Project Report

Ethan Prihar, Odell Dotson

Description of the Features and Design Strategies

The first feature returns the player whose piece is in the bottom left corner of the board. This feature is not very useful because having a piece in the bottom left corner of the board is not something that should determine if a player wins. This feature was implemented because it was required by the project guidelines.

The second feature returns which player has more pieces that aren't at the edge of the board. This feature should be useful because when a piece is at the edge of the board it has a lot less potential to be connected to. In general a piece toward the center would have a higher chance of being used to win than a piece on the side. This feature predicts that the player with the least pieces at the edge will win.

The third feature calculates a heuristic value based on how close all of a players pieces are to the center. This feature is similar to feature two because it also values pieces that are away from the edge of the board. This feature goes beyond feature two because each piece closer to the center has more weight in the heuristic calculation. The heuristic is calculated by adding one for every piece in an edge column. For every column closer to the center, a piece in that column is worth twice as much as a piece in the adjacent column closer to the edge of the board. Therefore, the number of pieces in a column is multiplied by 1, 2, 4, 8, 4, 2, or 1 respectively. The total value of all of the columns is evaluated and the player with the highest total is returned by the feature.

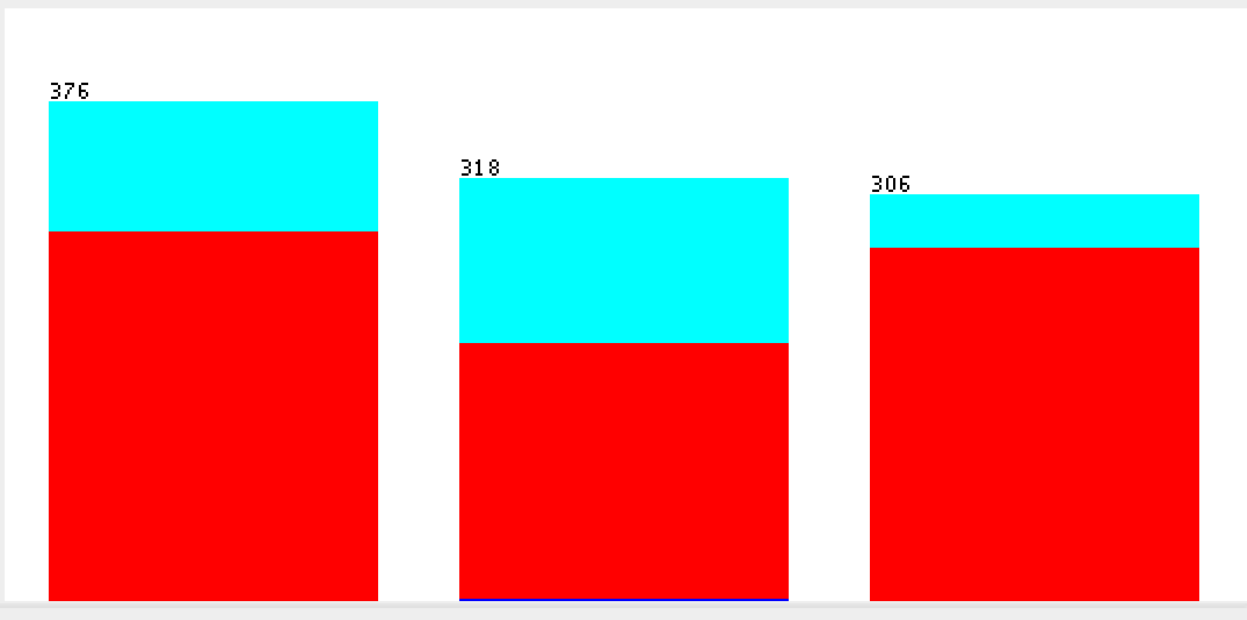
The fourth feature uses the same heuristic as feature three, but instead of returning whichever player has a higher heuristic, the feature returns the value of player one's heuristic minus player two's heuristic. This feature therefore returns a continuous function that represents how much of an advantage one player has over another because of a more centered positioning. This feature's usefulness will be compared to the previous feature's usefulness to determine whether it is better for this data set to use a continuous value to represent positioning advantage, or a categorized set of values.

