COMPUTATIONAL INTELLIGENCE AND DATA MINING TECHNIQUES USING THE FIRE DATA SET

Jeremy Storer

A Thesis
Submitted to the Graduate College of Bowling Green State University in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2016

Committee:
Robert C. Green II, Advisor
Robert Dyer
Jong Kwan Lee
ABSTRACT

Robert C. Green II, Advisor

Forest fires are a dangerous and devastating phenomenon. Being able to accurately predict the burned area of a forest fire could potentially limit biological damage as well as better prepare for ensuing economical and ecological damage. A data set from the Montesinho Natural Park in Portugal provides a difficult regression task regarding the prediction of forest fire burn area due to the limited amount of data entries and the imbalanced nature of the data set. This thesis focuses on improving these results through the use of a Backpropagation trained Artificial Neural Network which is systematically evaluated over a variety of configurations, activation functions, and input methodologies, resulting in approximately 30% improvements to regression error rates. A Particle Swarm Optimization (PSO) trained Artificial Neural Network is also evaluated in a variety of configurations providing approximately 75% improvement of regression error rates. Going further, the data is also clustered on both inputs and outputs using k-Means and Spectral algorithms in order to pursue the task of classification where near perfect classification is achieved when clustering on inputs is considered and an accuracy of roughly 60% is achieved when clustering on output values.
# TABLE OF CONTENTS

## CHAPTER 1 INTRODUCTION

## CHAPTER 2 BACKGROUND AND LITERATURE REVIEW

2.1 Montesinho Dataset .................................................. 3

2.2 Artificial Neural Networks ............................................ 4

2.2.1 Error Metric ...................................................... 6

2.3 ANN Training Methods ............................................... 6

2.3.1 Backpropagation .................................................. 6

2.4 ANN Classification ................................................... 10

2.5 Particle Swarm Optimization trained ANN ....................... 11

2.6 k-Means Clustering .................................................. 13

2.7 Spectral Clustering ................................................... 13

2.8 Literature Review .................................................... 16

## CHAPTER 3 PREDICTING FOREST FIRE SIZE WITH PARTICLE SWARM TRAINED NEURAL NETWORKS

3.1 Data Preprocessing ................................................... 21

3.2 Artificial Neural Network Architectures ......................... 23

3.3 Results ............................................................... 25

3.3.1 Backpropagation .................................................. 25

3.3.2 Particle Swarm Optimization Trained ANN .................. 31

3.4 Discussion and Comparison ....................................... 33

3.5 Conclusion .......................................................... 34

## CHAPTER 4 CLASSIFYING THE FOREST FIRE DATA SET WITH CLUSTERING AND COMPUTATIONAL INTELLIGENCE

35
LIST OF FIGURES

2.1 An ANN with 3 input perceptrons, 3 hidden perceptrons, and one one output perceptron including an additional bias perceptron on the input and hidden layers. . . . 5

2.2 An ANN with 3 Inputs, a 2 Perceptron hidden layer, and 1 output perceptron. . . . 7

3.1 Comparative histograms of data transformations as applied to outputs. . . . . . . . 22

3.2 Input architecture before and after applying 1-of-C encoding. . . . . . . . . . . . . 23

3.3 Minimum RMSE values using the Hard Sigmoid as the first layer activation function as paired with all second layer activation function permutations. . . . . . . . 28

3.4 Minimum RMSE values using the Linear as the first layer activation function as paired with all second layer activation function permutations. . . . . . . . 29

3.5 Minimum RMSE values using the Rectifier as the first layer activation function as paired with all second layer activation function permutations. . . . . . . . 29

3.6 Minimum RMSE values using the Sigmoid as the first layer activation function as paired with all second layer activation function permutations. . . . . . . . 30

3.7 Minimum RMSE values using the Hyper Tangent as the first layer activation function as paired with all second layer activation function permutations. . . . . . . . 30

3.8 Comparison of the 12 and 29 Input method with common statistical measurements using PSO as the training method. . . . . . . . . . . . . . . . . . . . . 32

3.9 12 and 29 Input method time to complete a training cycle using PSO as the training method. . . . . . . . . . . . . . . . . . . . . 32

4.1 Cluster Distributions of k-Means where k = 2. . . . . . . . . . . . . . . . . . . 38

4.2 Cluster Distributions of k-Means where k = 3. . . . . . . . . . . . . . . . . . . 39

4.3 Cluster Distributions of k-Means where k = 4. . . . . . . . . . . . . . . . . . . 40

4.4 Cluster Distributions of Spectral Clustering where k = 2. . . . . . . . . . . . . . 42
4.5 Cluster Distributions of Spectral Clustering where $k = 3$. . . . . . . . . . . . . . . 43
4.6 Cluster Distributions of Spectral Clustering where $k = 4$. . . . . . . . . . . . . . . 44
4.7 Graphical representation of the three clusters created from the Spectral Clustering
    algorithm using Outputs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
4.8 Graphical representation of the three clusters created from the Spectral Clustering
    algorithm using Inputs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.9 Graphical representation of the three clusters created from the k-Means Clustering
    algorithm using Outputs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
4.10 Graphical representation of the three clusters created from the k-Means Clustering
    algorithm using Inputs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Data attributes and ranges of the Montesinho data set</td>
<td>4</td>
</tr>
<tr>
<td>3.1</td>
<td>Common statistical data for the 12 Input method using backpropagation. Each activation function column represents all permutations of ANN architectures assuming corresponding function is used in first layer.</td>
<td>26</td>
</tr>
<tr>
<td>3.2</td>
<td>Common statistical data for the 29 Input method using backpropagation. Each activation function column represents all permutations of ANN architectures assuming corresponding function is used in first layer.</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Optimal ANN architectures for 12 Input Method for each of the five types of first layer activation functions using backpropagation as the training method.</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>Optimal ANN architectures for the 29 Input Method for each of the five types of first layer activation functions using backpropagation as the training method.</td>
<td>27</td>
</tr>
<tr>
<td>3.5</td>
<td>Statistical information of the 12 and 29 input method PSO trials.</td>
<td>33</td>
</tr>
<tr>
<td>4.1</td>
<td>The best accuracy scores for backpropagation and PSO ANN training methods for k-Means clustered data sets.</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>The best accuracy scores for backpropagation and PSO ANN training methods for Spectral Clustered Data Sets.</td>
<td>45</td>
</tr>
</tbody>
</table>
INTRODUCTION

Forest fires are an unpredictable and highly destructive force of nature [1]. They are able to create tremendous amounts of ecological and economic damage, while simultaneously endangering life. If it were possible to peer into the future and see these catastrophic events before they unfold it would greatly assist in mitigating the destructive impact of forest fires.

Artificial Neural Networks (ANN) attempt to predict the outcome of future events by learning from the past. To accomplish this, supervised training of the ANN is used. Supervised training entails a set of input dimensions with an expected output result being fed into an ANN so that it can be trained to identify trends or patterns. Once trained, the ANN can perform regression or classification on the data set depending on the format of the target output values. The type, amount, and overall characteristics of the data fed into an ANN is the most critical aspect to consider when attempting to train an ANN to reliably predict output results. If the amount of data is too small or heavily skewed, ANNs become very difficult to train. If the amount of data is small it is difficult to adequately train for all possible results. If the data is skewed the ANN will be bias towards a result that is overrepresented in the data set. This leads to difficulty converging on an optimum, which can result in inaccurate output predictions.

Datasets available for forest fire burn areas are limited in the number of instances that they contain. This is due, in part, to the fact that each area of the Earth has unique environmental and meteorological conditions. Thus, data from one part of the Earth will not necessarily be helpful for predicting the burn areas of forest fires at different geographical locations. Coupling this with forest fires being a fairly rare event leads to few dataset entries. To provide useful predictions of burn areas a method must be devised that is able to give reasonable predictions using a small, and often times skewed, data set.

The goal of this thesis is to use a small, skewed dataset that is highly difficult to perform a regression task on, provided by the Montesinho Natural Park, which is located in the northeast region of Portugal [2], and improve upon current state of the art results that are obtained when
predicting the burned area of a forest fire. This is accomplished through four primary contributions:

1. The application of 1-of-C encoding to this data set, an approach which has previously not been attempted;

2. A systematic evaluation and comparison of backpropogation and PSO-trained ANNs for regressing this dataset;

3. A systematic evaluation and comparison of various ANN structures and activation functions for regressing this data set; and

4. Using k-Means and spectral clustering to classify, as opposed to regress, this dataset.

By developing methods to improve upon the current results of this dataset it is hoped that the found techniques can be applied to other datasets that are of similar nature.

The structure of the remainder of this thesis is as follows: Chapter 2 will discuss current state of the art methods of approaching this dataset, difficulties involved in using the dataset, and other relevant background information; Chapter 3 contains the description and findings of an ANN that is trained to perform a regression task of predicting the burn area of a forest fire using both Backpropagation and Particle Swarm Optimization training methods using, an input technique that is novel to this data set, in addition to a systematic evaluation of the structure of the ANN as well as the activation functions used; Chapter 4 details the results of clustering the data set using a various number of clusters and multiple clustering algorithms in order to perform classification using ANNs; and Chapter 5 provides closing remarks and results.
BACKGROUND AND LITERATURE REVIEW

This chapter reviews those items that are significant to the work accomplished in this thesis including the Montesinho Fire Data Set, Artificial Neural Networks including backpropagation and particle swarm training techniques, and k-Means and spectral clustering techniques.

2.1 Montesinho Dataset

The dataset presented in this thesis was collected from January 2000 through December 2003 and built using two sources: First, a set of data collected by an inspector that is responsible for documenting Montesinho Park fire occurrences. Every time a forest fire occurred in the park several features were registered by this inspector; the time, date, the spatial location within a 9x9 grid of the park (inputs X, Y), the type of vegetation involved, components from the FWI system, and the total area burned [3]. Second, an organization collected information about wind speed and other weather patterns using a meteorological station at the center of Montesinho Natural Park. These two data sets were combined to form the one that is used in this thesis.

