Final Project - Power Line Partial Discharge Detection

Adrian Cross, Xuesong Fan, Aaron Wilson

December 13, 2019

Machine learning
COSC 522

Abstract

The phenomenon of partial discharge is a significant problem for medium voltage power lines, causing degradation of power lines leading to fires and power outages. This project attempts to distinguish signals, taken by a new meter designed at VSB, between partial discharges and normal waveforms. This is done by a variety of machine learning techniques giving an optimum result, using Mathews correlation coefficient, of 0.55 from using a back-propagating neural network. This compares to a value of 0.72 from the Kaggle leaderboards.

Contents

1 Introduction

1.1 Problem

Written by Adrian Cross

The problem worked on in this project is from the Kaggle website [2]. It involves the use of machine learning techniques, learned in the COSC 522 course, to detect partial discharge patterns in signals acquired from medium voltage overhead power lines. This detection is done by using a new meter designed at the ENET Centre at VSB. This is important research as these power lines run for hundreds of miles and are hard to manually inspect for damage that doesn't immediately lead to a power outage, such as a tree hitting the line or a flaw in the insulator. This type of damage leads to partial discharge phenomenon, which is where an electrical discharge which does not completely bridge the electrodes between an insulation system. These partial discharges cause slow degradation of the power line which, if left undetected, can eventually lead to a power outage or start a fire. Therefore by detecting these partial discharges early it can prevent the damage to the line in addition to reducing the cost for manually inspecting the power line.

In this project the signals are transformed into analyzable features, which are then put through a variety of machine learning techniques. Via the use of the MCC performance metric, discussed further in section 1.2, the performance of each technique can be compared with each other, as well as other techniques posted on Kaggle, for the best detection of whether partial discharges occur.

1.2 Performance Metric

Written by Adrian Cross

The performance metric used for this project is Matthews Correlation Coefficient (MCC), shown in equation 1.

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

Where TP is the number of true positives, TN is the number of true negative, FP is the number of false positives and FN is the number of false negatives. This is a binary class performance metric which attempts to quantify the quality of the classification performed depending on its confusion matrix [1]. The metric runs on a scale from -1 to +1 where -1 corresponds to a completely wrong prediction, +1 is a completely correct prediction and 0 is a completely random prediction. The MCC value for the machine learning techniques outlines in this project are compared with each other in section 12.2 and are compared with other Kaggle projects in section 12.4.

2 Preprocessing

2.1 Input Data

Written by Aaron Wilson

The input data consists of both a training and testing set; however the testing set is unlabeled. The training set contains 8712 examples, each of which is comprised of 800,000 data points. Fig. 1 shows examples of both a "normal" waveform and a waveform containing partial discharge patterns. The input data was collected by VŠB - Technical University of Ostrava, located in the Czech Republic. Each training example contains 1 50-Hz power cycle, sampled at 40 Megasamples per second (Msps).
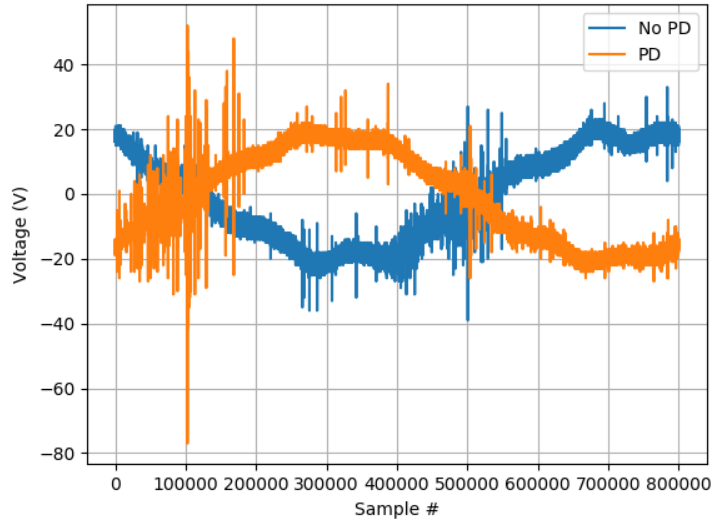
Figure 1: Examples of PD and Non-PD Waveforms

Prior to using the data set to train the various clasisfiers, a number of preprocessing steps needed to be performed. Each waveform (training example) contains 800,000 data points; hence attempting to process the data set in its original form would have been much too computationally intensive.

Both preprocessing steps were taken from [4]. They include applying a high-pass filter to remove the dominant low-frequency sinusoidal shape, and applying the Discrete Wavelet Transform (DWT) to remove the noise. Fig. 2 shows an example waveform that has gone through this process.
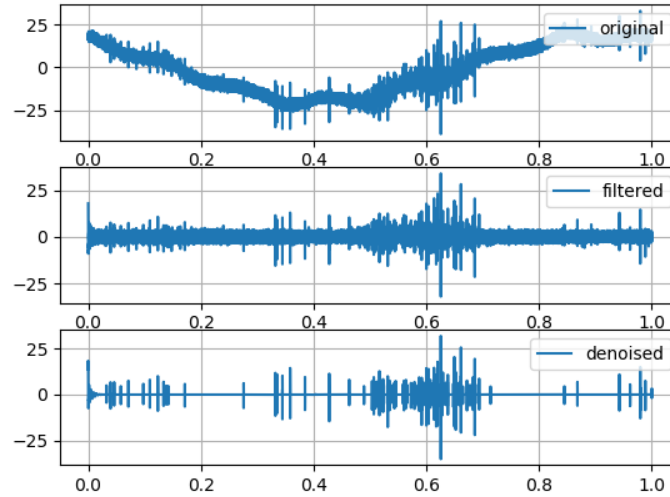

Figure 2: Denoising Process

2.2 High-Pass Filter

Written by Aaron Wilson

A 10th-order high-pass Butterworth filter with a low cutoff frequency of 10 kHz was first applied to the raw signal data. The filter is designed knowing that the original signals were sampled at a rate of 40 mega-samples per second

(Msps). The reasoning behind using this high-pass filter is to remove the fundamental frequency shape (50 Hz), therefore removing any phase offset between different training examples, as can be seen in Fig. 1.

