simple CNN Model:
    - Convolution layers (feature extraction)
    - Pooling layers (decrease the size of convolved feature maps, summarizes features)
    - Fully connected layers (flattening & classification)
    - Dropout (mitigates overfitting when all features are fully connected



Evaluate Network Performance with MCC

- Network performance is evaluated using Matthews Correlation Coefficient (MCC) between the predicted and observed response. The MCC is given by:

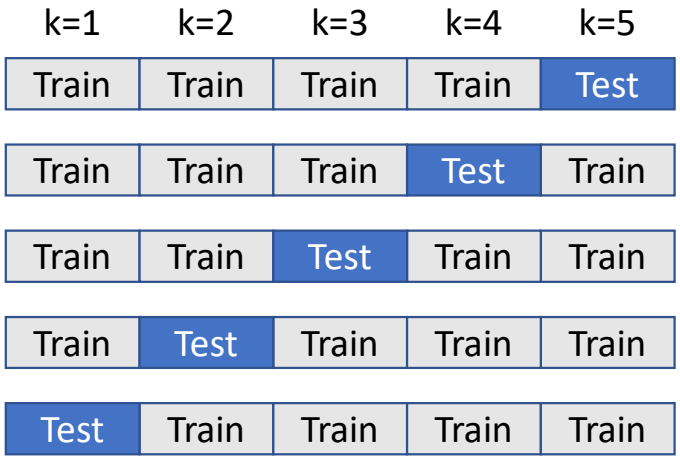
$$MCC = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

  where TP is the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives.

- Matthews Correlation Coefficient is an apt metric to assess a binary classification network, especially one with a highly unbalanced class where metrics like accuracy, precision, and recall don't adequately capture the effectiveness of a network.

Use k-Fold CV to Ensure Model Robustness to Test/Train Splits

- Use Ensemble k-Fold Cross-Validation to Validate Model Robustness to Test-Train Splits
  - Splits the data into k parts, Take k-1 parts as training data and 1 part for validation
  - Train and predict with the model k times over, holding out a different part each time.
  - Ensemble output produces probability of sample belonging to a class, not assignment
  - Average and use the prediction probabilities from the k scores to assign a class