The fifth feature tests who has "top control" of the columns of the board. "Top control" is defined as whoever has the highest piece on any given column. For every column where player 1 has top control, 1 is added to the top control variable and for every column that player 2 has top control, 1 is subtracted from this variable. Columns with no pieces in them do not count for top control at all and do not affect the top control score. When doing the first project, we found that often that your ability to make advantageous moves was dependent on how much control you had over the top edge of the board because you're only able to make plays that will become new edges on the top, interacting with the old top of the board.

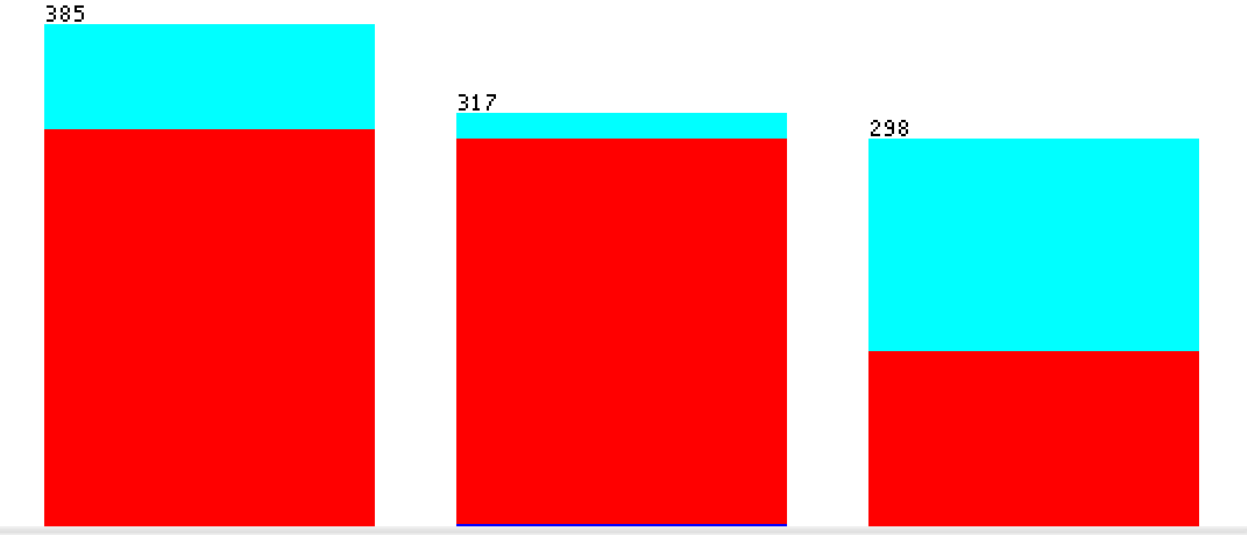
Analysis of the Features

To determine which features are better than others we can look at how well each feature predicts the winner of a board. To do this we show a histogram of every value the