The Fire Weather Index (FWI) is a Canadian system for rating fire danger [3] and contains six elements: the Fine Fuel Moisture Code (FFMC) which denotes the moisture content of surface litter that influences ignition and fire spread, the Duff Moisture Code (DMC) and Drought Code (DC), which represent the moisture content of shallow and deep organic layers, the Initial Spread Index (ISI), which is a score that correlates with fire velocity spread, the Build Up Index (BUI), which represents the amount of available fuel and the final FWI rating, which is derived from the other values.

The complete Monteshino dataset contains twelve input attributes and one output attribute, the burn area, which is measured in hectares (a hectare is equal to 10,000 \( m^2 \)). The inputs are X, Y, month, day, FFMC, DMC, DC, ISI, temperature, relative humidity, wind speed, and rain. FFMC, DMC, DC, and ISI are taken from the FWI system. The data set supplied for this thesis did not contain information on the BUI or final FWI rating. The attributes and ranges can be seen in Table
### Table 2.1: Data attributes and ranges of the Montesinho data set.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>X,Y coordinates</td>
<td>1 to 9</td>
</tr>
<tr>
<td>Month</td>
<td>Jan to Dec</td>
</tr>
<tr>
<td>Day</td>
<td>Mon to Sun</td>
</tr>
<tr>
<td>FFMC code</td>
<td>18.7 to 96.2</td>
</tr>
<tr>
<td>DMC code</td>
<td>1.1 to 291.3</td>
</tr>
<tr>
<td>DC code</td>
<td>7.9 to 860.6</td>
</tr>
<tr>
<td>ISI index</td>
<td>0 to 56.1</td>
</tr>
<tr>
<td>Temp ($C$)</td>
<td>2.2 to 33.30</td>
</tr>
<tr>
<td>Relative Humidity</td>
<td>15.0 to 100</td>
</tr>
<tr>
<td>Wind Speed ($km/hr$)</td>
<td>0.40 to 9.40</td>
</tr>
<tr>
<td>Rain ($mm/m^2$)</td>
<td>0.0 to 6.4</td>
</tr>
<tr>
<td>Burned area size ($ha$)</td>
<td>0.00 to 1090.84</td>
</tr>
</tbody>
</table>

2.2 Artificial Neural Networks

An ANN is a system that performs classification or regression tasks through a series of mathematical functions. The general basis of how ANNs function was theorized by Hebbs in 1949. Hebbs believes that if an axon of a cell is near enough to another to excite another cell, some growth process or metabolic change takes place in both cells so that when the first cell is excited it more reliably fires the other cell in response [4]. This biological concept is transferred to how ANNs learn by using weights, which act as cells adapting, and activation functions that tell certain neurons to turn on and off.

An ANN is structured into a few simple elements that then interact with each other to create a complex system capable of predicting output targets. In the smallest cases, there are three layers: input, hidden, and output. The number of hidden layers is variable. Each of these layers contains a number of perceptrons. The input layer has one perceptron that corresponds to each input dimension and the output layer has a perceptron that corresponds to each output dimension. In the case of regression, there is commonly one output perceptron which differs from classification as there are often as many perceptrons as there are classes. The number of perceptrons contained in the hidden layer is variable and challenging to determine the optimal amount.
A perceptron is the ANN version of a neuron. It takes several weighted inputs and sums them, and if the combined input exceeds a threshold it will activate and send an output. The output that is sent is determined by an activation function which is commonly between 0 and 1 or -1 and 1. Thus, for a single perceptron, eq. (2.1) can be used, where $y$ is the output, $\Phi$ is the activation function, $n$ is the number of connections to the perceptron, $w_i$ is the weight associated with the $i^{th}$ connection, $x_i$ is the value of the $i^{th}$ connection and $b$ is the bias perceptron.

$$y = \Phi \left( \sum_{i=1}^{n} w_i x_i + b \right)$$  \hspace{1cm} (2.1)

When multiple perceptrons are combined into different layers an ANN is formed as can be seen in Fig. 2.1.

In Fig. 2.1 layer $L_1$ is the input layer, $L_2$ the hidden layer, and $L_3$ the output layer. $L_2$ is known as a hidden layer because the input attributes are not being directly fed into it. Each of the lines is associated with a weight, which is used to multiply with the input value. The +1 value in layers $L_1$ and $L_2$ are bias nodes. Bias nodes allow the shifting of activation functions to the left or right. If eq. (2.1) is applied to each perceptron in each layer a concept known as forward propagation is accomplished. All ANNs used in this thesis use feed forward propagation to calculate values in the output layer. Once the data is fed forward an error metric must be used to evaluate the accuracy of the outputs and then, with that output result, train the ANN to reduce the error metric on the next iteration.
2.2.1 Error Metric

An error metric is used to decide how well an ANN is performing in regards to output prediction versus the target output. The primary error metric chosen for this thesis is root mean squared error (RMSE).

\[
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}
\]  

(2.2)

RMSE is a quadratic scoring rule which measures the average magnitude of error, where \( n \) is the total number of samples, \( y \) is the target value, and \( \hat{y} \) is the predicted value. RMSE squares errors before they are averaged giving a relatively high weight to large errors. As the Montesinho dataset contains several outlier values using RMSE will more severely punish missing predictions of outlier results [5].

2.3 ANN Training Methods

Many seemingly different problems in machine learning can be viewed as requiring the discovery of a computer program that produces a desired output for a set of particular inputs [6]. ANNs learn by altering their weights. Thus, any means of adjusting the weights based on the inputs can be a method of learning. In the case of this thesis, backpropagation and particle swarm optimization are the learning methods used. Backpropagation is used as it is among the most common methods of training an ANN. Trials have been done using it on the Montesinho dataset in the past, so a more systematic approach is desired to find the optimal ANN architecture when using backpropagation as the training method. Particle swarm optimization is chosen as it is radically different from backpropagation and provides a novel approach to training an ANN with the Montesinho data set. The following subsections will introduce how each of these concepts accomplishes training an ANN.

2.3.1 Backpropagation

Backpropagation is an algorithm used to teach feed forward ANNs in a supervised manner. It works by providing a set of inputs, with their corresponding target outputs, into the network,
calculating the actual outputs, and then backpropagating the calculated error using gradient descent [7]. This is useful for learning specific patterns of input and output data in order to be able to reproduce them afterwards even from slightly different or incomplete input data.

Figure 2.2: An ANN with 3 Inputs, a 2 Perceptron hidden layer, and 1 output perceptron.

A simple example can be given to demonstrate an ANN feeding forward and then backpropagating to help solidify understanding of the concept. The following example will refer to Fig. 2.2. \( P_i \) will refer to the perceptron, \( w_{ij} \) will refer to the weight connection between perceptron \( i \) and \( j \) which are represented by the lines in Fig. 2.2.

To begin the example, inputs are fed forward through the network as follows, where all weight values are initialized randomly:

\[
P_4 = \Phi(w_{14} * P_1 + w_{24} * P_2 + w_{34} * P_3)
\]
\[
P_5 = \Phi(w_{15} * P_1 + w_{25} * P_2 + w_{35} * P_3)
\]
\[
P_6 = \Phi(w_{46} * P_4 + w_{56} * P_5)
\]

After the feed forward is completed, backpropagation is used to begin the learning phase. A backpropagation step for a specific input set and target output starts by calculating the error at the output perceptron(s). This error is the difference between the target output and the predicted output multiplied with the activation function derivative on that output point. For instance, the sigmoid
function derivative is \( F'(x) = F(x) \ast (1 - F(x)) \). Thus,

\[
Output_j \text{Error} = (y_j - \hat{y}_j) \ast F'(\hat{y}_j)
\] (2.4)

This means that if the example is used the output error is calculated as:

\[
P_6 \text{Error} = (P_6 - \hat{P}_6) \ast \hat{P}_6 \ast (1 - \hat{P}_6)
\] (2.5)

After the error is calculated for the output layer, the error for each perceptron in the hidden layer(s) can be calculated by going backwards layer by layer. The error for a neuron in a hidden layer is the sum of the products between the errors of the perceptrons in the next layer and the weights of the connections to those perceptrons, multiplied by the derivative of the activation function.

\[
Hidden_i \text{Error} = \sum (OutputError_i \ast w_{ij}) \ast F'(HiddenOutput_i)
\] (2.6)

which, in the case of the example, results in:

\[
P_4 \text{Error} = (P_6 \text{Error} \ast w_{46}) \ast P_4 \ast (1 - P_4)
\]

\[
P_5 \text{Error} = (P_6 \text{Error} \ast w_{56}) \ast P_5 \ast (1 - P_5)
\] (2.7)

These errors are then used to calculate the changes of the weights as a result of the current input pattern and target outputs. The delta of a weight is the product of the input perceptron output value with the error of the output perceptron for that connection.

\[
\Delta w_{ij} = Output_i \ast Error_j
\] (2.8)
In the case of the example it results in the following:

\[
\Delta w_{46} = P_4 \times P_6 \text{Error} \\
\Delta w_{56} = P_5 \times P_6 \text{Error} \\
\Delta w_{14} = P_1 \times P_4 \text{Error} \\
\Delta w_{24} = P_2 \times P_4 \text{Error} \\
\Delta w_{34} = P_3 \times P_4 \text{Error} \\
\Delta w_{15} = P_1 \times P_5 \text{Error} \\
\Delta w_{25} = P_2 \times P_5 \text{Error} \\
\Delta w_{35} = P_3 \times P_5 \text{Error}
\]  

This system is repeated for all input patterns and the deltas are accumulated. At the end of a learning iteration, the actual weights are changed with the accumulated deltas for all the training patterns multiplied with a learning rate, typically between 0 and 1.

\[
\Delta w_{ij} \text{Final} = \sum \Delta w_{ij} \text{Input}_k \\
w_{ij} = w_{ij} + (\Delta w_{ij} \text{Final} \times \text{LearningRate})
\]  

To correspond with the example:

\[
\Delta w_{46} \text{Final} = \Delta w_{46} \text{Input}_1 + \Delta w_{46} \text{Input}_2 \\
w_{46} = w_{46} \text{LearningRate} \times \Delta w_{46} \text{Final}
\]
Thus, to train an ANN using Backpropagation the steps in Algorithm 1 must be followed.

