



# Artificial Intelligence

## UNIT ASSIGNMENT

Dale Stubbs - 14024149 | Computer Forensics and Security | 04/02/2017

## Contents

Weka.....	3
ZeroR.....	3
OneR.....	3
J48 Decision Tree .....	4
MultiLayer Perceptron (Neural Network) .....	5
1. Scenario 1 – Image Processing – MNIST Dataset .....	8
1.1. Introduction .....	8
1.2. Process.....	8
1.2.1. Equalising Distribution.....	8
1.2.2. Principal Component Analysis .....	8
1.2.3. PCA Models .....	9
1.2.4. PCA Encoding .....	9
1.2.5. Applying Filters.....	9
1.3. Experiments.....	10
1.3.1. Experiment One – Equalised Distribution Dataset.....	10
1.3.2. Experiment Two – PCA Models Only .....	11
1.3.3. Experiment Three – PCA Models with Binarization.....	13
1.3.4. Experiment Four – PCA Models with Sobel Filter .....	14
1.3.5. Experiment Five – PCA Models with Gaussian Filter .....	15
1.4. Comparison of results.....	16
1.5. Conclusion.....	17
2. Scenario 2 – Machine Learning – Mammography Dataset .....	18
2.1. Introduction .....	18
2.1.1. Attributes .....	18
2.2. Basic Pre-Processing .....	18
2.3. Experiments.....	19
2.3.1. Experiment Set One – Decision Trees – Basic Pre-Processing .....	19
2.3.1.1. Confidence Interval Pruning.....	19
2.3.1.2. Minimum Number of Objects Training.....	20
2.3.2. Experiment Set Two – Decision Trees – Improved Pre-Processing.....	22

2.3.3.	Experiment Set Three – Neural Networks – Basic Pre-Processing .....	24
2.3.3.1.	Hidden Layers .....	24
2.3.3.2.	Learning Rate .....	24
2.3.3.3.	Momentum .....	26
2.3.4.	Experiment Set Four – Neural Networks – Improved Pre-Processing .....	27
2.4.	Conclusion.....	29
	Images Used .....	30
	References .....	31

## Weka

Weka is a machine learning software that contains a multitude of different learning algorithms. *'The algorithms can either be applied directly to a dataset or called from your own Java code.'* (Cs.waikato.ac.nz, 2017)

This assignment will require the use of some of those algorithms such as:

- ZeroR
- OneR
- J48 Decision Tree
- MultilayerPerceptron

Each algorithm offers different attributes that can be altered in an attempt to create a more accurate classifier used.

### ZEROR

The ZeroR classifier is the most basic classifier available within Weka and is most useful when determining a baseline performance as it simply predicts the larger class of inputs. This classifier only uses the specified target class and ignores all the predictors. *'Although there is no predictability power in ZeroR, it is useful for determining a baseline performance'* (Chem-eng.utoronto.ca, 2017)

This algorithm has four attributes to alter, however, these attributes, when altered, do not alter the accuracy of the classifier. These attributes are:

1. numDecimalPlaces – This determines to how many decimal places the numbers used in the model are shown. The default value for this is two.
2. batchSize – This determines the number of instances to process when using batch prediction. The default value for this is 100.
3. debug – This determines if additional information is output to the console to inform the user of any bugs in the system. The default value for this is False.
4. doNotCheckCapabilities – This determines whether or not the capabilities of the classifier are checked before the classifier is built. The default value for this is False.

### ONER

This particular classifier is the next link in the chain of Weka classifiers as it uses the minimum-error attribute for prediction.

This algorithm has all of the adjustable attributes of the ZeroR classifier with one addition, minBucketSize. This attribute is used for limiting the complexity of the rules in order to decrease the potential for 'over training' the classifier. Overtraining (aka overfitting) is *'when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.'* (Brownlee, 2017). Effectively this means that the classifier has learned the training data so well that it cannot work well with new data that it sees.

## J48 DECISION TREE

This classifier will constitute a lot of the experiments I will do on the datasets. *'A decision tree is a decision support system that uses a tree-like graph decisions and their possible after-effect, including chance event results, resource costs, and utility'* (Bhargava, N., Sharma, G., Bhargava, R. and Mathuria, M., 2013). The J48 Decision Tree is used to create a C4.5 tree and has many more editable attributes than the OneR and ZeroR classifiers. This form of classifier is incredibly useful as its output is easy to understand by the user, it can handle multiple input types (ordinal, textual and nominal), it contains the ability to process incorrect or missing values and has a high performance with little effort. The editable attributes of the J48 Decision Tree contain the same attributes as the ZeroR classifier with the following additional parameters:

Parameter Name	Description	Default Value
binarySplits	Determines whether binary splits are used on nominal attributes when constructing the decision tree.	False
collapseTree	Decides whether or not parts of the tree are removed if they do not reduce training errors.	True
confidenceFactor	Alters the amount of post-pruning done to the decision tree.	0.25 (25%)
doNotMakeSplitPoint ActualValues	Ensures that the split point is not relocated to an actual data value when set to 'True'.	False
minNumObj	Responsible for determining what the minimum number of instances per leaf are required.	2
numFolds	Determines the quantity of data used for reduced-error pruning.	3
reducedErrorPruning	Determines whether reduced-error pruning is used as opposed to C4.5 pruning.	False
saveInstanceData	Determines whether or not to save the training data for visualisation purposes.	False
seed	Used to set the randomiser when reduced-error pruning is used.	1
subtreeRaising	Determines whether to consider the subtree raising operation when building the trees.	True
unpruned	Determines if pruning of the tree will be performed.	False
useLaplace	Determines if the counts at each leaf are smoothed based on Laplace.	False
useMDLcorrection	Determines if MDL correction is used when finding splits on numeric attributes.	True

Table 0-1: Parameters of the J48/C4.5 classifier

The experiments in this report will focus on adjusting only a few of these parameters as they are deemed to have a significant effect on the results. When using the J48/C4.5 decision tree the parameters I will be adjusting are the confidence factor and minimum number of objects.

The confidence factor (also known as confidence interval (CI)) is a value that is used in the pruning process of the classifier. This form of tree pruning is called post-pruning. Post-pruning is the process of building the tree *'first and then reduction of branches & levels of the decision tree is done'* (Patel, N. and Upadhyay, S., 2012). An error rate is calculated from the training data, a cynical upper limit for this value is added based on this confidence factor and the value is used to decide how much pruning can be done. When used within Weka, lowering this value decreases the amount of post-pruning done.

The minimum number of objects (MNO) is another important parameter I will be adjusting during my experimentation. This parameter is responsible for deciding how many instances that need to occur at each leaf of the tree. *'One simple way of pruning a decision tree is to impose a minimum on the number of training examples that reach a leaf.'* (Gerardnico.com, 2017) This form of pruning is a manual form of pre-pruning the tree when post-pruning has been disabled. When the MNO is set too low and post-pruning is enabled then the post-pruning makes the necessary adjustments for this error.