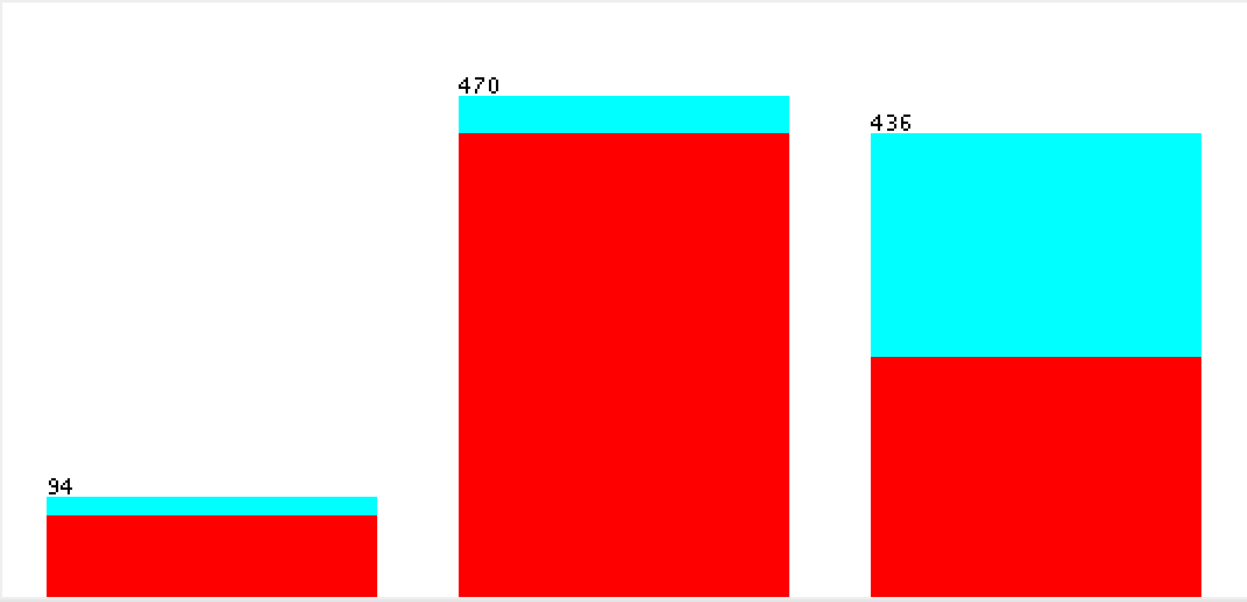
feature returns with how many times player 1 or player 2 won a game at each value of the feature. Player 1 won about 75% of its games, so if the histogram shows that at each value of the feature, about 75% of the time player 1 won, then we know that feature is useless. Speaking of useless features, below is a graph of feature 1.



As you can see from the data, every value of feature one leads to only a slight change in the win rate of player one, ranging from around 65% to 85% instead of 75%. There is only around a 10% change in the win rate of player 1 based on the value of feature 1 so this heuristic is not very good at predicting the winner. Feature 2 on the other hand has a larger change in the win rate of player one based on its value. The graph of Feature 2 is shown below.