2.3 Denoising via the Discrete Wavelet Transform (DWT)

Written by Aaron Wilson

A popular method of removing noise from a digital signal is by using the discrete wavelet transform (DWT). The DWT breaks down a signal into a series of "detail" and "approximation" coefficients by applying both a high-pass and low-pass filter, respectively, known as the "mother wavelet". For this work, the Daubechies 4 (known as "db4") wavelet was chosen. After the signal has passed through both filters, it is downsampled by a factor of 2. The process is then repeated on the new set of approximation coefficients, Fig. 3.
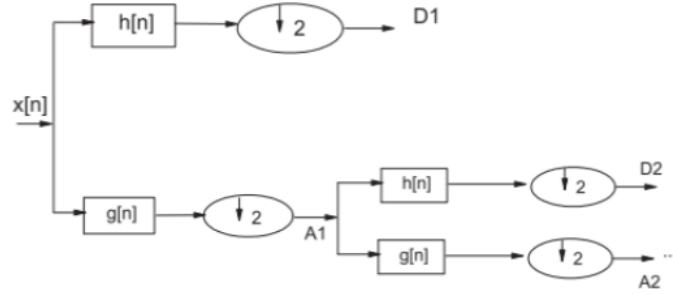

Figure 3: DWT Process [4]

In this work, the DWT was only performed once, i.e. to receive only a single set of detail coefficients and approximation coefficients. To "denoise" the signal, a threshold was applied to the obtained set of detail coefficients, c_d, determined by [4]:

$$T_{\text{hard}}(x) = \left\{ \begin{array}{ll} x & \text{if } |x| > \lambda \\ 0 & \text{otherwise} \end{array} \right. \tag{2}$$

where x represents the detail coefficients c_d, λ represents the threshold calculated by

$$\lambda = \frac{1}{0.06745 \times \text{MAD}(|c_d|)} \sqrt{2\log(n)}, \tag{3}$$

$\text{MAD}(\cdot)$ represents the *mean absolute deviation* calculation, and n represents the length of the input signal to the DWT. The resulting "denoised" signal is obtained by reversing the DWT process and creating a reconstructed signal with $T_{hard}(x)$ as the initial input.

2.4 Feature Extraction

Written by Aaron Wilson

From the denoised signal, twelve features were extracted. A table showing each of the features with a quick description is given in Table 1.

Table 1: Features Extracted from Denoised Signals

Feature	Description
Signal Mean	Mean of all 800,000 data points in denoised signal
Signal Standard Deviation	Standard deviation of denoised signal
Signal Skewness	How skewed the signal data points are with respect to a normal distribution
Signal Kurtosis	A measure of how prominent the signal distributions "tails" are
Number of Positive Peaks	A count of the total number of peaks greater than zero
Number of Negative Peaks	A count of the total number of peaks less than zero
Mean Peak Width	Mean value of peak widths
Mean Peak Height	Mean value of peak heights
Max Peak Width	Max value of peak widths
Max Peak Height	Max value of peak heights
Min Peak Width	Min value of peak widths
Min Peak Height	Min value of peak heights

The work performed in [4] shows a total of 15 features. Four of these features (signal entropy, detail coefficient entropy, approximation coefficient entropy, and fractal dimension) required extra knowledge to compute as well as possessed a very low "weight" factor according to the author, so it was decided to discard these features. Additionally, the author of [4] had one designated feature for "number of peaks"; in our work, this feature was broken down into two separate feature: number of positive peaks and number of negative peaks.

3 m-Fold Cross Validation

Written by Aaron Wilson
For all training and classification procedures performed in this work, m-fold cross validation is performed on the training set with $m = 10$. The training data is first randomly shuffled to prevent potential biasing, and broken into the m consecutive sub-groups. $m - 1 = 9$ of these groups were used for training, and the left-out group was used for testing.

4 Normalization and Dimensionality Reduction

4.1 Data Normalization

Written by Xuesong Fan
The training data set used in this project include 12 features (columns) as mentioned before. The last column (13th), indicates whether there is a fault on the power line, i.e., label. The data in other columns were normalized to make the features comparable, following the equation below:

$$nx_i = (x_i - \mu_i) / \sigma_i \tag{4}$$

where x_i is the sample in the dataset X with feature i, μ_i is the mean of feature i, σ_i is the standard deviation of feature i, and nx_i is the normalized sample in the normalized dataset nX with feature i.

4.2 Principal Component Analysis (PCA)

Written by Xuesong Fan
The PCA was used for the dimensionality reduction process on the normalized data sets. Covariance of the normalized data was calculated as:

$$\Sigma = \text{cov}(nX) = \overrightarrow{nX} \cdot \overrightarrow{nX}^T \tag{5}$$

where Σ is the covariance matrix of the normalized data set nX. The eigenvalues and eigenvectors of the covariance matrix were then calculated as:

$$[E, \lambda] = \text{eig}(\Sigma) \tag{6}$$

where $\vec{E} = [\vec{e_1}, \vec{e_1}, \ldots, \vec{e_d}]$ represents the eigenvectors and $\vec{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_d]$ represents the eigenvalues of the covariance matrix. Then a subset of the eigenvalues (major axes) were chosen such that the error rate is not greater than 0.10. The error rate could be calculated following:

$$\text{error rate} = \frac{\text{sum of discarded eigenvalues}}{\text{sum of all eigenvalues}} \tag{7}$$

The corresponding chosen eigenvectors could then be represented as $\vec{P} = [\vec{e_1}, \vec{e_1}, \ldots, \vec{e_m}]$, and the data for the reduced dimension can be calculated as:

$$\vec{pX} = \left(\vec{nX}^T \cdot \vec{P}\right)^T = \vec{P}^T \cdot \vec{nX} \tag{8}$$

To keep the error rate not greater than 10%, 8 out of the 12 dimensions were kept with the corresponding error rate of 7.60%. The data set was then denoted as pX.

