# Data Mining & Machine Learning
# (F20DL)

## Coursework 3

**By**
**Cameron McBride, Chris Walsh, Jordan Walker**

**Date: November 28th, 2018**

**Link to all documents including scripts, data Files etc. can be found at:**

**https://github.com/cdw2/data_mining_cw3**

***will be public after Submission Date.**

***All members of this coursework contributed equally to the final output of this document.***

## Tasks One and Two

In this part of the coursework, we have been asked to use a new dataset from what we have been using for the duration of the past two coursework namely the fer2018.arff and all of its relevant subsidiaries fer18EmotionX.arff. Given this, the new data set we will be working with is the big "emotion recognition" dataset created by the research group of Pierre-Luc Carrier and Aaron Courville which contains the following set of CSV's:

1. fer2017-training.csv and fer2017-testing.csv
2. fer2017-training-happy.csv and fer2017-testing-happy.csv

The dataset itself contains the same amount of examples, however, instead of the full 35887 in one file as used previously this new data set uses 28709 examples for training and 7178 for testing. Changes had to be made however to the initial scripts we had created as the files for happy were formed in an almost "R" fashion where the class(emotion) was at the end which varied from the initial data files we used which had the class(emotion) at the start and then the appropriate numeric values following. Changes also had to be accounted for that in both fer2017-training-happy.csv and fer2017-testing-happy.csv the entries are split by a comma rather than just a space between each entry which was used in all previous data files.

From the previous coursework(s), by use of various Python scripts and an overall wrapper that automates all the necessary operations to be completed, we can easily convert all the required files into the correct format so we can ultimately run the tasks needed for research.

To minimise the issue faced by using big data sets we have increased the heap size for Weka where we raised it to 6GB, and we have also used our implementation of reducing the image size from a 48x48 image to a 24x24 image by merging corresponding pixels. From tests carried out during Coursework 1 and 2, we have found that by using this technique we have not suffered a significant decrease in accuracy and some tests out-performed the full-sized images. In further addition, we have continued using SubSet Evaluation to get us the "best" pixel attributes that we will use throughout the Coursework.

During coursework two and a large number of tests that were needed to gain results that could be used for comparisons and conclusions we have implemented our scripts to run multiple tests in parallel. This dramatically reduces the time needed to gain all the necessary results and as a result, means we can spend more time analysing and understanding the results we may or may not receive in the duration of this document.

For task two we include all our scripts, results and screenshots in a shared Github folder that can be accessed by all those assessing this document after the project has been completed, and the deadline date has been passed. The Github file can be accessed at https://github.com/cdw3/data_mining_cw3. Once located the corresponding tasks and results are presented to the assessor and are laid out in a logical format and contain all items and appendices to the Coursework 3.

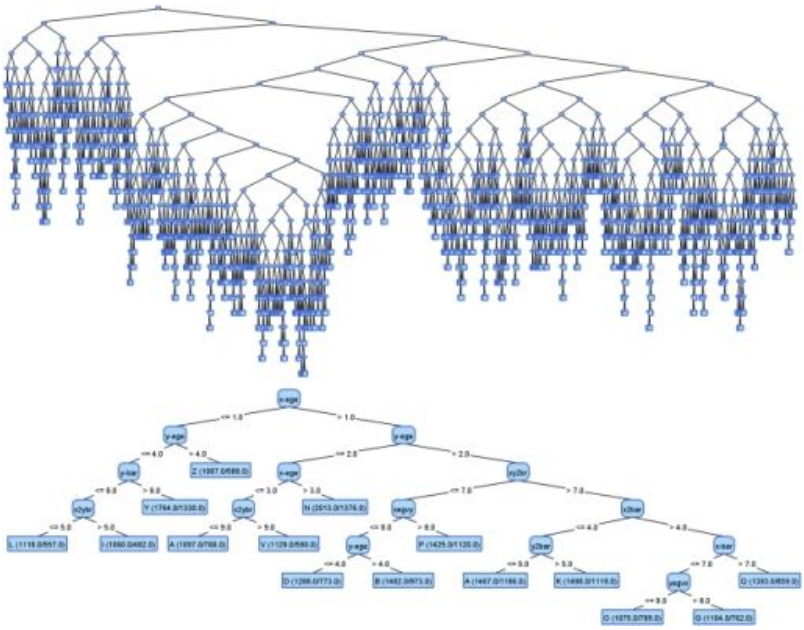
## ***Part 1 – Decision Trees***

In this section of the assignment, we will use decision trees which are a form of supervised learning algorithms and are essentially easy to understand and use. The idea by using trees is by use of the data set a predictive model can be mapped and visualised as a tree can be used. This form of classification aims to gain the highest level of accuracy with the most minimal number of decision splits possible.

The basic algorithm for learning decision trees is:

1. Load the training data
2. Select the attribute the gives the best split for decision
3. Create children from this parent based on the divide
4. Continue this method recursively until complete (or specific termination standard has been accomplished).

When building the tree manually, the main issue faced is what attribute/value to split on. However, for this, we employ entropy calculations which are used to measure of uncertainty to calculate the highest (or lowest dependant on the dataset) value which will be used to select the current attribute which will become the subsequent parent of all following nodes which are subject to evaluation.

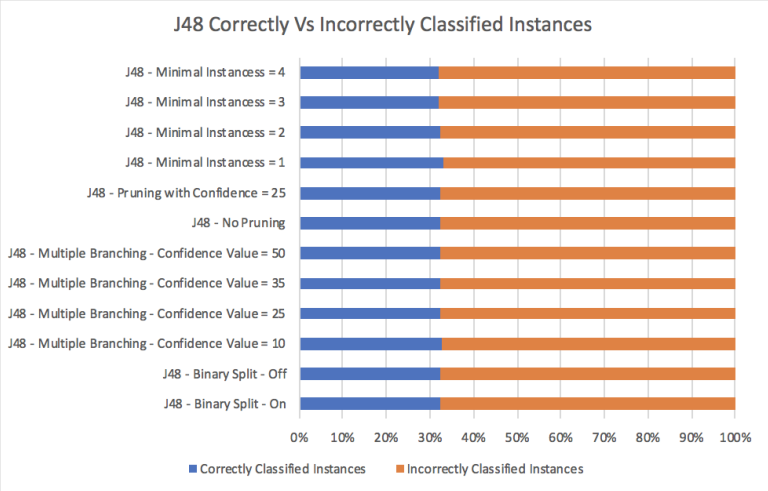


A tree classifier is what can be established as a recursive partition of the search space beginning with the initial root node which has no incoming edges where all following nodes will have at least one incoming. Upon each split from a node the instance space is then theoretically split into two or more. The end nodes don't always have to correlate to one class but can in some cases be used as a probability vector [2].

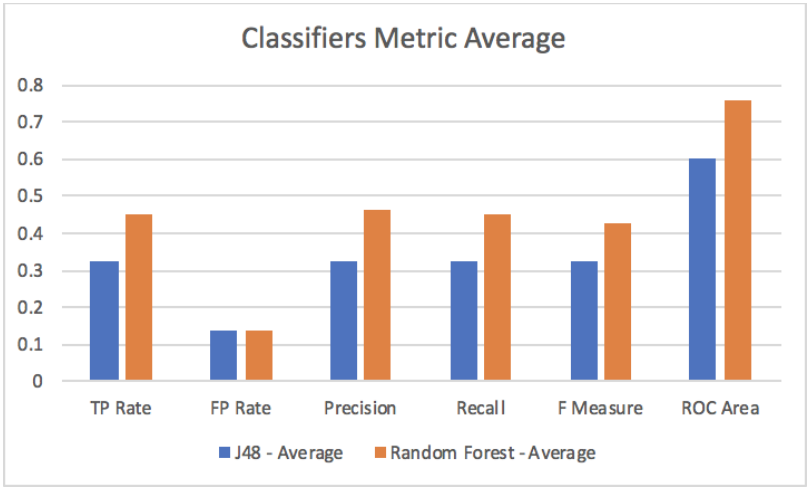
## Task Three

For this task, we were tasked to run the Decision Tree Classifier(s) J48 and Random Forest on the provided datasets using 10-fold cross-validation and to record and analyse the results we obtain upon completion. During these experiments we also used the User Classification on smaller subsets of the dataset.

During this task, we recorded various outputs including Accuracy, TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area (detailed meanings of each of these measurements can be found in the Appendix section A).



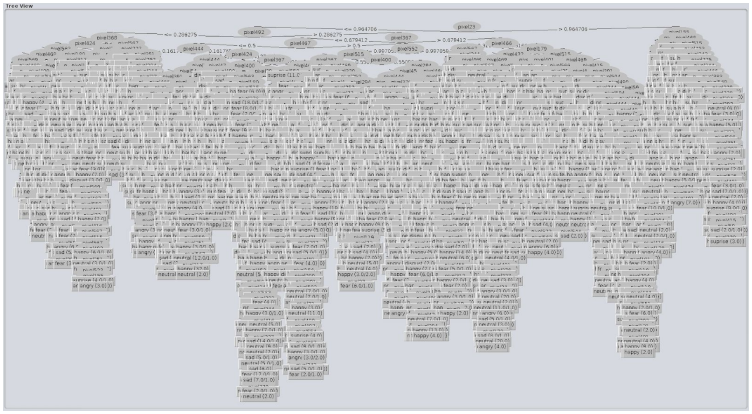

As seen in the left graph various modifications on the parameters of the J48 tree did very little to have much effect on the correctly classified instances with most results being within the range of 2% between the worst and best. This suggests that the dataset is not ideal for the j48 tree classifier as varying degrees of the parameter setting do not affect the overall performance drastically enough.




