\documentclass[a4paper,10pt]{article}

\usepackage[utf8]{inputenc}

\setlength{\parindent}{0pt}

\usepackage{url}

\def\UrlBreaks{\do\/\do-}

\usepackage{breakurl}

\usepackage[breaklinks]{hyperref}

\usepackage[margin=1 in]{geometry}

%\usepackage{tgbonum}

\usepackage[utf8]{inputenc}

\usepackage{natbib}

\usepackage{placeins}

\usepackage{graphicx}

\usepackage{csquotes}

\usepackage{times}

\usepackage{amsmath}

\usepackage{listings}

\usepackage{xcolor}

\usepackage{abstract}

\usepackage{bm} % ADDED BY AARON

\begin{document}

\section{Preprocessing}

\subsection{Input Data}

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.~\ref{fig:input\_data} shows examples of both a ``normal'' waveform and a waveform containing partial discharge patterns. The input data was collected by V\u{S}B - 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).\\

\begin{figure}

\centering

\includegraphics[scale=0.65]{pd\_vs\_no\_pd.png}

\caption{Examples of PD and Non-PD Waveforms}

\label{fig:input\_data}

\end{figure}

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 \cite{Vantuch}. 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.~\ref{fig:denoising\_process} shows an example waveform that has gone through this process.

\begin{figure}[h!]

\centering

\includegraphics[scale=0.65]{denoising.png}

\caption{Denoising Process}

\label{fig:denoising\_process}

\end{figure}

\subsection{High-Pass Filter}

A $10^{th}$-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.~\ref{fig:input\_data}.

\subsection{Denoising via the Discrete Wavelet Transform (DWT)}

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.~\ref{fig:dwt}.\\

\begin{figure}[h!]

\centering

\includegraphics[scale=0.65]{dwt.png}

\caption{DWT Process \cite{Vantuch}}

\label{fig:dwt}

\end{figure}

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 \cite{Vantuch}:

\[

T\_{hard}(x) =

\begin{cases}

x & \text{if $\left|x\right|>\lambda$} \\

0 & \text{otherwise}

\end{cases}

\]

where $x$ represents the detail coefficients $c\_{d}$, $\lambda$ represents the threshold calculated by

\begin{equation}

\lambda = \frac{1}{0.06745\times \text{MAD}(\left|c\_{d}\right|)}\sqrt{2\text{log}(n)},

\end{equation}

$\text{MAD}(\cdot)$ represents the \textit{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.

\subsection{Feature Extraction}

From the denoised signal, twelve features were extracted. A table showing each of the features with a quick description is given in Table~\ref{tbl:features}.

\begin{center}

\begin{table}[t!]

\caption{Features Extracted from Denoised Signals}

\label{tbl:features}

\begin{tabular}{|c|c|}

\hline

\textbf{Feature} & \textbf{Description}\\ \hline

Signal Mean & Mean of all 800,000 data points in denoised signal\\ \hline

Signal Standard Deviation & Standard deviation of denoised signal\\ \hline

Signal Skewness & How skewed the signal data points are with respect to a normal distribution\\\hline

Signal Kurtosis & A measure of how prominent the signal distributions ``tails'' are\\\hline

Number of Positive Peaks & A count of the total number of peaks greater than zero\\\hline

Number of Negative Peaks & A count of the total number of peaks less than zero\\\hline

Mean Peak Width & Mean value of peak widths\\\hline

Mean Peak Height & Mean value of peak heights\\\hline

Max Peak Width & Max value of peak widths\\\hline

Max Peak Height & Max value of peak heights\\\hline

Min Peak Width & Min value of peak widths\\\hline

Min Peak Height & Min value of peak heights\\\hline

\end{tabular}

\end{table}

\end{center}

The work performed in \cite{Vantuch} 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 \cite{Vantuch} 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.

\section{$m$-Fold Cross Validation}

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.

\section{Back-Propagation Neural Network (BPNN)}

Neural networks take the concept of the perceptron and expand it to many inter-connected neurons. Fig.~\ref{fig:nn} shows an example NN consisting of an \textit{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 \textit{train} the network to fit a certain set of data used as input, as well as characterizing \textit{new} (testing) data.

\begin{figure}[h!]