4.3 Fisher's Linear Discriminant (FLD)

Written by Xuesong Fan
The FLD method was used to generate the projection that best separates the projected data. For a two-class case, the within-class scatter matrix S_w can be calculated as:

$$\begin{aligned}\vec{S_W} &= \vec{S_1} + \vec{S_2} \\ &= \sum_t (nx_i - \mu_1) \cdot (nx_i - \mu_1)^T + \sum_t^{n_2} (nx_i - \mu_2) \cdot (nx_i - \mu_2)^T\end{aligned} \tag{9}$$

where n_1 and n_2 are the number of samples in each class, μ_1 and μ_2 are the mean values of each class. Since the covariance of the data sets can be calculated as:

$$\Sigma = \text{cov}(X) = \frac{1}{n-1} \sum_i^n (\vec{x_i} - \vec{\mu}) \cdot (\vec{x_i} - \vec{\mu})^T \tag{10}$$

The scatter matrix can then be presented as:

$$\vec{S_W} = (n_1 - 1)\text{cov}(\vec{nX_1}) + (n_2 - 1)\text{cov}(\vec{nX_2}) \tag{11}$$

Then the Canonical variate can be derived as:

$$\vec{w} = \vec{S_W}^{-1}(\mu_1 - \mu_2) \tag{12}$$

The projected data fX then could be calculated as:

$$\vec{fX} = \vec{w}^T \cdot \vec{nX} \tag{13}$$

5 Maximum Posteriori Probability (MPP)

Written by Xuesong Fan
The decision rules were derived using maximum a-posteriori probability, which can be expressed as:

$$P(\omega_i|\vec{x}) = \frac{p(\vec{x}|\omega_i) P(\omega_i)}{p(\vec{x})} \tag{14}$$

The discriminant function $g_i(x)$ is:

$$g_i(\vec{x}) = P(\omega_i|\vec{x}) \tag{15}$$

Since the normalization constant is fixed for a give x, then

$$g_i(\vec{x}) = p(\vec{x}|\omega_i) P(\omega_i) \tag{16}$$

Take the logarithm on both sides,

$$g_i(\vec{x}) = \ln p(\vec{x}|\omega_i) + \ln P(\omega_i) \tag{17}$$

Since the multivariate normal density is

$$p\left(\vec{x}|\omega_i\right) = \frac{1}{(2\pi)^{d/2}\left|\sum_i\right|^{1/2}} \exp\left[-\frac{1}{2}\left(\vec{x}-\vec{\mu}_i\right)^T \sum_i^{-1}\left(\vec{x}-\vec{\mu}_i\right)\right] \tag{18}$$

The discriminant function can be expressed as:

$$\begin{aligned} g_i(\vec{x}) &= \ln p\left(\vec{x}|\omega_i\right) + \ln P\left(\omega_i\right) \\ &= -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}\left(\vec{x}-\vec{\mu}_i\right)^T \Sigma_i^{-1}\left(\vec{x}-\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \end{aligned} \tag{19}$$

Since $-\frac{d}{2}\ln(2\pi)$ is a constant,

$$g_i(\vec{x}) = -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}\left(\vec{x}-\vec{\mu}_i\right)^T \sum_i^{-1}\left(\vec{x}-\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \tag{20}$$

5.1 Case 1

Given $\Sigma_i = \sigma^2 I$, the $-\frac{1}{2}\ln|\Sigma_i|$ term would be constant, then the discriminant function can be simplified as:

$$\begin{aligned} g_i(\vec{x}) &= -\frac{1}{2\sigma^2}\left(\vec{x}-\vec{\mu}_i\right)^T\left(\vec{x}-\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \\ &= -\frac{1}{2\sigma^2}\left(\vec{x}^T\vec{x} - 2\vec{\mu}_i^T\vec{x} + \vec{\mu}_i^T\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \end{aligned} \tag{21}$$

Since $\vec{x}^T\vec{x}$ is constant for a give $\vec{x}$, then

$$g_i(\vec{x}) = \frac{\vec{\mu}_i^T\vec{x}}{\sigma^2} - \frac{\vec{\mu}_i^T\vec{\mu}_i}{2\sigma^2} + \ln P\left(\omega_i\right) \tag{22}$$

Then, $\vec{x}$ would belong to class ω_i if $g_i(\vec{x}) > g_j(\vec{x})$; otherwise, it belongs to class ω_j.

5.2 Case 2

Given $\Sigma_i = \sigma^2 I$, the $-\frac{1}{2}\ln|\Sigma_i|$ term would be constant, then the discriminant function can be simplified as:

$$\begin{aligned} g_i(\vec{x}) &= -\frac{1}{2}\left(\vec{x}-\vec{\mu}_i\right)^T \Sigma^{-1}\left(\vec{x}-\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \\ &= -\frac{1}{2}\left(\vec{x}^T\Sigma^{-1}\vec{x} - 2\vec{\mu}_i^T\Sigma^{-1}\vec{x} + \vec{\mu}_i^T\Sigma^{-1}\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \end{aligned} \tag{23}$$

Since $\vec{x}^T\vec{x}$ is constant for a give $\vec{x}$, then

$$g_i(\vec{x}) = \vec{\mu}_i^T\Sigma^{-1}\vec{x} - \frac{1}{2}\vec{\mu}_i^T\Sigma^{-1}\vec{\mu}_i + \ln P\left(\omega_i\right) \tag{24}$$

Then, $\vec{x}$ would belong to class ω_i if $g_i(\vec{x}) > g_j(\vec{x})$; otherwise, it belongs to class ω_j.

5.3 Case 3

The discriminant function can be calculated as:

$$\begin{aligned} g_i(\vec{x}) &= -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}\left(\vec{x}-\vec{\mu}_i\right)^T \Sigma_i^{-1}\left(\vec{x}-\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \\ &= -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}\left(\vec{x}^T\Sigma_i^{-1}\vec{x} - 2\vec{\mu}_i^T\Sigma_i^{-1}\vec{x} + \vec{\mu}_i^T\Sigma_i^{-1}\vec{\mu}_i\right) + \ln P\left(\omega_i\right) \\ &= -\frac{1}{2}\vec{x}^T\Sigma_i^{-1}\vec{x} + \vec{\mu}_i^T\Sigma_i^{-1}\vec{x} - \frac{1}{2}\vec{\mu}_i^T\Sigma_i^{-1}\vec{\mu}_i - \frac{1}{2}\ln|\Sigma_i| + \ln P\left(\omega_i\right) \end{aligned} \tag{25}$$

Then, $\vec{x}$ would belong to class ω_i if $g_i(\vec{x}) > g_j(\vec{x})$; otherwise, it belongs to class ω_j.