In the right we can observe the difference in the average results generated by the confusion matrix produced by the various classifiers. What is most apparent is the overwhelming increase in specific metrics such as TP rating, Precision and ROC area indicating a much better performance by the Random Forest classifier. This is also derived from the observation that the FP rate between the J48 and Random Forest Implementation remains virtually the same with a slight decrease for Random Forest.

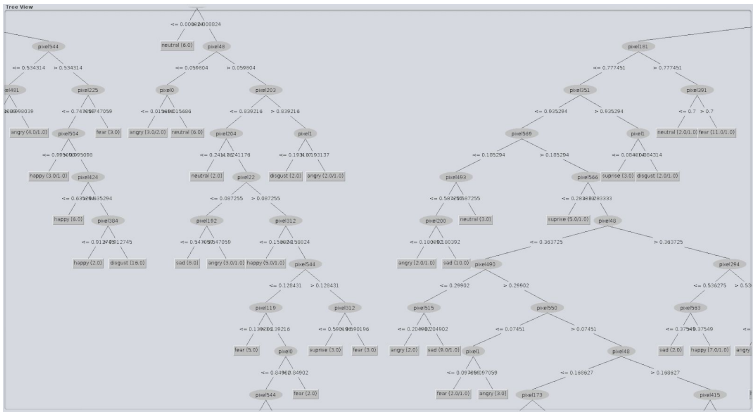
## Task Four

In task four we have been asked to visualize the obtained information from task three. This is done in means of further understanding the data and also in creating comparisons between the different Decision Tree Classifiers. To achieve this we used the inbuilt weka tree visualization tools to better explore and understand our trees. Below you can see the whole J48 tree that was constructed, it is quite a broad tree and quickly splits into many branches, identifying the multiple attributes and it is also quite deep showing the identification of the various features the combination of those attributes can produce.



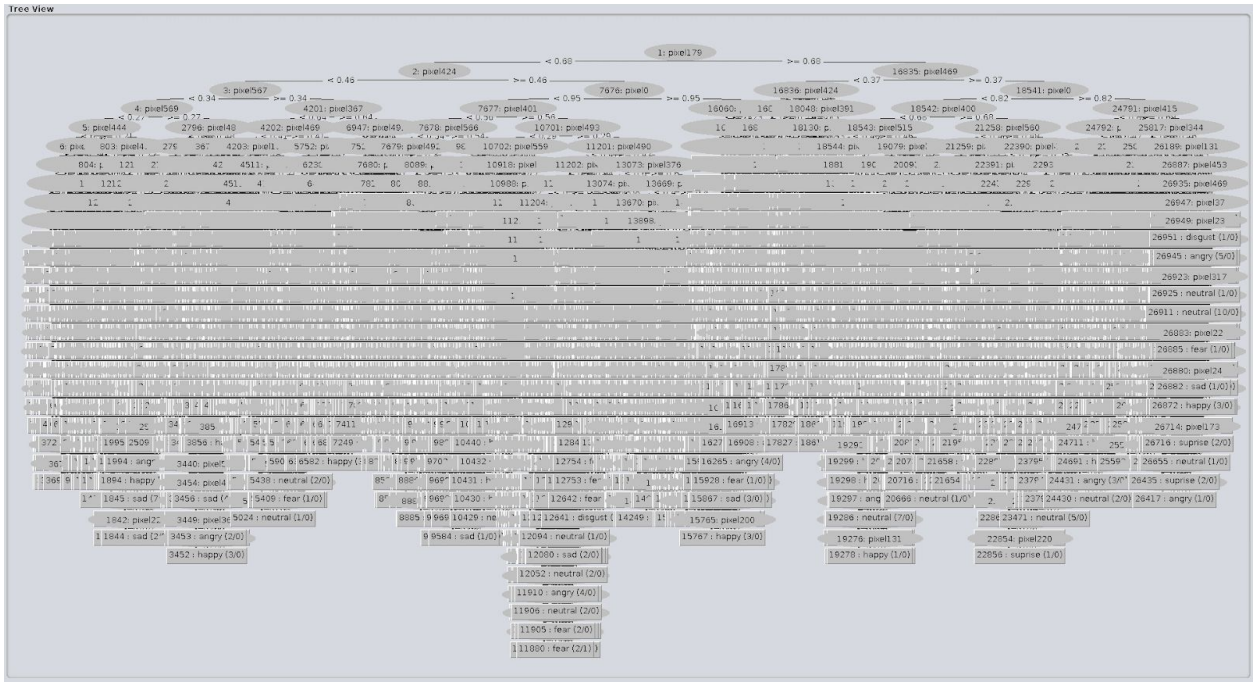
**J48 Whole**

Below you can see a portion of our zoomed J48 tree showing that it is mostly binary splitting that is occuring.



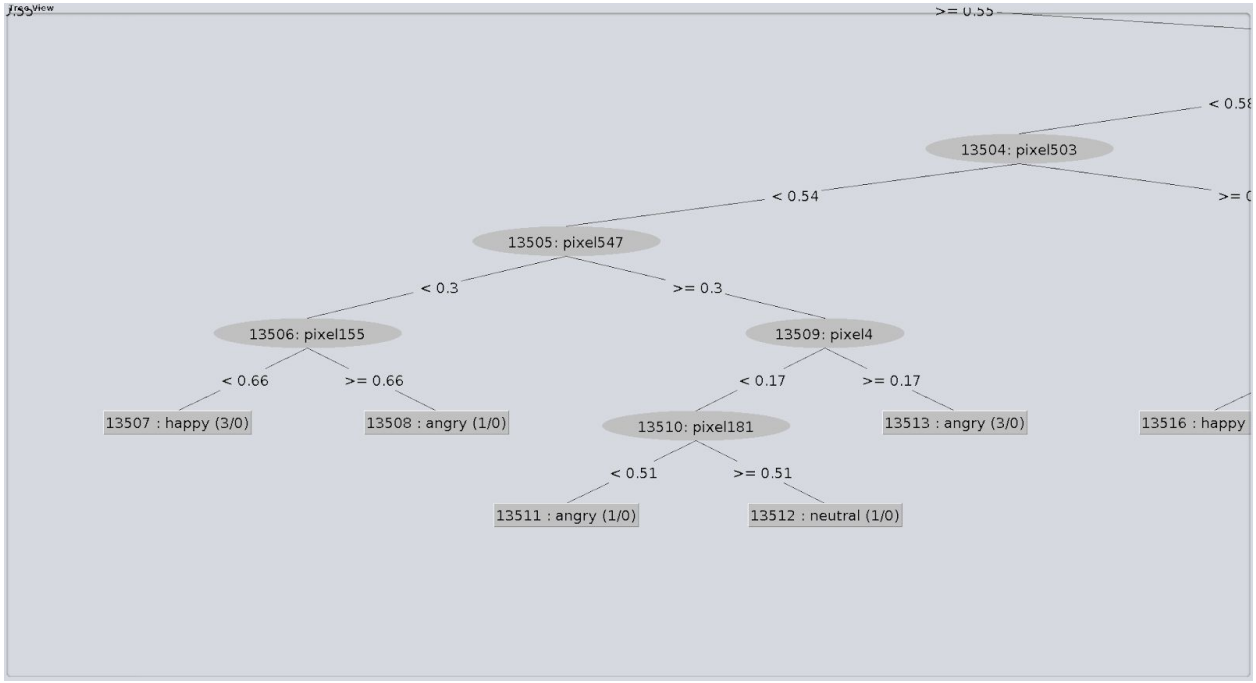
**J48 Zoomed**

Below you can see a visualisation of one of our whole random forest trees. Due to the way random forest works by aggregating a large number of trees you can see that this tree is much more complex.



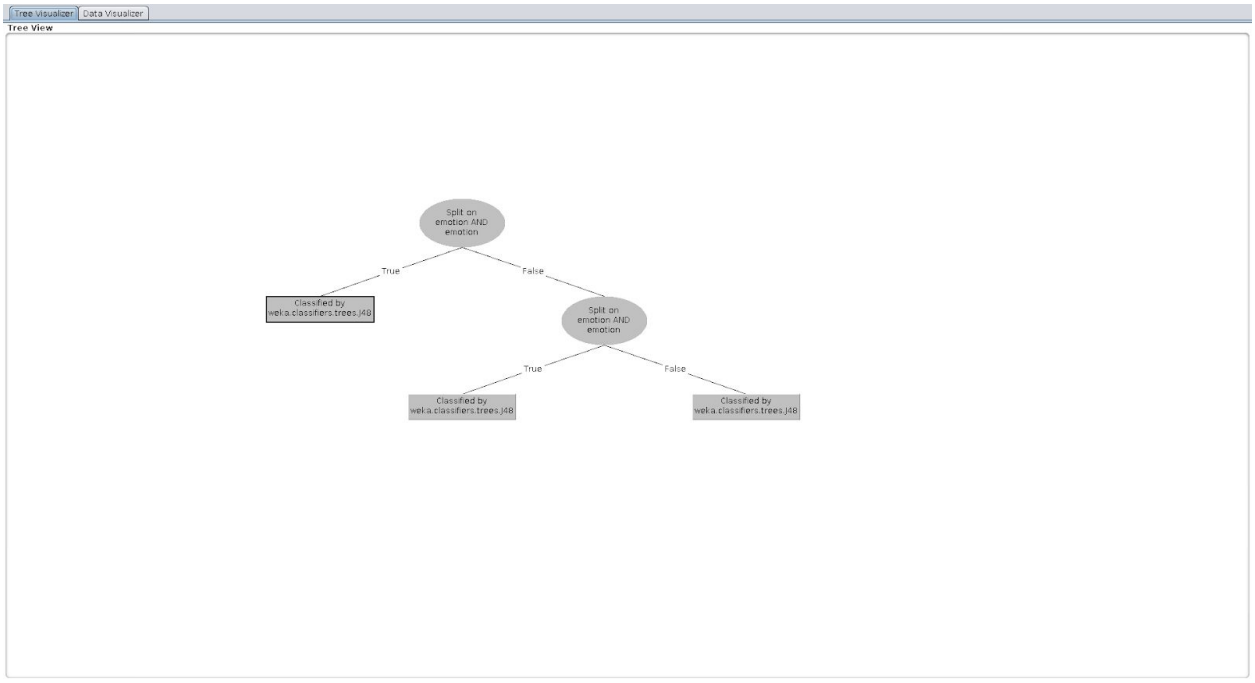

**Random Forest Whole**

When zoomed in you can see the probabilities calculated by the majority voting method used when aggregating the trees.



