Report

February 6, 2019

Machine Learning Engineer Nanodegree

Capstone Project

Victor São Paulo Ruela

1 Definition

1.1 Project Overview

Partial Discharge (PD) signals are electrical discharges that can occur inside the insulation of high voltage equipments. These signals have a repetitive nature and are confined to small regions, which in the long run can lead to irreparable damage to the equipment. Therefore, it is vital for the energy industry companies to monitor the occcurrance of PDs, in order to prevent accidents and guarantee a realiable energy transmission for its customers.

Measuring these signals in the field is already very challenging task, due to its low intensity and high noise levels from high voltage systems. However, this is just one part of the problem: based on the measurements, how can we predict some of its characteristics, such as faults, the level of damage, local of occurrence and possible causes? These are tasks that can be achieve with signal processing techniques and machine learning algorithms.

The dataset that is going to be used is available from the VSB Power Line Fault Detection Kaggle competition website.

It contains several examples of labeled PD signals (fault or undamaged). Each signal contains 800,000 measurements of a power line's voltage, taken over 20 milliseconds for each one of the three phases.

1.2 Problem Statement

The goal is to train a classification algorithm to predict for PD signal measurements if a power line damaged or not. I will be tackling this problem as a binary classification problem, also applying digital signal processing techniques to extract the most relevant features from the PD signals.

The first task will be to apply signal processing techniques in order to remove the background noise from the measurements and obtain a clear representation of the partial discharge signals. This can be done using digital filters (Butterworth, Chebyschev, etc.), the Fourier Transform and the Discrete Wavelet Transform (DWT). After that, new features will be extracted, such as amount of PDs and the frequency-domain content. If necessary, more features can be include based on the thesis from the competition's responsible [1].

For the training models, I pretend to compare binary classification models, such as Logistic Regression and Random Forests. I will work with simpler models in order to have more time available for the feature extraction, since this should be the most important taks on this porject. I expect to spend 70% of the time on the signal processing and feature extraction parts and 30% of the time on training models and tweaking parameters.

1.3 Metrics

The results will be evaluated with the Matthews correlation coefficient (MCC) between the predicted and the observed response:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{1}$$

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. This is a very suitable metric, since the dataset is probably unbalanced because faults are not very frequent.

This is the same metric used in the competition.

2 Analysis

2.1 Data Exploration and Visualization

The dataset is available at https://www.kaggle.com/c/vsb-power-line-fault-detection/data and contains the following files:

a) metadata_[train/test].csv - The signal general information and labels. Contatins the following columns:

- `id_measurement`: the ID code for a trio of signals recorded at the same time.
- `signal_id`: the foreign key for the signal data. Each signal ID is unique across both train and test, so the first ID in train is '0' but the first ID in test is '8712'.
- `phase`: the phase ID code within the signal trio. The phases may or may not all be impacted by a fault on the line.
- `target`: 0 if the power line is undamaged, 1 if there is a fault.

b) [train/test].parquet - The signal data. Each column contains one signal; 800,000 int8 measurements as exported with pyarrow.parquet version 0.11. More information about the parquet format can be found in https://acadgild.com/blog/parquet-file-format-hadoop.

The metadata file is the link between the actual PD signals measurements in the parquet file and its label. For example, to access the signal with ID "1", you just need to load the column "1" from the .parquet file. This is actually a very interesting feature, because it allows to optimize the memory consumption by processing one signal at a time.

The `metadata_train.csv` file contains `8712` rows, each one representing a signal available in the `train.parquet` file. The overall and per phase distribution for the labels are displayed in figure 1.

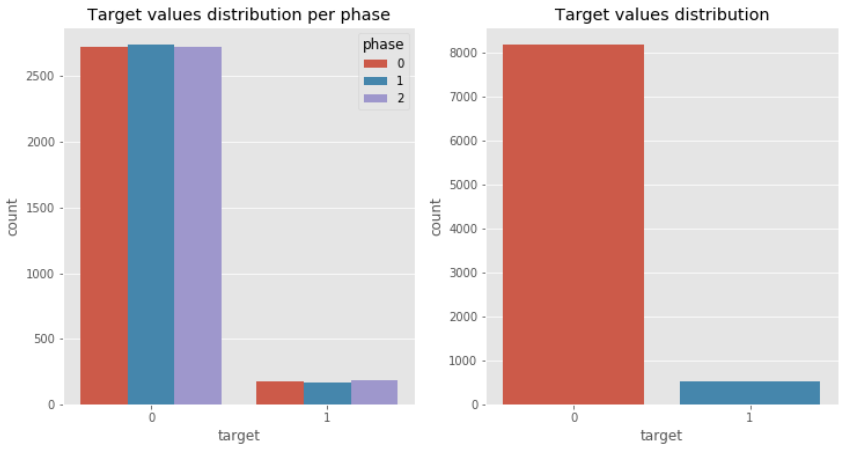

Figure 1: Label distribution

We can clearly see the data is very unbalanced: only `6.0262%` of the data contains examples of fault signals. This was expected, since a fault event is not very common. Looking at the distribution considering the phase of the signal, we can see that the labels are evenly distributed among them. Therefore, phase should not be an input variable to the models, since it won't add any relevante information.

