Group 8 UG
F20DL Data Mining and Machine Learning
Course Work 2 Report

Bernard Hollands – H00271461

Perry Anderson – H00270025

Calum McAdam - H00271918

Ram Attra - H00288794

Github Link: https://github.com/bhollands/F20DL_CW2_G8

(note this repo was private until after the deadline)

Contribution Declaration

Bernard Hollands – Worked on part 2: Made step 4 and 5 data sets, ran MLP experiments, wrote subsequent report sections, reduced size of data to be able to store them on GitHub.

Perry Anderson – Worked on Decision Trees with Calum, specifically on analysing results of using J48 classifier on cross validation and different variations of training and test data sets.

Calum McAdam – worked on Part 1: Decision Trees with Perry, focused on using random Tree algorithm to create the decision trees, also worked on the report sections focusing on random trees.

Ram Attra – Worked on part 2: producing cross validation and train and test set CNN's for evaluation, reporting the findings of my experiments in the suitable report sections.

1. Variation in performance with size of the train and testing sets

Decision Trees

J48

After conducting various experiments with the J48 classifier on various training and testing data sets, the best performance was found using 10-fold cross validation on the original data set (reduced and randomised). This yielded a classifier accuracy of 36.85%. When using the classifier on the same training data set but with the provided test set, the best model recorded an accuracy of a slightly lower 33.82%. When moving 4000 instances from the training set to the test set, the accuracy was very poor at 6.48%. This was to be expected as there is less data to train on and consequently more unknowns encountered in the test set. However, when 9000 instances were moved from the training set to the test set, the best model produced an unexpectedly higher accuracy of 19.33% – around 3 times the accuracy of what was recorded in the previous experiment.

From these results, it can be said that there was a confliction in terms of over-fitting. It was found that after removing 4000 instances from the training set, the accuracy from testing the classifier with the test set was significantly low. Changing the different parameters also saw a further decrease, albeit minimal, for e.g. when choosing not to prune the tree. In contrast, the training set with less instances and thus more in the test set seemed to perform better in terms of overfitting. In fact, it was found that altering the classifier parameters had a bigger impact on the accuracy, where using pruning and decreasing the confidence factor saw increases of up to 4-5%.

Random Tree

After running the Random tree classifier with the various testing and training sets, the best performance in term of accuracy on the original data that was reduced was with training set option in weka , this correctly classified 100% of the data. when using the provided test set was 28.8%. when moving 4000 instances from the original set to create a new test set the accuracy was only 4%and when moving 9000 instances to create new test data, the accuracy increased to 14.5%

after these tests it, there is drastic decline with each test, especially when moving 4000 instances, which dropped to the lowest accuracy, this was also true with moving 9000 instance which was also very low however it was almost triple the accuracy of the 4000.on the other end the weka training set was uncharacteristically high when compared to the rest with both 10-fold cross validation and the given test data only reaching a third of its accuracy

Neural Network

Multi-Layer Perceptron

Overall, the best performance was found using 10-fold cross validation with an accuracy of 97.21% as well as the highest average overall. The original sized datasets provided the next highest levels of overall accuracy. After switching the 4000 (after randomising to ensure not only 1 or 2 types of sign were transferred). The test accuracy was overall lower, with less data to train the network and more unknowns in the test set, meaning it sees less examples reducing how familiar it becomes with variations. This is only amplified when 9000 instances were transferred as there is only a handful of training instances left. This is very clear from figure 1.0 as you can see the average accuracies with each set while increasing the learning rate. There is very clear banding.

In terms of overfitting, it was found that the step 5 dataset suffered the most as it had not seen as much training data and the overall test accuracy suffered the most. The issue only became worse when the network became more complex i.e adding more epochs and weights to the classifier. This was mitigated by reducing the number of layers and the size of those layers as well as reducing the number of epochs as recommended here [6] and adding a 0.5 dropout to each as you can see if figure 1.1 it was quite effective. The overall accuracy went down however the overfitting is gone.

Compared to the linear classifier (Logistic Regression) the results (table [1]) where the average result is approximately 0.83 the MLP was not worth the extra effort for this dataset as the mlp took a lot of experimentation and tweaking to get above this level of accuracy especially with the step 5 data.

Convolutional

The best performance of 97% accuracy, was observed by the Convolutional Neural Network with the training set which had 4000 instances moved. The original testing and training sets achieved 96% accuracy, while when 9000 instances were moved to create new test data this dropped to 92%. The step 5 experiments found lower accuracies as less data to train the network meant more untrained test data, thus variations of data were not found increased leading to more inexperience of the model. Regarding overfitting, step 5 proved that the additional testing data without additional training meant worse accuracies. However, by reducing the complexity of the network accuracies improved (reduced overfitting). This was managed in two ways lowering the quantity of weights and lowering the value of said weights. Additionally, a more costly method to reduce overfitting was the introduction of more pools of images, essentially increasing the use of feature selection and disregarding false patterns (figure 6). Batch normalisation vs dropout, while the two were experimented neither significantly affected the model's overfitting, going against the hypothesis that batch normalisations regularisation is more effective in CNN's than dropout.

2. Variation in performance with change in the learning paradigm (Decision Trees versus Neural Nets)

Decision Trees

In general, when using J48, changing the learning parameters had little impact on the accuracy scores of the classifier on the different data sets. In many cases, changing the minimum number of instances per leaf did not affect the accuracy at all. Parameters that did create a change when altered provided an insignificant increase/decrease in accuracy (pruning, non-pruning, etc.). Binary splits, for the most part, were not used on larger sets due to the time it took to process. In cases where the set was smaller, this usually resulted in a decrease in accuracy.

Neural Network and Linear Classifier

Unlike J48 the multi-layer perceptron neural nets accuracy was highly dependent on the learning parameters. This can be seen in all the experiments run that are in the excel file. Later in the document there is a further exploration into the specific impacts of each learnings parameter and the effect on accuracy. Much more like the decision trees Linear Regression changing the number of iterations had little to no effect on the outcome.