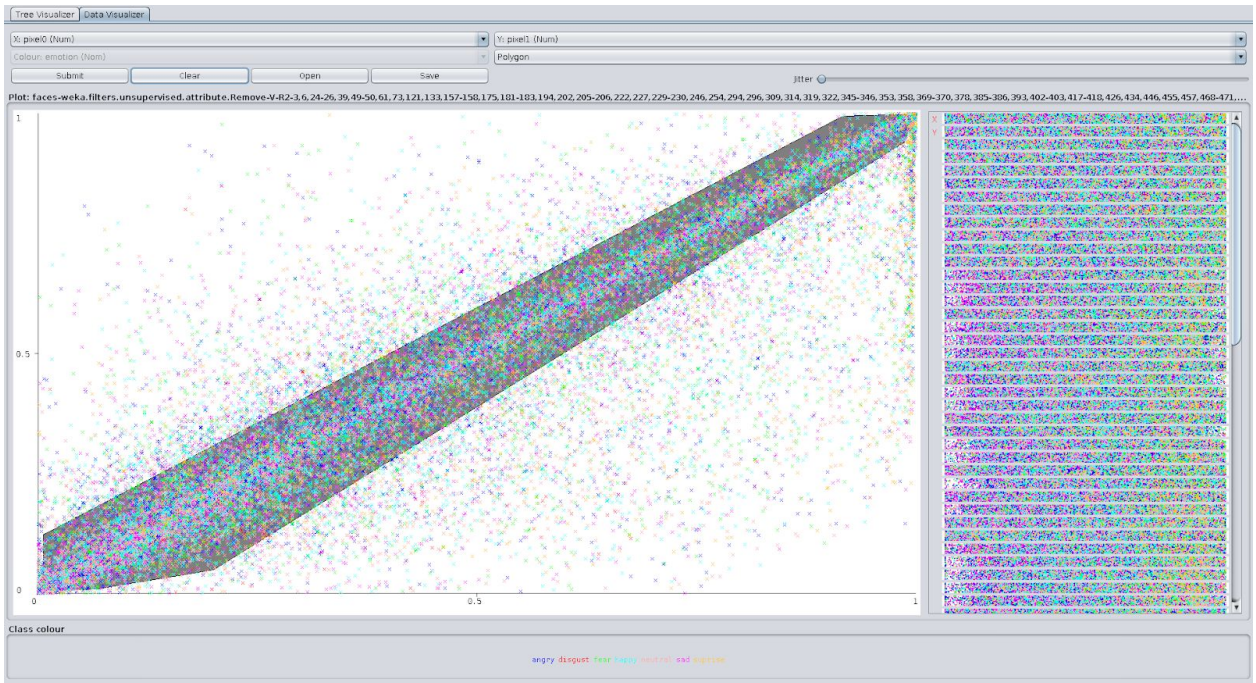

**Random Forest Zoomed**

Below is a visualisation of our user classifier, you can see we made two splits on the hardest to classify emotions then allowed the J48 algorithm to build the rest of the tree. This allowed us to construct trees semi-manually and investigate hypotheses we had about the data.




**User Classifier Tree**

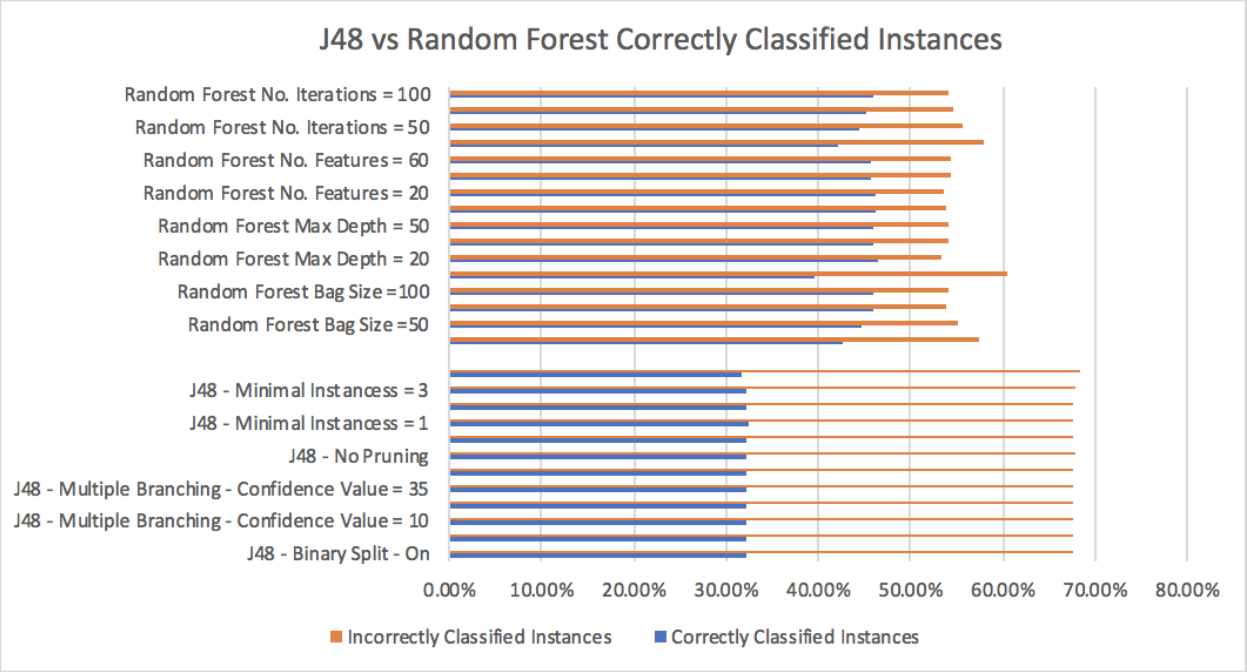
Below you can see how certain data is selected by drawing shapes around data points in the data visualisation tab.



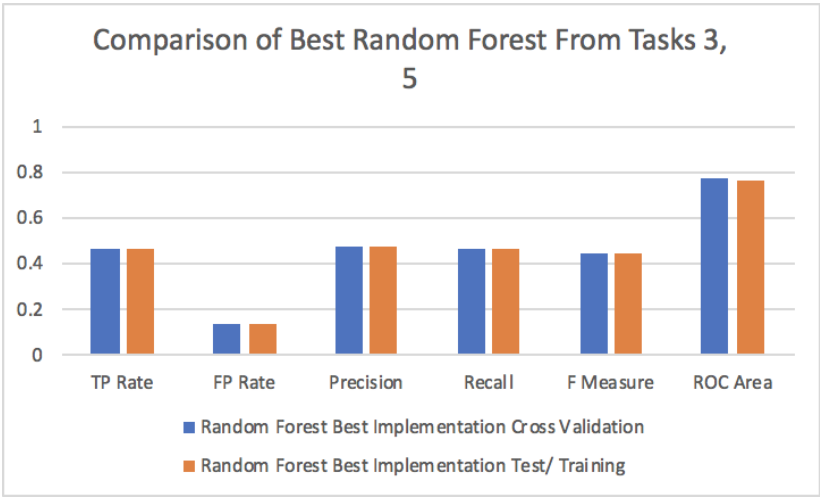

**User Classifier Data Selection**

**Task Five**

Task Five instructs that we instead of using Cross-Validation as done in Task 3, we are to use a Hold-Out method where we supply test and training data sets to the classifier. The objective of this is to record the data obtained and to then compare and contrast the results obtained.



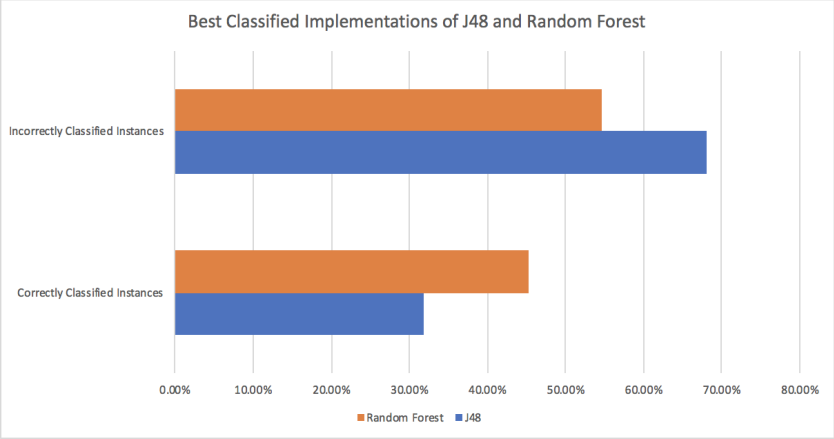

After completion of task five we can observe that the Random Forest Algorithm is still outperforming the J48 classifier by a large percentage increase. What is indicated by this is that the use of Random Forest is more ideally suited for the dataset.




