%citept or citepp

\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}

\definecolor{codegreen}{rgb}{0,0.6,0}

\definecolor{codegray}{rgb}{0.5,0.5,0.5}

\definecolor{codepurple}{rgb}{0.58,0,0.82}

\definecolor{backcolour}{rgb}{0.95,0.95,0.92}

\usepackage[utf8]{inputenc}

\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

\lstdefinestyle{mystyle}{

backgroundcolor=\color{backcolour},

commentstyle=\color{codegreen},

keywordstyle=\color{magenta},

numberstyle=\tiny\color{codegray},

stringstyle=\color{codepurple},

basicstyle=\ttfamily\footnotesize,

breakatwhitespace=false,

breaklines=true,

captionpos=b,

keepspaces=true,

numbers=left,

numbersep=5pt,

showspaces=false,

showstringspaces=false,

showtabs=false,

tabsize=2

}

\lstset{style=mystyle}

%\bibliographystyle{unsrtnat}

\bibliographystyle{abbrv}

%\setcitepstyle{authoryear,open={(},close={)}}

\title{Final Project - Power Line Partial Discharge Detection}

\author{Adrian Cross, Xuesong Fan, Aaron Wilson}

\begin{document}

%\fontfamily{qtm}\selectfont

\maketitle

\begin{center}

Machine learning

COSC 522

\end{center}

\begin{abstract}

\end{abstract}

\tableofcontents

\section{Introduction}

\subsection{Problem}

\textit{Written by Adrian Cross}

\\

The problem worked on in this project is from the Kaggle website \cite{Kaggle}. 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 \ref{MCC}, 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.

\subsection{Performance Metric}

\label{MCC}

\textit{Written by Adrian Cross}

\\

The performance metric used for this project is Matthews Correlation Coefficient (MCC), shown in equation \ref{MCC\_eq}.

\begin{equation}

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

\label{MCC\_eq}

\end{equation}

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 \cite{boughorbel2017optimal}. 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 \ref{MCC\_comp} and are compared with other Kaggle projects in section \ref{MCC\_Kaggle}.

\section{Preprocessing}

\subsection{Input Data}

\textit{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.~\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).\\

\FloatBarrier

\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}

\FloatBarrier

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.

\FloatBarrier

\begin{figure}[h!]

\centering

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

\caption{Denoising Process}

\label{fig:denoising\_process}

\end{figure}

\FloatBarrier

\subsection{High-Pass Filter}

\textit{Written by Aaron Wilson}

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)}

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

\FloatBarrier

\begin{figure}[h!]

\centering

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

\caption{DWT Process \cite{Vantuch}}

\label{fig:dwt}

\end{figure}

\FloatBarrier

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}:

\begin{equation}

T\_{\text {hard}}(x)=\left\{\begin{array}{ll}{x} & {\text { if }|x|>\lambda} \\ {0} & {\text { otherwise }}\end{array}\right.

\end{equation}

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}

\textit{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~\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}

\textit{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.

\section{Normalization and Dimensionality Reduction}

\subsection{Data Normalization}

\textit{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:

\begin{equation}

n x\_{i}=\left(x\_{i}-\mu\_{i}\right) / \sigma\_{i}

\end{equation}

where \(x\_i\) is the sample in the dataset X with feature i, \(\mu\_i\) is the mean of feature i, \(\sigma\_i\) is the standard deviation of feature i, and \(nx\_i\) is the normalized sample in the normalized dataset nX with feature i.

\subsection{Principal Component Analysis (PCA)}

\textit{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:

\begin{equation}

\Sigma=\operatorname{cov}(n X)=\overrightarrow{n X} \cdot \overrightarrow{n X}^{T}

\end{equation}

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

\begin{equation}

[E, \lambda]=\operatorname{eig}(\Sigma)

\end{equation}

where \(\vec{E}=[\overrightarrow{e\_{1}}, \overrightarrow{e\_{1}}, \ldots, \overrightarrow{e\_{d}}]\) represents the eigenvectors and \(\vec{\lambda}=\left[\lambda\_{1}, \lambda\_{2}, \ldots, \lambda\_{d}\right]\) 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:

\begin{equation}

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

\end{equation}

The corresponding chosen eigenvectors could then be represented as \(\vec{P}=[\overrightarrow{e\_{1}}, \overrightarrow{e\_{1}}, \ldots, \overrightarrow{e\_{m}}]\) , and the data for the reduced dimension can be calculated as:

\begin{equation}

\overrightarrow{p X}=\left(\overrightarrow{n X}^{T} \cdot \vec{P}\right)^{T}=\vec{P}^{T} \cdot \overrightarrow{n X}

\end{equation}

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.