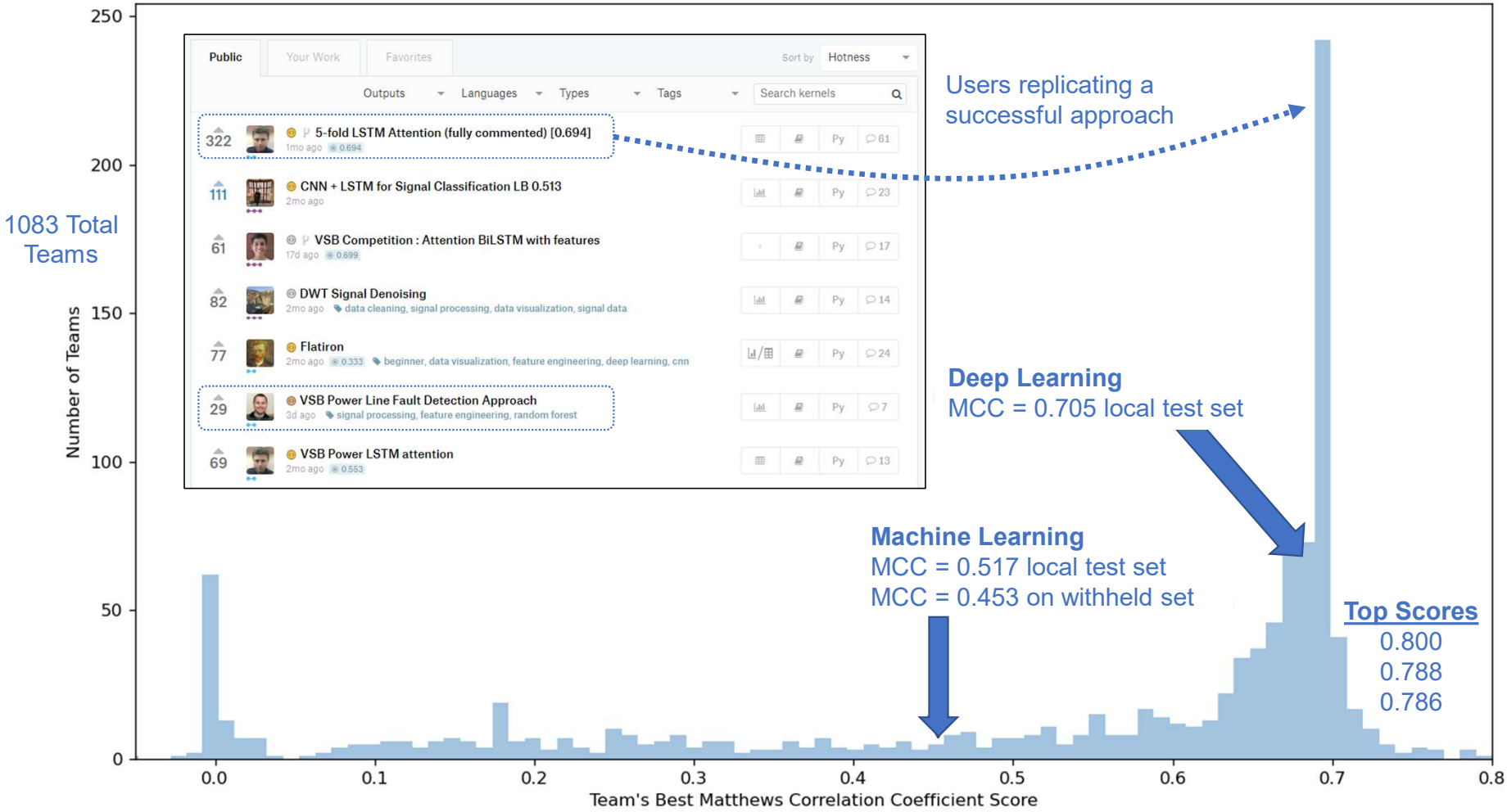


An example of 5-fold Cross Validation.

For each iteration, fit the model on the training folds and evaluate it using the test fold.

Retain the evaluation scores and probabilities between iterations but discard the trained model.

Results – Distribution of Public Leaderboard Scores

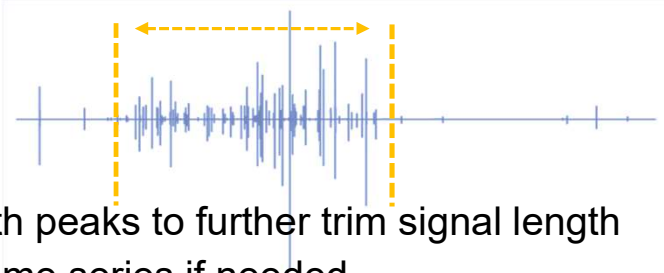


Interesting Observations and Lessons

- Use the right performance metric
  - Traditional metrics like accuracy, precision, recall, and F1 scores can be misleading
  - Matthews Correlation Coefficient – an effective binary classification performance measure - even for classes of very different sizes.
  - Pearson's Chi-Squared Test - a similar metric for higher order classification problems
- Account for the influence of test-train splits
  - With so few samples belonging to one of the classes, results can vary wildly
  - Rerun analyses while varying splits and random states to find robust performance predictions.
  - `StratifiedShuffleSplit` helps here, returning stratified randomized folds that preserve the percentage of records for each class.
- Use a model or framework built for the task, and tune it properly
  - Some models are more adept at handling unbalanced classes than others
  - Modify and iterate on the impact of class weights, don't rely on defaults
  - Ensemble Cross-Validation – robust test-train splits, produces probabilities not classes
  - Use Probabilities of Class Assignment to Under or Over Predict a Class

Possible Paths to Improve this Project

- Move from Detection in Single Cycles to an Accumulator or N-Cycle Detection
  - Detect persistent faults while mitigating sporadic effects like corona
  - Data should already exist: signal data is device or location tagged
- Perform Classification at the Measurement level instead of Signal Level
  - While not globally true, when a fault is present in one phase, the likelihood of fault present in one or both or the other phases is high – 90.3%.
  - Potentially trades a significant reduction in available training data points for improved classification
- Data augmentation
  - Using discarded sections as additional null target data – further imbalances class distribution
  - A second pass with different slicing and windowing
  - Take signals with known faults, use them to synthesize new signals
- Revisit Recurrent Neural Networks – LSTM
  - Take pre-processed signal, and window around the region dense with peaks to further trim signal length
  - Investigate the use of a compressive sampler to further reduce the time series if needed



References

1. Misak, S., et al. “A Complex Classification Approach of Partial Discharges from Covered Conductors in Real Environment.” *IEEE Transactions on Dielectrics and Electrical Insulation*, vol. 24, no. 2, 2017, pp. 1097–1104., doi:10.1109/tdei.2017.006135.
2. Vantuch, Tomas. *Analysis of Time Series Data*, 2018, dspace.vsb.cz/bitstream/handle/10084/133114/VAN431_FEI_P1807_1801V001_2018.pdf.