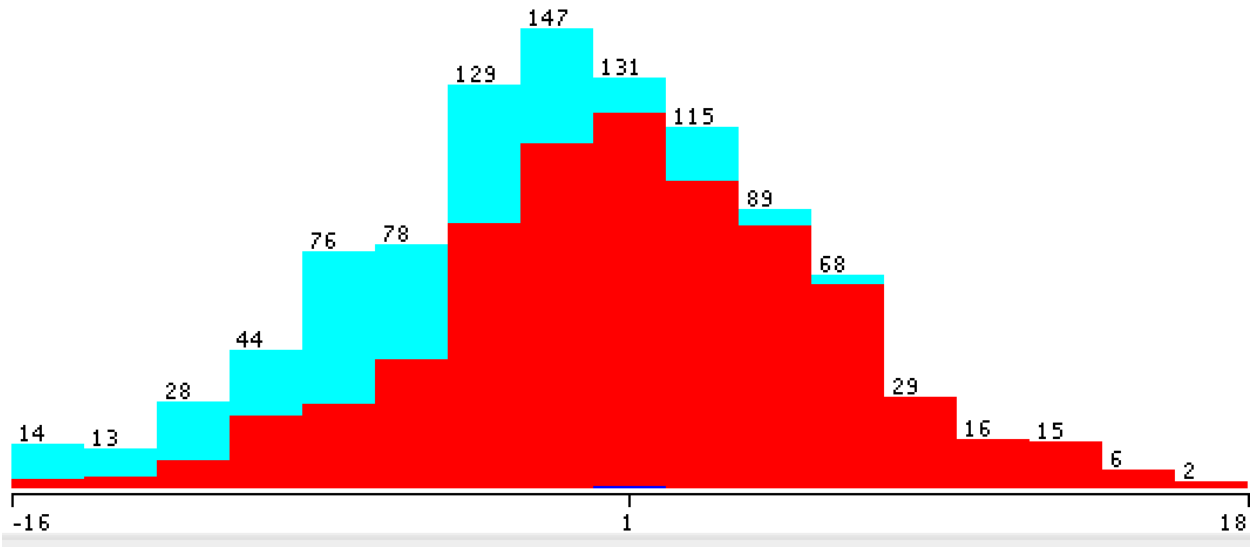


As shown by the graph, when feature 2 puts a data point into one category, player 1 wins 95% of the time, and when feature 2 puts a data point in a different category, player 1 wins 45% of the time. Both of these changes in the win rate of player 1 show that feature 2 can predict the winner much more accurately than feature 1. The issue with feature 2 is that it has one category that it cannot predict the outcome from accurately and it places over a third of the data into this category. Feature 2 is good, but only for two thirds of the data. Feature 3 on the other hand can almost as accurately predict the outcome of many more data points than feature 2 which is shown by the graph of feature 3 below.

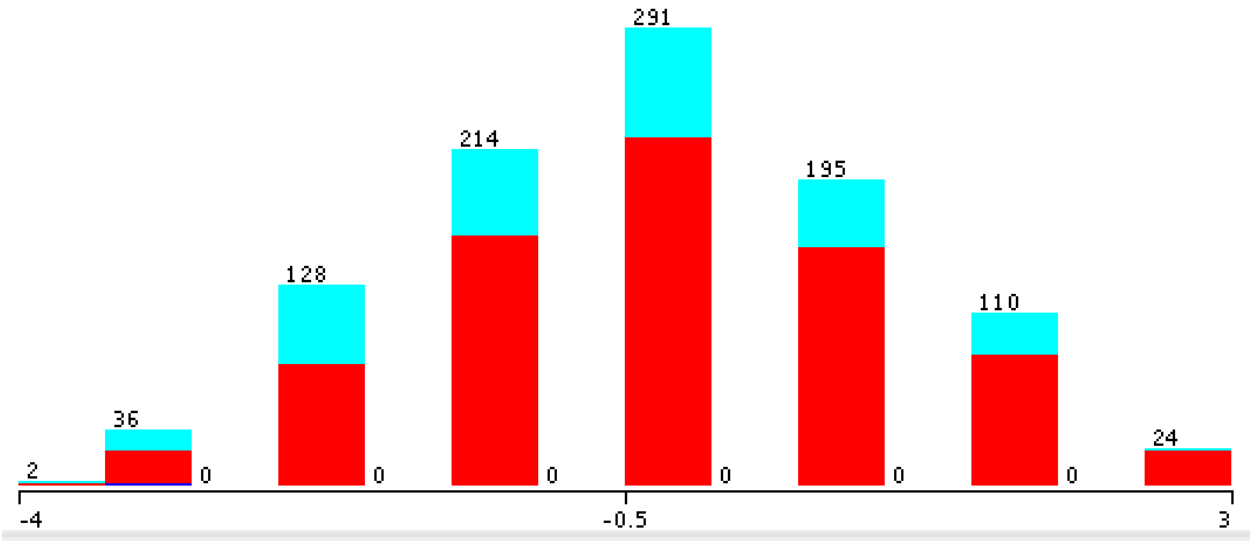


For each category in feature 3, there is almost a 25% change in the win rate of player 1 than if the predictions were random. Feature 3 can predict all of the data with this accuracy, whereas feature 2 could only predict two thirds of the data with this much accuracy.

Feature 4 creates a continuous output of values. The decision tree treats each value as a different category. Below is the graph of feature 4.



As shown by the graph, feature 4 has the potential to predict values more accurately than any of the previous graphs. For some values feature 4 can 100% accurately predict that player 1 will win, and that player 1 will win 15% of the time. There is only 30% of the data that feature 1 cannot predict accurately but it can more accurately predict the other 70% of the data than any of the other features. Feature 5 was not very good at predicting the winner and for every value of feature 5, there was not much change in the win rate of player 1 than if the winner was predicted randomly.



Based on the analysis of every feature, feature 4 is by far the most useful feature because it can predict whether or not player 1 will win with more accuracy than any of the other features. The only place feature 4 falls short is it's inability to accurate predict 30% of the data. This is where feature 3 is useful because it can predict the winner of all of the data almost as accurately as feature 4 can predict the winner of 70% of the data. So the 30% of

data that feature 4 cannot predict well can be predicted by feature 3, this will ensure all of the data is predicted with good accuracy.