**Algorithm 1: Backpropagation trained ANN algorithm**

1. Initialize weights with random values;
2. For each input and target output do:
   1. Compute the predicted output from the input;
   2. Compute output perceptrons error;
   3. Compute hidden perceptrons error;
   4. Compute weights deltas;
   5. Add the weights deltas to the accumulated delta;
3. Adjust weights and bias deltas using accumulated deltas;

2.4 ANN Classification

An Artificial Neural Network Classifier works in much the same way as discussed in previous section on ANN regression. It can use Backpropagation or PSO to perform the training. However, the last output layer contains a Softmax for its activation function. As an ANN classifier will have a perceptron representing each possible cluster regression cannot be used. The Softmax classifier calculates which class the input belongs to by determine which cluster is most likely. Every output perceptron will be rated between 0 and 1 with the perceptron closest to 1 being deemed the class after the Softmax is performed.

The Softmax classifier is the generalization of the binary Logistic Regression to multiple classes. It uses a function mapping \( f(x_i; W) = Wx_i \) where the error scores are interpreted as the non-normalized log probabilities for each class and use cross-entropy loss with the form of:

\[
L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)
\]  

(2.12)

where \( f_j \) represents the j-th element of the vector of class scores \( f \). The full loss for the data set is the mean of \( L_i \) over all training examples together with a regularization term \( R(W) \). The function:
\[ f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \]  

(2.13)
is known as the Softmax function. It takes a vector of arbitrary real-valued scores and creates a vector of values between 0 and 1 that sum to one.

The Softmax classifier is minimizing the cross-entropy between the estimated class probabilities and the true distribution, which in this interpretation is the distribution where all probability mass is on the correct class. As the cross-entropy can be written in terms of entropy and the Killback-Leilber divergence as \( H(p, q) = H(p) + D_{KL}(p||q) \) and the entropy of the delta function \( p \) is zero, this is also equivalent to minimizing the KL divergence between the two distributions. The cross-entropy wants the predicted distribution to have all of its mass on the correct answer.

2.5 Particle Swarm Optimization trained ANN

Particle Swarm Optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995 [8]. It is modeled after the behavior of flocking birds or schooling fish. PSO is relatively easy to implement and has few parameters that require adjusting for it to work optimally.

PSO is initialized with a group of random particles and then searches for optima by updating generations. In every iteration each particle is updated by following two best values. The first best value being the best solution that particular particle has achieved so far and the second being the best fitness value any particle has obtained so far, these are known as \( p_{best} \) and \( g_{best} \), respectively.

After the two best values are found, the particles update their velocity and positions. The velocities are updated via (2.14) where \( V_i \) represents the newly calculated velocity, \( w \) represents a constant called inertia weight, \( c1 \) and \( c2 \) are fixed values of 2.05 that were verified in a study by Shi [9], \( r1 \) and \( r2 \) represent randomly generated numbers between 0 and 1, \( Pos_{pbest} \) is the position of a particles best fitness value, \( Pos_{gbest} \) is the position of the global best fitness value of all particles, and \( Pos_{current} \) is the current particles position. \( w \) is often varied between 1.4 and 0.2 which decreases linearly depending on the training iteration number. The number of particles can
also be varied but usually is roughly two or three times the number of input dimensions.

\[ V_1 = \omega v_i + c_1 \cdot r_1 \cdot (P_{os_{\text{best},i}} - P_{os_{\text{current},i}}) + c_2 \cdot r_2 \cdot (P_{os_{g\text{best},i}} - P_{os_{\text{current},i}}) \]  

(2.14)

After the set of velocities is calculated, the new position of the particle is calculated by adding the corresponding velocity and current position point together. The general work-flow of PSO is as follows:

**Algorithm 2:** General PSO algorithm.

```plaintext
foreach particle do
    initialize particle with random position;
while exit criteria is not met do
    foreach particle do
        calculate particle’s fitness value;
        if the fitness value is better than the best fitness value in particle’s history set value as the new \( p_{\text{best}} \);
        choose the particle with the best \( p_{\text{best}} \) of all particles as the \( g_{\text{best}} \);
        if the value is better than the best value of global particles in history, set the best particle’s position as the new \( g_{\text{best}} \);
        calculate each particles new velocity and position;
```

A PSO trained ANN uses PSO to determine the optimal weights of the ANN. Every particle, \( P \), contains several important characteristics: the weights of all of the perceptron connections which represent the position of the particle, the current error measurement of the particle, the current velocity of the particle, and the global best error measurement and position.

On every iteration there is also a chance for each particle to die and then be randomly reinitialized. Kennedy and Eberhart inspired the use of this concept [10]. When a particle dies and is reinitialized it has a chance of reappearing at a location that is more optimal than where other particles currently reside. This prevents particles from converging at a local optima instead of the global optimum.
2.6 k-Means Clustering

K-Means is one of the most common unsupervised learning algorithms that solves clustering problems. It follows a straight forward path to classify a given data set into \( k \) number of clusters. The central idea of k-Means is to find \( k \) centroids, one representing each cluster. These centroids should be carefully placed as different centroid locations will create different results. Ideally, they should be placed as far away from each other as is logically possible.

After the location of the centroids are determined, each point belonging to a given data set is associated with the nearest centroid. Once no new points are pending, \( k \) new centroids are calculated as bary-centers of the clusters resulting from the previous grouping. After these new centroids are created, a new binding has to be done between the same data set points and the nearest new centroid. This loop is performed until the centroids cease moving or a maximum number of iterations is completed as convergence will not always occur.

K-Means clustering was first proposed by MacQueen in 1967 [11]. It is among the most commonly used clustering algorithms [12] and has been successfully used in a variety of topics, such as market segmentation [13] and astronomy [14] to cluster data points.

The k-Means algorithm is generally composed of the following steps:

1. Place \( k \) points into the space represented by the objects that are being clustered. These points represent initial group centroids.

2. Assign each object to the group that has the closest centroid.

3. When all objects have been assigned, recalculate the positions of the \( k \) centroids.

4. Repeat steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

2.7 Spectral Clustering

Spectral Clustering makes use of the spectrum (or eigenvalues) of the similarity matrix of the data. It examines the connectedness of the data, whereas k-means looks upon the compactness
to assign clusters [15]. This allows Spectral and k-Means clustering to find clusters of different shapes and styles.

The goal of Spectral Clustering is to cluster data that is connected but not necessarily compact or clustered within convex boundaries. Before the basic step of the algorithm are discussed a few preparation steps must first be taken.

First, an affinity is a metric that determines how close two points are in a given point space. The metric of this affinity must be guessed or learned. Thus, given two data points projected into matrix $\mathbb{R}^{n \times n}$ where $n$ is the total number of points, the affinity, $A_{ij}$ that is positive, symmetric, and depends on the Euclidean distance between data points is defined as:

$$A_{i,j} \simeq \exp(-\alpha ||x_i - x_j||^2)$$  \hspace{1cm} (2.15)

where $x_i$ and $x_j$ are two different points.

$A_{i,j} \simeq 1$ when points are close in $\mathbb{R}^{n \times n}$ and $A_{i,j} \to 0$ when points are far apart. Close data points then end up in the same cluster with data points in different clusters being far away. Data points in the same cluster may also be far away, even farther away than points in different clusters. The goal then is to transform the space so that when two points are close, they are always in the same cluster, and when they are far, they are always in different clusters.

Generally the Gaussian Kernel, $K$, is used directly or the Graph Laplacian, $Q$ is formed. Usually, the Graph Laplacian is described as a discrete approximation to the Laplacian from the Leplace-Beltrami operator. The non-normalized Graph Laplacian is defined as the difference of two matrices:

$$L_{i,j} = D_{i,j} - W_{i,j}$$  \hspace{1cm} (2.16)

where $D$ is the diagonal degree matrix assigned to the graph vertices, and $W$ is a matrix of positive weights, $W_{i,j}$ assigned to the graph edges. To evaluate the expectation value of $L$ with vector $x$:

$$x^tLx = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j} ||x_i - x_j||^2$$  \hspace{1cm} (2.17)
when using an adjacency matrix, the weights are all 1:

\[ Q_{i,j} = W_{i,j} = \delta_{i,j} \]  

(2.18)

where \( \delta_{i,j} \) represents the gaussian kernel. Apart from the diagonal and some normalization, the Graph Laplacian for an adjacency matrix is a low order approximation to the Gaussian Kernel, which is a good approximation when two points are in the same cluster.

All Laplacians, \( L \), share the concept of a degree matrix, \( D \). The degree matrix is a diagonal matrix that measures the degree at each node

\[ D_{i,i} = \sum_{j} a_{i,j} \]  

(2.19)

which can be thought of as a mean-field average of the affinity; \( D \) is a normalization factor for \( Q \) so that the cluster affinities are balanced across different clusters. The Laplacian can be defined as a Simple Laplacian, Normalized Laplacian, Generalized Laplacian, Relaxed Laplacian, Ng, Jordan, Weiss Laplacian, and note the related smoothed Kernel for k-Means Kernel Clustering.