6 Closest near neighbour approach (kNN)

Written by Adrian Cross

kNN stands for the closest nearest neighbour, non parametric approach to classifying data into classes. This is simply done by counting what training points are closest to the point being tested. If more training points close to the testing point belong to a specific class then that testing point also belongs to that class, weighted by the prior probability of the training points. The k number of points which are counted are determined by the k value inputted. Therefore if $k = 1$ you count the training data point closest to the testing point and consider the testing point to also be part of the same class, as shown in figure 4.
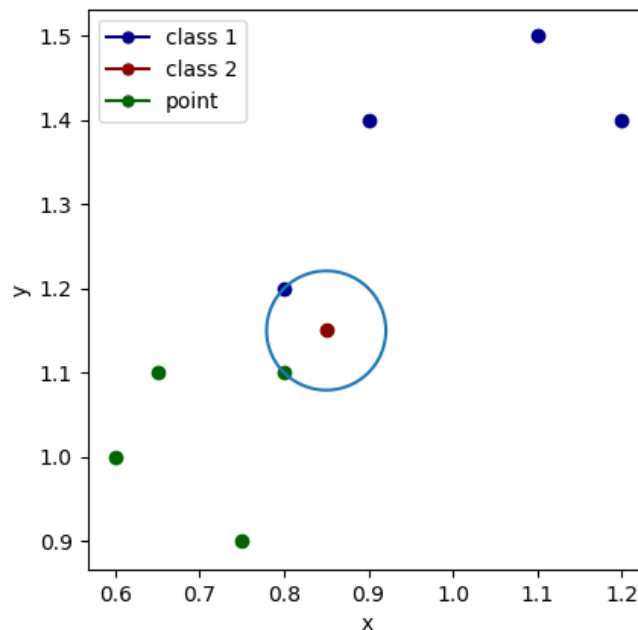

Figure 4: Example graph showing a kNN approach to classifying testing points with a value of k=1

In order to find the most efficient k value, the performance of the classifier needs to be balanced with the processing time. As the k value increases, so does the processing time, up to a k_{max} value which is equal to the total number of training data points. Statistically, the best k value will be around $\sqrt{k_{max}}$. However, in reality, this value can be quite different depending on the data being classified, therefore it is prudent to measure the performance of the classifier at different k values to find the optimal value.

7 Back-Propagation Neural Network (BPNN)

Written by Aaron Wilson

Neural networks take the concept of the perceptron and expand it to many inter-connected neurons. Fig. 5 shows an example NN consisting of an *intput layer* (the input x_i's), one hidden layer (the three circles in the center), and an output layer (the neuron connected to the output y). One of the reasons NNs are so flexible is the ability to change a variety of different parameters in the network, such as the number of hidden layers, the number of neurons in each unit, and the number of output units. This facilitates the ability to approximate almost any function imaginable. Each connection from neuron i to neuron j has its own associated weight, w_{ij}. These weights are used to *train* the network to fit a certain set of data used as input, as well as characterizing *new* (testing) data.

7.1 The Sigmoid Neuron

Each neuron in a NN performs some variation of a thresholding technique. One of the most popular choices of activation function is the *sigmoid function* [5]:
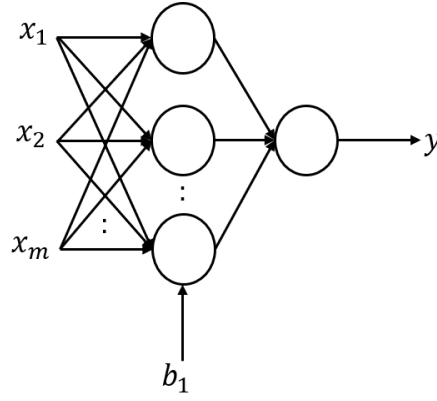
$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{26}$$

Figure 5: Neural Network with One Hidden Layer

The sigmoid function maps values into the continuous range of $[0, 1]$. Fig. 6 illustrates this phenomenon graphically. Values output from the sigmoid function that exceed 0.5 are classified as a "1", and "0" other wise (in the binary class case). The sigmoid function is particularly useful because it is *differentiable*, which is important in back-propagation.
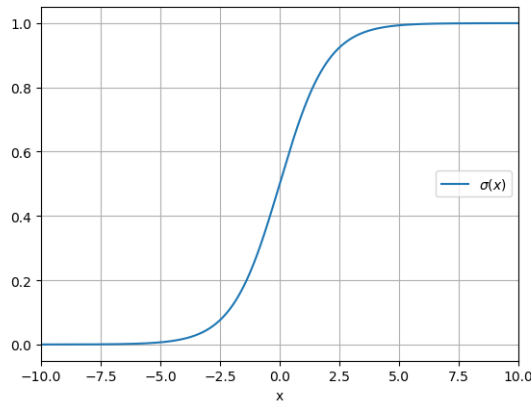

Figure 6: Sigmoid function

7.2 Gradient Descent

A neural network "learns" during training by updating its connection weights as it gathers new information about the data set. The weights, $\vec{w}$ are randomly initialized; taken from the standard normal distribution. As the network accepts more data, it is learning more about the properties that that particular data set was drawn from. To reflect these changes in the network, the weights are updated via *gradient descent*. Gradient descent (GD) uses the partial derivatives of the *cost function* to gradually update the system weights until a "stability" point is reached. Mathematically, [6]:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta \frac{\partial C}{\partial \boldsymbol{w}_k} \tag{27}$$

where η is the *learning rate*. The computation in (27) is repeated until the condition $|w_{k+1} - w_k| < \epsilon$ is achieved.