Overall, CNN's perform similarly across varying learning paradigms with any differences being insignificant, e.g., filter sizes above 3 affecting performance with deviations between each other by at most 0.5% figure 7. The only exception being the number iterations, which was observed that as it increased so did the accuracy, but this too meant an increase in computation time meaning an equilibrium between accuracy and number iterations had to be found before the CNN took too long for little to no significant improvements in accuracy.

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

J48

When testing the original data set with cross validation, the best model used pruning, with a confidence factor of 0.5, and a minimum of 2 instances per leaf – yielding a classifier accuracy of 36.85%. Decreasing the minimum number of instances produced a slightly worse accuracy, as did choosing to use reduced-error pruning or non-pruning. When testing with the test set, pruning was again the best option but this time with a lower confidence factor of 0.25. Less minimum instances and reduced-error pruning recorded lower accuracies.

When moving 4000 instances of training set instances to the test set, pruning was yet again used in the best performing model. However, on this occasion the use of error-reduced pruning generated the best result with

6.48% accuracy. Collapsing the tree and changing the minimum number of instances had no impact this model's accuracy, while using binary splits decreased it by 1%.

With 9000 training set instances moved to the test set, pruning was again preferred over non-pruning. Choosing a lower confidence factor, capped at 0.25, showed better results, with the best model producing an accuracy of 19.33%. When pruning, choosing error-reduced or changing the minimum number of instances did not change this accuracy. On the other hand, when not pruning, decreasing the minimum to 1 saw a small increase from 14.83 to 15.1%.

Random Tree

The various aspect of Random Tree that was changed to experiment for the best model was K value, to allow unclassified instances and the number of folds, the best model was the original one which had a k value of 0, didn't allow unclassified instances and the no. Of folds set to 0. While using tis on cross validation for the original data set the drop off was significant wit the original model being at 31% correctly classified and the new being at 7% this is the same for 9000 instances dropping from 14% to 7% and for the provided data set dropping from 28% to 0%

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

Multi-Layer Perceptron vs Logistic Regression:

There are many parameters that can be adjusted in a multi-layer perceptron (MLP) neural network, such as size of training and testing sets, the number of layers, learning rate, momentum, the size of the hidden layers and the number of hidden layers. There are many possible combinations of these parameters and some experimentation is needed to find the most optimal combination for a given set of data.

MLP Learning Rate - For each configuration, the learning rate has a very significant effect as show in on the output of the classifier as shown in [1] and [2]. If too big a learning rate is chosen then the results will be poor as you can see in figure 4.0 only adjusting the learning rate can either make the classifier very good or very poor.

MLP Momentum -To assess the momentum, all other parameters were held: learning rate is 0.01, layer size of 8 with 20 epochs, You can see in figure 4.1 that as the momentum increases the accuracy will rise till a threshold then it will drop off suddenly, making the classifier less accurate. This is shown in [3] as if there is too much momentum you will overshoot the local minima

MLP Number Of Layers - Altering the number of layers in the network drastically changes the length of time that it takes to give a result. Experimentation is usually required to find the optimal amount as explained in [4]. For this examination of this the number of hidden layers was increased from 1 to 10 with a size of 128 and 10 epochs. As you can see in figure 4.2 the results show that only increasing will not on its own increase the accuracy of the classifier.

MLP Hidden Layer Size- The hidden layer size is another parameter that dramatically increases the runtime for a network as there are more connections. For this experiment, the original train test split was used with 2 hidden layers and the size was increased from 8 to 4096. As you can see in figure 4.3 the results show no significant increase in accuracy however some overfitting issues do arise, and this is only amplified when the training set is reduced as in step 5.

MLP Number Of Epochs - By varying the number of epochs that the network runs this can have a huge effect on the length of time is takes to get a result as well as the result, so this is a key metric. Much like the number of layers it usually requires experimentation to find the right amount for the data to keep learning and minimise overfitting as discussed here [5]. While increasing the number of epochs the accuracy generally tends to go up, however this does make the classifier prone to overfitting. As you can see in figure 4.4 after 30 or 40 epochs the training accuracy always claims 0.99 however the overfitting does not reduce.

Convolutional:

In CNN's, the learning rate are vital to producing an effective model. Using train and test sets for evaluation it was found that a learning rate of 0.1 results in a good model performance, but this required a lot of Epochs, hence a rate of 0.01 was chosen as a trade-off for less epochs figure 8.1.

However, best result was discovered with the use of adaptive learning rates such as Adam. This effectively learnt the problem within 25 training Epochs at most, with it spending remaining training time making minor weight updates which would lead to the overfitting of the model.

With the use of previous learning parameters, the Epoch of the CNN could be kept much lower to prevent any overfitting. During testing as the Epoch increased so did accuracy figures 8.3 & 8.4 show that a 100% increase in the number of Epochs increased the accuracy by 3%. As discussed earlier this was found to be most effective at around 10 Epochs before computational time was wasted.

5. Variation in performance according to different metrics (TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area)

TP Rate/Recall

J48: In the first experiment with cross validation, class 1 had the highest TP rate/Recall at 0.500 – exactly half of all class 1 instances were correctly classified as being class 1. When moving 4000 instances to the test data set, class 3 was the only class to produce a non-zero TP rate. This can account for the poor accuracy of the model in general. When moving 9000 instances, class 1 was this time the only class to produce a non-zero TP rate, in fact, its TP rate was 1.0 meaning it did well to correctly predict the instances of that class, albeit at the cost of the other classes.

Random Tree: while testing with cross validation the true positive rate was at an average of 0.311, with the lowest being class 0 at 0.019 and the highest being class 1 at 0.475, the given test data's TP rate was at an average of 0.288, with the lowest being at class 0 with 0.00 and the highest being at class 1 with 0.422.the 9000 instance data sets TP rate average was at 0.175, with the lowest being class 2-8 at 0.00 and the highest was class 1 with 0.751. the 4000 instance sets average was at 0.045 with the lowest being class 4-8 at 0.00 and the highest being class 4 at 0.445

CNN: Performance varied largely between cross validation and train and test sets, with cross validation never achieving an average recall of greater than 0.1, and test and train achieving 0.96 on average across all experiments except for step 5. Labels 5 and 6 faired the best with recalls of greater than 0.96. Cross validation singled out label 1 with a recall of 1.0 every time. This is because every fold introduces more and more hyperparameters turning the model into pattern recognition only for label 1, an extreme form of overfitting.