## MULTILAYER PERCEPTRON (NEURAL NETWORK)

A Multilayer Perceptron (MLP) is a classifier that uses backpropagation to classify instances from a dataset. Backpropagation is a method used to train artificial neural networks (ANNs). It uses a two-phase approach. First calculating error values representing nodes contributions to original inputs then, using these errors, calculates the gradient of the loss function with regards to the weights in the network. This gradient is then used to create and update new weights in an attempt to minimise the loss function. *'At the heart of backpropagation is an expression for the partial derivative  $\partial C / \partial w$  of the cost function  $CC$  with respect to any weight  $w$  (or bias  $b$ ) in the network.'* (Nielsen, 2017). MLPs are considered the standard algorithm for any supervised learning pattern recognition process and this is why I have chosen to use them in my experiments within this report. Both scenarios will be attempting to gain the required information using pattern recognition. A table of all of the available parameters is shown below:

Parameter Name	Description	Default Value
GUI	Displays a Graphical User Interface when set to True. This allows for pausing and editing of the network during training.	False
autoBuild	Adds and connects hidden layers in a network.	True
batchSize	*As in ZeroR*	100
debug	*As in ZeroR*	False
decay	Causes the learning rate to decrease. The starting learning rate is divided by the epoch number to determine at what level the learning rate should be.	False
doNotCheck Capabilities	*As in ZeroR*	False
hiddenLayers	Defines the number of hidden layers in the network. This will only be used if autoBuild is set to True.	a
learningRate	The amount the weights are updated.	0.3



momentum	The momentum applied to the weights during updating.	0.2
nominalToBinaryFilter	Pre-processed the data with a filter.	True
normalizeAttributes	Normalises the attributes within the dataset.	True
normalizeNumericClass	Normalises the class if it is numeric.	True
numDecimalPlaces	*As in ZeroR*	2
reset	This allows the network to reset with a lower learning rate. Training will begin again if the network diverges from the answer.	True
seed	*As in the J48*. This seed is used for setting the initial weights of the connections between nodes.	0
trainingTime	The number of epochs to train through.	500
validationSetSize	The percentage of the dataset used for validation purposes.	0
validationThreshold	Set the threshold for the validation testing. Sets the number of times the validation can be worse before training is terminated.	20

*Table 0-2: Parameters of the Multilayer Perceptron Classifier*

Throughout my experiments I will alter some of these values whilst leaving the other values set to their defaults as they will provide a more accurate classifier when adjusted. The main focus of my experiments will use the adjusted values of the hidden layers, learning rate and momentum.

I will first examine the effect of having different numbers of hidden layers within my network. *'Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'* (Pages.cs.wisc.edu, 2017). There is a lot of information available on the internet that discusses the effects of having multiple hidden layers on a network and the best way to calculate the 'ideal' number of hidden layers, however, as my dataset has only five inputs to begin with, I believe only one experiment will be required to determine the 'ideal' number of hidden layers for my particular network.

The learning rate is also an important parameter that I will be adjusting during a set of experiments in the MLP experiment section. The learning rate determines the speed in which the network will 'learn'. Setting the learning rate too high may cause the network to overshoot the minimum error value, set it too low and it may never reach the minimum error value as the error value may increase before decreasing again also known as local minima. If the learning rate is set too low then it will never pass this local minima. The initial experiment set will be a rough estimation of what learning rates may be appropriate for this dataset and will be fine-tuned to find the optimum value as *'there comes a point at which reducing the learning rate any more simply wastes time, resulting in taking many more steps than necessary to take the same path to the same minimum'* (Wilson, D.R. and Martinez, T.R., 2001).

The final parameter I will focus some experiments on is the Momentum. The momentum adds a fraction of the previous weight update to the current one which allows the system to avoid converging to a local minima. *'When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum.'* (Willamette.edu, 2017). Setting the momentum too high may cause the optimum error value to be overshoot and, again, too low could allow the network to get trapped in a local minima. The experiments will use values around the default to begin with and will be fine-tuned from there until the optimum momentum has been found.



# 1. Scenario 1 – Image Processing – MNIST Dataset

## 1.1. INTRODUCTION

The MNIST dataset is a collection of 70,000 (60,000 used for training and 10,000 used for testing) handwritten numbers ranging from zero through to nine. The aim of this section is to develop a proof of concept by creating an Artificial Intelligence (AI) classifier that can correctly discriminate between the number four and all other numbers.

## 1.2. PROCESS

### 1.2.1. Equalising Distribution

The initial problem faced when processing this dataset is the volume of images to be processed. The full dataset contains 60,000 entries, of which only 9% are fours. The dataset needs to be distributed equally and must contain an equal number of fours and non-fours. This was completed using my 'imagePreparation.m' MatLab function (Appendix 1). This function not only equally distributes the number of fours and non-fours but also distributes all of the non-four digits equally between the remaining numbers (0, 1, 2, 3, 5, 6, 7, 8 and 9) using 649 images of each of the other numbers with the exception of the number seven which has 650 images. As the number of fours is 5842, when combined, all of the remaining numbers equal 5842. This is the most equal form of distribution that I believe is possible and therefore this is the distribution I will use in my image processing.

### 1.2.2. Principal Component Analysis

My choice on which form of processing I would subject the images to was my largest concern when beginning the assignment. I chose to use Principal Component Analysis (PCA) to process the images as this seemed to be the most appropriate form of analysis. Used in pattern recognition and face recognition as well as other areas, PCA is used to determine the rate of similarity and variance between objects within a dataset. Data with high dimension components that cannot be represented via graphical means relies upon this form of analysis in order to establish patterns within the data. *'The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables, while retaining as much as possible of the variation present in the data set.'* (Jolliffe, 2007)

The method used within PCA is relatively simple, the mean of the data (in this scenario the data is images) is calculated (add all of the images together and divide it by the number images). The mean is then subtracted from each image individually and all of the images are then converted into columns. These columns are then grouped into a matrix to allow the computation of the covariance matrix. Then eigenvectors are calculated, which are the principal components of the images, and each eigenvector is given an eigenvalue the eigenvectors importance.

### 1.2.3. PCA Models

Models of each of the individual numbers are created and used to encode the images of a particular model. For example, the model created for the number one is created by using a percentage of the numbers ones in the dataset. I, when creating the models, choose the percentage of ones to be used. In order to create a more detailed set of Principal Components (PCs), the PC models need to be created using the full 60,000-entry dataset. This will give more non-four images to use when creating the models for those numbers as, if I were to use the equalised dataset, there would only be 649/650 images to be used. When creating the PC models I chose to use 99% of the available images to provide models that are more accurate.