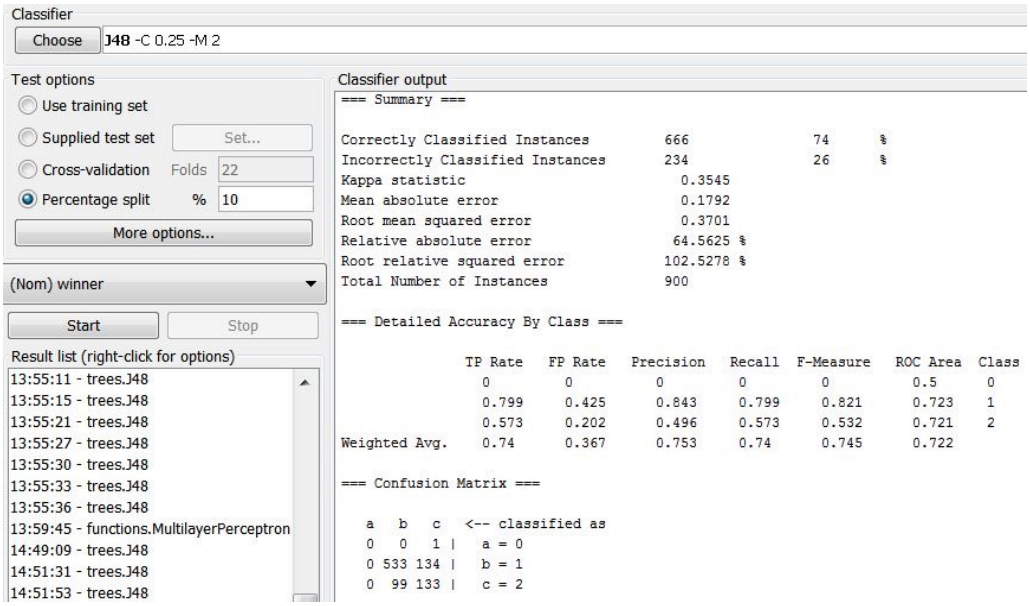
The Value of Cross Validation:

Cross-validation is used to prevent overfitting. Holdout percents will train a system on a portion of the data, and then test that trained model against the holdout percent, testing the validity of the model. This helps prevent overfitting because it is observable how badly off the prediction is on data that it has never seen before. This prevents it from becoming overfit on that data.

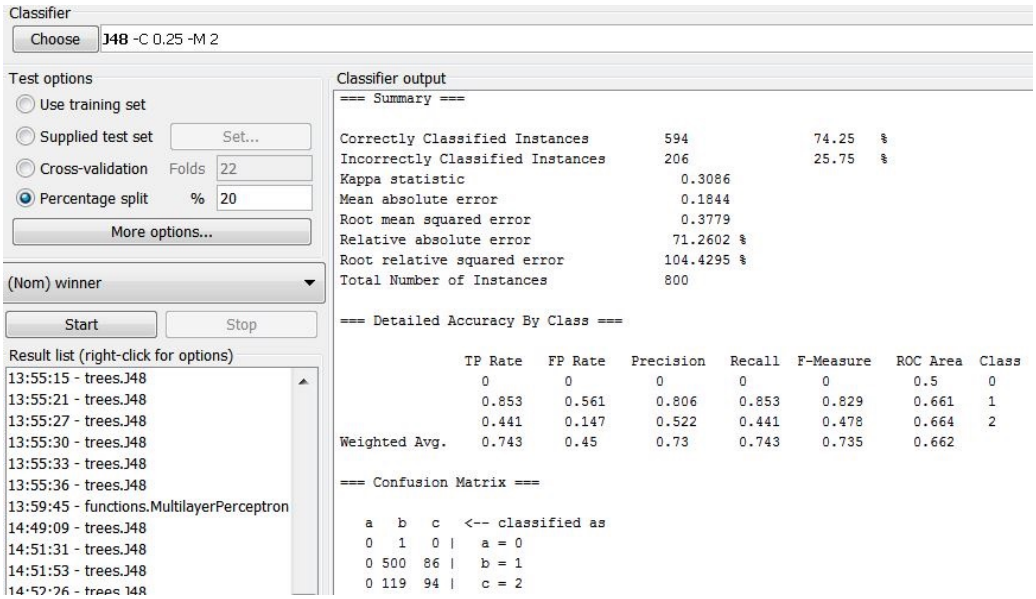
K-fold cross validation is essentially holdout cross-validation that runs multiple times, on the data split up in different ways. This prevents overfitting because across a full k-folded run, all the data will be run upon multiple times but each datapoint will also have been left out at some point in the testing. This helps to prevent overfitting because the data that is left out is not overfit for.