Eah signal in the `train.parquet` contains `800,000` measurements of a power line's voltage, taken over `20` milliseconds. Thefore, the sampling rate is `40Mhz`. A sample signal PD signal for each phase can be seen in the figure 2.

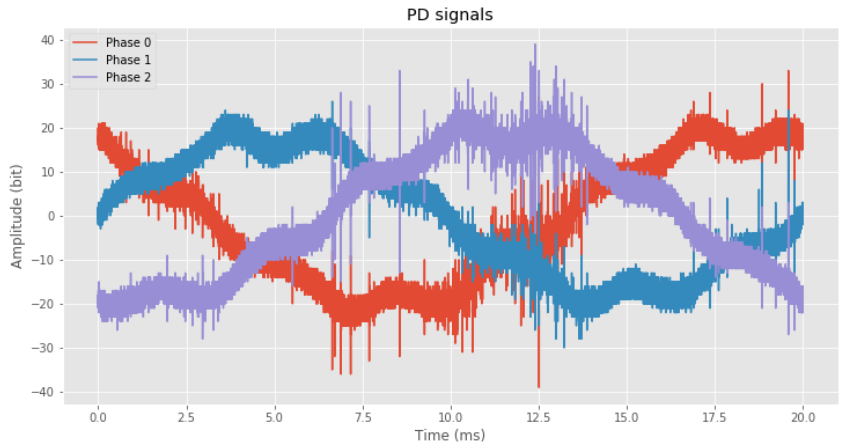

Figure 2: PD Signals for each phase

Figure 3 shows a comparison between a fault and normal example signals.
Based on these examples, the following can be inferred about PD signals:

- There is a lot of background noise, which can be incorrectly identified as a PD pattern. Therefore, it is necessary to denoise this signal ass the first step for the analysis.
- It's not easy to visually observe an actual PD pattern, since it happens very fast.

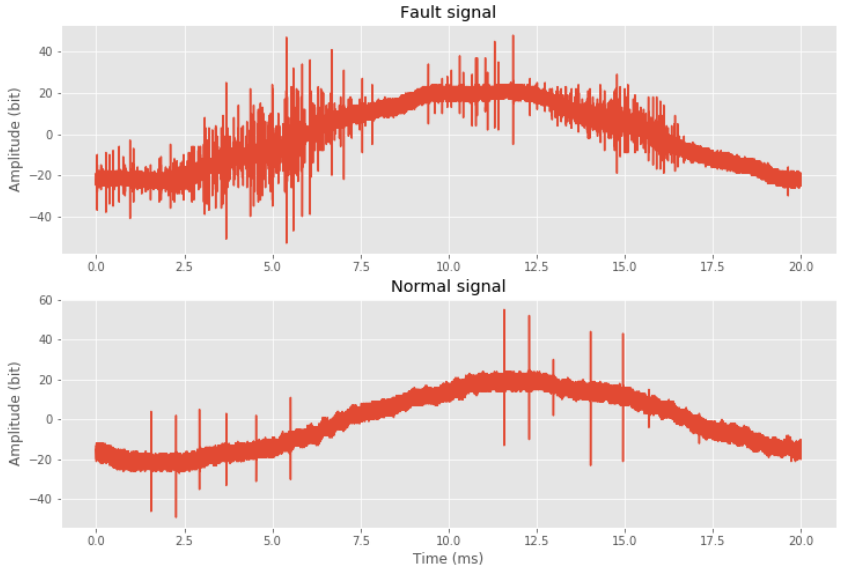

Figure 3: Fault and normal PD Signal

- The start and end of the measuremente cycle is different depending on the phase.

Moreover, the raw time-based data format should not be used for the machine learning algorithms, since this would lead to have `800,000` features as inputs to the model.Therefore, signal processing techniques will have to applied to extract features that can accurately represent the signal.

2.2 Algorithms and Techniques

The following classifiers will be teste and the one with the highest cross-validation score on the dataset will be chosen: Logistic Regression, kNN, Random Forest and Gradient Boosting. They are well known techniques for the binary classification task and are expected to perform well on this dataset, since they do not require very large datasets to work.

Random Forest This is an ensemble learning method that uses decision trees as the weak learners. It is suitable for both classification and regression problems. It works by fitting several decision trees from randomly taken subsets of the data, using the general bagging procedure. Moreover, it also adds more randomness to the model when growing the trees: instead of always searching for the best feauture while splitting a node, it searches for the best feature among a random subset of features. Therefore, it is very flexible and can build very complex decision boundaries, having a good performance in most problems.

Logistic Regression It's a linear model that uses a logistic function to model a binary dependent variable. It will define a linear decision boundary, therefore it is more suitable for linearly separable problems. For example, the logistic function can be defined as the standard sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

This model outputs the probability of the dependent variable from one or more independent variables. Therefore, in order to predict the class, we need to define a threshold and assume that any probability below belongs to class 0 and above to class 1.

kNN This is a very simple, non-parametric (does not make any assumptions on the uderlying data distribution), lazy learning algorithm that can be used for both classification and regression problems. The prediction is done by finding the k closest data points to a new example and assing as the output the most common class among them.