### 1.2.4. PCA Encoding

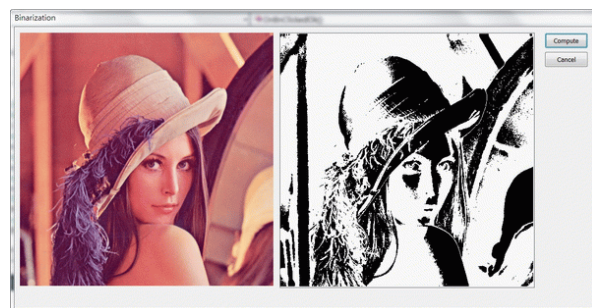
The images within the equalised dataset were then encoded using the models created earlier. As there are 10,000 images that are used for testing purposes. I attempted to create an equal number of images that were used for training purposes. This was completed by using a percentage of 86% when encoding the images when using the 'encodePCModels.m' (Appendix 2). Once the PCA encoding had been applied the outputs generated by MatLab for both the training images and testing images were combined into one Comma Separated Values (CSV) file ready to be input into Weka for further experimentation. The percentage of training images to testing images when using 86% inside the PCA encoding was 50.285%. This was as close to a 50% split of training/testing data I was able to generate and therefore was the percentage I chose to use in my encoding.

### 1.2.5. Applying Filters

Following the previous step, I then chose to apply filters to the dataset created in order to establish if these had any effect on the overall performance. The filters I applied were Binarizing, Smoothing using a Gaussian Filter and Edge Detection using a Sobel Filter. The following sub-sections will discuss the details of each of the filters.

#### 1.2.5.1. Binarization

Binarization is the process of creating a binary image from a pixel image. This is achieved using Otsu's algorithm of Thresholding. Otsu's algorithm follows six steps the first of which is to obtain a histogram of the image. Otsu's algorithm selects an appropriate threshold that will be used to determine if a pixel value should be converted into a zero or a one. For example, if a threshold of 0.5 was used to binarize an image then all of the pixels with a value greater than 0.5 would be converted into a one and all pixels with a value of less than 0.5 would be converted into a zero. The threshold determined by Otsu's algorithm in my script was 0.8 which was then applied to the binary filter which was then used to convert the PCA encoded images into binary.



*Image 1:1: An example of converting a colour image to binary form*

### 1.2.5.2. Gaussian Filter

A Gaussian filter applies a smoothing effect to the images and blurs the edges of the image creating a much cleaner image with fewer details that could be used when training the classifier. This filter also reduces the amount of image noise that is produced by the apparatus that originally captured the image. This filter works by applying a 'point-spread' function using a convolution mask. The output of a Gaussian filter is a 'weighted average' of each pixels neighbourhood with the average weighted more towards the centre of the mask.



Image 1:2: An example of applying a Gaussian Filter

### 1.2.5.3. Sobel Filter

The Sobel filter, when applied to an image, creates a new image with emphasised edges contained within the original image. This filter uses two 3x3 masks which are applied to the original image. One is used to find horizontal edges and the other is used to find vertical edges. The output from a Sobel filter is an image that represents the 'likely' edges of an item contained within the image displayed as white lines with the contents of those edges, as well as the remainder of the image, being a darker colour. The Sobel filter used in my script took no additional parameters for processing.



Image 1:3: An example of applying a Sobel Filter

## 1.3. EXPERIMENTS

A series of experiments were then performed on an unedited for of the dataset in order to establish the minimum requirements for my classifier including accuracy, tree size, number of leaves and time taken to build when experimenting with a Decision Tree (J48/C4.5) classifier. I have chosen to use the J48 classifier as this classifier offers simple to understand results, they perform better on larger datasets than other classifiers and they also offer a more 'human-like' approach to making a decision.

### 1.3.1. Experiment One – Equalised Distribution Dataset

The first experiment I ran was the baseline experiment for validation of the minimum accuracy required for my experiments. This experiment used a J48 decision tree with default parameters and the equalised dataset created in step 2.1 of the image processing section. This dataset was used as the unedited dataset was too large to test accurately on the available computers.

### 1.3.1.1. Results of Experiment One

Confusion Matrix 'Hit'	Confusion Matrix 'False Alarm'	Confusion Matrix 'Miss'	Confusion Matrix 'True Reject'	Tree Size	Number of Leaves
8500	518	45	937	607	304

Table 1-1: Decision Tree using reduced size dataset

```

=== Summary ===

Correctly Classified Instances      9437           94.37 %
Incorrectly Classified Instances    563            5.63 %
Kappa statistic                     0.7383
Mean absolute error                 0.0606
Root mean squared error            0.231
Relative absolute error             12.128 %
Root relative squared error        46.2031 %
Total Number of Instances          10000

```

Figure 1.i: Summary of Decision Tree performance in Experiment One

This experiment provided me with a basic accuracy percentage of 94.37% required in order to conclude that my pre-processing had improved accuracy within Weka. The time taken to build the model based on the input dataset is also something that I was interested in as the time taken to build is an important factor when attempting to discover if the pre-processing of the data has aided Weka in all areas as opposed to only increasing the accuracy of the classifier. The time to build baseline in this experiment was 34.43 seconds.

### 1.3.2. Experiment Two – PCA Models Only

The next experiment was run on a dataset created by simply encoding the images using the PCA models created in step 2.3. This would provide me with a basic set of results obtained when only using PCA encoding on the images.

#### 1.3.2.1. Results of Experiment Two

Confusion Matrix 'Hit'	Confusion Matrix	Confusion Matrix 'Miss'	Confusion Matrix	Tree Size	Number of Leaves
------------------------	------------------	-------------------------	------------------	-----------	------------------

```

=== Summary ===

Correctly Classified Instances      9955           99.55 %
Incorrectly Classified Instances     45            0.45 %
Kappa statistic                     0.991
Mean absolute error                 0.0085
Root mean squared error            0.0652
Relative absolute error             1.6948 %
Root relative squared error        13.0376 %
Total Number of Instances          10000

```

	'False Alarm'		'True Reject'		
4953	23	22	5002	283	142

*Table 1-2: Decision Tree using reduced size dataset with PCA encoding*

As shown in the above table and figure the results of this experiment were much better than the previous experiment in all aspects including the time taken to build. This particular experiment only required 0.59 seconds to build the model which is much faster than the previous result.

*Figure 1.ii: Summary of Decision Tree performance in Experiment Two*

### 1.3.3. Experiment Three – PCA Models with Binarization

The next experiment was run on a dataset created by simply encoding the images using the PCA models created in step 2.4.1. This would give me an insight into whether or not applying a filter to the PCA encoded output would make any difference when experimenting. The results were astounding.