In this graph we can observe that by using a test / training split the Random Forest Classifier slightly underperforms compared to the Random Forest enabled with Cross Validation. This is assumed that the 90 / 10 split endorsed by Cross Validation helps the model be more accurate compared to the 70/30 split automatically set by Weka's Holdout.

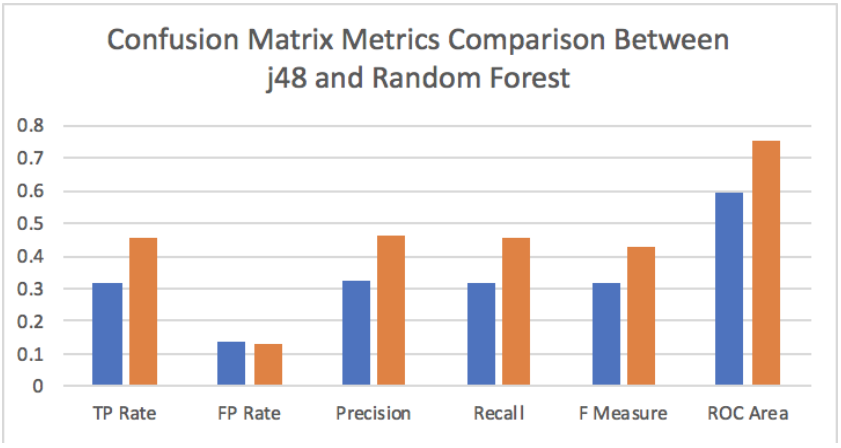
## Task Six

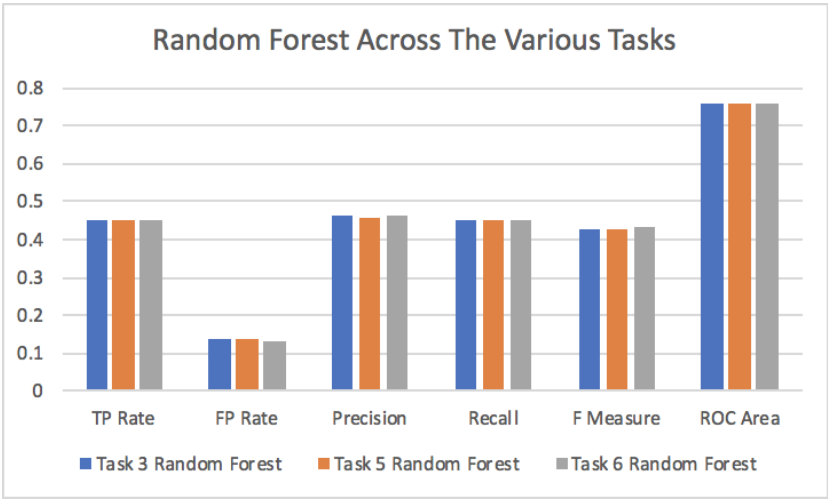
In task six we have been specified to create our own test/ training data set which we will use for our classifiers. The split is 70 / 30 where the 70% is dedicated to training and the remaining 30% is dedicated to testing. Following we have documented our interesting findings.




Again it is observed that Random Forest outperforms J48 on this dataset with over a 10% increase in correctly classified instances compared to J48's best classification result.

The confusion matrix produced by both the best results obtained from both J48 and Random Forest in task six reinforces that Random Forest still outperforms J48 in relation to classifying the given data set accurately.



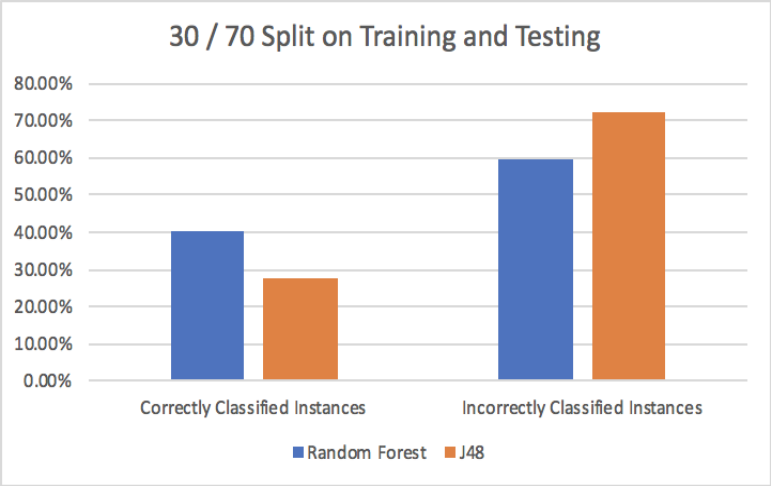




As one can see Random Forest performs well across all tasks gaining similar results from all the metrics given as an average from all tests ran and their output confusion matrix.

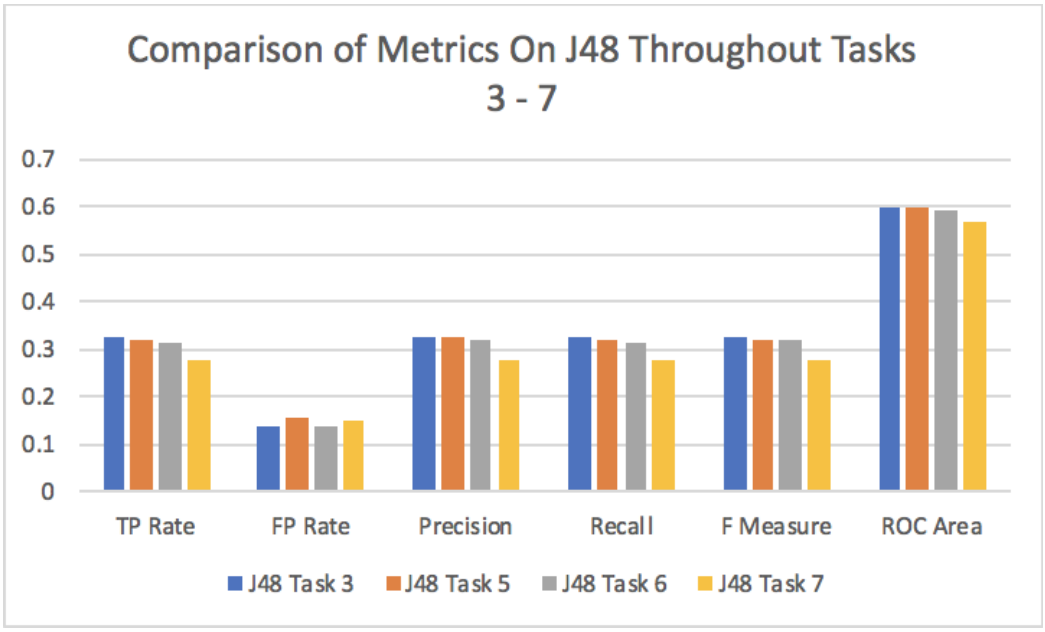
This indicates a true representation of the classifiers ability and is not a result from how we prepared the data.

**Task Seven**

For task seven we were asked to do another split on the data set this time a 30 / 70 split was asked for meaning 30% would be used for training and 70% for testing. Following are our results from these tests.




Throughout these tests we have seen that J48 has not been an ideal classifier for this complex dataset. For this example however what could be assumed is that J48 is suffering from a lack of training and hence is performing poorly against Random Forest whos consistent results in further indicating that overfitting is not occuring within this classification method.



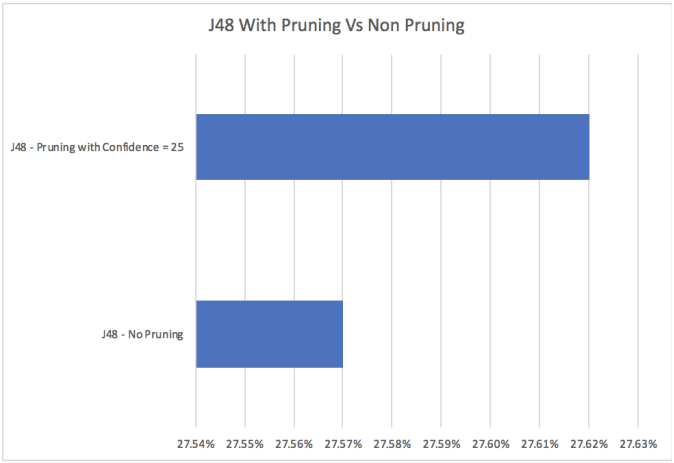

After the initial results of task three we assumed that J48 was the poorest classifier for this dataset based against Random Forest. We can observe that the method of data preparation has an effect to the overall performance of the model compared to Random Forest which can create a more desirable model for the complex data set we are using.

## Task Eight

When working with tree classification models the term over-fitting refers to the classifier tightly fitting the given training data so much that it would be inaccurate in predicting the outcomes of the untrained data.

In decision trees, overfitting can occur when the tree itself has been built to perfectly satisfy all cases within the training data set which in turn creates strict rules the model must follow and results in branches for this sparse data.