FP Rate/Inverse Recall

J48: In the first experiment, class 1 had the highest FP rate at 0.239, with classes 2-4 ranging between 0.141-0.215. This is seen in the visualisation of classifier errors (Figure 5.1) where there is a large concentration of instances predicted to be in classes 1-4 for the same labelled classes. When moving 4000 instances, class 1 had the highest FP rate at 0.357. Classes 4-9 all produced 0 for the FP rate, meaning that all instances were predicted for classes 0-3 (Figure 5.2). While this produced a better accuracy, it also affected the accuracy of the other classes. When moving 9000 instances, class 1 had the only non-zero FP rate at 1.0. This meant that all instances had been classified as class 1, and this can be seen in the visualisation (Figure 5.3).

Random Tree: using cross validation on the original set gave an average FP rate of 0.176 with the highest being class 1 with 0. 309.The provided test data gave an average of 0.187 with the highest being was class 1 with

0.299.the test set with 9000 instances gave an average of 0.153, with the highest being class 0 with 0.208. and the 4000-instance set gave an average of 0.032 with the highest being class 3 with 0.342

CNN: Like before cross validation singled out label 1, with a FP of 0 and most train and test sets had FP less than 10%. Labels 0 and 7 had the worst FP rates for every test by at least 20%, with the worst results coming from step 5 with 0.85 and 0.77 respectively.

Precision

J48: In experiment 1, precision was highest for classes 1-4 as expected, given they also recorded a similar range of values for the TP and FP rate. When moving 4000 instances, class 3 was obviously the only class to record a value for precision (0.173), as it was also the only class to record non-zero values for the TP rate. When moving 9000 instances, class 1 was the only class to have a non-zero value for both TP and FP rates, so as a result it had the only value for precision at 0.193.

Random Tree: using cross validation with the original data set gave an average precision of 0.296, with the highest being from class 4 with a precision of 0.365, when using the provided test data the average precision was 0.271 with the highest being from class 4 with a precision of 0.396. when removing 4000 instances to create a new test set, the average precision was 0.134 with the highest being class 3 at 0.134 and when moving 9000 instances the average was 0.183 with the highest being class 1 at 0183

CNN: Cross validation had the lowest precision on average with 0.02, which label 1 got the highest accuracy of 0.23. For test and train sets the precision for every experiment except certain Epochs for step 5 stayed above 0.91. With the lowest precision 0.88 being recorded for Epoch = 5 in step 5. Label 5 had the greatest success in hitting 1.0 more than any other label.

F Measure

J48: In the first experiment, classes 1-4 recorded similar values for F-Measure in the range 0.375-0.434. As F-Measure factors in the values for both precision and recall, this meant that classes 3 and 1 in the 4000 and 9000 instance experiments respectively had the only non-zero values for F-measure.

Random Tree: The F-measure with Cross validation was averaged at 0.292, with the highest at class 1 with 0. 387.The given test data averaged at 0.273, with the highest at class 1 wit 0.354. with 4000 instances averaged at0.206 with the highest being 0.206 at class 3 and 9000 instances averaged at 0.295 with the highest being at 0.295

CNN: F-Measure for cross validation across experiments averaged 0.23, with the highest score going to label 1 with 0.37. The test data had far more success with F-measure averaging at the lowest 0.83 for the step 5 experiment, and the highest 0.97 being reached numerous times.

ROC Area

J48: With many classes recording zero for TP and FP rates in the 4000 and 9000 instances experiments, this resulted in these classes achieving 0.500 for the ROC area. Since the TP and FP rates are of the same value, they share the same line that splits the area in half. In the 9000 instances experiment, class 1 also recorded 0.500 for the ROC area – this is because it recorded 1.0 for both the TP and FP rates.

Random Tree: cross validation ROC Area averaged at 0.624 with the highest being 0.690. the given data set averaged at 0.585 with the highest at 0.686, 4000 instances averaged at 0.506 with the highest at 0.564, 9000 instances averaged at 0.487 with the highest at 0.5

CNN overfitting highlighted by cross validation & train and test

From results it can be observed that train and test sets have a lot more variance in errors, when cross validation has more biased errors with it singling out multiple times label 1. With these results cross validation is not useful in selecting a model, as it exacerbates feature selection into overfitting to only test data that fits label 1, a behaviour of CNN's as they tend to overfit rather than underfit.

Appendix and References

[1] https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[2] https://www.jeremyjordan.me/nn-learning-rate/

[3] https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

[4] https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[5]https://www.researchgate.net/post/How_to_determine_the_correct_number_of_epoch_during_neural_network_training

[6] https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

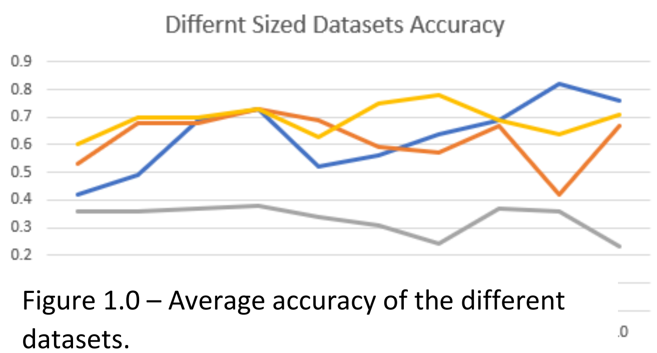


Figure 1.0 – Average accuracy of the different datasets.