#### 1.3.3.1. Results of Experiment Three

Confusion Matrix 'Hit'	Confusion Matrix 'False Alarm'	Confusion Matrix 'Miss'	Confusion Matrix 'True Reject'	Tree Size	Number of Leaves
273	4702	0	5025	3	2

Table 1-3: Decision Tree using reduced size dataset with PCA encoding with Binarization

```

=== Summary ===

Correctly Classified Instances      5262           52.62 %
Incorrectly Classified Instances    4738           47.38 %
Kappa statistic                    0.0479
Mean absolute error                 0.4879
Root mean squared error             0.4939
Relative absolute error             97.5766 %
Root relative squared error         98.7671 %
Total Number of Instances          10000

```

Figure 1.iii: Summary of Decision Tree performance in Experiment Three

As can be seen in the above table and figure applying the filter actually had a dramatic effect on the results generated. The speed in which this model was build was impressive, however, the model lacked accuracy. The time taken to build this model was a mere 0.08 seconds.

### 1.3.4. Experiment Four – PCA Models with Sobel Filter

The next experiment was completed using the PCA encoded image outputs with a Sobel Filter applied that was created in step 2.4.3. This experiment would give me insight in whether applying edge detection to PCA encodings would yield any improvement on the results.

#### 1.3.4.1. Results of Experiment Four

Confusion Matrix 'Hit'	Confusion Matrix 'False Alarm'	Confusion Matrix 'Miss'	Confusion Matrix 'True Reject'	Tree Size	Number of Leaves
2012	2963	761	4264	7	4

Table 1-4: Decision Tree using reduced size dataset with PCA encoding with Sobel Filter

```

=== Summary ===
Correctly Classified Instances      6276           62.76 %
Incorrectly Classified Instances    3724           37.24 %
Kappa statistic                    0.2535
Mean absolute error                 0.457
Root mean squared error             0.4782
Relative absolute error             91.3945 %
Root relative squared error         95.645 %
Total Number of Instances          10000

```

Figure 1.iv: Summary of Decision Tree performance in Experiment Four

As can be seen in the above set of table and figure, the result is poor. The accuracy has improved from the previous experiments results but they are still very much less accurate than the results of PCA model encoding only. The time taken to build this model was 0.11 seconds which, as in the previous experiment, is a dramatic improvement on the earlier set of results, with the exception of the binarized dataset, however, the overall performance of the tree was significantly worse again.



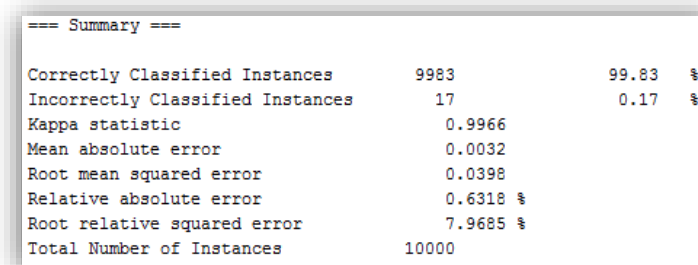
### 1.3.5. Experiment Five – PCA Models with Gaussian Filter

The final experiment was completed using the PCA encoded dataset with a Gaussian filter with a standard deviation of 0.8 had been applied. Again, this would provide insight into whether smoothing of the dataset post-encoding would yield better results.

#### 1.3.5.1. Results of Experiment Four

Confusion Matrix 'Hit'	Confusion Matrix 'False Alarm'	Confusion Matrix 'Miss'	Confusion Matrix 'True Reject'	Tree Size	Number of Leaves
4970	5	12	5013	179	90

Table 1-5: Decision Tree using reduced size dataset with PCA encoding with Gaussian Filter



```

=== Summary ===
Correctly Classified Instances      9983           99.83 %
Incorrectly Classified Instances    17             0.17 %
Kappa statistic                    0.9966
Mean absolute error                 0.0032
Root mean squared error             0.0398
Relative absolute error             0.6318 %
Root relative squared error         7.9685 %
Total Number of Instances          10000

```

Figure 1.v: Summary of Decision Tree performance in Experiment Five

As you can see from the above table and figure, the classifier was much more accurate during this experiment. There was a significant decrease in the size of the tree and number of leaves and the time taken to build the model using this dataset was 0.53 seconds. All aspects of the classifier have been significantly improved.

#### 1.4. COMPARISON OF RESULTS

	Confusion Matrix 'Hit'	Confusion Matrix 'False Alarm'	Confusion Matrix 'Miss'	Confusion Matrix 'True Reject'	Tree Size	Number of Leaves	Correctly Classified Instances	Incorrectly Classified Instances	Mean Absolute Error	Time Taken to Build Model
Unedited Dataset	8500	518	45	937	607	304	9437 (94.37%)	563 (5.63%)	0.0606	34.43 seconds
PCA Models Only	4953	23	22	5002	283	142	9955 (99.55%)	45 (0.45%)	0.0652	0.59 seconds
PCA with Binarization	273	4702	0	5025	3	2	5264 (52.64%)	4738 (47.38%)	0.4879	0.08 seconds
PCA with Sobel Filter	2012	2963	761	4264	7	4	6276 (62.76%)	3724 (37.24%)	0.457	0.11 seconds
PCA with Gaussian Filter	4970	5	12	5013	179	30	9983 (99.83%)	17 (0.17%)	0.0032	0.53 seconds

*Table 1-6: Comparison table of all experiment results collected*

The table above contains all of the collected data from the experiments conducted. All of the experiments were completed using a J48/C4.5 Decision Tree in Weka with default parameters. I ran two further experiment with the Confidence Factor changed but these yielded results exactly the same as the previous experiments so I decided to omit the from this report. The best two pre-processed datasets processed by Weka were the PCA Model encoded dataset and the PCA Model encoded dataset with a Gaussian filter applied. Both of these datasets provided an improved accuracy rating, however, the Gaussian filter seemed to have the most significant impact on the accuracy when compared to the PCA encoding alone. When the Gaussian filter was applied the number of 'Hit's on the confusion matrix increased by 17 over the PCA encoding only and the number of 'False Alarm's combined with the 'Miss's' dropped from 44 down to 17 decreasing by more than 50%. This allows me to say that applying a Gaussian filter after encoding the dataset with the PCA models in the best form of pre-processing I did.

## 1.5. CONCLUSION

Upon concluding this scenario I can determine that image processing can dramatically improve the performance of a J48/C4.5 Decision Tree in Weka. The overall performance of Weka was significantly improved in all areas after completing the image processing before classifying the data using an AI classifier. The best form of pre-processing I found throughout my experiments completed was to first create a set of PCA models using 99% of the available images from a specific number (zero to nine), distribute the dataset equally among four's and non-four's as well as equally amongst the non-four's (649 of each other number), encode the equalised dataset with the PVA models created and then apply a Gaussian filter to the output. This pre-processing gained me an accuracy of 99.83% which was better than the other forms of pre-processing I tried.