If good clusters can be identified then the Laplacian, \( L \), is approximately block-diagonal, with each blocking defining a cluster. Thus, if we were to have three major clusters \( (C_1, C_2, C_3) \), it would be expected that:

\[
\begin{bmatrix}
L_{1,1} & L_{1,2} & L_{1,3} \\
L_{2,1} & L_{2,2} & L_{3,2} \\
L_{3,1} & L_{3,2} & L_{3,3}
\end{bmatrix}
\sim
\begin{bmatrix}
L_{1,1} & 0 & 0 \\
0 & L_{2,2} & 0 \\
0 & 0 & L_{3,3}
\end{bmatrix}
\]  

(2.20)

where \( L_{C_1,C_1} \) corresponds to a subblock for cluster \( C_1 \), etc. These blocks let us identify clusters with non-convex boundaries. It is also expected that the three lowest eigenvalues and eigenvectors \( (\lambda_i, v_i) \) of \( L \) each correspond to a different cluster. This occurs when each eigenvector corresponds to the lowest eigenvector of some subblock of \( L_{c,c} \). That is, if \( Lv_i = \lambda_i v_i \) are the lowest eigenvalue, eigenvector pairs in the full space, and \( L_{C_1,C_1}v_{C_1} = \lambda_{C1}v_{C1} \) is the lowest eigenvalue,
eigenvector pair of block $C_1$, then $v_{C_1}$ is a good approximation to one of the lowest three $v_i$. Likewise for subblocks $C_2$ and $C_3$. More importantly, this also restricts the eigenvalue spectrum of $L$, so that the set lowest three full space eigenvalues consists of the lowest subblock eigenvalues $\lambda_i, i = 1, 2, 3 = \lambda_{C_1}, i = 1, 2, 3$. Also, to identify $k$ clusters, the eigenvalue spectrum of $L$ must have a gap. Frequently, this gap is hard to find.

With all of this in mind we can break the Spectral Clustering algorithm into several steps:

1. Project data into $R^{n \times n}$.

2. Define an Affinity matrix $A$, using a Gaussian Kernel, $K$ or an Adjacency matrix $A_{i,j} = \delta_{i,j}$

3. Construct Graph Laplacian, $L$, from $A$

4. Solve an Eigenvalue problem, such as $Lv = \lambda Dv$

5. Select $k$ eigenvectors $\{v_i, i = 1, ..., k\}$ corresponding to the $k$ lowest (or highest) eigenvalues $\{\delta_i, i = 1, k\}$, to define a $k$-dimensional subspace $P^t L P$

6. Form clusters in this subspace

2.8 Literature Review

Multiple works have attempted to regress this data set using a variety of methods. One of the more extensive attempts of predicting forest fire burn areas with the Montesinho data set is from Paulo Cortez and Anibal Morais [16]. They use five different models in an attempt to find one suitable for regressing the data: Multiple Regression (MR), Decision Trees (DT), Random Forests (RF), ANNs, and Support Vector Machines (SVM). Each of the five models were then tested using four different sets of inputs. The four input models were derived to determine if any of the variables had impact on the error rates if they were to be eliminated from the input stage.

Cortez and Morais believe that SVM has a theoretical advantage over an ANN due to the absence of local minima in the model optimization phase. They loosely confirmed their beliefs by showing their strongest testing results were obtained using SVM. Even though the SVM results
were the best, with an root means squared error (RMSE) value of 64.7, it was by a fairly insignif-
ificant amount as the ANN trained with backpropagation obtained an RMSE of 66.9. These differ-
ences could be attributed to a non-optimal ANN architecture. Non-optimal in this case mean-
ing that if different activation functions or different a number of hidden layers or perceptrons were
to be used results could be improved.

The use of five different training models and four input methods resulted in 20 different systems
tested. Their RMSE rates varied between the range of 64.7 and 68.9. This is a relatively small
difference between all of their results. This could mean that taking away input items may have little effect on the RMSE scores. Their methodology did not account for outliers very well so they are able to predict many of the smaller sized fires but struggle with fires of larger size.

Safi and Bouroumi [17] predicted forest fire burn areas of the Montesinho data set as well. They use backpropagation to train the ANN and aim to minimize the global error on the output layer. A variety of architectures are tested with varying levels of perceptrons and hidden layers. Their best results were obtained using a single hidden layer ANN consisting of 35 perceptrons that resulted in a 10% rate of error.

Safi and Bouroumi also found that their results hinge on the number of training iterations performed. There is no obvious pattern to the number of training iterations that should be performed but results vary depending on the the choice. For instance, at 10,000 training iterations they have a roughly 5% variation in the error rate, while at 50,000 training iterations they have roughly a 60% variation in error rate. These numbers are found using identical ANN architectures demonstrating that the number of training iterations could be something important to consider. However, that large amount of training iterations on a data set so small could likely result in over-fitting and the found results should be documented multiple times to verify the number of iterations is indeed what is effecting the error score.

Castelli, Vanneschi, and Popovic [18] proposed an intelligent system based on genetic pro-
gramming for the prediction of burned areas. They only consider data relating to FWI character-
istics and meteorological data in order to build predicting models, employing geometric semantic
genetic operators for genetic programming. Strong results are obtained using this method along with testing other methods such as Random Forests, Radial Basis Function Networks, ANNs, and basic Linear Regression. The authors calculate error using mean absolute error (MAE) between predicted and target values. Their ANN system obtains a MAE of 33.8 while their geometric semantic genetic programming method obtains a 12.9 MAE. These findings are very strong but do not contain RMSE values. If RMSE is used as the error metric the numbers would most likely increase due to the outlier nature of the data set. Their work showed that methods other than ANNs are viable for regressing this data set.

Jain and Bhatia [19] use Support Vector Regression to regress the Montesiho data set. The goal of their research is to identify the best parameter settings using a grid-search and pattern search technique as well as to compare the prediction accuracy among the models using different data sorting methods, random sampling and cross validation. They found that E-SVR performs better using various fitness functions and variance analysis. Their best RMSE result being 63.51 with the E-SVR method using 10-fold cross validation.

Wang, Wang, Due, Wang, Liang, Zhou, and Huang [20] once again attempted to regress the data set and use a variant of particle swarm optimization (PSO), known as Immune Particle Swarm Optimization (IPSO) with Support Vector Regression (SVR), to execute the task. The immune portion of the algorithm name refers to the Immune Algorithm (IA). IA is a meta-heuristic search algorithm, which is concerned with keeping diversity of the solution group and avoiding particles from becoming stuck in local optimums when used in conjunction with PSO [21]. SVR differs from SVM work [16] as it finds a hyperplane which can accurately predict the distribution of information, but not the plane used to classify the data.

This work uses RMSE as the fitness value measurement of each particle in the PSO algorithm. This was done in an attempt to avoid over fitting and to get a high generalization performance value. A novel approach was also presented by having an affinity for each particle to clone itself. Thus, a particle with a larger affinity would clone more offspring to protect its good genes and accelerate the convergence rate of the algorithm. An aspect of mutation was also introduced into
the PSO algorithm to assist in more reliably finding the global optimum. After the clonal copy and mutation process, the best individual mutated particle for each cloned group is selected to compose the next generation. The authors found that there is nearly no difference when using PSO versus IPSO with both achieving an RMSE of approximately 64.1. Thus, using PSO or IPSO alone did not net a very significant decrease in RMSE compared to using a backpropagation trained ANN. Though related to this work, the work presented in this thesis is substantially different because as opposed to using the PSO algorithm alone. PSO is used in conjunction with an ANN to act as the training algorithm instead of backpropagation.

A PSO trained ANN has not been used on the Montesinho fire data set before. However, many trials have been done on various tasks with a PSO trained ANN that demonstrate its validity as a training method. Xue-tong [22] used a PSO trained ANN to predict the prices of petroleum throughout time. He found that using a PSO trained ANN gave better results than that of backpropagation. The primary finding in this work was that convergence occurs at a faster and more consistent rate when using PSO as opposed to backpropagation to train an ANN.

Chau [23] attempts to predict accurate water stages of the Shing Mun River. Accurate water stage prediction would allow the pertinent authority to issue flood advisories or warnings in a more timely manner which would allow earlier planning for evacuations. Two separate data sets are used for trials in Chau’s research. Both results confirm that a PSO trained ANN performs better than a backpropagation trained ANN having lower RMSE scores. The backpropagation based ANN had a 0.29 RMSE and the PSO-based ANN a 0.16 RMSE for one trial and a 0.24 versus 0.14 RMSE in the other trial. Both of these results show statistical improvement while using a PSO trained ANN.

Mohammadi and Mirabendini [24] used a data set from the University of California at Irvine (UCI) machine-learning database [25] that contains 21 attributes and 1,000 instances to determine if an instance has good or bad credit. They found that a PSO trained ANN had the lowest mean-square error (MSE) and best accuracy among Levenberg-Marquardt, Gradient Descent, Gradient Descent with Momentum, Gradient Descent with Adaptive Learning Rate, and Gradient Descent with Momentum and Adaptive Learning Rate. This gives PSO winning marks against even more
Niu and Li [26] show the merit of using a hybrid PSO-ANN approach as well. They performed tests using the Iris, New-Thyroid, and Glass data sets and then train an ANN using Backpropagation, Genetic Algorithms, and PSO-ANN for each data set. In all cases the hybrid PSO-ANN outperformed the other methods obtaining both the highest accuracy and lowest MSE scores.
PREDICTING FOREST FIRE SIZE WITH PARTICLE SWARM TRAINED NEURAL NETWORKS

This chapter proposes a new approach to regressing the Montesinho dataset that combines multiple computational intelligence and data mining techniques. First, the input method of the dataset is different than any previous works as all categorical inputs use 1-of-C encoding. This results in expanding the dataset from 12 inputs to 29. Second, a PSO trained ANN is used to attempt the regression of determining burn areas of the forest fires. An exhaustive and systematic evaluation of the use of backpropagation in conjunction with multiple activation functions is also completed.