Gradient Boosting This is an ensemble model, typically having decision trees as the weak learners, for regression and classification problems. It is a boosting technique, in which the models are sequentially trained based on the errors from previous iterations. The main idea behind it is that we are inserting new models that can correct the errors from previous ones, by training them on the residuals. This is different from AdaBoost, where we add greater weights to wrong predictions. A more detailed explanation can be found in [6].

The models be evaluated with the default parameters, as described in their documentation page. The chosen model will use as inputs the features extracted from the raw PD signals and the respective labels (see the `Data Preprocessing` section). A grid search will be executed in order to tune its hyperparameters and find the optimal classifier. For example, for the Random Forest, the following hyperparameters can be optimized:

- `n_estimators`: The number of trees in the forest
- `max_features`: The number of features to consider when looking for the best split:
- `max_depth`: The maximum depth of the tree
- `min_samples_split`: The minimum number of samples required to split an internal node
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node

2.3 Benchmark

The benchmark model will be a naive predictor that will output that the power line is always undamaged (false). Based on the data discussion from previous sections, this model will have `93.9738%` accuracy. This model will also have a baseline value of `0.0` for the Mathews Correlation Coefficient.

3 Methodology

3.1 Data Preprocessing

Feature Extraction As discussed in the previous sections, signal processing techniques must be applied in order to extract the most relevant information from the signals. The first step is applying a filter to the signal in order to remove the background noise. An analysis of the most common PD denoising techniques is available in [2]. For this project, a high pass filter and a Discrete Wavelet Transform (DWT) denoising technique will be used.

The high pass filter will remove all frequency content below a certain threshold. This is very important to remove the 50Hz baseline frequency from the power line, and also some low frequency noise. The threshold that is going to be used is 10kHz, since PD patterns are only observed above this value.

The DWT is the most used signal processing technique for PD denoising, having several papers studying its application available in the literature. It works by decomposing the signal in several scales and levels based on a pre-selected mother Wavelet. The framework for PD denoising consists in:

- Apply the DWT to the noisy signal until a decomposition level where its possible to distinguish the DP signal. This will calculate the wavelet coefficients c for each level.
- Chosse an appropriate threshold T_d for selecting the coefficients for each level. This can be done either with soft or hard thresholding.
- Recover the denoised signal using the inverse discrete Wavelet transform from the coefficients selected previously.

A more detailed description regarding the Wavelet denoising framework can be found in [1,3]. For this project, the a hard-thresholding approach from [1] will be used. The limit $T_{d,j}$ is calculated based on the Mean Absolute Deviation (MAD), using the following equations:

$$\sigma = \frac{1}{0.6745} MAD(|c|)$$

$$T_d = \sigma\sqrt{2\log n}$$

where n is the lenght of the signal. The selected mother wavelet will be the `Debauchy 4`, since it was indicated in [2] as the most similar to the PD pattern.

The high pass filter and DWT denoising will be applied for every signal in the database. After this step, we can start extracting new features to represente this signal. An filtered signal example can be seen in figure 4.

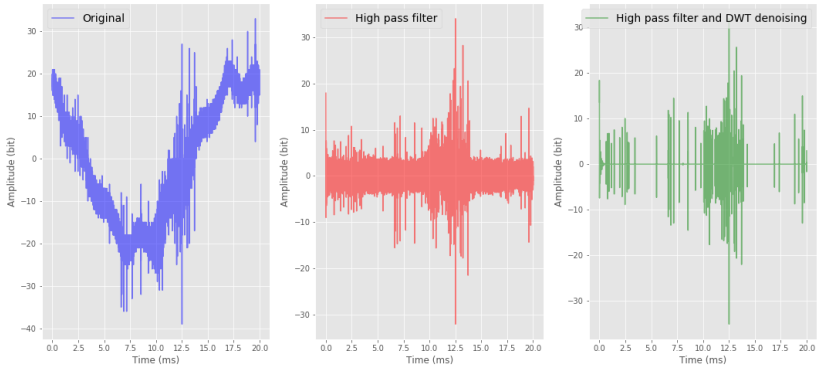

Figure 4: Denoised PD Signal

Afther that, time and frequency domain features will be extracted from the denoised signal. According to [4], the most relevant time domain features are:

- Number of peaks

- Mean width of peaks
- Max width of peaks
- Min width of peaks
- Mean height of peaks
- Max height of peaks
- Min height of peaks

The signal will be divided into 4 sections of size `200,000` and these features will be extracted for each subset, as suggested in [4]. This is done because the appearance of peaks are not random, i.e, they are clustered in specific subparts of the sinusoidal signal. This yields a total of 28 features extracted.

In order to extract these features, the functions find_peaks and peak_widths from the `scipy` package will be used. It was necessary to fine tune the input parameters for both functions, and after several trials the following parameters were selected:

1. **find_peaks**
    - prominence: `10` (The prominence of a peak measures how much a peak stands out from the surrounding baseline of the signal and is defined as the vertical distance between the peak and its lowest contour line.)
    - distance: `50` (Required minimal horizontal distance in samples between neighbouring peaks)
2. **peak_widths**
    - rel_height: `0.9` (Relative height at which the peak width is measured as a percentage of its prominence)

In figures 5 and 6 it is possible to see the peaks from an example signal and also have a closer look at the width and height of a peak.