Resulting from the when new data is given to the tree to "predict" the accuracy of the given model is not fully accurate. However, by means of parameter settings such as using pruning which is a technique where once the training has been completed the branches for sparse data are removed such that the overall accuracy is not disturbed.




As seen in the figure above the increase of correctly classified instances increases when pruning is used however only slightly. To fully utilise this feature other parameters have to be adapted to allow for a better overall performance.

Overall we label the success rates we got with Random Forest implementation is of a better performance due to it being harder for a Random Forest to actually overfit. This is due to the way that random forest works. It constructs its final tree by aggregating many trees grown using a random sample of the training data. This means that each individual tree is grown with a potentially small subset of the training data, (dependant on algorithm parameters), making it unlikely to overfit. When all the trees in the "forest" are aggregated together you get a tree capable of classifying all of the data that has been seen by the many trees, but with a low level of overfitting because of the method in which each individual tree has been constructed.

## Conclusion of Decision Trees

From our research we have found that trees as an overall have these advantages and disadvantages:

| Advantages | Disadvantages |
|---|---|
| Trees are graphical which allows for easy viewing the sequential pattern that has been generated and finding where each attribute is in order of importance. | The decision tree performs a divide and conquer technique which works well with small and non-complex datasets but can impose major issues when working with larger more complex datasets. |
| Trees generally can be computed quickly which allows for fast analysis when comparing new data and new methods of initial parameters. | Trees are essentially greedy and because of this, there can be very subject to error from noise contained within the dataset. |
| Decision trees can handle both nominal and numerical values | Most of the algorithms require that the target attribute will have only discrete values. |

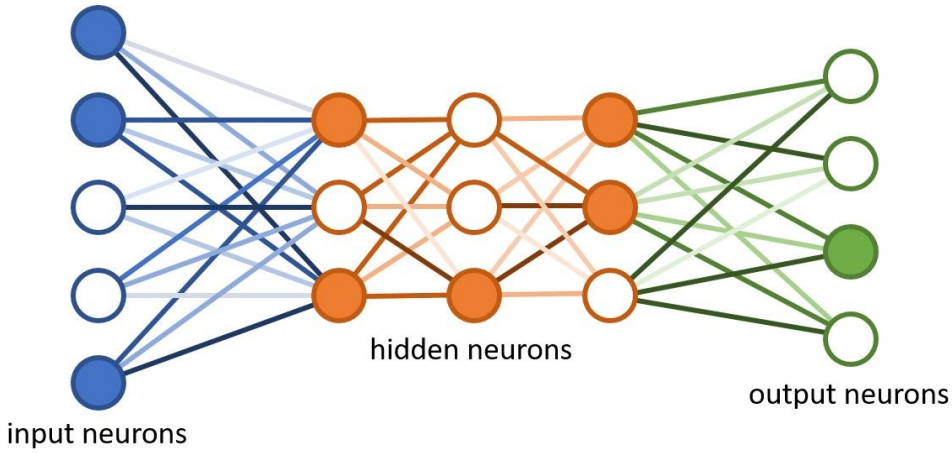
Overall the decision tree can be thought of as a simplistic idea to solve the classification problem. Because of its simplicity it allows for wide use and is a model that is not restricted to the logic contained in transistors of the users device. What is meant by this is that it does not take a specific knowledge compared to many of the other classifiers that have been used throughout this course and anyone who can follow rules can make their own classification tree in practice.

The process of a rule based classifier where every node asks a questions where the entry is then passed to the next node and so on based on the answer the entry gives essentially works in many data sets where there is more outlying rules that can further separate the data and given the data set is correctly entered or is small the accuracy of a decision tree can be very high without succumbing to the overfitting problem imposed into classifiers.

However it has to be taken into consideration that even though a decision tree in the right circumstances can work as a substantial classifier the issue remains that finding an optimal tree is impossible. The reason for this is due to to finding an optimal tree being an NP-complete problem. Many of the decision tree models utilise an heuristic based approach which essentially is a greedy algorithm which is used to search the hypothesis space. This results in trees being able to be generated quickly and retrospectively computationally inexpensive but with no guarantee that the tree generated is better than another [4].

## ***Part 2 – Linear Classifiers and Neural Networks***

For this section of the assignment, we will use Artificial Neural Networks (ANN) which are classification modules that are based on the the biological organ the brain to process information and make informed predictions through either supervised or unsupervised methods. The novelty of an ANN is in the structure which includes a number of processing neurons which can act in parallel to produce results and just like the biological organ they aim to replicate they learn from examples i.e. training data. For the duration of this assignment, the Neural Networks will be supervised as we are providing the network with labelled training instances.

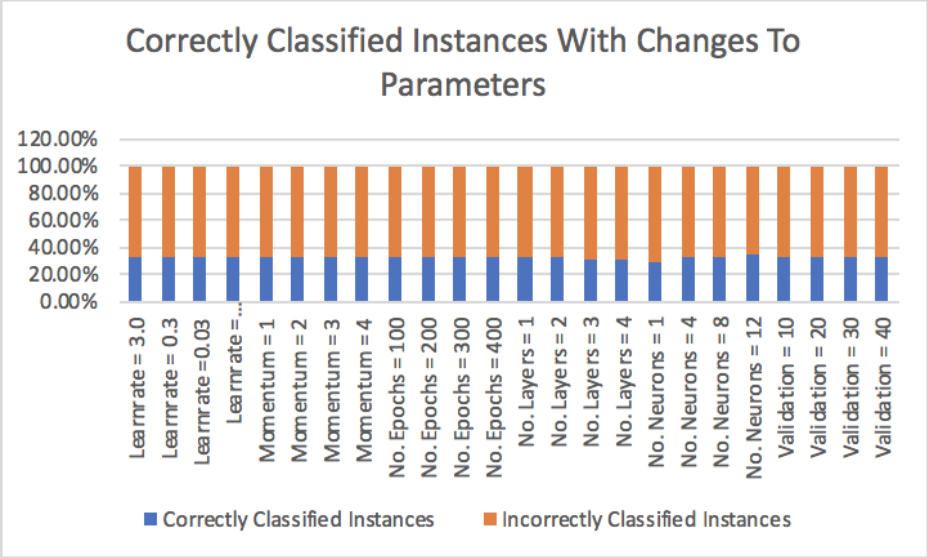



Items passed into Neural Network through the input neurons known as the input layer generally flow through a set of nodes laid out in organised layers. The nodes that encompass each layer contain what is known as an activation function such as TanH or Sigmoid.

The hidden layers containing the hidden neurons are where the processing is calculated by use of weighted connections between each of the nodes. These weighted connections can be modified by the learning rule and are most likely updated at each epoch (iteration) and can help increase the overall accuracy of the model.

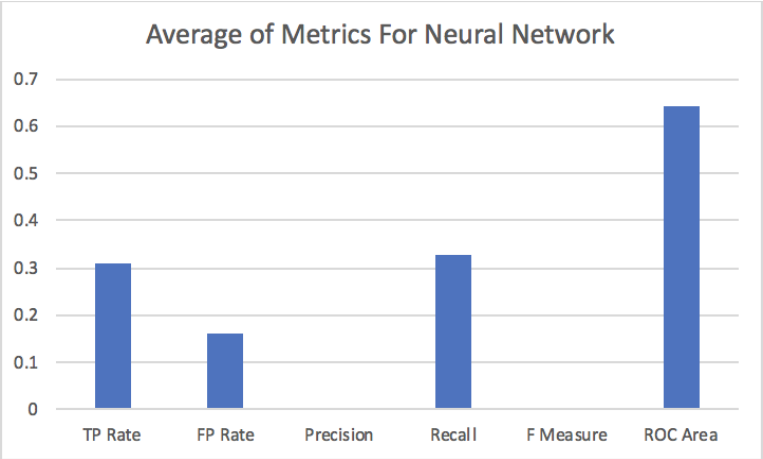
**Task Three**

For the duration of our tests we modified the following values to collect data on the various parameter settings. Firstly we changed the learning rates which we took data from Wekas preset value of 0.3 which was modified to test on 3, 0.03 and 0.003. We also altered the momentum, the number of epochs, the number of neurons and the number of layers, you can see graphs of these results below.