3.1 Data Preprocessing

The data set from Montesinho has proven very difficult to use as a source for training an ANN to predict the burn area of other fires in the area as there are only 517 entries in the set making it relatively small. The dataset is also heavily right skewed with a large percentage of the burn areas in the dataset being 0.0. This is due to the fact that anything with less than 100 $m^2$ of burn area is considered, for the purposes of the data gathering, statistically insignificant and is tallied with a 0.0 burn area. The skewed nature and small size of this dataset make it very difficult to predict fires of larger sizes accurately as they are outliers and therefore heavily underrepresented.

For the data to be used in the training of an ANN, some amount of preprocessing had to be performed to mitigate some of the problems inherit in the dataset. The first problem addressed was that of the skewed nature of the data: 247 out of the 517 entries contain a 0.0 entry for the output. To attempt to reduce the amount of skew a logarithm transform is applied to the outputs in the form of $y = \ln(output + 1)$ to bring in the tail. The results of this application can be seen in Fig. 3.1. As can be seen, after the log transform is applied the skew is lessened considerably but is still problematic, even if much less so than previously.

The second issue of the data set to be handled is normalization of the numerical fields. In this
case, the numerical fields would be the inputs X, Y, FFMC, DMC, DC, ISI, temp, RH, wind, and rain, as well as the output post transform. The normalized value, $e_i$, for variable $E$ in the $i^{th}$ row is calculated as seen in eq. (3.1) where $E_{min}$ is the minimum value for variable E and $E_{max}$ is the maximum value for variable $E$. This results in a normalization where all numerical values are between 0 and 1 which is an optimal range to use when training an ANN and limits the ANN favoring certain data fields over others due to the usages of different scales.

$$Normalized(e_i) = \frac{e_i - E_{min}}{E_{max} - E_{min}}$$

(3.1)

Finally, data fields that are categorical in nature (e.g. day and month) are encoded into a binary array instead of the traditionally normalized values. This is known as 1-of-C encoding [27, 28]. This means that instead of normalizing between 0 and 1 each category is given its own slot. As an example, the day field is broken into each day of the week starting with Sunday. This means that if it were to be Tuesday, for instance, it is represented as a binary array of $[0, 0, 1, 0, 0, 0, 0]$ and Thursday would be $[0, 0, 0, 0, 1, 0, 0]$. The months field is done in a similar fashion starting with January. Doing this resulted in a total of 29 inputs as opposed to the original 12 as can be seen in Fig. 3.2. The data is processed the same way in the following chapter.
3.2 Artificial Neural Network Architectures

One of the contributions of this thesis is determining if variations in ANN architectures effect the RMSE rate of the regression problem for this dataset. There are several items used for the permutations of the ANN architectures. All of the ANNs contain only a single hidden layer. The amount of nodes in this hidden layer is varied between 2 and 200 for backpropagation trained ANN and 2 and 100 for particle swarm optimization trained ANNs. The activation functions are also varied. There are five activation function used in the different ANN architectures: Rectifier (eq. (3.4)),Hyper Tangent (eq. (3.3)), Hard Sigmoid, Sigmoid (eq. (3.2)), and Linear (eq. (3.5)). Each of these activation functions could occur in two different locations – either between the input and hidden layers or between the hidden layer and output. This results in 25 possible configurations of activation layers. The PSO trained ANN uses fewer variations in architectures as viewing the results of the large backpropagation study shows little correlation between activation function and RMSE. The PSO trained ANN used hyper tangent for both activation functions. Both the backpropagation and PSO trained ANN use the 12 and 29 input methods to determine if input method had any effect on RMSE values as well. With all of these variations in mind this results in 9,900 different backpropagation systems along with 196 PSO trained ANN systems for a grand total of 10,096 unique architectures tested for the regression task. The activation function equations can

(a) Before 1-of-C encoding. (b) After 1-of-C encoding

Figure 3.2: Input architecture before and after applying 1-of-C encoding.
be seen in eqs. (3.2) – (3.5). The hard sigmoid function, which is not listed, is an element-wise approximation of the sigmoid function.

\[ f(x) = \frac{1}{1 + e^{-x}} \]  \hspace{1cm} (3.2)

\[ f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \]  \hspace{1cm} (3.3)

\[ f(x) = \ln(1 + e^x) \]  \hspace{1cm} (3.4)

\[ f(x) = a + bx \]  \hspace{1cm} (3.5)

To aid in the task of systematically evaluating backpropagation architectures, a python library, Keras [29], is used to take away some of the burden of coding a very large amount of independent ANN architectures. Keras provides all of the most common and several less often used activation functions along with easy implementation of different ANN architectures. A series of loops are set up to allow all of the permutations of hidden layer perceptrons and activation functions to occur and output to relevantly named files. Once all of the files are finished being created they are statistically analyzed within MATLAB and graphs created using Plotly. All Backpropagation trials are cross validated using 5-Fold validation with 300 training epochs for each fold.

There were no suitable libraries are available for a PSO trained ANN in Python so it is created from scratch. Common PSO values are used for variables with \( C1 = C2 = 2.05 \), 20 particles, 50 max epochs, and an exit error of 17. The inertial weight, \( w \), is linearly decreased from 1.5 to 0.4 based on the epoch number. A probability of death was also added to each particle at 5%. If a particle “dies” it randomly reinitializes. Fewer variations of PSO trained ANN were performed due to the fact that each trial took much longer to run along with the low correlation between activation function and RMSE score found in backpropagation trials.
The same code structure is used for all of the trials of PSO trained ANNs to keep measurements consistent. Results of the PSO trained ANN are also output to files and analyzed using MATLAB with the graphs created by Plotly.

All of the single hidden layer Backpropagation and PSO trained ANN are performed on the same hardware using the same programming language. The system that is used is a 2013 MacBook Pro with a 2.3 GHz Intel Core i7 CPU, 16 GB of 1600 MHz DDR3 RAM, and an NVIDIA GeForce GT 750M 2048 MB GPU. All tests were conducted by using a combination of Python, Keras, and Theano inside of the PyCharm IDE.

3.3 Results

3.3.1 Backpropagation

Tables 3.1 and 3.2 contain information discussed in this section. First, the variance levels of each input method will be looked at to determine the volatility of the RMSE values for each base layer activation function. For the 12 Input method, four out of the five activation functions (Rectifier, Hyper Tangent, Sigmoid, and Linear) had very similar variances at 7.83, 7.76, 8.78, and 8.36, respectively. Meaning, there is minimal amounts of outlier RMSE values and that the vast majority of the values in these four categories occur near the mean. Hard Sigmoid had a large variance value of 10,645,926. During the trials, at least one large outlier RMSE value occurred. This is confirmed by looking at the maximum value of Hard Sigmoid for the 12-Input method which is 93,148.42. This large value alone accounts for the large upward swing in variance. If there are other RMSE values near this large value, it will cause the variance to quickly climb.

Next, some common statistical measurements will be examined for the 12-Input method. The median is the first measurement to view and is being approached before the mean due to the large amount of trials and possible large outlier values like what occurred in the Sigmoid and Hard Sigmoid fields, which both had very large max values. The median values of all activation functions are close together with a range between 53.88 and 54.21. This shows that most of the trial results had similar results except for perhaps the Sigmoid and Hard Sigmoid fields. Sigmoid
Table 3.1: Common statistical data for the 12 Input method using backpropagation. Each activation function column represents all permutations of ANN architectures assuming corresponding function is used in first layer.

<table>
<thead>
<tr>
<th>Activation</th>
<th>Rectifier</th>
<th>Tanh</th>
<th>SGM</th>
<th>Hard SGM</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance</td>
<td>7.83</td>
<td>7.76</td>
<td>8.78</td>
<td>10,645,926</td>
<td>8.36</td>
</tr>
<tr>
<td>Mean</td>
<td>52.87</td>
<td>52.94</td>
<td>99,296,320,853</td>
<td>219.58</td>
<td>52.75</td>
</tr>
<tr>
<td>Min</td>
<td>43.50</td>
<td>44.14</td>
<td>44.03</td>
<td>45.31</td>
<td>43.87</td>
</tr>
<tr>
<td>Q1</td>
<td>51.72</td>
<td>51.83</td>
<td>52.144</td>
<td>52.64</td>
<td>51.33</td>
</tr>
<tr>
<td>Median</td>
<td>53.67</td>
<td>53.72</td>
<td>54.00</td>
<td>54.21</td>
<td>53.62</td>
</tr>
<tr>
<td>Q3</td>
<td>54.88</td>
<td>54.95</td>
<td>55.55</td>
<td>55.37</td>
<td>54.81</td>
</tr>
<tr>
<td>Max</td>
<td>57.33</td>
<td>57.19</td>
<td>93,400,333,303,845</td>
<td>93148.42</td>
<td>57.54</td>
</tr>
</tbody>
</table>

Table 3.2: Common statistical data for the 29 Input method using backpropagation. Each activation function column represents all permutations of ANN architectures assuming corresponding function is used in first layer.

<table>
<thead>
<tr>
<th>Activation</th>
<th>Rectifier</th>
<th>Tanh</th>
<th>SGM</th>
<th>Hard SGM</th>
<th>Linear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance</td>
<td>7.81</td>
<td>6.87</td>
<td>1.09e+32</td>
<td>144,504</td>
<td>7.83</td>
</tr>
<tr>
<td>Mean</td>
<td>52.81</td>
<td>53.01</td>
<td>3,961,252,082</td>
<td>81.23</td>
<td>52.86</td>
</tr>
<tr>
<td>Min</td>
<td>52.48</td>
<td>44.66</td>
<td>44.23</td>
<td>44.25</td>
<td>43.52</td>
</tr>
<tr>
<td>Q1</td>
<td>51.31</td>
<td>51.94</td>
<td>52.03</td>
<td>51.96</td>
<td>51.61</td>
</tr>
<tr>
<td>Median</td>
<td>53.63</td>
<td>53.71</td>
<td>54.11</td>
<td>53.98</td>
<td>53.73</td>
</tr>
<tr>
<td>Q3</td>
<td>54.85</td>
<td>54.73</td>
<td>55.42</td>
<td>55.42</td>
<td>54.89</td>
</tr>
<tr>
<td>Max</td>
<td>57.22</td>
<td>56.90</td>
<td>3,269,161,898,563</td>
<td>9039.46</td>
<td>57.32</td>
</tr>
</tbody>
</table>

and Hard Sigmoid have mean values of 99,296,320,853 and 219.58, respectively. This indicates that there are several values that occur a large distance away from the median, especially with the sigmoid field. The other three activation fields had very steady RMSE results values as the median and mean values are nearly equal. This means that Rectifier, Hyper Tangent, and Linear all produce steady results while Sigmoid and Hard Sigmoid are prone to outlier occurrences. It is therefore better to not use Sigmoid or Hard Sigmoid as the first activation function in an ANN if using the 12-Input method.