GD is a mathematical way of finding global minima. Given a starting point, the "direction" of travel is determined using the steepest downward direction of the curve (i.e. $\frac{\partial C}{\partial w_k}$). This is analogous to placing a ball along the slope of a hill; it will roll down the steepest direction until it reaches a valley.
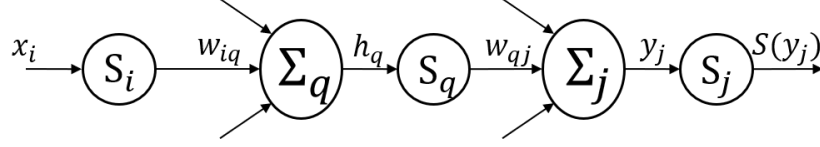
Figure 7: Generalized 3-Layer Neural Network

7.2.1 Stochastic Gradient Descent

When the amount of data needed to train a network becomes substantially large, it may be worthwhile for the designer to employ *stochastic gradient descent* in place of the standard gradient descent algorithm. Stochastic gradient descent approximates $\frac{\partial C}{\partial w_k}$ by using a small subset, or "mini-batch" of the training samples at a time. Using a smaller number of training examples allows for quicker learning, [5]. In this project, stochastic gradient descent is used with a mini-batch size of $m = 10$.

7.3 Back-Propagation

Computation of $\frac{\partial C}{\partial w_k}$ can be quite cumbersome, especially as the size of the NN increases. Because C does not directly depend on $\vec{w}_k$, the chain rule must be utilized. Relevant expressions may be defined using the generalized 3-layer NN depicted in Fig. 7 [6].

The cost function C may be defined using the least-squares technique that measures the squared difference between the "ground truth" labels T_j and the network outputs $S(y_j)$:

$$C = \frac{1}{2} \sum_j \left(T_j - S(y_j)\right)^2 \tag{28}$$

The partial derivative term in (27) for layer $q - j$, may be expressed (using the chain rule) as [6]:

$$\frac{\partial C}{\partial w_{qj}} = \frac{\partial C}{\partial S_j} \frac{\partial S_j}{\partial y_j} \frac{\partial y_j}{\partial w_{qj}} \tag{29}$$

where $\frac{\partial C}{\partial S_j} = T_j - S_j$, $\frac{\partial S_j}{\partial y_j} = S(y_j)(1 - S(y_j))$ [5], and $\frac{\partial y_j}{\partial w_{qj}} = S_q(h_q)$. Substituting these expressions into (29),

$$\frac{\partial C}{\partial w_{qj}} = (T_j - S(y_j)) \left[S(y_j)(1 - S(y_j))\right] S_q(h_q) \tag{30}$$

Moving backwards to the layer $i - q$, the equation for $\frac{\partial C}{\partial w_{iq}}$ may be written as:

$$\frac{\partial C}{\partial w_{iq}} = \left[\sum_j \frac{\partial C}{\partial S_j} \frac{\partial S_j}{\partial y_j} \frac{\partial y_j}{\partial S_q}\right] \frac{\partial S_q}{\partial h_q} \frac{\partial h_q}{\partial w_{iq}} \tag{31}$$

$$\frac{\partial C}{\partial w_{iq}} = \left[\sum_j (T_j - S_j)\left(S(y_j)(1 - S(y_j))\right) w_{qj}\right] S(h_q)(1 - S(h_q)) x_i \tag{32}$$

Back-propagation allows the network to "learn from its mistakes" based upon a certain initialization of the weight and bias values. It differs from the "feed-forward" operation in which an output is computed through the various neurons and weighted inputs. Back-propagation feeds the output *back* through the network to adjust the weights such that the network properly fits the data set.

8 Random Forest

Written by Xuesong Fan

Decision tree is a predictive model to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).[7] Tree models where the target variable can take a discrete set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Although this model is simple to understand and interpret, it has many limitations and the performance may not be very promising.

Random forest is an ensemble learning method which corrects the overfitting problem in the decision tree. It improves the classification by combining classifications of randomly generated training sets and randomly selecting features at each node to determine a split. By applying random forest, the performance would be considerably improved in this project.

9   Support Vector Machine (SVM)

Written by Adrian Cross

SVMs are a classification technique which aims to separate the two classes defined in this problem. One of the key advantages of SVMs are they are very adaptable to the distribution shape of the data points. This is in contrast to other classification techniques, such as MPP shown in section 9, because it allows for a variety of different boundary shapes, via the defined kernel, in addition to a singular separable line. Figure 8 shows an example of an SVM with a linear kernel.
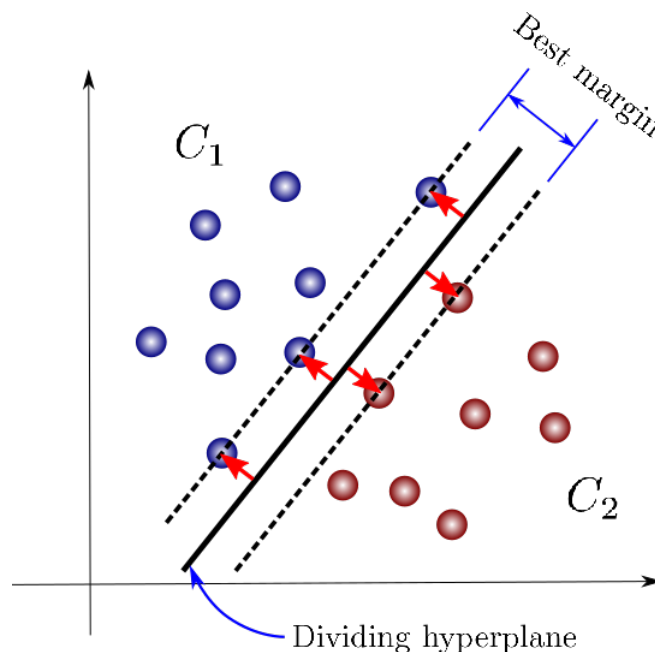

Figure 8:   Example of how a linear SVM can seperate 2 classes in a 2D case (from https://towardsdatascience.com/support-vector-machines-for-classification-fc7c1565e3)

Figure 8 also shows two other parameters used in an SVM which need to be optimized. The margin is the separation between the closest data points belonging to either class. The best margin aims to maximize the distance between these two boundaries, depending on prior probabilities. The other parameter is the physical dividing hyperplane which aims to seperate the classes with the least amount of misclassification of the testing samples.

10   K-Means Clustering

Written by Adrian Cross

Data points can be sorted into different clusters based on the other data points within a data set. These different clusters can then be assigned a class value of 1 or 0 based upon the input training data. Then the testing data can be applied to the clusters to give them a predicted class label.