## 2. Scenario 2 – Machine Learning – Mammography Dataset

### 2.1. INTRODUCTION

This scenario introduces us to the Mammogram dataset available online. The following report will show the steps taken from pre-processing the dataset through to experimentation on that dataset in order to establish which classifier is best at determining the correlation between the five available attributes (Bi-Rads, Age, Shape, Margin and Density) and the determining attribute (Severity). Each of the attributes will be discussed in the following section. The complete dataset is made up of 961 instances 516 of which contain a benign mass and 445 that contain a malignant mass. Of these 961 instances, 162 attributes are missing from 131 cases.

#### 2.1.1. Attributes

Each of the instances contained within the dataset contains six attributes as stated in the introduction to this report. Some of these instances have missing inputs for some of the attributes and it is my aim to decide whether to fill these attributes in with appropriate values or to discard them completely from the dataset before attempting to classify them. These attributes also contain what are known as ‘outliers’. *‘An outlier is an observation that lies an abnormal distance from other values in a random sample from a population. In a sense, this definition leaves it up to the analyst (or a consensus process) to decide what will be considered abnormal. Before abnormal observations can be singled out, it is necessary to characterize normal observations.’* (Itl.nist.gov, 2017). As stated in the citation, outliers are attributes that fall outside of the average values generated for that particular attribute. These can either be larger or smaller than the average and they can be caused by one of two things: genuine variations in the data and incorrect error input that are known as ‘blunders’.

### 2.2. Basic Pre-Processing

For the original set of experiments, I only applied one form of pre-processing. This was to remove all of the instances that contained missing attributes and the obvious outliers from the Bi-Rads attribute. I found only three obvious sets of outliers from this attribute five zero’s, two six’s and a 55. As the scale of Bi-Rads in the assignment brief stated that Bi-Rads was supposed to be on a scale of one to five I decided that these three instances were outliers and removed them. There were 815 instances remaining once this had been completed. After completing this step I then ran some mathematical equations over the remaining instances to ensure that the dataset still had an equal distribution of severity attributes and found that I had actually improved the dataset split between benign and malignant cases (51.9% and 48.1% respectively) when compared to the original dataset (53.69% and 46.31%). This provided me with a good base set to run the first set of experiments over the data.

## 2.3. EXPERIMENTS

### 2.3.1. EXPERIMENT SET ONE – DECISION TREES – BASIC PRE-PROCESSING

The first set of experiments I chose to run consisted of the ZeroR, OneR and J48 classifiers all with default parameters using the 10 fold Cross-Validation as stated in the assignment brief. The results of this experiment are as follows:

Classifier	Percent Correct	Percent Incorrect	Mean Absolute Error
ZeroR	51.90	48.10	0.50
OneR	83.07 v	16.93*	0.17 *
J48	83.44 v	16.56 *	0.24 *

*Table 2-1: Output of ZeroR, OneR and J48 using edited dataset*

As this table shows the J48 decision tree is the most accurate and therefore the next experiment will be to adjust the parameters of the J48 in order to establish the most accurate decision tree.

#### 2.3.1.1. Confidence Interval Pruning

The first step is to adjust the confidence factor (CF). As a base set, I will run five different CFs. These are 0.05, 0.1, 0.15, 0.2 and 0.25. The results are as follows:

Confidence Factor	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0.25	83.44	16.56	6.70	3.85
0.05	83.55	16.45	6.82	3.91
0.1	83.54	16.46	7.20	4.10
0.15	83.62	16.38	7.36	4.18
0.2	83.44	16.56	7.70	4.35

*Table 2-2: The effect of altering the confidence factor of the J48*

This table shows that the when the CF was set to 0.15 the accuracy improved. As a result of this next step will be to re-run the experiment with the CFs set to 0.11, 0.13, 0.17, 0.19 and 0.15 as the benchmark.

Confidence Factor	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0.15	83.62	16.38	7.36	4.18
0.11	83.56	16.44	7.26	4.13
0.13	83.56	16.44	7.30	4.15
0.17	83.52	16.48	7.42	4.21
0.19	83.44	16.56	7.60	4.30

*Table 2-3: The effect of smaller confidence factors*

This experiment shows that the CF of 0.15 is still the most accurate from this set of values. The next experiment will use the CF values of 0.14 and 0.16 in order to determine if a value closer to 0.15 will provide more accurate results.

Confidence Factor	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0.15	83.62	16.38	7.36	4.18
0.14	83.55	16.45	7.36	4.18
0.16	83.57	16.43	7.36	4.18

*Table 2-4: The effect of closer to optimal CFs*

This experiment shows no improvement than when using a CF of 0.15 and therefore I will be using this value to generate the optimal classifier for this dataset.

### 2.3.1.2. Minimum Number of Objects Training

The next set of experiments I chose to run consisted of minimum number of object (MNO) training. Again, using the 10 fold Cross-Validation. The first experiment consists of multiple values used in the MNO parameter with the J48. The results are as follows:

Minimum Number of Objects	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0	83.44	16.56	9.26	5.13
1	83.44	16.56	9.26	5.13
2	83.39	16.61	8.00	4.50
3	83.43	16.57	7.88	4.44
4	83.45	16.55	7.88	4.44
5	83.39	16.61	7.90	4.45

*Table 2-5: The results of the MNO adjustments*

The above table shows a slight improvement when the MNO is set to four. The next experiment will determine if adjusting the MNO will have any effect on the results when the CF is set to its optimal setting of 0.15.

Minimum Number of Objects	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0	83.57	16.43	7.94	4.47
1	83.57	16.43	7.94	4.47
2	83.62	16.38	7.36	4.18
3	83.63	16.37	7.36	4.18
4	83.62	16.38	7.34	4.17

*Table 2-6: The effect of adjusting the MNO when the CF is set to 0.15*

As the above table demonstrates, altering the MNO of this classifier raises the accuracy of the J48 by 0.01%. Therefore I can determine that the best Decision tree possible for this dataset has a CF of 0.15, with the MNO set to 3. This classifier has an accuracy of 83.63%, and a tree size of 7.36 with 4.18 leaves. The next logical step would be to experiment using the Multilayer perceptron using this dataset in order to determine if the accuracy when using this dataset can be improved without further pre-processing at this time. The tree constructed when using the above values is shown in the image below.