My Kaggle kernel and GitHub repos have been set public:

Kaggle Kernel: https://www.kaggle.com/jeffreyegan/vsb-power-line-fault-detection-approach

GitHub: https://github.com/jeffreyegan/VSB_Power_Line_Fault_Detection

Thank you!

Modify Class Weights used in Model Training to Remove Bias

- Set and Tune Class Weights
  - Enables cost-sensitive learning to remove a bias favoring the dominant class
  - Impose a costly penalty on minority class misclassification
  - Initially set weights based on ratios of data points in each class
  - Iterations on tailoring the weights can improve model performance

- Example from this Project

  Default Weights are Uniform:

```
class_weight = dict({0:1.0, 1:1.0})  # default weights
```

  Initial set the weights based on ratio of samples in training data.
  Recall 213 / 8712 signals have faults present thus set fault weight 8499/2134 = 39.90

```
class_weight = dict({0:1.0, 1:39.9})  # initial, adjusted weights
```

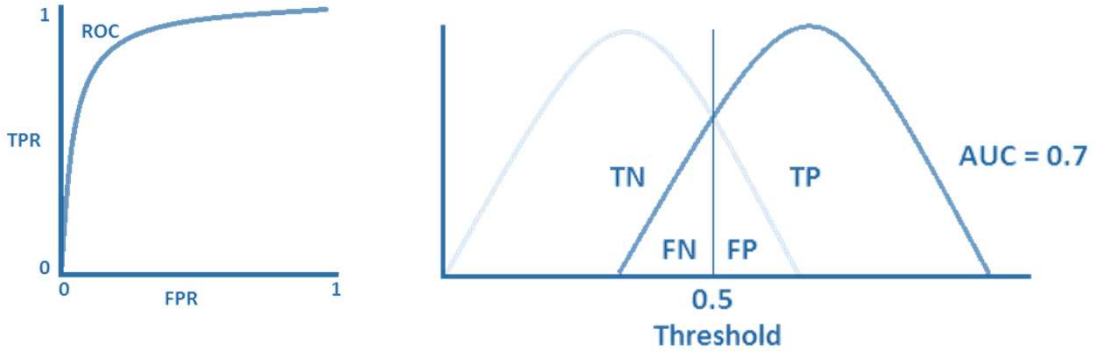
  Iterate and optimize weights through Monte Carlo analysis

```
class_weight = dict({0:0.5, 1:2.0})  # optimized, adjusted weights
```

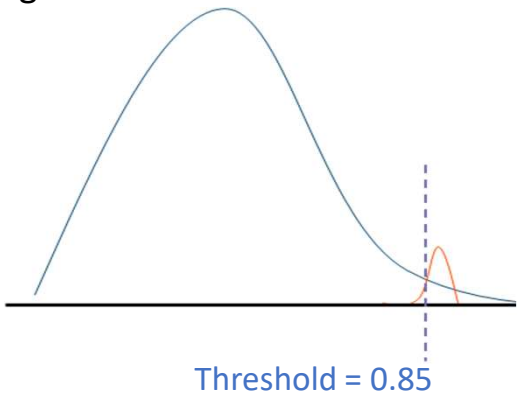
Adjust Thresholds Used for Class Prediction

- Adjust Thresholds on Classification Probabilities to Over/Under Predict a Class
  - Another method to improve minority class performance
  - Default threshold is 0.5 for binary classification,
  - Use sklearn.metrics.roc_curve to find and set more appropriate thresholds
    - `fpr, tpr, thresholds = metrics.roc_curve(y, y_prob, pos_label=1)`
  - In the final Partial Discharge detection model, the threshold is set to 0.85