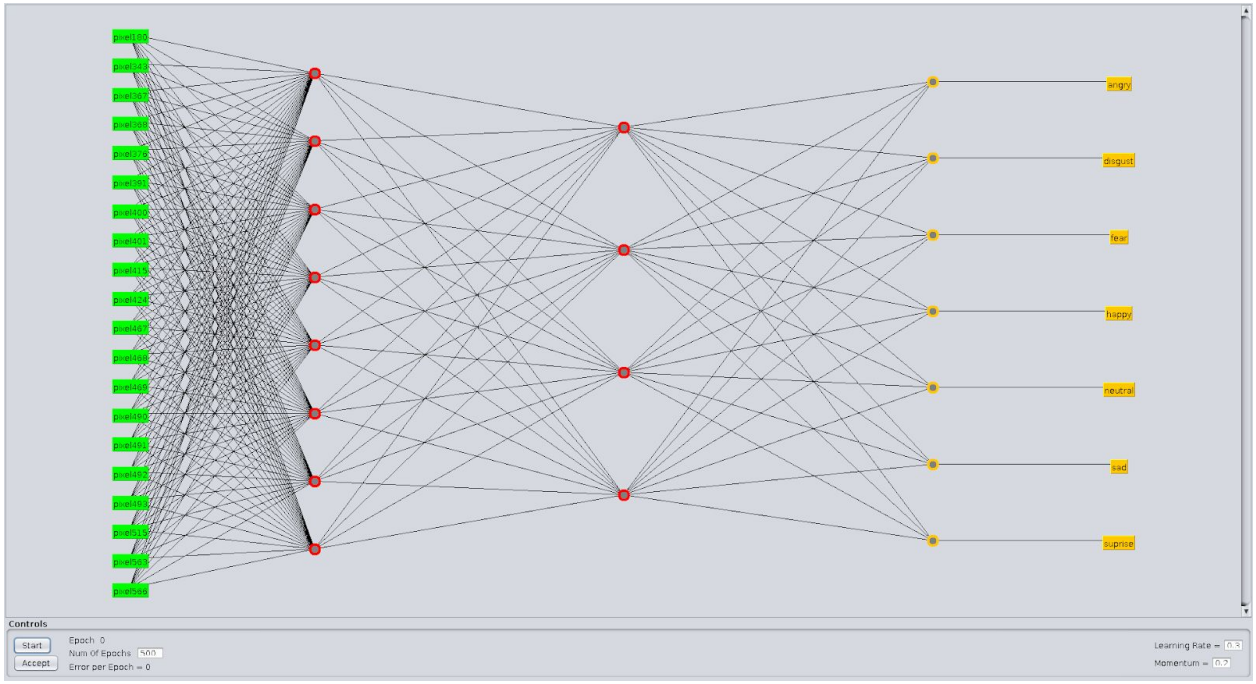

What we encountered very quickly when working with Neural Networks is how deterministic they are as a terms of classification model. Although in the graph above indicates not much of a change in correctly classified instances when changing various parameters it is found that it is not just the increasing of one parameter that will be a result of better performance but more a combination.

We also found that when using Neural Networks observing the generated confusion matrix shows two missing values. For this example our Neural Network could not identify the emotion “disgust” which meant that we received values of “?” which is characterised as undefinable. This then effects the whole column meaning a value for precision and F Measure cannot be given.



## Task Four

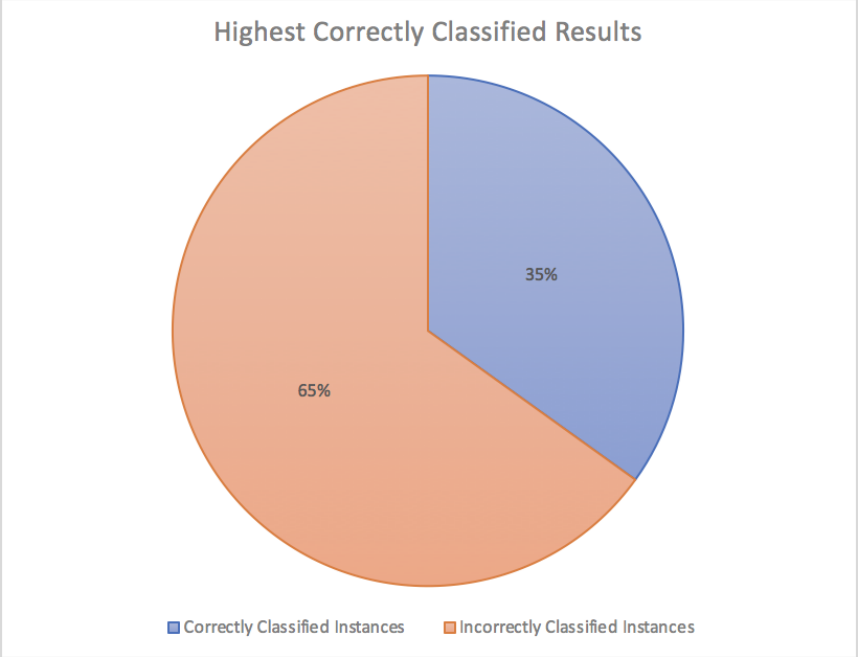
Below you can see a visualisation of the MLP neural network in weka, you could add new nodes by left clicking on the background, you could break connections by selecting a node then right clicking on the connected node and make them by left clicking. You could select multiple nodes at once and and and remove connections. This allowed us to manually build networks when required.



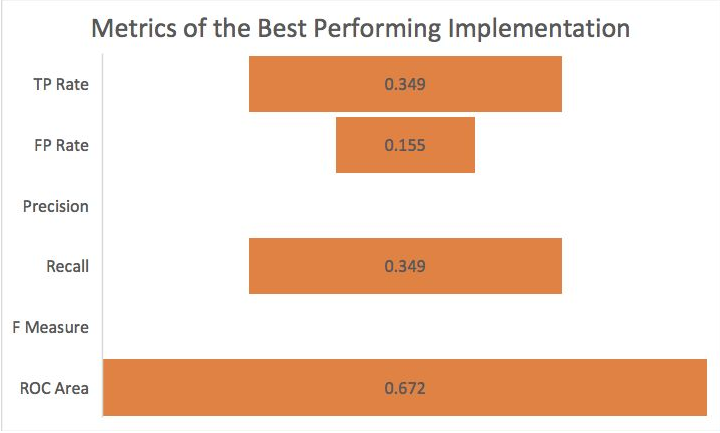

**Multilayer Perceptron Weka**

## Task Five

Task Five instructs that we instead of using Cross-Validation as done in Task 3, we are to use a Hold-Out method where we supply test and training data sets to the classifier.



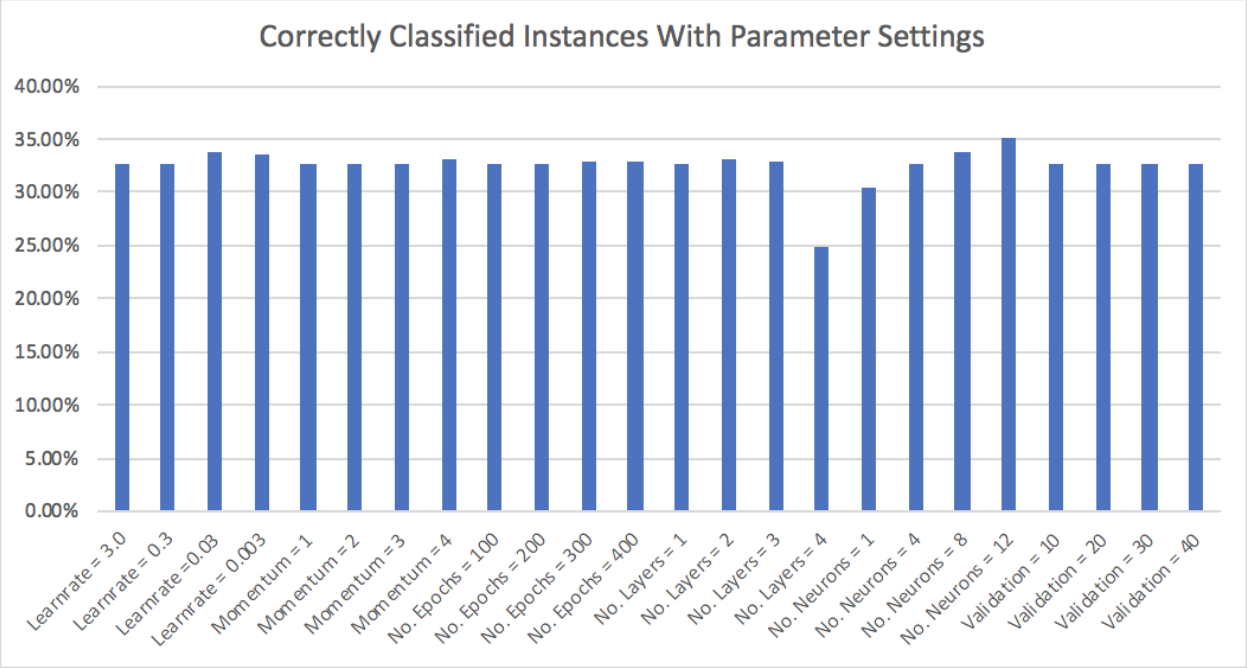

When collecting our results we found that most of our implementations were in the 30 - 35% range which was very similar to the results obtained in task three. The Pie Chart above shows a abstract visualization of our best performing implementation from task five.



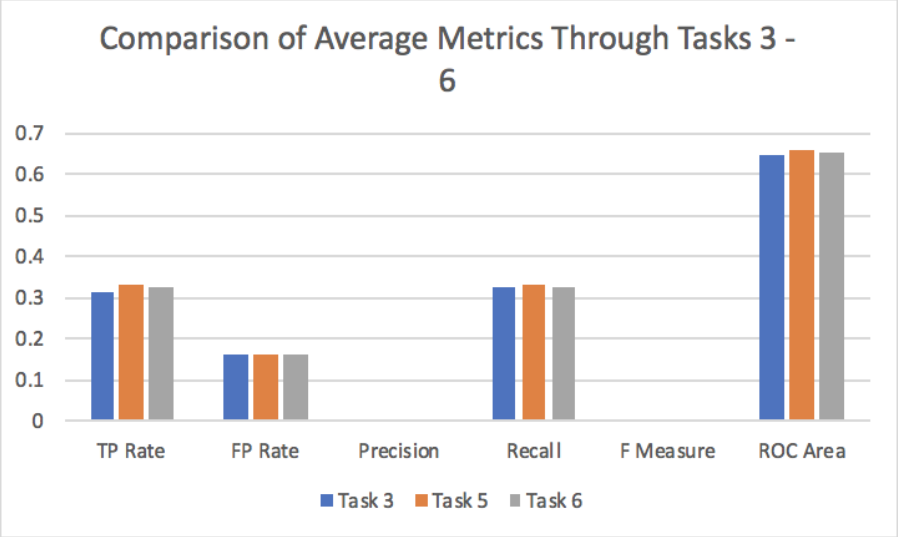

From viewing its confusion matrix we can observe from the data collected from running Neural Networks to this stage of implementation and the testing carried out in Part One that the Neural Network performs between the J48 algorithm and Random Forest. This we propose is due to over fitting in both the Neural Network and J48 classification.