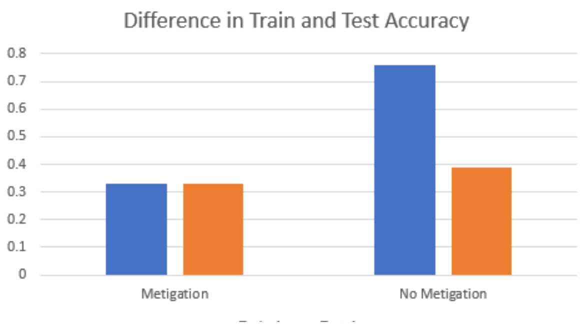


Figure 1.1 – Difference in overfitting with and without steps taken to reduce it

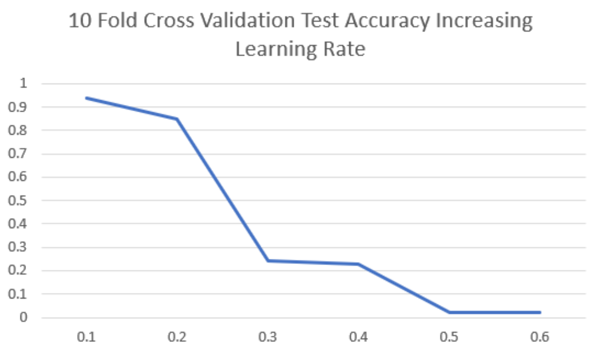


Figure 4.0 – Shows increasing learning rate can be detrimental to the accuracy of a network

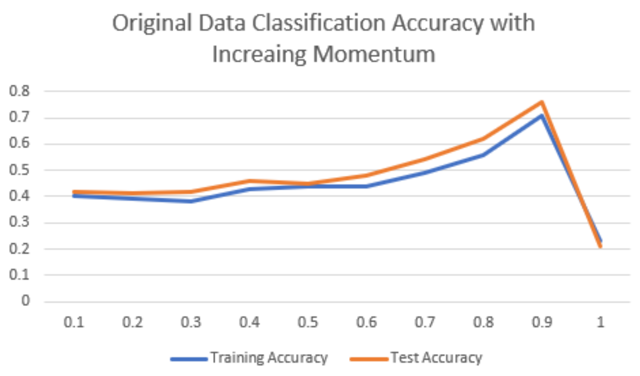


Figure 4.1 – Shows increasing momentum

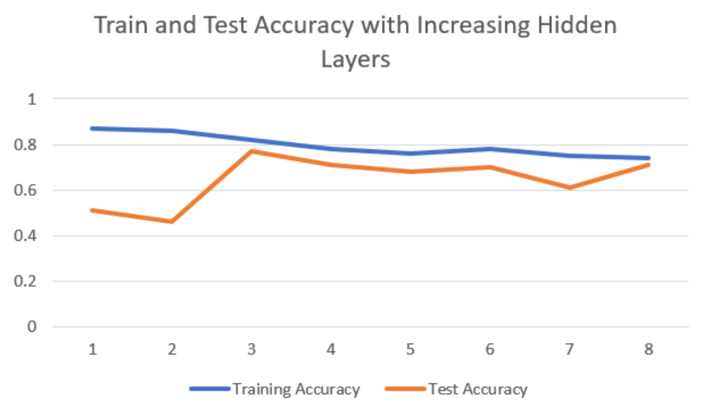


Figure 4.2 – Shows train and test accuracy increasing the number of hidden layers

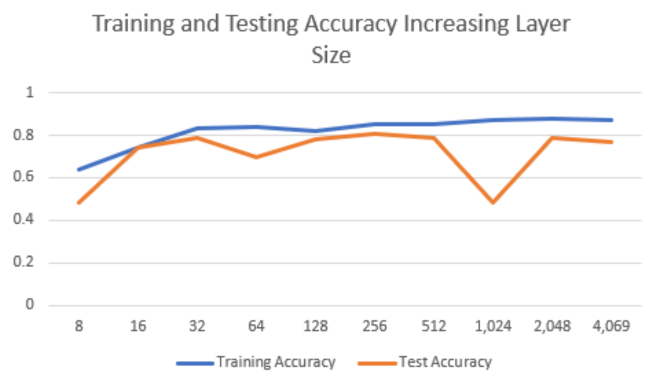


Figure 4.3 – Shows train and test accuracy increasing the size of the hidden layers

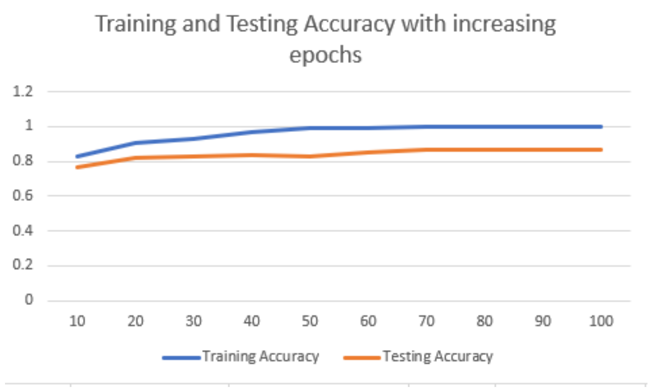
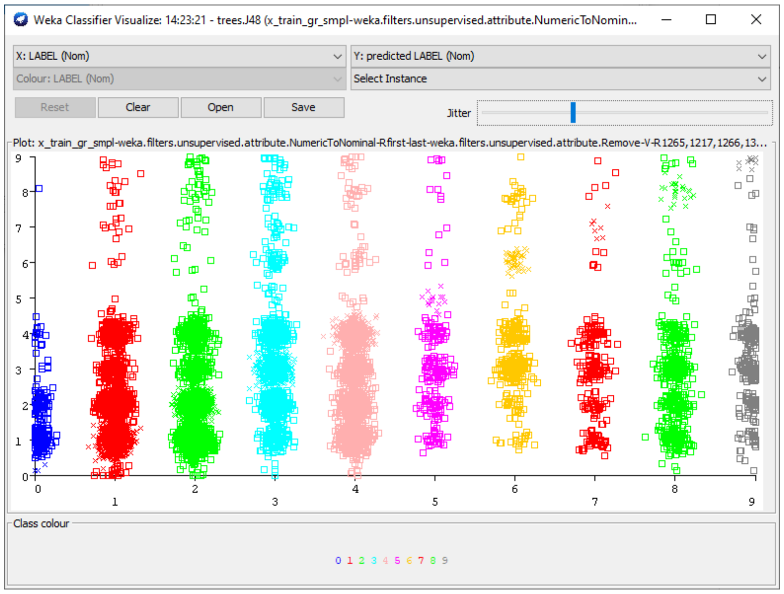


Figure 4.4 – Shows increasing number of epochs



Weka Classifier Visualize: 14:23:21 - trees.J48 (x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNomin...