Example ROC & Threshold with Balanced Classes

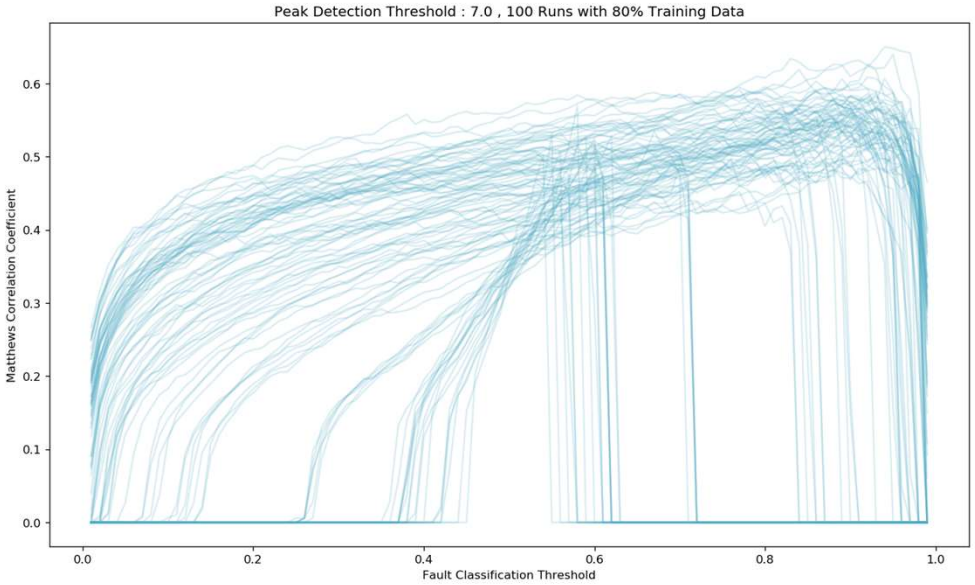


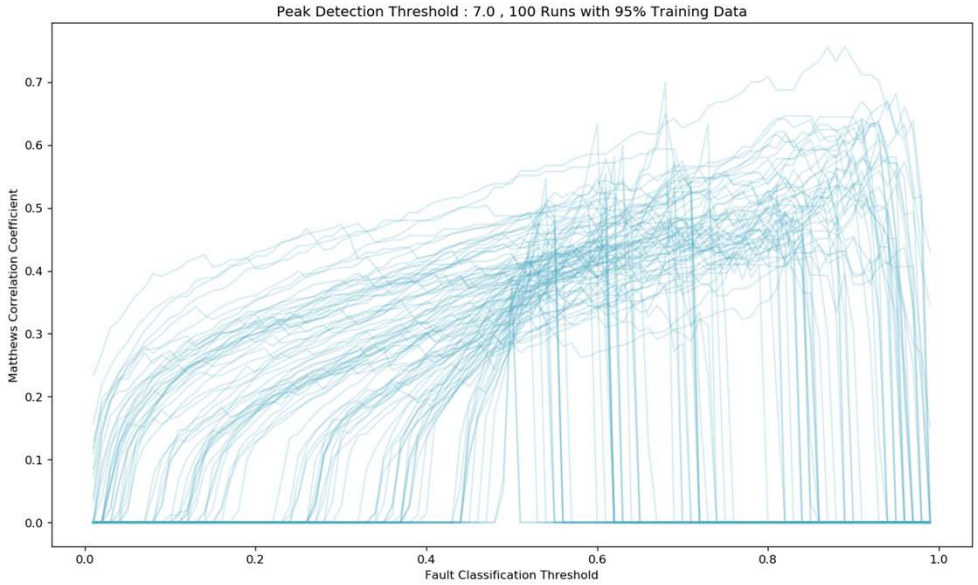
Setting Threshold for Imbalanced Class



Concerns with Training/Test Splits on Unbalanced Data

Perform Monte Carlo trials that vary the `random_state` and `test_size` parameters in sklearn's `train_test_split` function and assess the impact of different thresholds on performance.





The goal is to find and set thresholds for Peak Detection and Fault Classification that routinely yield higher Matthews Correlation Coefficient scores and are robust to varied divisions of the training data.

Revisiting Random Forest – Tuning for Unbalanced Classes

- Use Ensemble k-Fold Cross-Validation
  - Splits the data into k parts, Take k-1 parts as training data and 1 part for validation
  - Train and predict with the model k times over, holding out a different part each time.
  - Ensemble output produces probability of sample belonging to a class, not assignment
  - Average and use the prediction probabilities from the k scores to assign a class
- Set and Tune Class Weights
  - Enables cost-sensitive learning to remove a bias favoring the dominant class
  - Impose a costly penalty on minority class misclassification
  - Initially set weights based on ratios of data points in each class
  - Iterations on tailoring the weights can improve model performance
- Adjust Thresholds on Classification Probabilities to Over/Under Predict a Class
  - Another method to improve minority class performance
  - Adjust probability thresholds for making a classification