**TASK 6.**

For this task we were asked to create our own training and testing dataset by moving 9000 instances from the original training set into the testing set, making the split around 70% training and 30% testing.



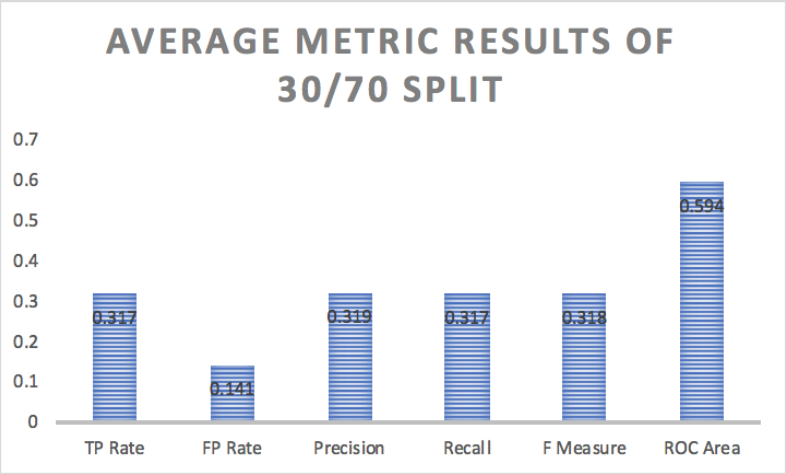

Above you can see the classification accuracy for the various tests we ran on the MLP for the new training and testing datasets. Note the large dip for 4 hidden layers. This is due to overfitting in the MLP.




We can see here that by observing the different average results of metrics produced by the MLP implementations that by using a Hold Out method i.e. using Testing and Training Data we can observe better performance within our model.

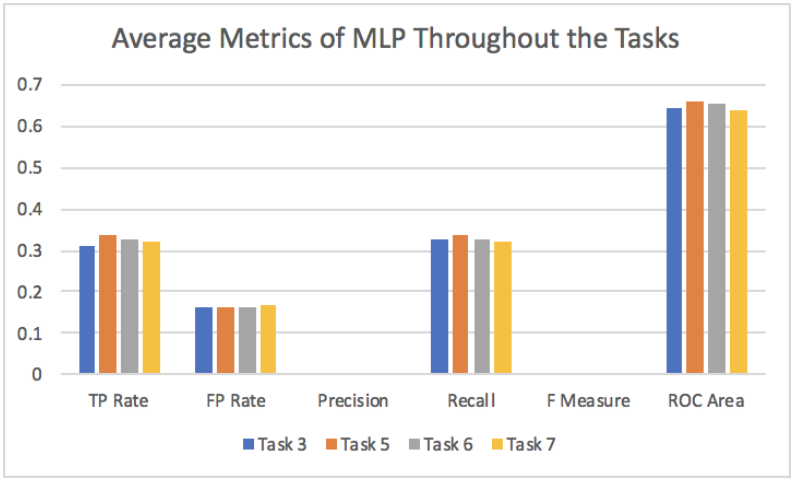
**TASK 7.**

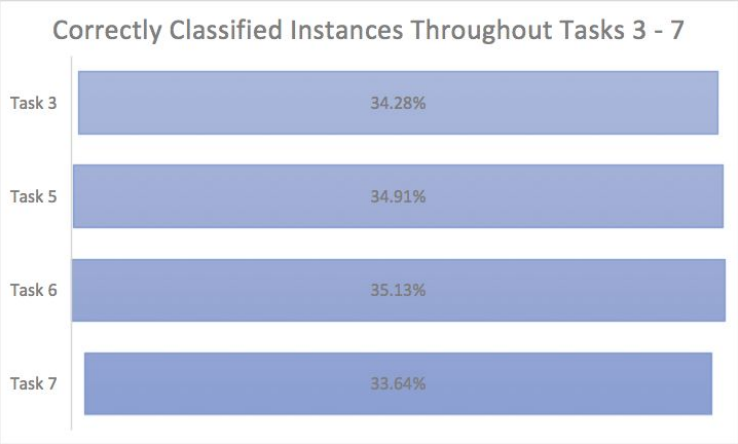
In this task we were asked to once again create a new training and testing dataset and split our training data with a new ratio. This time with 30% for training and 70% for testing.




From the left graph can see the average metrics from the confusion matrix for this new training and testing ratio.
We can see compared to the results gathered in Tasks 5 and 6 that the overall average of these measurements has decreased apart from FP rate which has increased as expected.

The right graph shows the average metrics tracked throughout the various tasks. What we can observe is that Hold-out and a 70/30 split on the data (which is wekas default for hold out) correlates in a higher average across the metrics.




This can also be observed in the number of correctly identified instances across the tests and we can conclude that when the data set is split too much then the model becomes over fitted and doesn't perform near its optimal.

**TASK 8.**

Overfitting occurs when the classifier is too specific to training data. Often the cause of this is noisy data within the training set. As a result, the network size, weight values and connections are not optimal for new test data. This causes poorer performance in the context of correctly classified instances. A real-world implication with training networks regards the amount of data available. Since large networks generally require a lot of data to train, there may not be enough data to train networks with different architectures for the purpose of model combination.

Due to the speed of using large, complex networks, it is difficult at test time to combine predictions of different networks to reduce overfitting. One technique used to counter this is Dropout[3]. This technique randomly drops units from the network during training, along with their connections. It is like sampling a subset of the network, which consists of the units which were not dropped. At test time, an un-thinned network with smaller weights is used to approximate the effect of averaging the predictions of the thinned networks created with the training data.

The dropout approach is still subject to the same tuning issues found in neural networks. Effective heuristics for applying dropout include: having a larger network size to compensate for the dropped units; using a high learning rate and/or high momentum to speed up learning; max-norm regularisation to constrain network weights; and adjusting the dropout rate - the probability of retaining a unit. Combining these factors should provide a higher number of correctly classified instances by reducing overfitting in training the network. This approach has a number of recorded implementations with improved performance on varying datasets [3] - ranging from vision and speech recognition tasks to document classification - and these help to demonstrate that the dropout technique may be employed in a range of domains.

## Conclusion of Linear Classifications and Neural Networks The Overfitting Problem

After completing the tasks and analyzing the results it becomes evident to see that all classifications models that can be considered to have complexity are liable to the issue of overfitting. This can happen for many reasons including the wrong parameters being set for the model, the data set, the data set for the classifier etc.

During the project we have been working with deterministic models as opposed to stochastic models with the main difference that a deterministic model is fully dependant on the parameters set by the researcher where as stochastic models the parameters are set but there is no guarantee due to the inherited randomness associated with their design implementation. This then means that multiple runs of our models with the same parameters will yield the same result.

Because of this reason of working with deterministic models we can establish that we can remove a vast amount of overfitting by changing the parameters of the various models. This has been seen through our results and the visualised graphs we have produced throughout this assignment.

### *Overall Conclusion of Test Results*

**1. "Variation in performance with the size of the training and testing sets"**

For cross validation using the whole training set, the average accuracy of the J48 Algorithm was 32.41%. The average accuracy of the Random Forest Algorithm was 45.14%. The average accuracy of the MLP was 32.49%. After switching to hold out validation, keeping 70% of the data for training and 30% for testing the accuracy of the J48 algorithm dropped slightly to 31.66%. With hold out validation at the same ratio the Random Forest algorithm dropped to 44.06% and the MLP increased slightly to 32.58%. This shows that with the larger training dataset the MLP has overfitted. The reason for the drop in accuracy in the Tree algorithms is that cross validation is training on 9 folds of the training data, approximately 90% of it, and testing on the remaining 10%. The tree algorithms must not have overfitted on 90% of the training data.

We then moved onto hold out validation with a 30:70 ratio for training and testing. The accuracy of the J48 algorithm at this ratio was 27.57%. This clearly shows that the classifier is under fitted at this training ratio. At the 30:70 ratio the Random Forest Classifiers accuracy was 39.55%. Again showing a drop in accuracy and underfitting of the classifier. At this ratio the accuracy of the MLP was 31.94% showing that at this point the classifier is under fitted compared to the 70:30 ratio.

**2. "Variation in performance with the change in the learning paradigm"**

The average accuracy of the tree classification algorithms trained and tested on all six emotions was 37.30%. The average accuracy of the MLP was 32.61. Therefore it can be seen that the Tree classification algorithms outperform the MLP for multiple classes of this data.

The average accuracy of the tree classification algorithms classifying on only one emotion was 78.8%. The accuracy of the MLP was 77.31%. Again the Tree classification algorithms outperform the Neural Network.

We suspect that this might be the result of the design of the WEKA MLP and our selection of attributes and configuration of the network hyperparameters. This highlights the importance of a systematic search to determining MLP hyperparameters before general use in classification tasks.

**3. "Variation in performance with varying learning parameters in Decision Trees"**

The first J48 setting that we investigated was binary splitting. This specifies whether to use binary splits or not on nominal data. If true the tree is "grown" by considering a split on one nominal value versus all others. This results in only two branches from any node. We found no difference in accuracy with splitting on and off.