X: LABEL (Nom)
Y: predicted LABEL (Nom)
Colour: LABEL (Nom)
Select Instance

Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-R1265,1217,1266,13...

Figure 5.1 – Visualisation of classifier error for J48 with the best model on the original dataset.

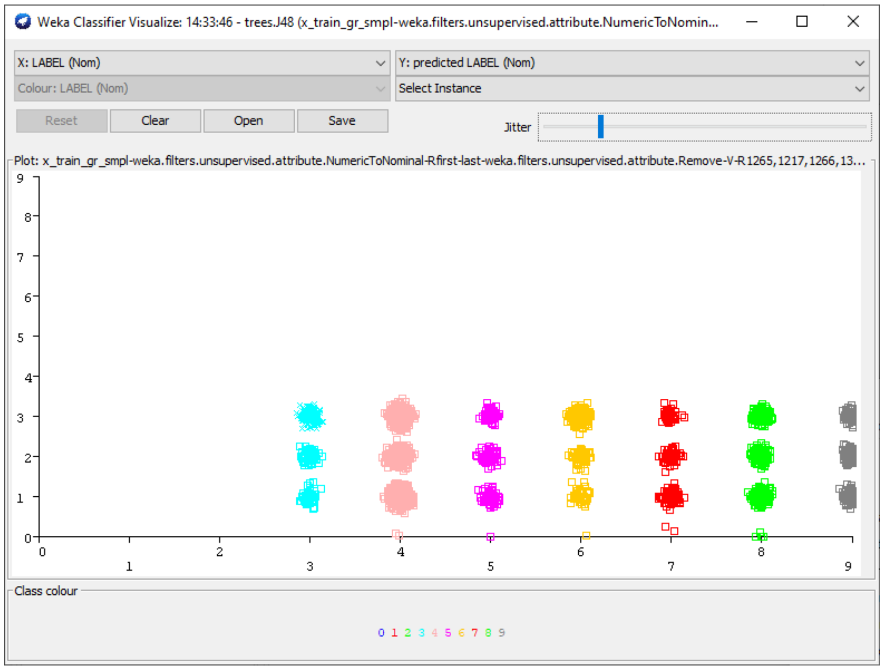

Weka Classifier Visualize: 14:33:46 - trees.J48 (x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNomin...
X: LABEL (Nom)
Y: predicted LABEL (Nom)
Colour: LABEL (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-R1265,1217,1266,13...
Class colour
0 1 2 3 4 5 6 7 8 9

Figure 5.2 – Visualisation of classifier error for J48 with the best model after moving 4000 instances from training set to testing set.