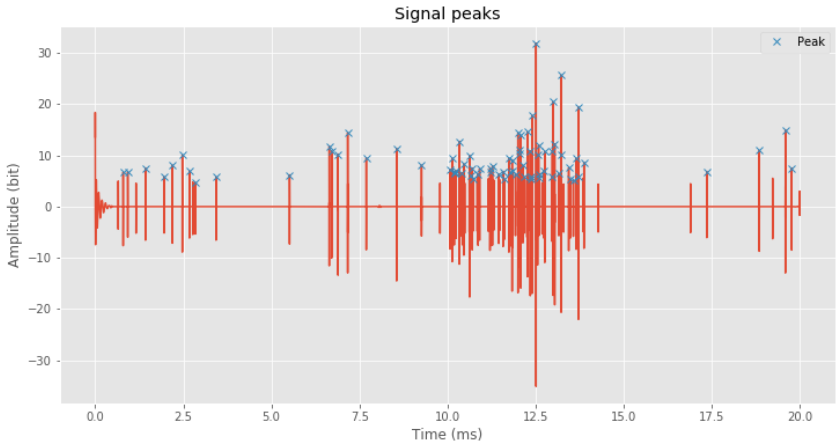

Figure 5: PD signal selected peaks

The frequency domain features are based on the power spectral density (PSD) of each denoised signal. The PSD will not be calculated based on the raw signal because the noise contamination

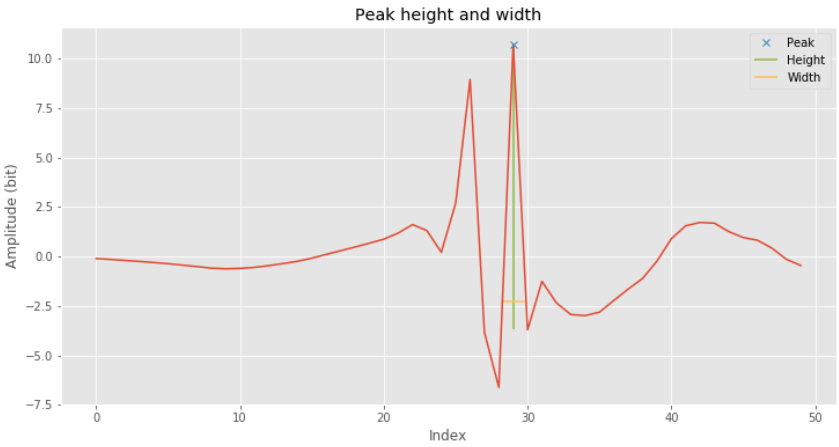

Figure 6: PD signal peak height and width

in higher frequencies will introduce undesired bias to the data. The PSD will be split into 4 equals sections of 5 MHz (Nyquist frequency divided by 4), such that we can extract the following features from the signal at different frequency ranges:

- Sum of power spectrum
- Maximum value
- Mean value

This will be calculated with the welch function from the `scipy` package. Welch's method calculates an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms. An example for a faulted and normal signal can be seen in figure 7. This task yields more 16 extracted features.

After applying these two feature extraction, we finally have the dataset that is going to be used for training our prediction models. This dataset has 40 numerical features for each signal and their corresponding labels.

3.2 Feature Transformation

The dataset has features with very different magnitudes (e.g: number of peaks and peak width), therefore the dataset will first be normalized to the 0-1 range. Figure 8 shows the distribution of each feature after the normalization.

It can be seen that almost all features distribution are skewed, which means the data does not follow a normal distribution. Therefore, the natural logarithm transformation should be applied to the dataset.

3.3 Feature Selection

The next pre-processing step should be the feature selection using a tree-based approach. The idea behind this is that in every node in a decision tree is a condition is tested on a single feature, which is called the impurity. Thus, when training a tree, we can calculate how much each feature

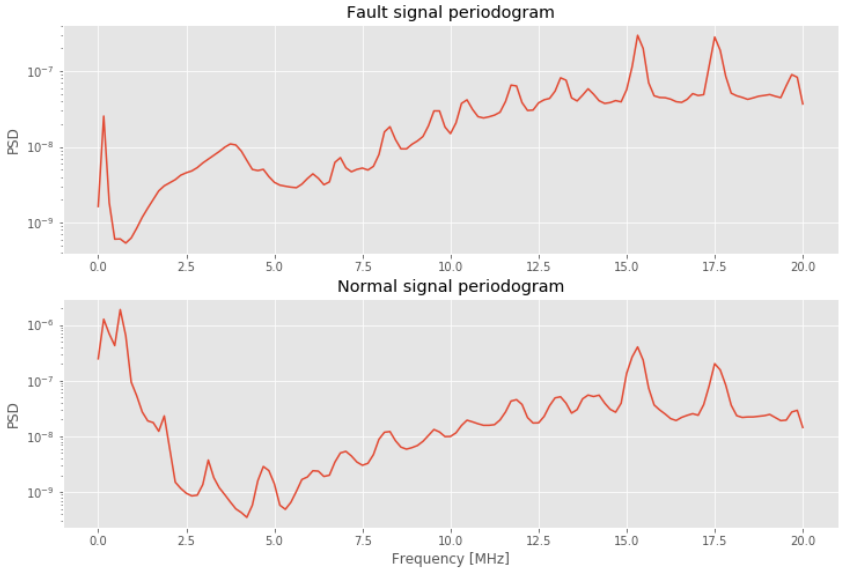

Figure 7: Power spectral density comparison

decreases the impurity in a tree. Finally, if we build an ensemble of trees, we can average these values on different subsets of the data and estimate the importance of the variables. For this task, the ExtraTreeClassifier model will be used with the default parameters.