Decision Tree Testing

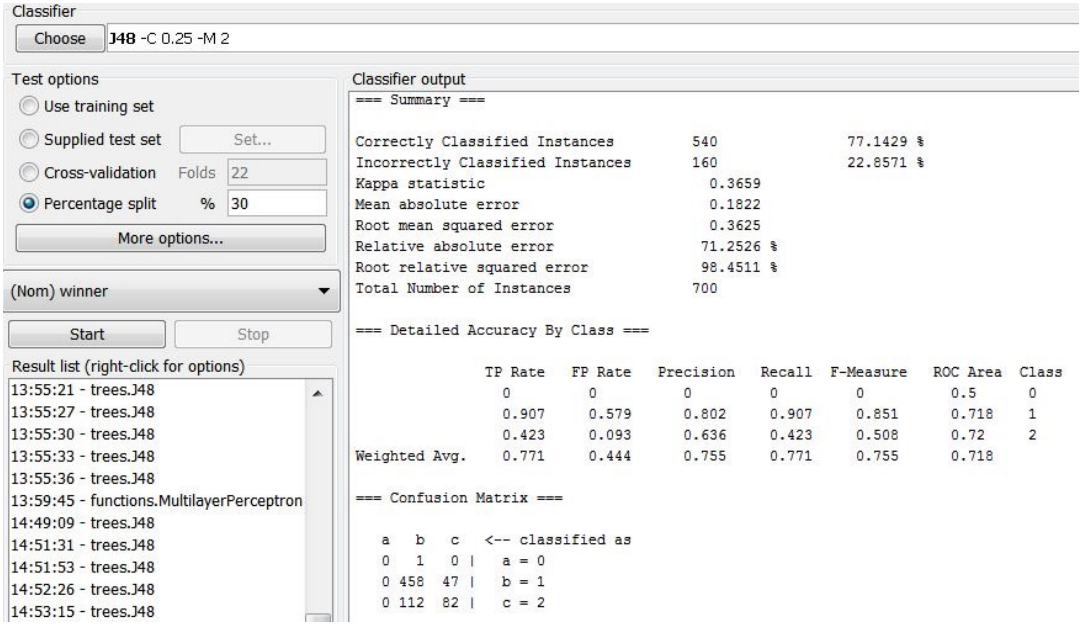
To create the optimal decision tree we used two methods of cross validation. First we held out a certain percent of the data and created a tree with the remaining data, then tested on the held out data. We started with a hold out of 10% and increased this percent until the accuracy of the tree began to decrease. Below is the screenshot of the 10% holdout test.