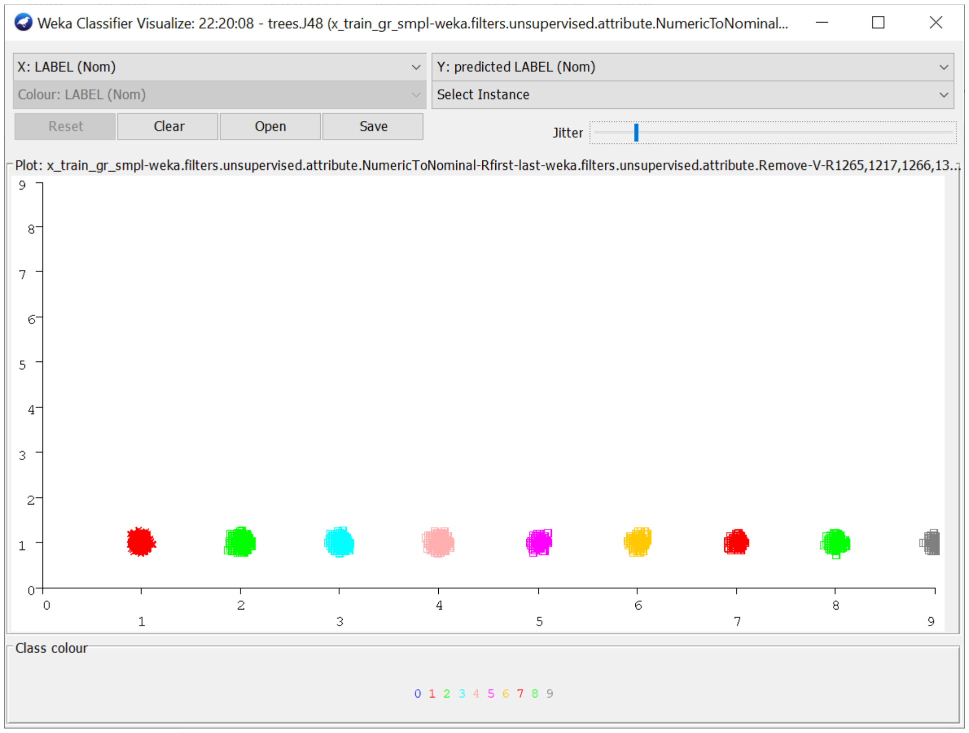
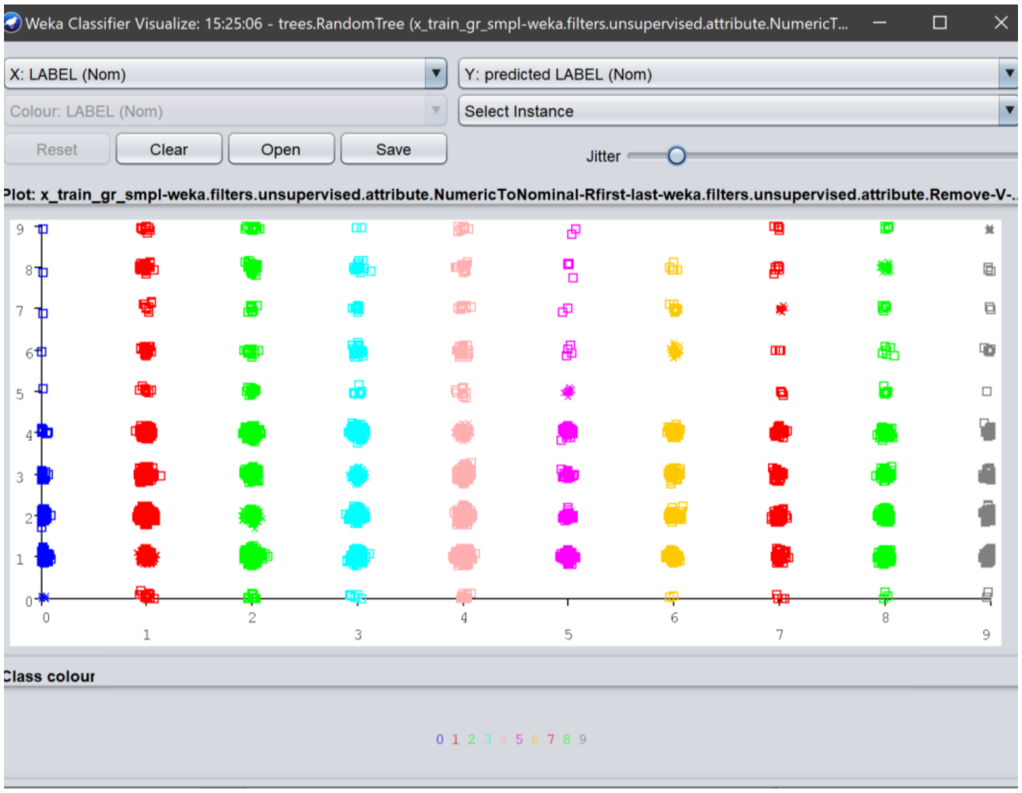
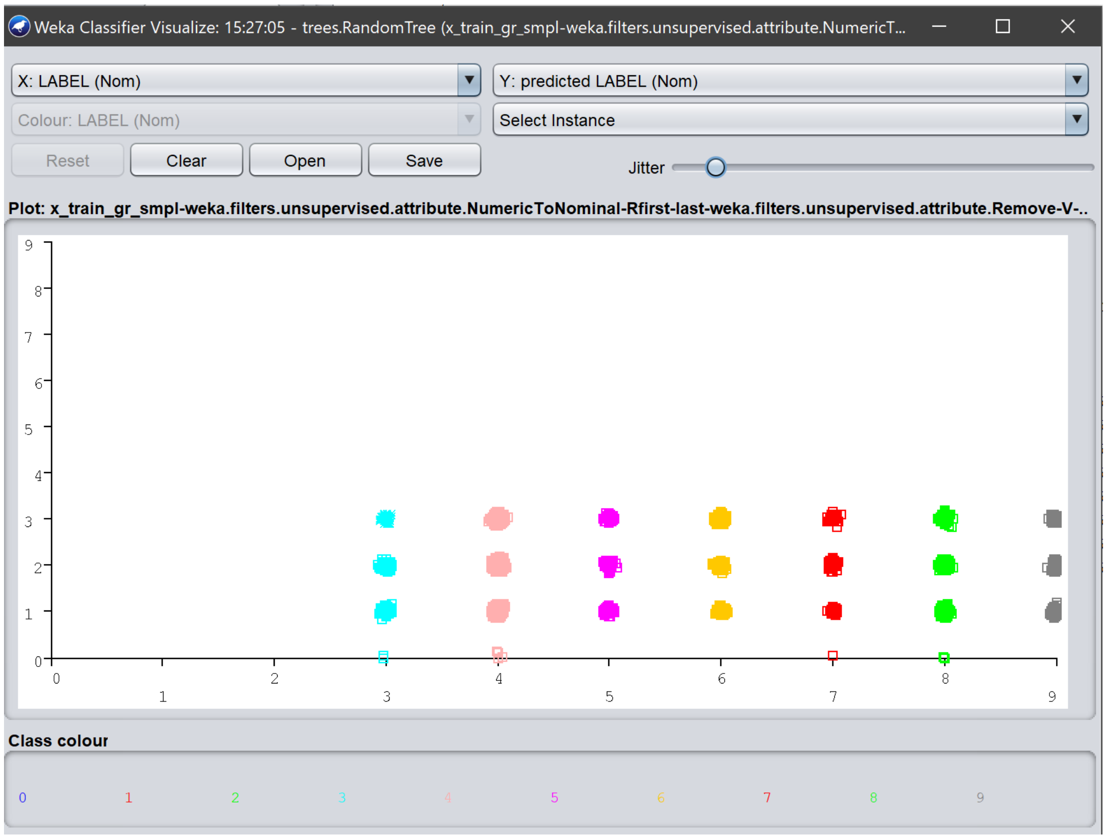

Weka Classifier Visualize: 22:20:08 - trees.J48 (x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal...
X: LABEL (Nom)
Y: predicted LABEL (Nom)
Colour: LABEL (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-R1265,1217,1266,13...
Class colour
0 1 2 3 4 5 6 7 8 9

Figure 5.3 – Visualisation of classifier error for J48 with the best model after moving 9000 instances from training set to testing set.



Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-..

Figure 5.4 – visualisation of classifier errors for random tree with cross validation



Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-..