After applying this feature selection algorithm to the dataset, the importance of each feature can be seen in figure 9.

Based on this result, the following features will be selected for training the prediction models: `number_of_peaks_p1`,`number_of_peaks_p2`, `number_of_peaks_p3`,`number_of_peaks_p4`, `max_height_of_peaks_p1`,`max_height_of_peaks_p2`,`max_height_of_peaks_p4`,`max_height_of_peaks_p3`, `sum_spectrum_p1`,`max_spectrum_p1`, `mean_spectrum_p1`

3.4 Implementation

The implementation process starts by extracting the features for each signal in the `train.parquet` file, as described before. Each signal needs to be denoised and then the features can be extracted. The signal denoising functions are in the `signal_denoising.py` file and the feature extraction in the `feature_extraction.py` file. This is the most time-consuming task, since each signal took around 0.5 seconds to be loaded and processed, thefore taking up to 1.2 hours to process all the signals in the training dataset (8712). This was done in the `execute_signal_processing.py` file, which will generate the output file `all_features.csv` with the extracted features. Next, this file is loaded and the feature scaling, transformation and selection are executed. This task, and the model learning are available in the `data_processing_and_modeling` Jupyter notebook.

The remaining process is very straightforward, and consists in the following steps:

- Load the dataset with the features already extracted transformed and selected into memory
- Split the dataset into training (80%) and test (20%) sets

- Evaluate several models using 10 fold cross-validation on the training set
- Select the trained model with the highest MCC cross validation score
- Train the selected model on training set
- Use grid search to select the optimal hyperparameters
- Evaluate the model MCC score on the test set

During the implementation, the following challenges had to be tackled in order to produce the final results:

- Selecting the peak `prominence` threshold needed a very carefull analysis: low values could lead to noise being classified as a peak and high values would ignore most of the peaks. This needed to be done by a trial and error approach until a good value was found.
- Selecting the `peak_widths` parameters was also tricky: low values could classify noise as peaks and high values would ignore all peaks. Since the PD pulse is very fast, this variable was very diffficult to be defined such that only the peaks of interest were detected. This also needed to be done by a trial and error approach.
- The data processing routine takes too long time. Every time a mistake was done in the parquet processing code, I had to start over and wait up to an hour to finish the processing. In order to improve this, I tried to optimize the code using mulithreaded processing, but it became quite complex and didn't work as expected.
- After finishing the feature extraction, having a first look at the data showed that the model seemed impossible to be solved, because there was no clear class separation. After selecting the features and applying the necessary transformations, the data started to make more sense and a good model was learnt.
- Setting the `random_seed` parameters for the models was important to make sure the results were reproducible.

3.5 Refinement

As mentioned in the Benchmark section, the naive predictor achieved a poor MCC score of `0.0`. To get an initial result, a few models using default hyperparameters were tested agains the dataset: kNN, Gradient Boosting, Random Forest and Logistic Regression. The results can be seen in figure 10.

We can see that all the proposed models achieved a better score than our benchmark. Since the Random Forest model achieved a high score with only 10 trees in the forest, it will be selected as the model to be optimized. The grid search algorithm will be executed on the following hyperparameters configuration:

- `n_estimators`: [10, 30, 50, 100, 150, 200, 300]
- `max_depth`: [10, 30, 50, 70, 100, None]
- `min_samples_split`: [2, 5]
- `min_samples_leaf`: [1, 2, 4]

After executing the Grid Search for every combination of these parameters, the optimal model achieved a MMC score of `0.9639` on the training set and `0.5691` on the test set.