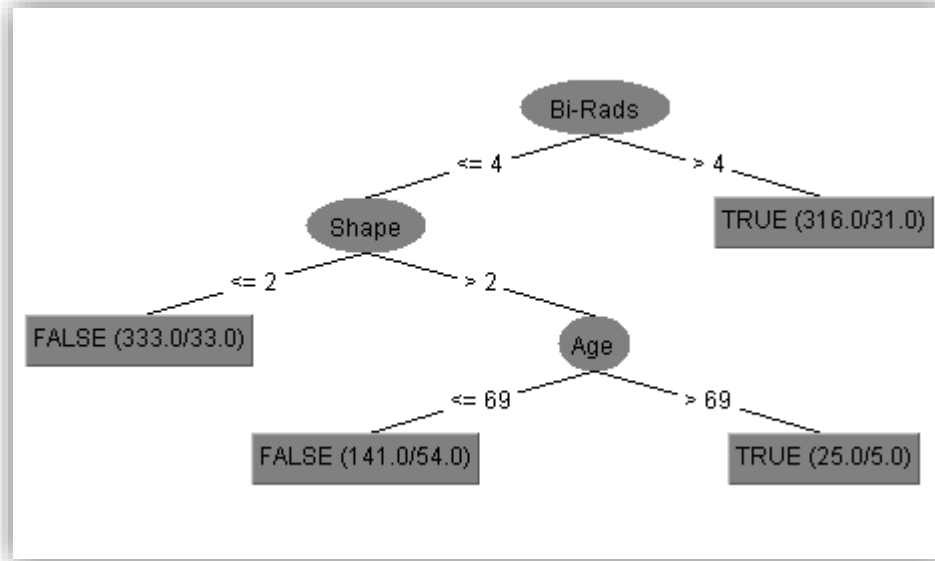


Figure 2.i: Image of Decision Tree generated with a CF of 0.15

The next step I will take is to edit the data within the dataset to match that of the above tree. The most obvious thing I noticed when analysing the tree created is the missing fields. There is no mention of the 'Margin' or 'Density' fields from within the dataset. I will separate the Bi-Rad field into two groups: greater than four ( $>4$ ) and less than or equal to four ( $\leq 4$ ), separate the Shape field into two groups: greater than two ( $>2$ ) and less than or equal to two ( $\leq 2$ ) and separate the Age field into two groups: greater than 69 ( $>69$ ) and less than or equal to 69 ( $\leq 69$ ). I will also be disregarding the Margin and Density fields to discover if this has any effect on the overall accuracy that currently sits at 83.63%.



### 2.3.2. EXPERIMENT SET TWO – DECISION TREES – IMPROVED PRE-PROCESSING

After completing the steps mentioned in the final paragraph of the previous experiment set I will no experiment on the newly edited dataset to determine if any improvement has been made. Again the first experiment undertaken will be the baseline experiment comparing the ZeroR, OneR and J48 default parameter classifiers.

Classifier	Percent Correct	Percent Incorrect	Mean Absolute Error
ZeroR	51.90	48.10	0.50
OneR	83.07 v	16.93 *	0.17 *
J48	84.91 v	15.09 *	0.23 *

*Table 2-7: Output of ZeroR, OneR and J48 using newly edited dataset*

As the results show the accuracy of the J48 when using the newly pre-processed dataset has increased by 1.28% when compared to the optimal J48 of the previous experiment set. The next step is to determine if the accuracy can be further improved when using the J48 classifier. As with the previous experiment set, I will begin with adjusting the CF values. The first experiment will use the CF values of 0.05, 0.1, 0.15 and 0.2 which will have the CF value of 0.25 as their benchmark.

Confidence Factor	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0.25	84.91	15.09	7.00	4.00
0.05	84.91	15.09	7.00	4.00
0.1	84.91	15.09	7.00	4.00
0.15	84.91	15.09	7.00	4.00
0.2	84.91	15.09	7.00	4.00

*Table 2-8: The effect of altering the confidence factor of the J48*

As the above table demonstrates, adjust the CF value for this dataset has no effect on the decision tree being constructed. The next step will be to determine if the MNO values, when altered, will have any effect on the decision tree.

Minimum Number of Objects	Percent Correct	Percent Incorrect	Tree Size	Number of Leaves
0	84.91	15.09	7.00	4.00
1	84.91	15.09	7.00	4.00
2	84.91	15.09	7.00	4.00
3	84.91	15.09	7.00	4.00
4	84.91	15.09	7.00	4.00
5	84.91	15.09	7.00	4.00

*Table 2-9: The results of the MNO adjustments*

Again the experiment shows no change to any aspect of the decision tree. I can only conclude here that the pre-processing techniques deployed on this particular dataset do not allow the J48 to be manipulated at all as it allows the J48 to construct the optimal decision tree using the default parameters.

```
J48 pruned tree
-----

Bi-Rads = >4: TRUE (316.0/31.0)
Bi-Rads = <=4
|  Shape = >2
|  |  Age = <=69: FALSE (141.0/54.0)
|  |  Age = >69: TRUE (25.0/5.0)
|  Shape = <=2: FALSE (333.0/33.0)

Number of Leaves   :    4
Size of the tree   :    7

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      692           84.908 %
Incorrectly Classified Instances    123           15.092 %
Kappa statistic                    0.6963
Mean absolute error                 0.2347
Root mean squared error             0.3436
Relative absolute error             47.0066 %
Root relative squared error         68.7679 %
Total Number of Instances          815

=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0.778    0.085    0.894     0.778    0.832     0.702    0.871    0.830    TRUE
                0.915    0.222    0.816     0.915    0.863     0.702    0.871    0.843    FALSE
Weighted Avg.   0.849    0.156    0.854     0.849    0.848     0.702    0.871    0.837

=== Confusion Matrix ===

  a  b  <-- classified as
305 87 |  a = TRUE
 36 387 |  b = FALSE
```

Figure 2.ii: Full summary of the optimal J48/C4.5 from Weka Explorer

My next step will be to put these datasets through the Multilayer Perceptron classifier and determine if the accuracy can be further improved.

### 2.3.3. EXPERIMENT SET THREE – NEURAL NETWORKS – BASIC PRE-PROCESSING

This series of experiments will determine the accuracy of a Neural Network when applied to the basically pre-processed dataset. The first experiment will be the baseline experiment using the ZeroR, OneR and Multilayer Perceptron (MLP) classifiers. The results are as follows:

Classifier	Percent Correct	Percent Incorrect	Mean Absolute Error
ZeroR	51.90	48.10	0.50
OneR	83.07 v	16.93 *	0.17 *
MLP	83.49 v	16.51 *	0.22 *

*Table 2-10: Output of ZeroR, OneR and MLP using edited dataset*

As the table above demonstrates the MLP is significantly more accurate than the ZeroR classifier. There are three parameters that will be altered during the following set of experiments. These are the number of hidden layers, learning rate and momentum. The first step will be to alter the number of hidden layers as this will be the quickest set of experiments to conclude.