For the 29-input method, the median values are again very close in values, similar to the 12 Input method. They range between 53.63 and 54.11. Mean values for Rectifier, Hyper Tangent, and Linear are also close together at 52.81, 53.01, and 52.86. Sigmoid and Hard Sigmoid once
again suffered from outlier values occurring which raised the mean values to 3,961,252,082 and 81.23, respectively. It is likely that Sigmoid had many large value RMSE encounters as the max value is very large. Hard Sigmoid does not have as large of a max nor mean relatively speaking. This means that outlier occurrences were most likely fairly rare. Hard Sigmoid and Sigmoid both have very large variance rates due to the amount or severity of outlier occurrences. An outlier event most likely occurs when the training set fed contains too large of a ratio of target data that is of value 0.0. It is possible that nearly all training data could have 0.0 output targets and very few of the test data items would then have the 0.0 values. This would result in a circumstance of a very large RMSE value occurring.

<table>
<thead>
<tr>
<th>Input Method</th>
<th>Input Activation</th>
<th>Number Hidden Nodes</th>
<th>Output Activation</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hard SGM</td>
<td>52</td>
<td>Linear</td>
<td>45.31</td>
</tr>
<tr>
<td></td>
<td>Linear</td>
<td>21</td>
<td>Rectifier</td>
<td>43.88</td>
</tr>
<tr>
<td></td>
<td>Rectifier</td>
<td>127</td>
<td>Hard Sigmoid</td>
<td>43.50</td>
</tr>
<tr>
<td></td>
<td>Tanh</td>
<td>22</td>
<td>Rectifier</td>
<td>44.14</td>
</tr>
<tr>
<td></td>
<td>SGM</td>
<td>116</td>
<td>Hard Sigmoid</td>
<td>44.03</td>
</tr>
</tbody>
</table>

Table 3.3: Optimal ANN architectures for 12 Input Method for each of the five types of first layer activation functions using backpropagation as the training method.

<table>
<thead>
<tr>
<th>Input Method</th>
<th>Input Activation</th>
<th>Number Hidden Nodes</th>
<th>Output Activation</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hard SGM</td>
<td>187</td>
<td>Rectifier</td>
<td>44.25</td>
</tr>
<tr>
<td></td>
<td>Linear</td>
<td>61</td>
<td>Rectifier</td>
<td>44.52</td>
</tr>
<tr>
<td></td>
<td>Rectifier</td>
<td>75</td>
<td>Rectifier</td>
<td>42.48</td>
</tr>
<tr>
<td></td>
<td>Tanh</td>
<td>199</td>
<td>Hard Sigmoid</td>
<td>43.71</td>
</tr>
<tr>
<td></td>
<td>SGM</td>
<td>86</td>
<td>Hard Sigmoid</td>
<td>44.23</td>
</tr>
</tbody>
</table>

Table 3.4: Optimal ANN architectures for the 29 Input Method for each of the five types of first layer activation functions using backpropagation as the training method.

The minimum RMSE scores of each activation pair with each input method are more interesting and pertinent and can be seen in Figs. 3.3–3.7. These figures present the minimum RMSE values of the 12 and 29 input methods grouped by base and secondary activation functions. Each base activation function has its own graph with its secondary activation function minimums displayed.
Figure 3.3: Minimum RMSE values using the Hard Sigmoid as the first layer activation function as paired with all second layer activation function permutations.

The figures show, more often than not, the 29-Input method outperforms the 12-Input method and is also, more often than not, the overall minimum RMSE value of the group. The only base activation function performing consistently poorer using the 29-Input method is the Hyper Tangent group. All other base activations show favoring of the 29-Input method. This solidifies that the 29-Input method when used in conjunction with backpropagation training is more likely to net the best global RMSE score. In this case the lowest score found overall contains the 29-Input method followed by a Rectifier activation function which is followed by 75 hidden layer perceptrons and another Rectifier activation function for the second layer. The lowest 12-Input RMSE score also presented with the Rectifier activation function as the first layer function. This shows a trend that it is best to use Rectifier as the first activation function. The second depends on which input method is used and will need to be adjusted accordingly. The other lowest scoring architectures can be seen in Tables 3.3 and 3.4.
Figure 3.4: Minimum RMSE values using the Linear as the first layer activation function as paired with all second layer activation function permutations.

Figure 3.5: Minimum RMSE values using the Rectifier as the first layer activation function as paired with all second layer activation function permutations.
Figure 3.6: Minimum RMSE values using the Sigmoid as the first layer activation function as paired with all second layer activation function permutations.

Figure 3.7: Minimum RMSE values using the Hyper Tangent as the first layer activation function as paired with all second layer activation function permutations.
3.3.2 Particle Swarm Optimization Trained ANN

Statistical data for PSO trained ANN is captured inside Table 3.5. The 12-Input method has a variance of 11,787.66 versus 1,331.53 for the 29-Input method. This shows much more volatility in the 12-Input method. The optimal number of hidden layer perceptrons for the 29 Input method is 21 and the optimal number for the 12 Input method for is 8. These are both small numbers of hidden layer perceptrons indication that a larger network is not indicative of superior performance when using an hybrid PSO-ANN.

The relatively large difference in mean RMSE values is important to note. Even though the 12-Input method produced the overall lowest RMSE score it has a high mean value of 88.43. This is far from the minimum 15.65 value and coupled with the large variance it is fair to assume that the 12-Input method is highly volatile. This means that it is capable of finding a better overall RMSE score but is also more likely to have larger amounts of undesirable outcomes in the process. With this in mind the 29-Input may still be more desirable to use even though it has a slightly higher RMSE value to to it being far more consistent with predictions than the 12-Input method. Fig. 3.8 helps to visualize the volatility of the 12-Input method. The large difference in mean and larger median value clearly demonstrate how the 12-Input method is much more volatile than the 29-Input method. The drastic difference in variance levels further solidifies this.

The major disadvantage to using the 29-Input method comes from its time scaling. Fig. 3.9 shows how the time to complete a training duration increases as the number of perceptrons in the hidden layer increases. The two different lines represent quadratic fits from each data set of result. It can be seen that the 12-Input method is much closer to linearly increasing than the 29-Input method. Not only is the 29-Input method always longer in duration than the 12-Input method it is increasing at nearly an exponential rate. This means that scaling up the very large or deep ANNs will result in extremely long training times if using the 29-Input method. The user must judge for themselves if the longer training durations is worth the more consistent results. Typically, the extended training time will be worth the more consistent results as the ANN is not designed to be run online. It should only need trained as more data becomes available, which most likely will
Figure 3.8: Comparison of the 12 and 29 Input method with common statistical measurements using PSO as the training method.

Figure 3.9: 12 and 29 Input method time to complete a training cycle using PSO as the training method.
be infrequently. The user choosing the 12-Input methods may net better results in the end but the better results also may never occur due to the volatility of the results of the 12-Input method.

<table>
<thead>
<tr>
<th></th>
<th>12-Inputs</th>
<th>29-Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Variance</strong></td>
<td>11787.66</td>
<td>1331.53</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td>88.43</td>
<td>31.18</td>
</tr>
<tr>
<td><strong>Min</strong></td>
<td>15.65</td>
<td>16.12</td>
</tr>
<tr>
<td><strong>Q1</strong></td>
<td>24.31</td>
<td>21.51</td>
</tr>
<tr>
<td><strong>Median</strong></td>
<td>32.06</td>
<td>25.08</td>
</tr>
<tr>
<td><strong>Q3</strong></td>
<td>224.40</td>
<td>28.24</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>448.28</td>
<td>237.48</td>
</tr>
</tbody>
</table>

Table 3.5: Statistical information of the 12 and 29 input method PSO trials.

3.4 Discussion and Comparison

Backpropagation and PSO trained ANNs have shown they can both perform admirably in the regression task of predicting the burn area of a forest fire. PSO, however, showed superior results in its RMSE score with a 15.65 minimum versus the lowest backpropagation RMSE score of 42.48. This large difference can be attributed to the fact that PSO is less likely to get stuck in local minima than backpropagation. PSO also brings about more randomness and chance when training an ANN due to the chances for particle death as well as each particle moving about the search space in a slightly different manner, resulting in improved weight values.

The better RMSE values of PSO also comes with a price – the training and testing iterations take much longer to execute than the backpropagation method. This also means that PSO trained ANN will have a much more difficult time scaling up to larger network sizes such as deep learning neural networks which can contain many different hidden layers each containing many perceptrons. The backpropagation trials are lasted around 15-20 seconds per training epoch and only slightly increased as the number of perceptrons increased. This allows this style of ANN to scale to larger sizes much easier.

The vast majority of the backpropagation activation function architectures had very low volatility, along with the 29-Input PSO method. The 12-Input PSO method is among the more volatile systems it however achieved the best RMSE score. Its volatility could be the reason for providing
the best RMSE score. As this task has little to no correlation between inputs and the outputs the volatility could result in the particle appearing in or moving to parts of the search space that other methods simply do not attempt to pursue. Conversely, the opposite may occur as well. Meaning at times the 12-Input PSO may never enter the global optimum position as the particles are moving about to hectically.