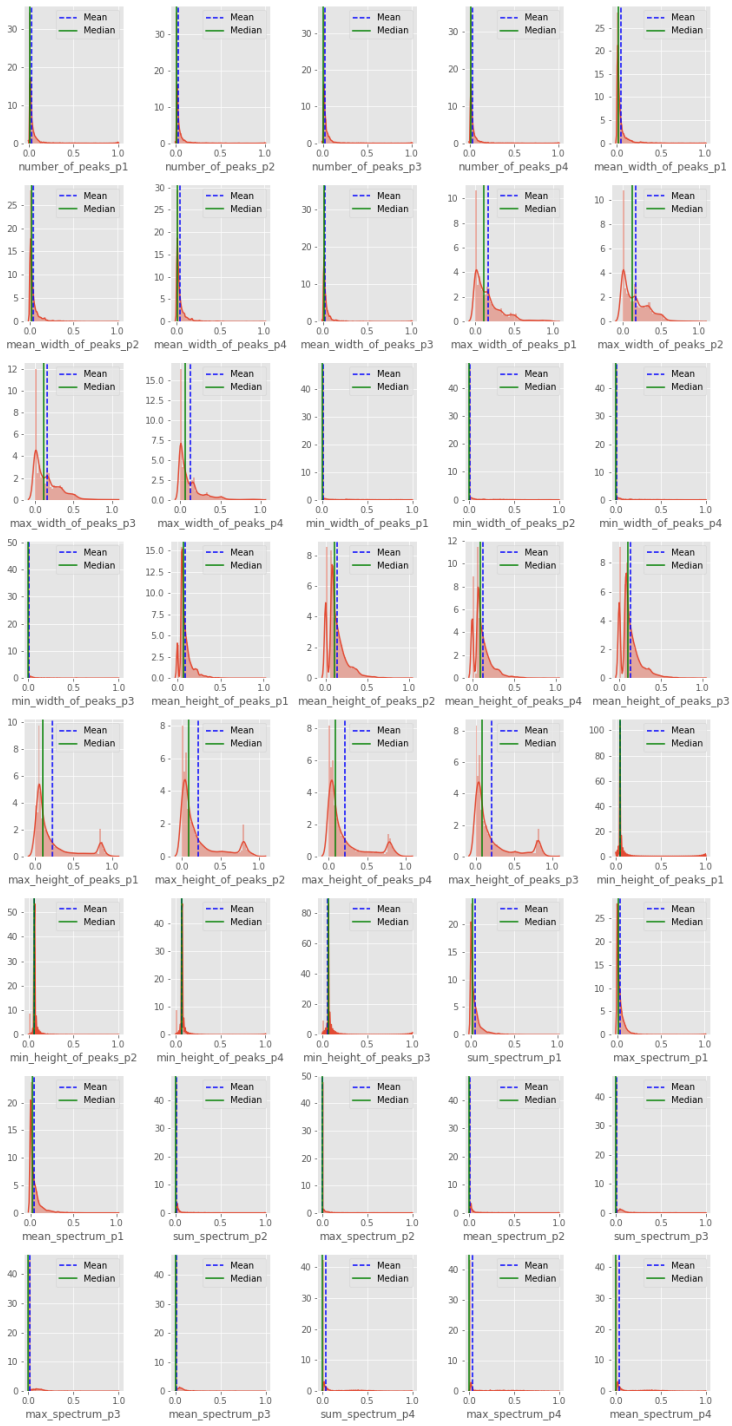

Figure 8: Scaled Features

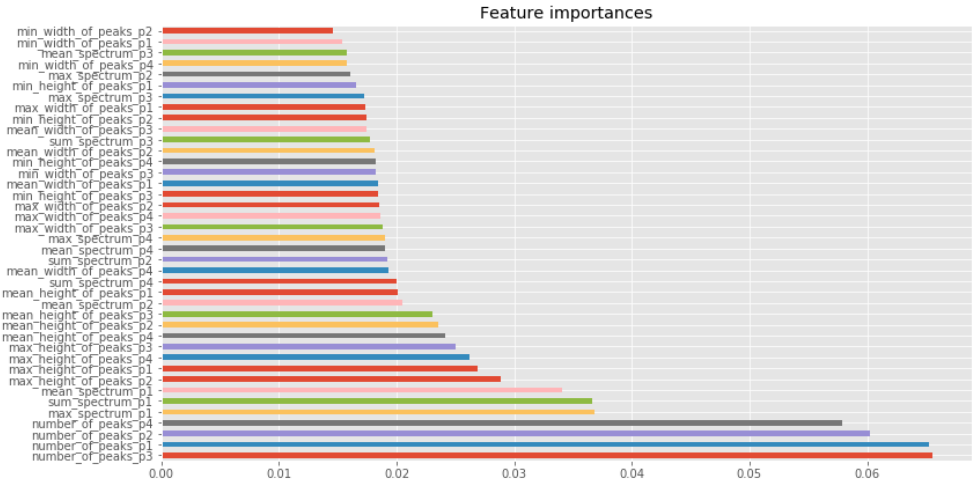

Figure 9: Feature Importance

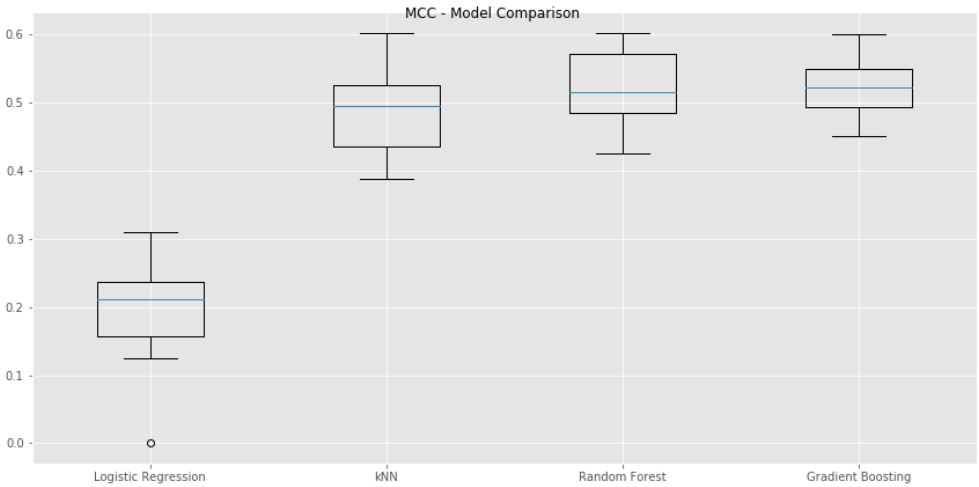

Figure 10: Model Cross-validation results

4 Results

4.1 Model Evaluation and Validation

The final optimal hyperparameters for the random forest classifier are summarized below:

- `n_estimators`: 150
- `max_depth`: 30
- `min_samples_split`: 5
- `min_samples_leaf`: 1

The model was chosen because it achieved the highest MCC score in the training data. The hyperparameters are also very reasonable, not having any very extreme values selected. To verify the robustness of the final model, the test data was evaluated, achieving a MCC score was `0.5691`, which is a good value. However, it suggests that there was some overfitting on the training data, since the training score was `0.9639`.

Figure 11 shows the results of the grid search algorithm. It is possible to see that the model achieved a higher score variation on the training than on the test set. This means that the model is very robust to predict unseen data, because even the models that achieved the worst training score had a good performance on the test set.

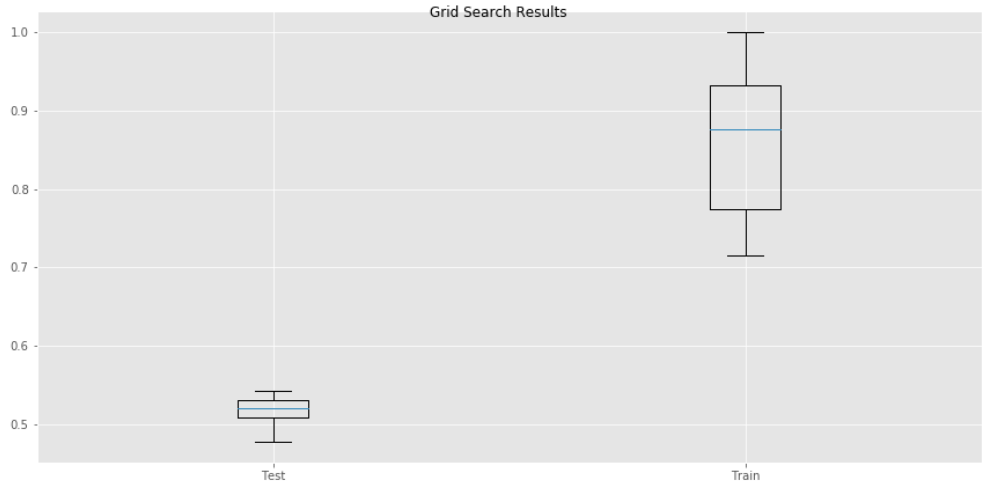

Figure 11: Grid search results for the test and training set

4.2 Justification

Comparing to the benchmark model, the final solution was much better. The test MCC score values was increased from `0` to `0.6`, achieving the objective of this project. Figure 12 displays the confusion matrix of the test set evaluations.

The analysis of the confusion matrix shows that the model did a very good job classifying the normal signals. Its biggest challenge was to be able to correclty classify the faulted signals: only 48% of the examples were correctly classified as faulted, which was the reason the MMC score was not very high. This could be an effect of the very unbalanced dataset.

The model was also evaluated against Kaggle's test set, achieving a score of `0.469`. This placed me among the top 500 competitors!

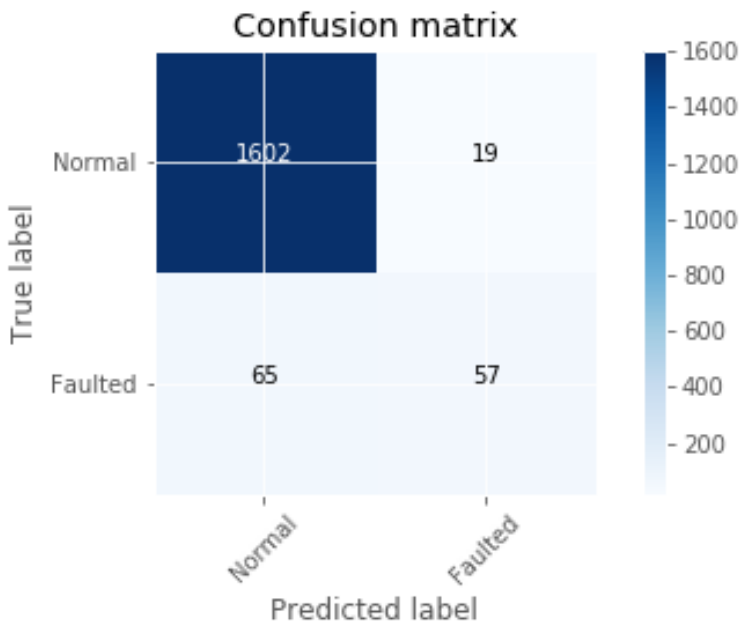

Figure 12: Confusion Matrix

5 Conclusion

5.1 Free-Form Visualization

The biggest challenge of this project is to find a representation of the PD signal that is good enough to separate between a faulted and normal signal. Figure 13 shows an example of signals classified as faulted and normal.