Before the k-means algorithm can be performed, the original cluster centers need to be initialized. This requires two inputs, the first is the number of clusters needing to be initialized, the second is to define how the cluster centers should be distributed. This algorithm works by randomly assigning the cluster centers, depending on the number of clusters used, and then running the algorithm to get the cluster centers.

The k-means algorithm is the clustering algorithm used in this project. This algorithm works by [6]:

1. Begin with an arbitrary assignment of samples to clusters or begin with an arbitrary set of cluster centers and assign samples to nearest clusters

2. Compute the sample mean of each cluster

3. Reassign each sample to the cluster with the nearest mean

4. If the classification of all samples has not changed, stop; else go to step 2.

This algorithm continues until a pre assigned number of iterations has been reached, or until the algorithm has converged so that the samples are no longer changing.

Once the algorithm is complete there are a number of cluster centers with new positions and class labels based on what was derived from the k-means algorithm. The testing data is then applied to these cluster centers, assigning them a label based on their closest cluster centers label [3].

11 Experimental Setup

11.1 MPP

Written by Xuesong Fan

Different prior probabilities were used for the training data to evaluate the performance of MPP classifier. The accuracy curves on the left side of figure 9 show the improvement of performance while increasing the prior probability of the class 0 for different data sets and different cases in MPP classifier. However, it can be observed from the ROC curves in figure 9 that with the increment of true positive rate, the false positive rate also increases. The overall performance is not quite promising although the accuracy seems very high when the prior probability of class 0 is fairly large. Since the prior probability is based on the training data set, which is $P(\omega_0) \approx 0.94$, the MCC was actually used to compare the performance between classifiers, rather than accuracy.
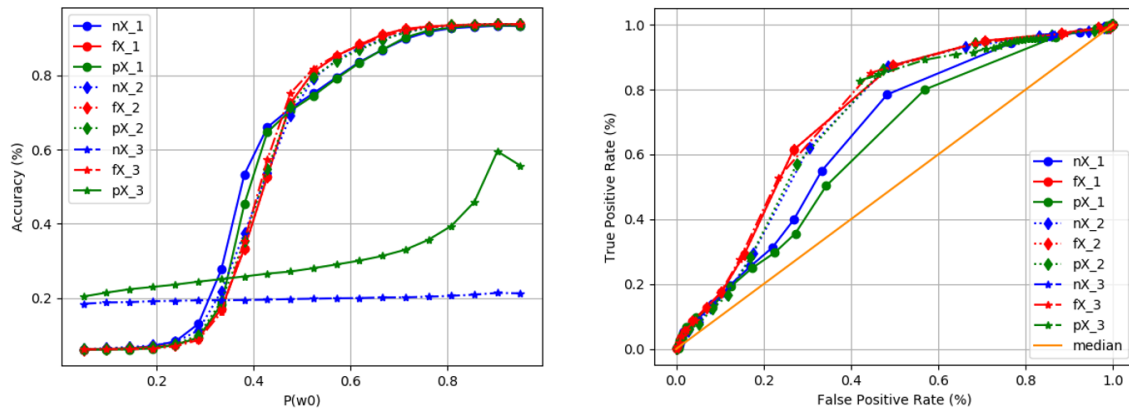

Figure 9: Graphs showing accuracy as a function of prior probability and ROC curves for MPP

11.2 kNN

Written by Adrian Cross

In order to attain a good kNN value to apply to the data set the k value was varied and the accuracy recorded. This variation is shown in figure 10. From this you can see that the accuracy stabilizes once the k value reaches 8 with an accuracy percentage of 95, 95 and 94 for nX, pX and fX respectively.
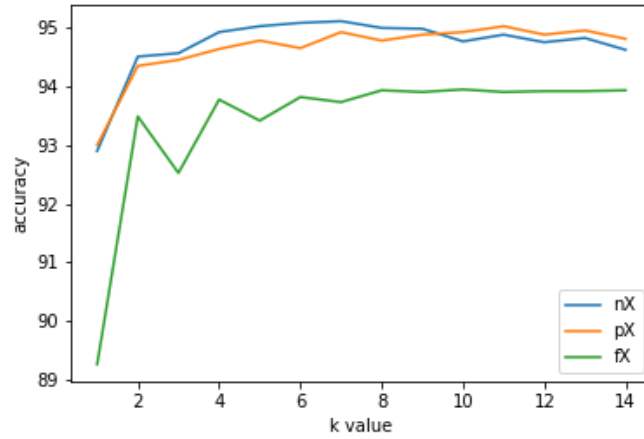
Figure 10: Graph showing variation of accuracy with k value

Using these results it was decided that a value of $k = 10$ should be used for this analysis.

11.3 Back-Propagation Neural Network

Written by Aaron Wilson

A BPNN was constructed using the Python Keras API (Application Programming Interface), from Tensorflow. Optimal hyperparameters selected include learning rate $\eta = 3.0$, number of hidden layers = 1, number of hidden-layer neurons = 3, 30 training epochs, and sigmoid activation functions at each neuron. The stochastic gradient descent optimizer was used.
Not sure what else is needed here? Maybe some justification on why these parameters were chosen?

11.4 Random Forest

Xuesong Fan

11.5 SVM

Written by Adrian Cross
The SVM technique has been used on the data to see which kernel is the best at discriminating between the classes. Figure 11 shows how the accuracy has varied for a variety of different kernels. These kernels include a linear, 2nd order and 3rd order polynomial which follow a fitted line graph of 1st, 2nd and 3rd order. The Radial Based Function (RBF) kernels function is shown in equation 33 and the sigmoid function is shown in equation 34.