3.5 Conclusion

This chapter has demonstrated the use of 1-of-C encoding, performed a systematic evaluation of ANN architectures and activation functions, and used a PSO-ANN algorithm in a targeted effort to improve regression results for the Monteshino dataset. Using PSO-ANN to train the Montesinho data set proves to be a large improvement compared to past efforts. With a low average RMSE score of 15.65 this shows a tremendous improvement over the RMSE scores of previous authors’ works. This low RMSE score was found using the 12-input PSO-ANN. Even with this in mind, the 29-Input PSO-ANN should be chosen to best regress this data set consistently. The 12-input PSO-ANN is difficult to recommending using as there is a decent chance it may never reach the global optimum of the search space reliably. It is worth using the 29-Input method, even if the RMSE score is ever so slightly higher, to more reliably scrub the search space.

This also confirms that the 29-Input method is better to use in the case of performing a regression task on this data. The 29-Input method sees more consistently lower RMSE results all across the board with less variance. The 1-of-C encoding method should always be at least attempted on small and skewed data sets to see if it can improve results on individual cases. It is shown that it is effective in this case. The extra input fields assisted the ANNs in better learning to predict the output values.
CLASSIFYING THE FOREST FIRE DATA SET WITH CLUSTERING AND
COMPUTATIONAL INTELLIGENCE

Clustering and classification are two of the fundamental tasks in Data Mining. Classification is used as a supervised learning method, while clustering is often used for unsupervised learning. This means that the goal of clustering is descriptive while the goal of classifying is predictive [30]. Typically, clustering algorithms are used when items have an at least somewhat obvious set of similarities or differences to use to decide to which cluster they belong. The Montesinho data set, however, is a bit more ambiguous. None of the input dimensions have a strong correlation to the output value nor each other. This means that classifying this data set proves difficult. To attempt to classify the Montesinho data set it must first be clustered and then the results of the clustering used to train an ANN to classify.

This chapter uses, for the first time, k-Means and Spectral clustering to cluster this dataset on both input and output data in order to more accurately classify the data. The outputs present a more straightforward clustering task as they present only one, rather meaningful, dimension, meaning that the clustering algorithms can decide where they should splice that single dimension into sections to determine the clusters without much difficulty. However, this does not necessarily reflect a real world situation as clustering a data point after knowing its result would ultimately not prove very helpful in predicting burn area conditions. Yet, the findings are not without merit since the more supervised data will result in better future training of ANNs. The data set is also classified on its inputs, both the 12 and 29 input variants. This is a much more challenging task due to the large dimensionality and lack of correlation between dimensions. No clustering work has been performed on this data set in the past, so all results are contributions toward further understanding how classifiers can be applied to the Montesinho data set which can be extrapolated to other small, and highly imbalanced data sets that are difficult to regress and/or classify.
4.1 Methodology

Several different numbers of clusters and two clustering algorithms are used in the clustering trials. The two different clustering algorithms used are k-Means and Spectral Clustering. Each of the two clustering algorithms is used to cluster the data into 2, 3, and 4 clusters. These clusters are made on both the inputs and the outputs of the data. Each number of clusters is evaluated using both the 12 and 29 Input methods (as discussed in the previous chapter) to see what impact the input method has on clustering tasks. This results in a total of 24 new data sets that are used to train the various ANN architectures which are trained using backpropagation and PSO.

After the clusters are created on either the inputs or outputs, these results are fed into a series of ANN architectures as in the previous chapter. However, only the first layer activation function is varied as the second is locked as a Softmax function for classification [31]. Once again, a PSO trained ANN and backpropagation are used to perform the classification task.

The classification is completed using the python library Sci-Kit Learn [32] which has k-Means and Spectral Clustering implementations. These implementations generate text files with the cluster information to be used for the ANNs in the form of a new singular field with the cluster number that replaces the current output. This cluster number is fed into the networks in a 1-of-C encoding method. That is, if the cluster were to be two in a field of three clusters the ANN would be fed [0 1 0] as the target value, representing that the data should be in cluster two.

After being trained, the ANNs provide a calculation of the percentage accuracy in classifying the clusters according to eq. (4.1) where \( \hat{y} \) is the total number of correct predictions and \( y \) is the total number of samples tested. This equation comes from the ANNs predicting the class of a set of data and comparing against the target of the class. If the predicted class matches the target it is considered successful. The number of successes is then divided by the total amount of items in the training set to result in the accuracy score. This accuracy number will provide the basis for the discussion of the results and how clustering performed as an approach to provide helpful
predictions of the Montesinho data set.

\[
\hat{y}/y = \text{accuracy} \tag{4.1}
\]

All accuracy results are derived from cross validation. In the case of this classification task, stratified k-Folds, with \( k = 5 \), is used for cross validating. This is to ensure that each cluster is represented equally when training and testing the data. Otherwise, it is highly likely that the training would be over-fitted to certain clusters and under-fitted to others.

4.2 Results

Results are broken up into the two clustering algorithms used. Further, each clustering algorithm has a section detailing the results of clustering on the Input and Outputs along with cluster distribution of each method.

4.2.1 K-Means Clustering

When evaluating clustering algorithms, distributions of the clusters gives an insight into why results may be the way they are when training an ANN to classify. The skewed nature of the data set makes it difficult to create clusters of even sizes when clustering on the outputs. Clustering on the Inputs may create clusters of similar size but has its members have a low correlation to the output target.

The clustering using k-Means with two clusters selected can be found in Fig. 4.1. The output shape is as expected. The large right skew of the data set leads to a right skew of the clustering. When clustering on the inputs some changes are evident as both contain a left skew, however the 29 Input method skew presents as much less significant.

When using three clusters in the k-Means algorithm (Fig. 4.2) the skew of the clustering on the outputs of the data worsens. This is due to the fact that the amount in cluster zero remains roughly the same but the remaining data set members are split apart. Clustering on the Inputs shows two very different groupings. The 12 Input method displays a roughly linear decrease in cluster size while the 29 Input method’s clusters are all almost the same size.
Figure 4.1: Cluster Distributions of k-Means where $k = 2$. 

(a) Clustering on 12 Inputs. 

(b) Clustering on 29 Inputs. 

(c) Clustering on Outputs.
Figure 4.2: Cluster Distributions of k-Means where k = 3.
When using four clusters in the k-Means algorithm (Fig. 4.3) the skew of the clustering on the outputs worsens still. It is assumed that this pattern will continue as the number of clusters increases due to the nature of the data set. As the number of clusters grows, group zero always will remain the same size with the remainder of the members being divided into more and more clusters. Clustering on the Inputs shows that the 12-method resembles the same linear decrease with the new group being a slight step up while the 29-Input method loses its group member evenness and takes on a more random appearance.

Measuring the accuracy of the methods clustering using input values is an interesting proposition. The measured values are high, however, clustering on the inputs of the data set does not tightly correlate to the burn areas of the data set. The best accuracy scores for the k-Means clustered data can be seen in Table 4.1. The PSO trained ANNs shows superior results for both the 12 and 29 Input methods when training on the clustered outputs. The Backpropagation trained ANNs
fared better when training on the clustered Inputs.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Backpropagation</th>
<th>PSO-ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.613175</td>
<td>0.651081</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.617952</td>
<td>0.670312</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.974258</td>
<td>0.916482</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.996435</td>
</tr>
<tr>
<td>3 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.575067</td>
<td>0.596516</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.558438</td>
<td>0.610864</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.984652</td>
<td>0.924682</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.938752</td>
</tr>
<tr>
<td>4 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.504682</td>
<td>0.578972</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.516485</td>
<td>0.556482</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.954652</td>
<td>0.894652</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.904685</td>
</tr>
</tbody>
</table>

Table 4.1: The best accuracy scores for backpropagation and PSO ANN training methods for k-Means clustered data sets.

4.2.2 Spectral Clustering

The clustering performed by the Spectral Clustering algorithm looks very similar to that of the clustering accomplished by the k-Means algorithm. The results of this clustering are shown in Figs. 4.4 - 4.6. When two clusters are desired the data favors one cluster over the other on both Input and Output clustering.

When three clusters are selected and performed on the outputs the there is still a large favoring of one cluster, though it is not as severe as the k-Mean circumstances. Clustering on the inputs of the 29 Input method provides fairly even cluster size and the 12 Input method has two cluster of similar size and one that is much smaller than the other two.

Four clusters presents a difficult situation on the outputs. One cluster size is extremely large compared to the other three. This causes difficulty in training an ANN due to over-representation of that cluster. The 12-Input method proves to have the most balanced cluster sizes which provides easier training and the 29-Input method has three clusters of similar size and one somewhat smaller.
(a) Clustering on 12 Inputs.  
(b) Clustering on 29 Inputs.  
(c) Clustering on Outputs.

Figure 4.4: Cluster Distributions of Spectral Clustering where $k = 2$. 
Figure 4.5: Cluster Distributions of Spectral Clustering where $k = 3$. 
Figure 4.6: Cluster Distributions of Spectral Clustering where $k = 4$. 

(a) Clustering on 12 Inputs.  
(b) Clustering on 29 Inputs.  
(c) Clustering on Outputs.
The best accuracy scores of the trials that used Spectral Clustering, Table 4.2, showed similar patterns to the k-Means trials. Once again, PSO trained ANN performed better when training on data that was clustered by output. The backpropagation trained ANNs performed better on data that is clustered by Inputs. Overall, the scores were slightly better for the Spectral Clustered data.

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Backpropagation</th>
<th>PSO-ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.608167</td>
<td>0.669435</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.593479</td>
<td>0.680374</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.994358</td>
<td>0.937807</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.967255</td>
</tr>
<tr>
<td>3 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.572540</td>
<td>0.589369</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.541732</td>
<td>0.601427</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.971023</td>
<td>0.915902</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.944095</td>
</tr>
<tr>
<td>4 Clusters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12-Outputs</td>
<td>0.527634</td>
<td>0.573466</td>
</tr>
<tr>
<td>29-Outputs</td>
<td>0.504675</td>
<td>0.534678</td>
</tr>
<tr>
<td>12-Inputs</td>
<td>0.916482</td>
<td>0.906428</td>
</tr>
<tr>
<td>29-Inputs</td>
<td>1.000000</td>
<td>0.919645</td>
</tr>
</tbody>
</table>

Table 4.2: The best accuracy scores for backpropagation and PSO ANN training methods for Spectral Clustered Data Sets.