Visually, it is pratically impossible to classify each signal. In fact, the faulted signal has less peaks and is less noisy than the regular one! However, after applying feature extraction and data pre-processing techniques, it was possible to find relevant features that are able to separate them. This also summarizes a very interesting and exciting characteristic of this dataset: how a signal having 800,000 samples can be represented by only 11 features and be correclty classified only with this information. Furthermore, it higlights the importance of the data pre-processing in any data science project.

5.2 Reflection

The process used for this project can be summarized using the following steps:

1. An interesting and relevant problem with public data sets was found
2. The data was downloaded, the signals denoised and the features extracted
3. A benchmark was created for the classifier
4. Several classifiers were evaluated and the best one selected
5. The selected classifier was trained using the available train data, using grid search to optimize the hyperparameters
6. The test data was evaluated against the optimal classifier
7. The model predictions were submitted to the Kaggle website

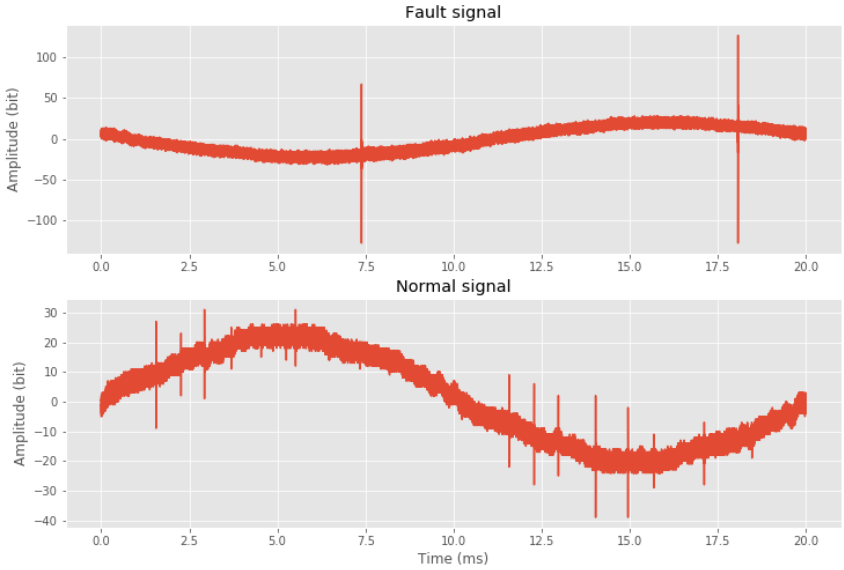

Figure 13: Fault and normal signals

Step 2 was the most difficult and time-consuming task, because in order to correctly define the denoising and feature extraction techniques I had to study the available literature on PD signal processing. Since I already had a good background on the problem domain, I knew where to seek the necessary information and this became a very straight-forward task, even though being quite complex. I also decided to spend more time studying the feature selection and transformation, in order to polish my skills on these tasks and also produce a final dataset with better quality. In fact, this was the most interesting and important step in the project, since I could effectively see the impact of a good data preparation step on the model learning.

Steps 4 and 5 took less time than expected, since my choice of features proved to be good and the models achieved better results than the benchmark on the first trials.

5.3 Improvement

To achive better results, I believe the first step is trying to improve the quality of the PD denoising and feature extraction algorithms. In [1], the author presents a series of signal representation techniques that can be used to extracted more relevant features from the PD signals. There are also a very large literature on PD denoising techniques, which could also be explored. Due to their complexity and the project's time constraints, I decided to only use the denoising and feature extraction techniques I was most familiar with.

As for the models improvements, I believe it is worth exploring more the Gradient Boosting algorithm, since it also achieved good results on the initial trial and also works very well in practice. This could improve the overfitting observed on the training dataset and improve the current benchmark with random forests.

6 References

[1] http://dspace.vsb.cz/bitstream/handle/10084/133114/VAN431_FEI_P1807_1801V001_2018.pdf

[2] S. Sriram, S. Nitin, K.M.M. Prabhu, and M.J. Bastiaans. Signal denoising techniques for partial discharge measurements. IEEE Transactions on Dielectrics and Electrical Insulation, 12(6):1182–1191, 2005

[3] Ma, X., Zhou, C. and Kemp, I.J., 2002. Interpretation of wavelet analysis and its application in partial discharge detection. IEEE Transactions on Dielectrics and Electrical Insulation, 9(3), pp.446-457.

[4] S. Misák, J. Fulnecek, T. Vantuch, T. Buriánek and T. Jezowicz, "A complex classification approach of partial discharges from covered conductors in real environment," in IEEE Transactions on Dielectrics and Electrical Insulation, vol. 24, no. 2, pp. 1097-1104, April 2017.

[5] Liaw, A. and Wiener, M., 2002. Classification and regression by randomForest. R news, 2(3), pp.18-22.

[6] Friedman, J.H., 2001. Greedy function approximation: a gradient boosting machine. Annals of statistics, pp.1189-1232.