$$y = exp(-\gamma||x - x'||^2) \tag{33}$$

$$y = tanh(< x, x' > +r) \tag{34}$$

Where r and γ are values which can vary based on the input data. It is important to note that the sigmoid function defined here is slightly different than the function defined in section 7.1.
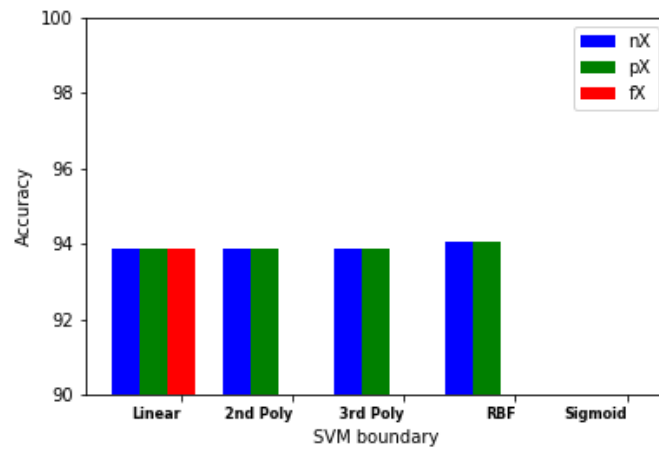
Figure 11: Graph showing variation of accuracy with different SVM kernels

Each kernel has a similar accuracy, except for the sigmoid function which much lower accuracy. The fX normalized data does not apply to the non linear function, due to the number of dimensions. The RBF function has a slightly higher accuracy and therefore is applied to the data.

11.6 K-Means Clustering

Written by Adrian Cross

The number of clusters required for our analysis is first gained via the elbow method [3]. This gives a cluster value of 6 as the optimum amount. Next the algorithm is investigated to see how many iterations should be looped over, or whether it needs a set limit in the case of it converging. Figure 12 shows a variation of accuracy with number of iterations.
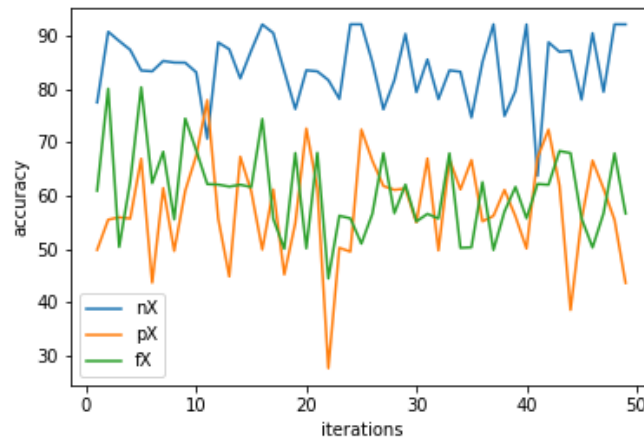

Figure 12: Graph showing variation of accuracy of clusters with number of iterations

From figure 12 the accuracy has a large amount of noise from iteration to iteration, therefore we don't expect a clear result from this method. Nevertheless a hard limit of 50 has been put on the number of iterations when calculating final MCC values on the data set.

12 Results

12.1 Classifier Execution Time Comparisons

For execution time comparison between classifiers, each classifier was run on the same machine [Inter(R) Core(TM) i7-7800X CPU @ 3.50GHz with 32.0 GB RAM]. Each time value encapsulates cross-validation. Fig. 13 shows the relative comparisons between classifiers.
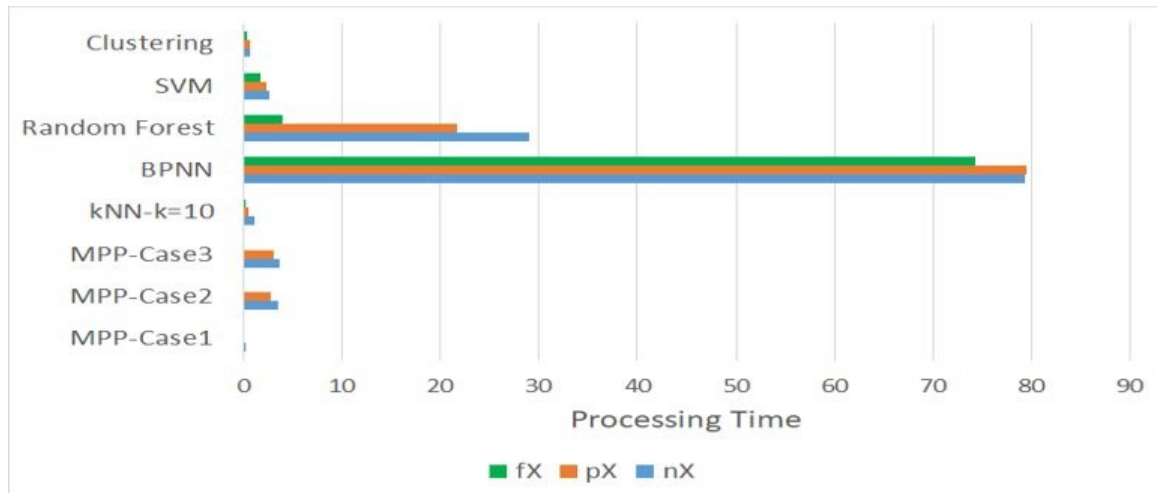

Figure 13: Execution Times for Various Classifiers

It can be seen from Fig. 13 that the BPNN classifier far and away was the most computationally intensive, the random forest classifier taking the next-most amount of time. Every other classifier required only five seconds or less to fully classify the data.