\centering

\includegraphics[scale=0.65]{nn.png}

\caption{Neural Network with One Hidden Layer}

\label{fig:nn}

\end{figure}

\subsection{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 \textit{sigmoid function} \cite{nielsen}:

\begin{equation}

\sigma(z)=\frac{1}{1 + e^{-z}}

\end{equation}

The sigmoid function maps values into the continuous range of $\left[0, 1\right]$. Fig.~\ref{fig:sigmoid} 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 \textit{differentiable}, which is important in back-propagation.

\begin{figure}[h!]

\centering

\includegraphics[scale=0.5]{sigmoid.png}

\caption{Sigmoid function}

\label{fig:sigmoid}

\end{figure}

\subsection{Gradient Descent}

A neural network ``learns'' during training by updating its connection weights as it gathers new information about the data set. The weights, $\bm{\vec{w}}$ 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 \textit{gradient descent}. Gradient descent (GD) uses the partial derivatives of the \textit{cost function} to gradually update the system weights until a ``stability'' point is reached. Mathematically, \cite{qi}:

\begin{equation}

\bm{w}\_{k+1} = \bm{w}\_{k} - \eta \frac{\partial C}{\partial \bm{w}\_{k}}

\label{eqn:gd}

\end{equation}

where $\eta$ is the \textit{learning rate}. The computation in \eqref{eqn:gd} 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.

\subsubsection{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 \textit{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, \cite{nielsen}. In this project, stochastic gradient descent is used with a mini-batch size of $m = 10$.

\subsection{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 $\bm{\vec{w}}\_{k}$, the chain rule must be utilized. Relevant expressions may be defined using the generalized 3-layer NN depicted in Fig.~\ref{fig:3nn} \cite{qi}.\\

\begin{figure}

\centering

\includegraphics[scale=0.65]{3nn.png}

\caption{Generalized 3-Layer Neural Network}

\label{fig:3nn}

\end{figure}

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 $S(y\_{j})$:

\begin{equation}

C=\frac{1}{2}\sum\_{j}\left(T\_{j} - S(y\_{j})\right)^{2}

\end{equation}

The partial derivative term in \eqref{eqn:gd} for layer $q-j$, may be expressed (using the chain rule) as \cite{qi}:

\begin{equation}

\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}}

\label{eqn:chain\_rule1}

\end{equation}

where $\frac{\partial C}{\partial S\_{j}}=T\_{j} - S\_{j}$, $\frac{\partial S\_{j}}{\partial y\_{j}}=S(y\_{j})(1 - S(y\_{j})$ \cite{nielsen}, and $\frac{\partial y\_{j}}{\partial w\_{qj}} = S\_{q}(h\_{q})$. Substituting these expressions into \eqref{eqn:chain\_rule1},

\begin{equation}

\frac{\partial C}{\partial w\_{qj}}=(T\_{j} - S(y\_{j}))\left[S(y\_{j})(1 - S(y\_{j}))\right]S\_{q}(h\_{q})

\end{equation}

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

\begin{equation}

\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}}

\end{equation}

\begin{equation}

\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}

\end{equation}

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 \textit{back} through the network to adjust the weights such that the network properly fits the data set.

\section{BPNN Setup}

\textbf{\*\*\*AJ I'm not sure how you want to incorporate this next section, but it should be with the setups for the other classifiers - not its own section. I'm just writing it like this to keep it separate from the above background sections.\*\*\*}

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.

\textbf{\*\*\*Not sure what else is needed here? Maybe some justification on why these parameters were chosen?\*\*\*}

\section{Classifier Execution Time Comparisons}

For execution time comparison between classifiers, each classifier was run on the same machine \textbf{\*\*\*Xuesong, can you put some spec on your laptop here? Processor type, amount of RAM, CPU clock speed, etc.\*\*\*}. Each time value encapsulates cross-validation. Fig.~\ref{fig:times} shows the relative comparisons between classifiers.

\begin{figure}

\centering

\includegraphics[scale=0.8]{times.jpg}

\caption{Execution Times for Various Classifiers}

\label{fig:times}

\end{figure}

It can be seen from Fig.~\ref{fig:times} 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.

\textbf{\*\*\*Anything else here?\*\*\*}

\end{document}