#### 2.3.3.1. Hidden Layers

Hidden Layers	Percent Correct	Percent Incorrect	True Positive Rate
a	83.49	16.51	0.81
i	82.86	17.14	0.81
o	83.89	16.11	0.81
t	82.65	17.35	0.81
0	84.38	15.62	0.82
1	84.38	15.62	0.83
2	83.89	16.11	0.81
3	83.49	16.51	0.81
4	83.41	16.59	0.81

*Table 2-11: The results of editing the number of hidden layers in the network*

The above table shows the most accurate MLP to have either zero or one hidden layer. As the MLP with one hidden layer has a true positive rate of 0.83 while the zero hidden layer MLP has a true positive rate of 0.82 I will be using this former MLP for further experiments.

#### 2.3.3.2. Learning Rate

The next series of experiments will use a single hidden layer in the MLP and will use various levels of Learning Rate (LR) to determine to optimum LR for this specific dataset. The default parameter for this is 0.3 so I will be using 0.3 as my base LR and experiment with the LR set to the following values: 0.1, 0.2, 0.4 and 0.5 to determine if any of these are more accurate.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.3	84.38	15.62	0.83
0.1	84.59	15.41	0.83
0.2	84.33	15.67	0.83
0.4	84.42	15.58	0.83
0.5	84.36	15.64	0.83

*Table 2-12: Results of initial adjustments of the Learning Rate*

This result shows that an increased accuracy is achieved when using an LR of 0.1. The next step is to adjust this value to determine if a more accurate MLP can be found. The next set of will use the base value of 0.1 with adjust LR values of 0.05, 0.07, 0.09, 0.11, 0.13 and 0.15.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.1	84.59	15.62	0.83
0.05	84.66	15.34	0.83
0.07	84.62	15.38	0.83
0.09	84.62	15.38	0.83
0.11	84.53	15.47	0.83
0.13	84.41	15.59	0.83
0.15	84.30	15.70	0.83

*Table 2-13: Results of secondary adjustments of Learning Rate*

This experiment shows that with an LR of 0.05 a better classification is achieved. The next step will be to focus the numbers down to closer to 0.05 and attempt to achieve a better classification percentage. The next set of LR values to be experiment with are 0.03, 0.04, 0.06 and 0.065 as 0.7 was tested in the previous experiment. The results of these tests will be compared to the results of the LR of 0.05.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.05	84.66	15.34	0.83
0.03	84.76	15.24	0.83
0.04	84.64	15.36	0.83
0.06	84.66	15.34	0.83
0.065	84.64	15.36	0.83

*Table 2-14: Results of next adjustments of Learning Rate*

Again, this experiment found a more accurate LR so the next experiment will use the LR of 0.03 as its base and test the LR values of 0.02 0.025 and 0.035.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.03	84.76	15.24	0.83
0.02	84.79	15.21	0.83
0.025	84.76	15.24	0.83

*Table 2-15: Results of Learning Rates closer to the most accurate*

Again this experiment found a more accurate LR when using 0.02 as its value. The next experiment will use the LR values of 0.02 as the benchmark with the values 0.015, 0.017, 0.019, 0.021 and 0.023 being tested.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.02	84.79	15.21	0.83
0.015	84.86	15.14	0.83
0.017	84.81	15.19	0.83
0.019	84.79	15.21	0.83
0.021	84.82	15.18	0.83

*Table 2-16: Results of Learning Rates closer to the most accurate*

Again, a more accurate LR value has been found so the next experiment will use the LR values: 0.013, 0.014, 0.016 and 0.016 being compared to the LR value of 0.015.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.015	84.86	15.14	0.83
0.013	84.82	15.18	0.83
0.014	84.85	15.15	0.83
0.016	84.84	15.16	0.83
0.017	84.81	15.19	0.83

*Table 2-17: Results of Learning Rates closer to the most accurate*

As the above table shows, no improvement was made during this experiment. The next experiment will use the LR values 0.0149 and 0.0151 to determine if smaller values closer to the best one will provide more accurate results.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.015	84.86	15.14	0.83
0.0149	84.86	15.14	0.83
0.0151	84.86	15.14	0.83

*Table 2-18: Results of Learning Rates closer to the most accurate*

As you can see no improvement was made closer to the value of 0.015. Therefore my optimal MLP has an LR value of 0.015. The next logical step will be to adjust the Momentum parameter and determine if an even more accurate MLP can be found.

### 2.3.3.3. Momentum

The next series of experiments will adjust the Momentum (M) and attempt to construct a more accurate MLP for this dataset. The first experiment will use an MLP with a single hidden layer and an LR of 0.03 as its default parameters. When M is left at its default value the resulting accuracy is that of the last experiment of the previous experiment series (84.86%). This will be used as my baseline for the next test and I will use the following values for M: 0.1 and 0.3.

Momentum	Percent Correct	Percent Incorrect	True Positive Rate
0.2	84.86	15.14	0.83
0.1	84.84	15.16	0.83
0.3	84.81	15.19	0.83

*Table 2-19: Initial results of adjusting the Momentum*

As can be seen in the above table adjusting the momentum after adjusting both the number of hidden layers and LR has decreased accuracy of the MLP. Therefore I can conclude that this dataset has an optimal accuracy rating of 84.39% and the optimal MLP has a single hidden layer, an LR of 0.03 and the momentum of 0.2. This MLP has a better accuracy than the optimal J48 decision tree when using the basic pre-processed dataset and is therefore deemed as the better classifier. The next step will be to attempt to improve the accuracy generated by the J48 when using the improved dataset.

#### 2.3.4. EXPERIMENT SET FOUR – NEURAL NETWORKS – IMPROVED PRE-PROCESSING

This next set of experiments was completed using the improved dataset created using the groupings outlined at the end of the first set of experiments. Again the first experiment will compare the accuracy of the ZeroR, OneR and MLP classifiers all with default parameters.

Classifier	Percent Correct	Percent Incorrect	Mean Absolute Error
ZeroR	51.90	48.17	0.50
OneR	83.07 v	16.93 *	0.17 *
MLP	84.91 v	15.09 *	0.23 *

*Table 2-20: Output of ZeroR, OneR and MLP using improved dataset*

The initial experiment shows that the MLP actually has the same accuracy level of the J48 when using the same dataset. My next step is to adjust the number of hidden layers. I will test the following values: 0, 1, 2 and 3 as these yielded the most significant results in the previous experiment set.

Hidden Layers	Percent Correct	Percent Incorrect	True Positive Rate
0	84.77	15.23	0.78
1	84.66	15.34	0.79
2	84.91	15.09	0.78
3	84.91	15.09	0.78

*Table 2-21: The results of editing the number of hidden layers in the network*

As the table above suggests the accuracy is not improved during this experiment. The next experiment will use two hidden layers and will have the M parameter adjusted to the following values: 0.2 and 0.4 as the default value for this parameter is 0.3.

Momentum	Percent Correct	Percent Incorrect	True Positive Rate
0.3	84.91	15.09	0.78
0.2	84.91	15.09	0.78
0.4	84.91	15.09	0.78

Table 2-22: Effect of Momentum adjustment on improved dataset

Again, this experiment shows no improvement. The final experiment will be to adjust the learning rate to determine if this has any effect on the accuracy of the classifier.