12.2 MCC

For the MCC value calculations each classifier was run on the same machine, as outlined in section 12.1. Figure 14 shows how the MCC values stack up against each other for the machine learning classifiers outlined in this project.
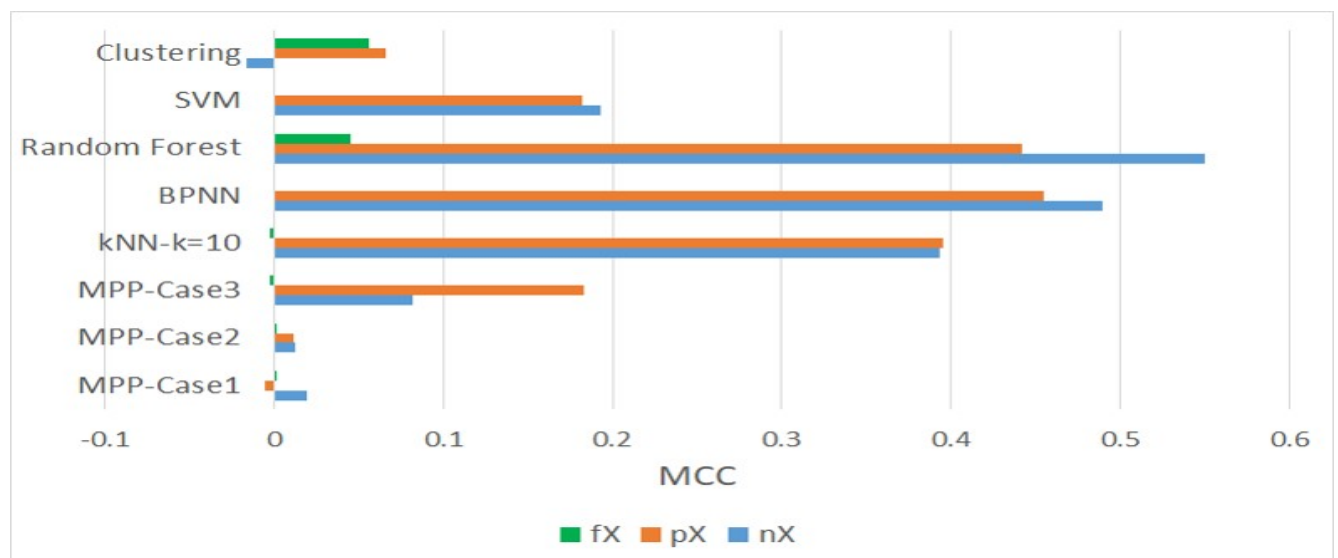

Figure 14: MCC for Various Classifiers

From this bar chart the random forest has the best MCC value of approximately 0.55 for the nX data. The second best classifier was the BPNN for and MCC value of 0.49. It can also be seen that the fX reduced data removes too

16

much information from the data set, as it reduces the MCC value to a maximum value of 0.05 in the clustering algorithm, which is far too small for any distinguishing classes. The pX data also removes some of the data, decreasing the maximum MCC value to 0.45 which, although it is less than the optimum 0.55, may still be useful, depending on required parameters, due to the decrease in processing time.

12.3 Classifier Fusion

Written by Aaron Wilson

Often certain classifiers will perform better than others in certain aspects. To exploit the best-performing attributes of multiple classifiers, the classifiers may be *fused* together. There are multiple methods available to perform this task, but for this project, Naïve Bayesian classifier fusion was performed. Given a set of predicted labels produced by classifier k, $\boldsymbol{s}_k$, the "fused" label may be estimated by applying Bayes' Theorem:

$$P(\omega_k|\boldsymbol{s}) = \frac{p(\boldsymbol{s}|\omega_k)}{p(\boldsymbol{s})} = \frac{P(\omega_k)\prod_{i=1}^{L}p(s_i|\omega_k)}{p(\boldsymbol{s})} \tag{35}$$

where $P(\omega_k) = \frac{N_k}{N}$ is the prior probability of class ω_k, N_k is the number of samples in class ω_k, N is the total number of samples, $p(s_i|\omega_k) = \frac{cm_{k,s_i}}{N_k}$ is the pdf of predicted labels from classifier i belonging to class ω_k, and cm_{k,s_i} is the confusion matrix for classifier i at row k. The above equation may be approximated with [6]

$$P(\omega_k|\boldsymbol{s}) \approx \frac{1}{N_k^{L-1}}\prod_{i=1}^{L}cm_{k,s_i} \tag{36}$$

Classifier fusion in this project was performed using the results of the BPNN (0.49 MCC) and the Random Forest classifier (0.55 MCC). As a result, MCC value of the fusion was only 0.47 - leading us to believe that classifier fusion was NOT helpful in improving the performances from these two classifiers.

12.4 Comparison to Kaggle

Written by Adrian Cross

The Kaggle website has a leaderboard showing the best MCC value for different groups. Figure 15 shows a screenshot of this leaderboard after the competition has completed.

Figure 15: Kaggle leaderboard after completion of competition showing a leading MCC value of 0.71899

The score column shows the leading MCC values of each classification method. our maximum value of 0.55 is obviously lower than the leading value of 0.72, however the leading value also uses more complicated feature extraction techniques from the original signal data, which is beyond the scope of this project.

13 Conclusion

Written by Adrian Cross

In this project features have been extracted from signal sources using signal processing techniques. The dimensions of the data have then been normalized, to produce the nX data set, and reduced using PCA, to produce the pX data set, and FLD, to produce the fX data set. After this normalization the three different data sets have been put through separate machine learning techniques, kNN, SVM, MPP (with all three cases considered), BPNN, random forest and k-means clustering as well as a fusion of random forest and BPNN. The outputted, classified data has then been compared, using the MCC, to see how the reduced datasets compare to each other and to find the best classification technique for this data. It was found that the nX data classified using the Random forest method produced the best MCC value of 0.55.

The best MCC value of 0.55 derived from this project is less than the leaderboard result on the Kaggle website of 0.72. Future studies would improve upon this MCC number by involving a much more in depth extraction of features, such as those performed in [4]. This would increase the information retained when converting from signal to data, thus increasing the overall accuracy of the final result. However, due to the time constraints and technical knowledge required, this analysis is not within the scope of this project.

References

[1] S. Boughorbel, F. Jarray, and M. El-Anbari. Optimal classifier for imbalanced data using matthews correlation coefficient metric. *PloS one*, 12(6):e0177678, 2017.

[2] Kaggle. Vsb power line fault detection. `https://www.kaggle.com/c/vsb-power-line-fault-detection`. Accessed: 12/13/2019.

[3] T. S. Madhulatha. An overview on clustering methods. *arXiv preprint arXiv:1205.1117*, 2012.

[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. *IEEE Transactions on Dielectrics and Electrical Insulation*, 24(2):1097–1104, 2017.

[5] M. Nielsen. Using neural nets to recognize handwritten digits [electronic resourse]–electronic data. *Mode of access: http://neuralnetworksanddeeplearning. com/chap1. html*, 2016.

[6] University of Tennessee. Machine learning cosc. `http://web.eecs.utk.edu/~hqi/cosc522/`. Accessed: 12/13/2019.

[7] Wikipedia. Decision tree learning. `https://en.wikipedia.org/wiki/Decision_tree_learning`. Accessed: 12/13/2019.