Figure 5.5 - visualisation of classifier errors for Random Tree after moving 4000 instances from training set to new test set

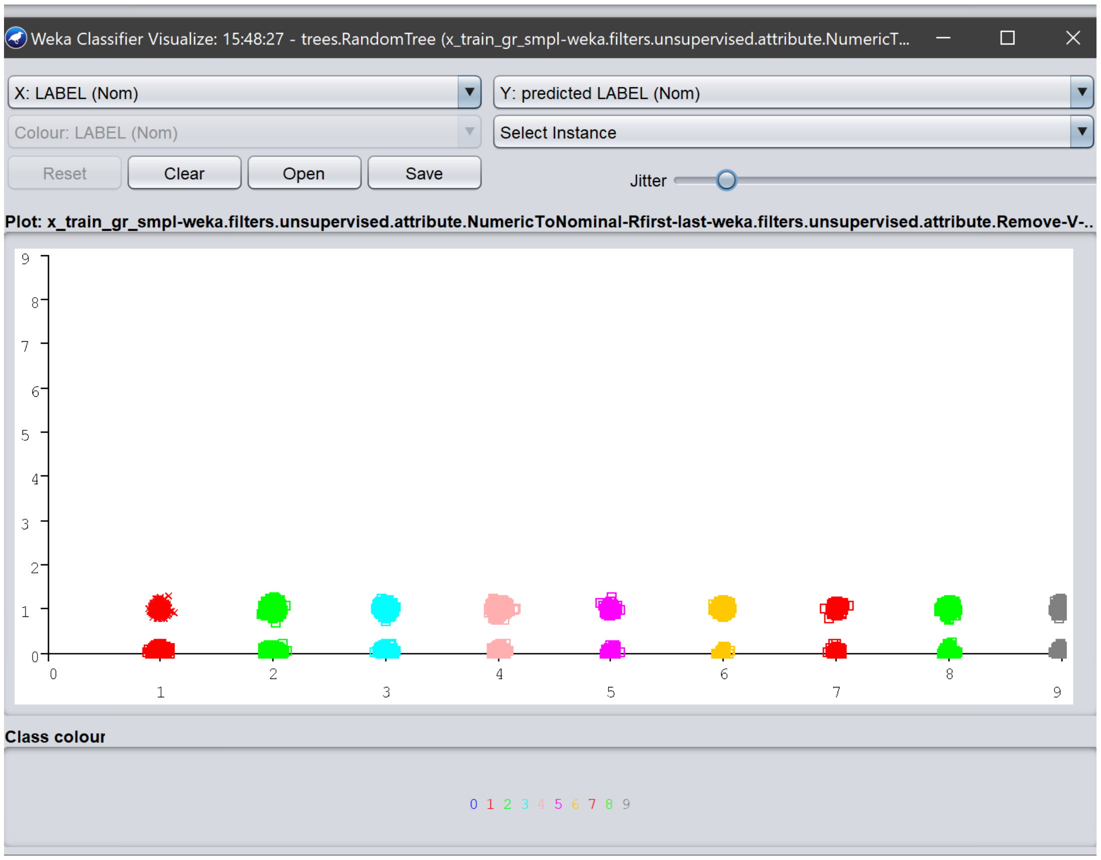
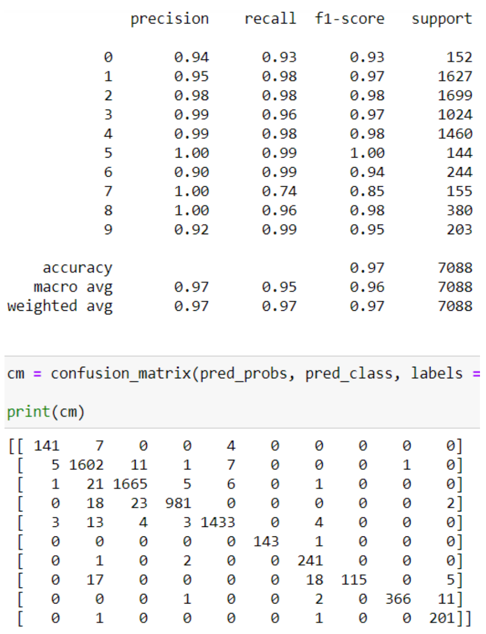

Weka Classifier Visualize: 15:48:27 - trees.RandomTree (x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericT...
X: LABEL (Nom)
Y: predicted LABEL (Nom)
Colour: LABEL (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: x_train_gr_smpl-weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last-weka.filters.unsupervised.attribute.Remove-V-..
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
Class colour
0 1 2 3 4 5 6 7 8 9

Figure 5.6 - visualisation of classifier errors for Random Tree after moving 9000 instances from training set to new test set

Data set	**Format**	**Classifier Accuracy**
Original Data	10 Fold Cross Validation	0.8450
Original Data	Train Test Split	0.8666
Step 4 Data (±4000)	Train Test Split	0.8901
Step 5 Data (±5000)	Train Test Split	0.7489

Table 1 – Logistic Regression accuracy scores for different data sets



```
              precision    recall  f1-score   support

           0       0.94      0.93      0.93       152
           1       0.95      0.98      0.97      1627
           2       0.98      0.98      0.98      1699
           3       0.99      0.96      0.97      1024
           4       0.99      0.98      0.98      1460
           5       1.00      0.99      1.00       144
           6       0.90      0.99      0.94       244
           7       1.00      0.74      0.85       155
           8       1.00      0.96      0.98       380
           9       0.92      0.99      0.95       203

    accuracy                           0.97      7088
   macro avg       0.97      0.95      0.96      7088
weighted avg       0.97      0.97      0.97      7088
```

```
cm = confusion_matrix(pred_probs, pred_class, labels =

print(cm)
```

```
[[ 141    7    0    0    4    0    0    0    0    0]
 [    5 1602   11    1    7    0    0    0    1    0]
 [    1   21 1665    5    6    0    1    0    0    0]
 [    0   18   23  981    0    0    0    0    0    2]
 [    3   13    4    3 1433    0    4    0    0    0]
 [    0    0    0    0    0  143    1    0    0    0]
 [    0    1    0    2    0    0  241    0    0    0]
 [    0   17    0    0    0    0   18  115    0    5]
 [    0    0    0    1    0    0    2    0  366   11]
 [    0    1    0    0    0    0    1    0    0  201]]
```