Learning Rate	Percent Correct	Percent Incorrect	True Positive Rate
0.3	84.91	15.09	0.78
0.1	84.91	15.09	0.78
0.2	84.91	15.09	0.78
0.4	84.91	15.09	0.78
0.5	84.91	15.09	0.78

Table 2-23: Effect of adjusting the learning rate on the improved dataset.

As the final experiment above shows there has been no increase in the accuracy of the classifier in this final set of experiments. I can again determine, as in the J48/C4.5 experiment set, that this dataset allows the classifier to create the optimal MLP with all the parameters having the default values. The following figure is the representation of this MLP using the Weka Explorer with the GUI enabled:

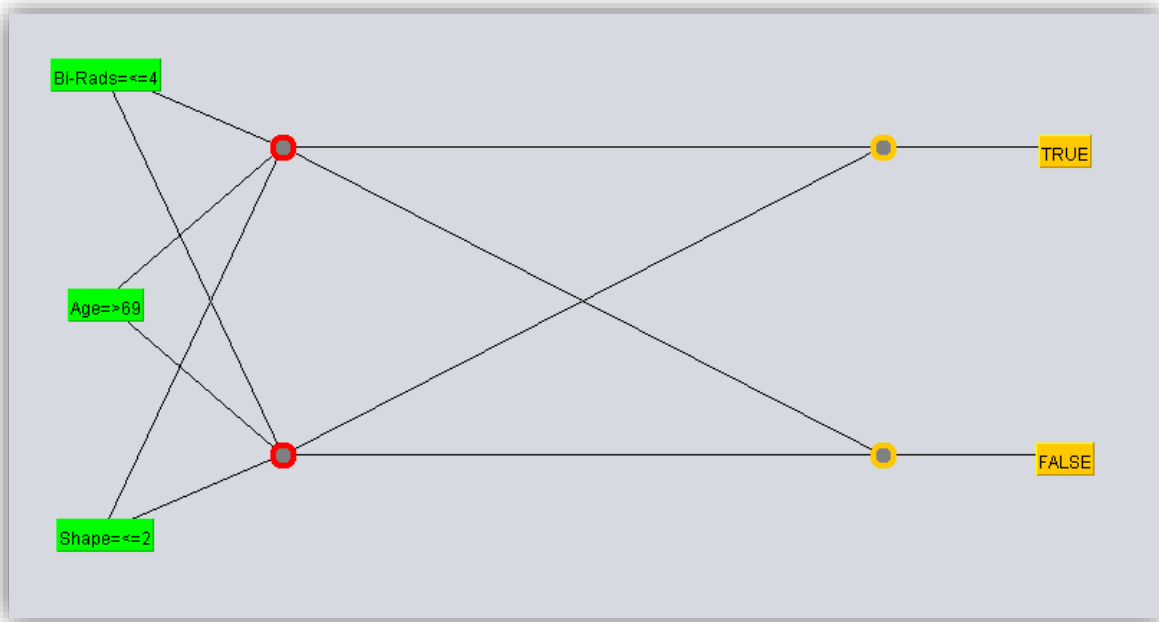


Figure 2.iii: Optimal MLP as shown in Weka Explorer



## 2.4. CONCLUSION

After completing this scenario I can conclude that the pre-processing of data before submitting it to an AI classifier can dramatically improve the performance of the classifier. Not by increasing the accuracy, but by allowing the user to build the optimal classifier with the default settings of their chosen classifier. When I conducted experiments on the 'basic' pre-processed dataset it took five experiments to create the optimal Decision Tree and nine experiments to create the optimal Neural Network using the J48 and Multilayer Perceptron classifiers respectively. The key to this pre-processing seems to be the grouping of sets of attributes together within a field. Grouping the Bi-Rads, Age and Shape fields, as well as disregarding the Margin and Density fields, allowed me to construct the best performing Decision Tree and Neural Network with little to no effort. I can conclude that after completing all of the above experiments that I cannot achieve an accuracy rating of higher than 84.91% using either the J48 or MLP classifiers and, as such, can declare both classifiers as equally idyllic for use when predicting malignant masses from the mammogram dataset.

## Images Used

Image 1: Image obtained from <https://futureoflife.org/background/benefits-risks-of-artificial-intelligence/>

Image 2: Image obtained from <https://www.codeproject.com/Tips/865252/How-To-Do-Simple-Image-Binarization>

Image 3: Image obtained from <http://blog.mmast.net/python-image-processing-libraries-performance-opencv-scipy-scikit-image>

Image 4: Image obtained from <http://dana.loria.fr/doc/image.html>

## References

Cs.waikato.ac.nz. (2017). Weka 3 - Data Mining with Open Source Machine Learning Software in Java. [online] Available at: <http://www.cs.waikato.ac.nz/ml/weka/> [Accessed 9 Apr. 2017].

### *ZeroR*

Brownlee, J. (2017). Overfitting and Underfitting With Machine Learning Algorithms - Machine Learning Mastery. [online] Machine Learning Mastery. Available at: <http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/> [Accessed 18 Feb. 2017].

Bhargava, N., Sharma, G., Bhargava, R. and Mathuria, M., (2013). Decision tree analysis on j48 algorithm for data mining. Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering, 3(6).

Patel, N. and Upadhyay, S., (2012). Study of various decision tree pruning methods with their empirical comparison in WEKA. International journal of computer applications, 60(12).

Gerardnico.com. (2017). Data Mining - Pruning (a decision tree, decision rules) [Gerardnico]. [online] Available at: [http://gerardnico.com/wiki/data\\_mining/pruning](http://gerardnico.com/wiki/data_mining/pruning) [Accessed 9 Apr. 2017].

Nielsen, M. (2017). Neural Networks and Deep Learning. [online] Neuralnetworksanddeeplearning.com. Available at: <http://neuralnetworksanddeeplearning.com/chap2.html> [Accessed 9 Apr. 2017].

Pages.cs.wisc.edu. (2017). A Basic Introduction To Neural Networks. [online] Available at: <http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html> [Accessed 9 Apr. 2017].

Wilson, D.R. and Martinez, T.R., (2001). The need for small learning rates on large problems. In Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on (Vol. 1, pp. 115-119). IEEE.

Willamette.edu. (2017). Momentum and Learning Rate Adaptation. [online] Available at: <https://www.willamette.edu/~gorr/classes/cs449/momrate.html> [Accessed 9 Apr. 2017].

Jolliffe, I. (2007). Principal component analysis. New York: Springer.

Itl.nist.gov. (2017). 7.1.6. What are outliers in the data?. [online] Available at: <http://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm> [Accessed 9 Apr. 2017].