\subsection{Fisher's Linear Discriminant (FLD)}

\textit{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{equation}

\begin{aligned} \overrightarrow{S\_{W}} &=\overrightarrow{S\_{1}}+\overrightarrow{S\_{2}} \\ &=\sum\_{t}\left(n x\_{i}-\mu\_{1}\right) \cdot\left(n x\_{i}-\mu\_{1}\right)^{T}+\sum\_{t}^{n\_{2}}\left(n x\_{i}-\mu\_{2}\right) \cdot\left(n x\_{i}-\mu\_{2}\right)^{T} \end{aligned}

\end{equation}

where \(n\_1\) and \(n\_2\) are the number of samples in each class, \(\mu\_1\) and \(\mu\_2\) are the mean values of each class. Since the covariance of the data sets can be calculated as:

\begin{equation}

\Sigma=\operatorname{cov}(X)=\frac{1}{n-1} \sum\_{i}^{n}(\overrightarrow{x\_{i}}-\vec{\mu}) \cdot(\overrightarrow{x\_{i}}-\vec{\mu})^{T}

\end{equation}

The scatter matrix can then be presented as:

\begin{equation}

\overrightarrow{S\_{W}}=\left(n\_{1}-1\right) \operatorname{cov}(\overrightarrow{n X\_{1}})+\left(n\_{2}-1\right) \operatorname{cov}(\overrightarrow{n X\_{2}})

\end{equation}

Then the Canonical variate can be derived as:

\begin{equation}

\vec{w}=\overrightarrow{S\_{W}}^{-1}\left(\mu\_{1}-\mu\_{2}\right)

\end{equation}

The projected data fX then could be calculated as:

\begin{equation}

\overrightarrow{f X}=\vec{w}^{T} \cdot \overrightarrow{n X}

\end{equation}

\section{Maximum Posteriori Probability (MPP)}

\label{MPP}

\textit{Written by Xuesong Fan}

\\

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

\begin{equation}

P\left(\omega\_{i} | \vec{x}\right)=\frac{p\left(\vec{x} | \omega\_{i}\right) P\left(\omega\_{i}\right)}{p(\vec{x})}

\end{equation}

The discriminant function \(g\_{i}(x)\) is:

\begin{equation}

g\_{i}(\vec{x})=P\left(\omega\_{i} | \vec{x}\right)

\end{equation}

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

\begin{equation}

g\_{i}(\vec{x})=p\left(\vec{x} | \omega\_{i}\right) P\left(\omega\_{i}\right)

\end{equation}

Take the logarithm on both sides,

\begin{equation}

g\_{i}(\vec{x})=\ln p\left(\vec{x} | \omega\_{i}\right)+\ln P\left(\omega\_{i}\right)

\end{equation}

Since the multivariate normal density is

\begin{equation}

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]

\end{equation}

The discriminant function can be expressed as:

\begin{equation}

\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 \left|\Sigma\_{i}\right|-\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}

\end{equation}

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

\begin{equation}

g\_{i}(\vec{x})=-\frac{1}{2} \ln \left|\Sigma\_{i}\right|-\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)

\end{equation}

\subsection{Case 1}

\textit{Written by Xuesong Fan}

\\

\section{Closest near neighbour approach (kNN)}

\textit{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 \ref{scatter\_plot}.

\FloatBarrier

\begin{figure}[!htb]

\centering

\includegraphics[width=9cm]{scatter\_plot.png}

\caption{Example graph showing a kNN approach to classifying testing points with a value of k=1}

\label{scatter\_plot}

\end{figure}

\FloatBarrier

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.

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

\textit{Written by Aaron Wilson}

\\

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{Random Forest}

\textit{Written by Xuesong Fan}

\\

\section{Support Vector Machine (SVM)}

\textit{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 \ref{MPP}, because it allows for a variety of different boundary shapes, via the defined kernel, in addition to a singular separable line. Figure \ref{SVM\_ex} shows an example of an SVM with a linear kernel.

\FloatBarrier

\begin{figure}[h!]

\centering

\includegraphics[scale=0.45]{SVM\_ex.png}

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

\label{SVM\_ex}

\end{figure}

\FloatBarrier

Figure \ref{SVM\_ex} 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.

\section{K-Means Clustering}

\textit{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 \cite{qi}:

\begin{enumerate}

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

\item Compute the sample mean of each cluster

\item Reassign each sample to the cluster with the nearest mean

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

\end{enumerate}

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.

\section{Experimental Setup}

\subsection{MPP}

\textit{Xuesong Fan}

\subsection{kNN}

\textit{Adrian Cross}

\subsection{Back-Propagation Neural Network}

\textit{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.

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

\subsection{Random Forest}

\textit{Xuesong Fan}

\subsection{SVM}

\textit{Adrian Cross}

\subsection{K-Means Clustering}

\textit{Adrian Cross}

%Don't think we need all these sections for results

\section{Results}

\subsection{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.

\FloatBarrier

\begin{figure}

\centering

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

\caption{Execution Times for Various Classifiers}

\label{fig:times}

\end{figure}

\FloatBarrier

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.

\subsection{MCC}

\FloatBarrier

\begin{figure}

\centering

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

\caption{MCC for Various Classifiers}

\label{fig:times}

\end{figure}

\FloatBarrier

\subsection{Classifier Fusion}

\textit{Aaron Wilson}

\subsection{Comparison to Kaggle }

\label{MCC\_Kaggle}

\textit{Adrian Cross}

\section{Conclusion}

\textit{Adrian Cross}

\begin{lstlisting}[language=Python]

\end{lstlisting}

\bibliography{Bib}

\end{document}