Figure 6 – Effects of pooling, and reduction of weights

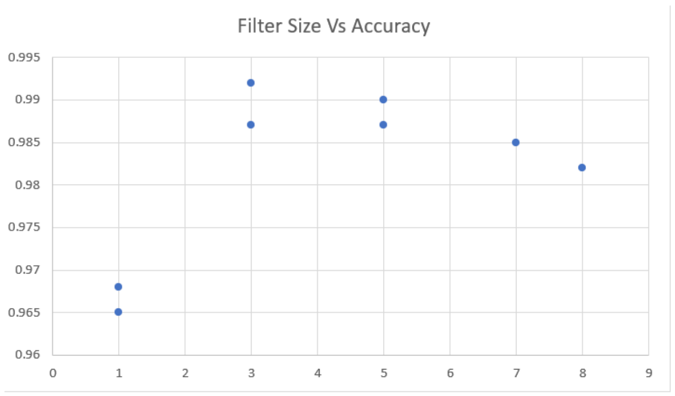


Figure 7 – Filter Size on accuracy

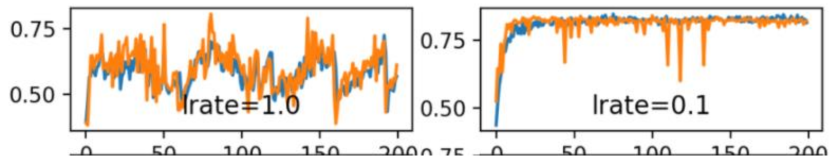


Figure 8.1- Learning rate on accuracy

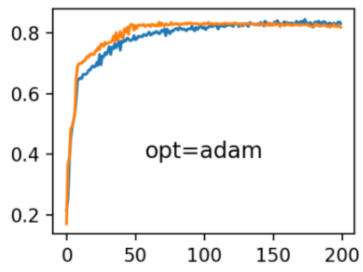
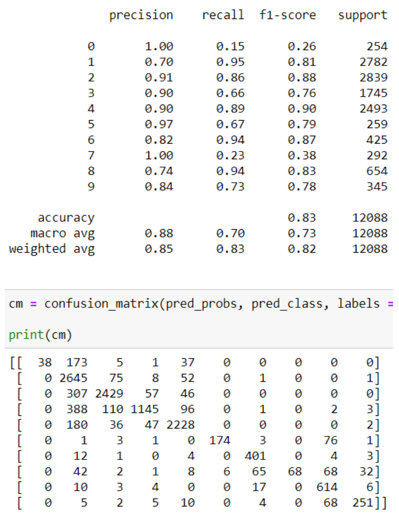


Figure 8.2 – Adam Optimiser

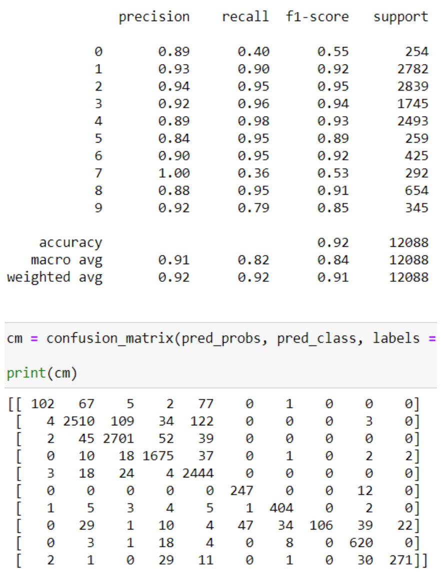


```
              precision    recall  f1-score   support

           0       1.00      0.15      0.26       254
           1       0.70      0.95      0.81      2782
           2       0.91      0.86      0.88      2839
           3       0.90      0.66      0.76      1745
           4       0.90      0.89      0.90      2493
           5       0.97      0.67      0.79       259
           6       0.82      0.94      0.87       425
           7       1.00      0.23      0.38       292
           8       0.74      0.94      0.83       654
           9       0.84      0.73      0.78       345

    accuracy                           0.83     12088
   macro avg       0.88      0.70      0.73     12088
weighted avg       0.85      0.83      0.82     12088
```

```
cm = confusion_matrix(pred_probs, pred_class, labels =

print(cm)
```

```
[[  38  173    5    1   37    0    0    0    0    0]
 [   0 2645   75    8   52    0    1    0    0    1]
 [   0  307 2429   57   46    0    0    0    0    0]
 [   0  388  110 1145   96    0    1    0    2    3]
 [   0  180   36   47 2228    0    0    0    0    2]
 [   0    1    3    1    0  174    3    0   76    1]
 [   0   12    1    0    4    0  401    0    4    3]
 [   0   42    2    1    8    6   65   68   68   32]
 [   0   10    3    4    0    0   17    0  614    6]
 [   0    5    2    5   10    0    4    0   68  251]]
```

Figure 8.3 – Epoch 5



```
              precision    recall  f1-score   support

           0       0.89      0.40      0.55       254
           1       0.93      0.90      0.92      2782
           2       0.94      0.95      0.95      2839
           3       0.92      0.96      0.94      1745
           4       0.89      0.98      0.93      2493
           5       0.84      0.95      0.89       259
           6       0.90      0.95      0.92       425
           7       1.00      0.36      0.53       292
           8       0.88      0.95      0.91       654
           9       0.92      0.79      0.85       345

    accuracy                           0.92     12088
   macro avg       0.91      0.82      0.84     12088
weighted avg       0.92      0.92      0.91     12088
```

```
cm = confusion_matrix(pred_probs, pred_class, labels =

print(cm)
```

```
[[ 102   67    5    2   77    0    1    0    0    0]
 [   4 2510  109   34  122    0    0    0    3    0]
 [   2   45 2701   52   39    0    0    0    0    0]
 [   0   10   18 1675   37    0    1    0    2    2]
 [   3   18   24    4 2444    0    0    0    0    0]
 [   0    0    0    0    0  247    0    0   12    0]
 [   1    5    3    4    5    1  404    0    2    0]
 [   0   29    1   10    4   47   34  106   39   22]
 [   0    3    1   18    4    0    8    0  620    0]
 [   2    1    0   29   11    0    1    0   30  271]]
```

Figure 8.4 – Epoch 10