4.3 Discussion and Comparison

It is surprising how similar k-Means and Spectral clustering performed in all circumstances. Spectral Clustering creates clusters on the basis of connectivity while k-Means creates clusters based on compactness. It is assumed that these two clustering methods would produce drastically different results when clustering the data. However, the differences between the two are small and in fact nearly identical in regards to clustering of the outputs outside of a few discrepancies in the middle range of output values. The similarities in clusters could be attributed to the data set not favoring connectivity nor compactness which would result in neither clustering algorithm performing necessarily different than the other.

Graphs of the clustering allow easier visualization of why data clustered on the inputs is per-
(a) Outputs clustered using 29-Input.

(b) Outputs clustered using 12-Input.

Figure 4.7: Graphical representation of the three clusters created from the Spectral Clustering algorithm using Outputs.
Figure 4.8: Graphical representation of the three clusters created from the Spectral Clustering algorithm using Inputs.
Figure 4.9: Graphical representation of the three clusters created from the k-Means Clustering algorithm using Outputs.

(a) Outputs clustered using 29-Input.

(b) Outputs clustered using 12-Input.
Figure 4.10: Graphical representation of the three clusters created from the k-Means Clustering algorithm using Inputs.

(a) Inputs clustered using 29-Input.

(b) Inputs clustered using 12-Input.
forming better than data clustered on the outputs. To perform graphs of the data, principal component analysis is applied to the data set to narrow it down to three components that then allow a three dimensional graph of the clusters to be created.

Looking at Figs. 4.7 – 4.10 it can be seen that when using the 29 Input method to perform clustering three very distinct clustering shapes appear. When clustering on the inputs these shapes become logically assigned clusters. This leads to the belief that if the correlation of the inputs can be successfully mapped to the outputs results could dramatically increase. This thought is echoed by the data in tables 4.1 and 4.2. When classifying on the inputs of the 29 input method the backpropagation ANN is able to classify the data with 100% accuracy. The downside to this finding is that when analyzing the data more closely it is difficult to determine why the clusters were assigned in this way. The output values of the corresponding clusters do not make enough logical sense for this interesting result to be applied to real world scenarios.

It can be seen that the figures that use the 12-Input method do not clearly define themselves into clusters. This phenomenon appears to hamper accuracy scores as the 29-Input method provides superior accuracy scores for when clustering on either the inputs or outputs. This provides insight as to why the 29-Input method performs better throughout the thesis. It gives the data set a more solid structure which most likely allows the ANN to learn more efficiently and effectively.

To classify the data set via a backpropagation or PSO trained ANN proves to be very challenging. Stratified k-Folds cross validation is used in an effort to represent all of the classes statistically even in an attempt to avoid over training the ANNs on clusters that are much larger than others. There is a trend that the accuracy decreases as the number of clusters increases. This is most likely occurring due to the fact that there is always going to be one large cluster when clustering on the outputs due to the skew nature of the data set. This skew will result in one cluster that is always large and does not shrink as more clusters are added resulting in a disproportionately larger cluster as more clusters are created.
4.4 Conclusion

The Montesinho data set has not had clustering methods applied to it before. By clustering the data into clusters of sizes two, three, and four on the inputs and outputs of the data set, using both 12 and 29 input methods, it is seen that classifying the data on outputs is equally as or more difficult than regressing it, though when using the inputs the classification task becomes much easier.

Both backpropagation and PSO trained ANNs struggle to classify the data when it is clustered on the outputs. With the best results being very close for both clustering methods. The best results for k-Means being 67%, 61%, and 55% for 2, 3 and 4 clusters respectively. With the Spectral Clustering methods garnering 66%, 60%, and 53%. The small differences in scores can be attributed to random chance when the ANN is learning and is hard to say whether one clustering method performs better than the other when used for a classifying ANN.

Both clustering methods drew there best results from PSO trained ANN when clustering on the inputs. A highly interesting finding however is how consistently backpropagation was able to classify when clustering on the inputs of the 29 input method. All results had a 100% classification success rate.

It is possible that different clustering algorithms, such as Affinity Propagation [33], could outperform these algorithms. This is due to the nature of the Affinity Propagation clustering algorithm determining the optimum number of clusters pragmatically. The Affinity Propagation algorithm could struggle however, as the data set dimensions do not correlate well with each other. This could result in many clusters of small size which would also pose a challenging task for a classifying ANN to perform.
CONCLUSION AND FUTURE WORK

The Fire Data Set provided by Montesinho Natural Park provides an incredibly difficult classification or regression task due to several factors. First, it is highly right skewed. This leads to a specific target output, in this case 0.0, being over represented making the training of ANNs difficult as they are not exposed enough to other output values to derive patterns. Second, the size of the data set is an issue. At only 517 entries it is a small set of data. The limited amount of data makes training difficult as the already small dataset is made even smaller for training as the data is divided between instances used for training and testing. Third, there is very little correlation between the input dimensions and the target output. If the input dimensions were more tightly correlated with the outputs it would result in easier training of ANN. The correlation and small size of the data set can not be changed. However, the imbalanced nature is lessened when applying a log transform to the outputs of the data.

The current state-of-the-art with regards to the Montesinho data set have RMSE values of near 60. This is a very poor result and will not assist in the predicting of burn areas of forest fires in a real world scenario. This thesis has dramatically lowered this RMSE score in hopes of being able to apply the findings to other data sets that are small and imbalanced, resulting in four contributions that combine computational intelligence and data mining techniques:

1. A novel input structure derived from the data set. Instead of the listed 12-Inputs, 29 are created by expanding categorical data into their own dimensions. Meaning that fields such as month are broken into 12 dimensions where a one value indicates the selected month and a zero value indicates months that are not selected;

2. An exhaustive, comparative study on activation functions and neural network architectures using the backpropagation method in conjunction with these two input methods. These results demonstrate that the 29-Input method performs better than the 12-Input method for backpropagation trained networks with the activation functions and number of hidden layer
nodes providing minimal differences;

3. A PSO trained ANN is introduced and performs at a much higher level than the backpropagation trained network with the lowest RMSE value of 15.65 for 12 input method and 16.12 for the 29 input method. Even though the 12 input had the overall lowest minimum, the 29 input method is still the preferred input style as its results are much more consistent and less volatile throughout different ANN architectures; and

4. Clustering of the data set using two separate methods, k-Means and Spectral, is performed. Using these two methods the data set is clustered using two, three, and four clusters with each of the cluster sizes being clustered on the inputs as well as the output dimensions of the data set. These clustering results are then used to train a PSO-ANN to predict the class of the data. Results demonstrate that it is very difficult to train an ANN to reliably predict the class of a data set when clustered on the outputs with resultant accuracy values being between 53% and 68% depending on the amount of clusters. However, when using the data that is clustered on the inputs, the results approach or obtain 100% accuracy. The 29-Input method is once again proven to work more reliably than the twelve. Clusters generated and trained using the 29-Input method obtained the best accuracy scores including the 100% accuracy on inputs.

5.1 Future Work

RMSE may ultimately be able to further be improved for the Montesinho data set. There are several methods to be attempted that have not yet been applied:

- Evolutionary undersampling and oversampling assists in learning with imbalanced data [34]. It is possible to reduce the set of samples (undersampling) or replicate minority class examples (oversampling). Undersampling with imbalanced datasets can be considered as a prototype selection procedure with the purpose of balancing datasets to achieve a higher classification rate, avoiding the bias toward majority class examples. This is highly applicable to Montesinho data set as it is highly imbalanced with a very large proportion of 0.0
output values. Oversampling could help as well with the replication of lower represented output classes which would lessen the effect of all of the 0.0 output results.

- Another method to consider is deep learning and hierarchical neural networks. A deep learning neural network is a neural network with more than one hidden layer and can have many, many perceptrons. This growth of the network allows more possible pattern recognition and can help in recognizing less obvious patterns to data. This could help lower the RMSE results of the Montesinho data set as the data set lacks obvious patterns which may limit how well a single hidden layer neural network can detect them. A hierarchical neural network provides a way for the neural networks to have arbitrary connections between layers. This means that perceptrons connections can be grouped together in novel ways such as having all of the perceptrons representing the FWI system connect to a specific group of perceptrons in a different layer while the spatial X, Y coordinates could be connected to their own perceptrons and layer. These two unique layers could then combine into a different layer. Building these kinds of custom connection systems further allows detections of more abstract patterns.

- Metaheuristics can also be considered and applied in different ways. In this thesis, PSO was used as the metaheuristic to train the ANNs. Other metaheuristics can be used to both train ANNs and used to cluster data. Metaheuristics for clustering include simulated annealing which can be used to train the k-Means algorithm [35]. The simulated annealing algorithm can be used to both initialize the k-Means algorithm as well as search for the global optimum without as high of a likelihood of becoming stuck in local optima. Other metaheuristics other than PSO can be used to train ANNs as well such as Generalized Extremal Optimization, variable neighborhood search, and canonical genetic algorithms [36]. The use of different metaheuristics in training ANNs could find different patterns to the data much like using different metaheuristics to cluster with k-Means.

While the work completed by this thesis has dramatically lowered the RMSE scores of the Montesinho data set through the use of PSO trained ANNs, there is room for improvement on
many fronts. Continually improving the RMSE of the Montesinho data set will provide insight into improvement on other small, skewed, and difficult to regress data sets which may then be applied in a variety of real-world scenarios.
BIBLIOGRAPHY