The next J48 setting that was investigated was Pruning Confidence Value. This determines how certain the algorithm has to before pruning certain branches of the tree. The higher the confidence value the more certain the algorithm has to be that the data you are learning from is a good representation of all possible instances. We found that accuracy was highest with a confidence value of 10, it then decreased from 20 to 50. But the accuracy did not change from 20 to 50. This suggests that the confidence value was between 10 and 20 for the majority of the data and the tree did not get pruned above the 10 percent setting.

The next J48 setting we investigated was turning pruning on and off, we found no difference in accuracy with pruning on and off. This is because we used the weka default setting of 25% confidence and as we discovered above, we think the pruning confidence rate was between 10% and 20%. Therefore pruning was effectively always off in our tests.

The final J48 setting we investigated was the minimum number of objects. This is the minimum number of observations that are allowed at each leaf of the tree. We found the accuracy was best at 1 instance per leaf, at 33.14%, then decreased to 31.87% at 4 instances per leaf.

The first random forest attribute that we investigated was bag size. This is the percentage of the training data that is randomly sampled to build each tree within the forest. We found that the accuracy increased as more of the training data was randomly sampled. We think this is due to the larger percentage allowing a more representative sampling of the data.

The next random forest attribute that we investigated was maximum tree depth. We discovered that a shallow tree depth of 10 produced a low accuracy, as did a deep tree depth at 50. The best accuracy was found at depth 30. We think this because at this depth the tree had grown to be specific enough to classify well, but had not yet overfitted and become too specific as was the case at depth 50.

The next random forest attribute that we investigated was number of features, this is the number of attributes that are randomly chosen from the data to build the tree. We varied this from 10 to 60. We

found that the best accuracy was with around 20 features. Interesting this correlated with the best number of attributes we found ourselves using the CFSSubsetEval attribute selection method. We think this is because those are the 20 best attributes to classify the data, and more attributes than that are just introducing noise and making it harder to classify.

The next random forest attribute that we investigated was number of forest iterations, this is the number of random forests that are grown to find the best accuracy. As you would expect the accuracy consistently increased from 42.77% to 45.99% when varied from 25 to 100 iterations.

**4. "Variation in performance with varying learning parameters in Neural Networks"**

The first neural network hyperparameter that we investigated was Learning Rate. We altered this by an order of magnitude 4 times from 3.0 to 0.003. The test results demonstrated that a higher learning rate resulted in a lower accuracy. We suspect this is due to the larger steps taken by the gradient descent algorithm at higher learning rates. Larger steps means that it might miss a local cost minima.

The second hyperparameter that we investigated was momentum. We found that a lower momentum setting resulted in a higher accuracy. We think this is due to the gradient descent algorithm taking smaller steps as it approaches a local minimum and therefore falling into the minimum rather than stepping over it.

The third hyperparameter that we investigated was Number of Epochs. We varied this from 100 to 400. An interesting effect happened here. The accuracy increased from 100 to 200 epochs, but then began to fall again from 200 to 400. We suspect that this is due to the network beginning to overfit.

The fourth hyperparameter that we investigated was number of hidden network layers. We found that the accuracy increased up until 2 hidden layers, then began to decrease, with a 10% loss in accuracy with 4 hidden layers. We suspect this is dude to the network overfitting on specific features.

The fifth hyperparameter that was investigated was number of neurons. We discovered that the accuracy consistently increased with an increase in neurons, however there was a point of diminishing returns, with only 1 neuron, the accuracy was 30.43%. With 8 jumped to 33.40% but with 12 it only increased to 33.64%, even with 20 it only increased to 34.82%. We suspect this is because there are only a certain number of features to learn within the small images, so you reach a point where more neurons will not significantly improve accuracy.

The final hyperparameter that we investigated was validation threshold. We varied this from 10% to 40% but did not see any effect on our network accuracy or confusion matrix attributes.

**5. "Variation in performance according to different metrics (TP Rate, FP Rate, Precision, Recall, FMeasure, ROC Area)"**

We found that random forest had a better average True Positive Rate than J48 at 0.451 compared to 0.324. The tree algorithms had a better true positive rate overall compared to the MLP whos average TP rate was 0.311.

We found that random forest again had a better False positive rate at 0.137 compared to J48's 0.140. Both tree algorithms again outperformed the MLP's FP rate of 0.162.

We found that random forest had the best precision at 0.463 compared to the J48 algorithm at 0.325. The MLP precision could not be calculated as it could not classify one or more of the classes and therefore will not give an average precision value.

We found that random forest again had the best recall of the Tree Algorithms at 0.451 vs J48's 0.324. The MLP's recall did slightly better than J48 at 0.325.

We found that random forest again had the best FMeasure at 0.428 vs J48's 0.325. The MLP's FMeasure could not be calculated due to one or more classes not being classified.

We found that random forest once again had the best ROC area at 0.758 vs J48's 0.601. The MLP did a little better than J48 at 0.643.

Given the results of our experiments, we have found that the best algorithm for this data appears to be the Random Forest classifier. We think this is because they limit the effect of overfitting by aggregating many decision trees built on a smaller random sample of the overall data.

***Highlighting of Key Programming Implementation Aspects***

For the duration of this project we have created a script that can utilise many minor subscripts in the Python programming language. This "wrapper" was used through the project and with the addition of adding parallel capabilities to the wrapper we were able to run multiple tests at once and then receiving the output as a text file without having to use the Weka GUI.

By developing these pieces of software we allowed ourselves to be faster and more efficient when it came to completing the tasks given in the specifications through courseworks 1, 2 and 3. The main features of our software is as follows:

1) Threaded Parallelism

In order to efficiently run multiple tests at once we decided to multi-thread our test functions. We built own thread class to accept and instantiate the functions to run our algorithm tests. This is shown below:

```
class myThread (threading.Thread):
  def __init__(self, threadID, name, function, args=None):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.function = function
    self.args = args
  def run(self):
    print ("Starting " + self.name)
    self.function(self.args)
```

This meant that we could create multiple threads to run several tests with different parameters simultaneously such as varying the maximum number of parents as shown below.

```
#bag_size_percent
 thread1 = myThread(1, "run_rf", rf_function, ["-P", "25", "-I", "100", "-num-slots", "4", "-K", "0", "-M", "1.0", "-V", "0.001", "-S", "1"])
 thread2 = myThread(2, "run_rf", rf_function, ["-P", "50", "-I", "100", "-num-slots", "4", "-K", "0", "-M", "1.0", "-V", "0.001", "-S", "1"])
 thread3 = myThread(3, "run_rf", rf_function, ["-P", "75", "-I", "100", "-num-slots", "4", "-K", "0", "-M", "1.0", "-V", "0.001", "-S", "1"])
thread4 = myThread(4, "run_rf", rf_function, ["-P", "100", "-I", "100", "-num-slots", "4", "-K", "0", "-M", "1.0", "-V", "0.001", "-S", "1"])
```

2) Our Wrapper

We wrote two classes to automate our classification tasks, "classify.py" which is a set of wrapper classes to load the datasets and call certain weka classes based on parameters that we feed in from "run_all.py". Run_all.py is where we setup our tests by specifying the classifier name and a list of options to provide to weka to run on it.

3) Scripts to Select, Reduce and Randomise Data Sets

We wrote several scripts called "part3_attribute_reduction.py" to reduce the dimensionality of the data. We also wrote scripts called, "TestTrain.py" to select a certain percentage of the data for training and

testing, and ones called “shuffle.py” and “makeR.py” to randomise the data to provide a more representative sample in the training and test data.

4) Conversion from CSV to ARFF format Script.

We wrote a script called “csv_arff.py” to convert the fer2017 csv files to the weka .arff format by using the python csv library to parse the original file, writing out attribute data in the weka header format, then recursively iterating over the data to normalise and convert and output it to a new file in the arff format.

## ***Bibliography***

[1] - OPENI (2012). J48 Decision Tree. [image] Available at: https://openi.nlm.nih.gov/detailedresult.php?img=PMC3316502_pone.0033812.g002&req=4 [Accessed 21 Nov. 2018].

[2] - Rokach, L. and Maimon, O. (2005). Decision Trees. 1st ed. [ebook] Researchgate, pp.167 - 169. Available at: http://file:///Users/jordanswalker/Downloads/Decision_Trees.pdf [Accessed 23 Nov. 2018].

[3] - Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15 (2014): 1929-1958. Available at: https://www.semanticscholar.org/paper/Dropout%3A-a-simple-way-to-prevent-neural-networks-Srivastava-Hinton/6c8b30f63f265c32e26d999aa1fef5286b8308ad [Accessed 27 Nov. 2018].

[4] - Mines.humanoriented.com. (2018). Decision Tree Classifier. [online] Available at: http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/lguo/decisionTree.html [Accessed 27 Nov. 2018].

## ***Appendix***

### *A) Tables of Reference*

***Weka output measurements and their description.***

| Measurement | Description |
|---|---|
| TP Rate | This measurement is the rating of true positives which corresponds to the instances correctly classified to the given class. |
| FP Rate | This is the rating of the false positives which corresponds to the instances which have falsely been classified to a class. |
| Precision | This is the proportion of instances that are true of a class divided by the total instances of that class. |
| Recall | The proportion of the instances that have been classified to a given class and divided by the actual total in that class. |
| F Measure | This is a combined measurement which is the calculation:<br>2 * precision * recall / (precision + recall) |
| ROC (Receiver Operating Characteristics) Area | This is an area measurement and can be used to observe how the classifier is performing overall. |