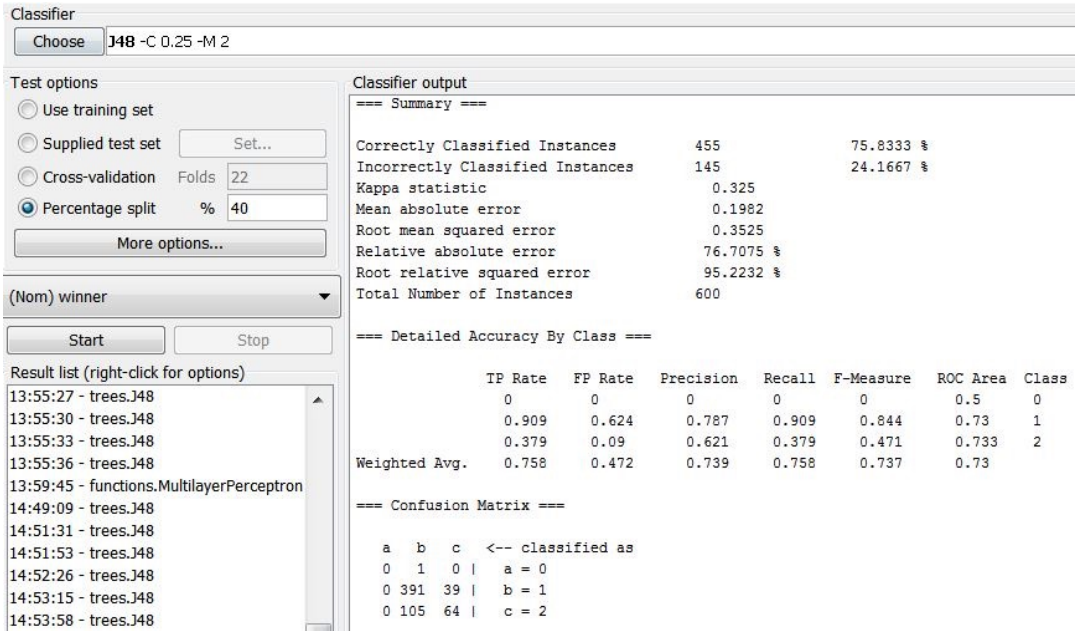
As shown, holding out 10% of the data lead to 74% accuracy. When increasing to 20% hold out the accuracy increased. Below is the 20% holdout test.



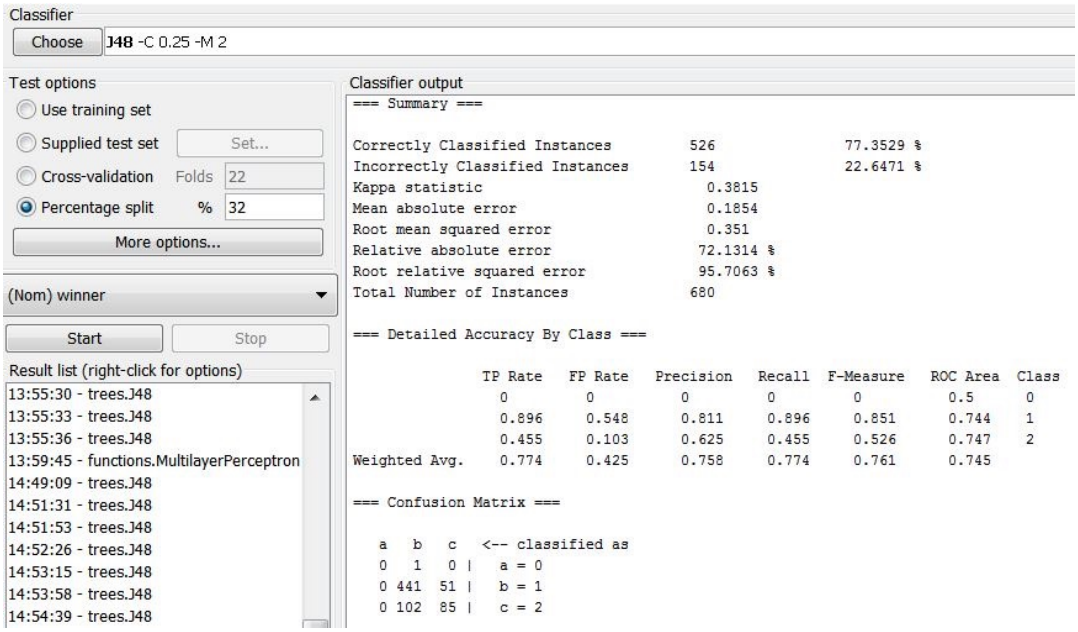
The accuracy of the prediction with 20% hold out had increased from 10% by 0.25%. The accuracy of the prediction continued to increase as the hold out percent was increased to 30%. The screenshot of the 30% hold out test is shown below.



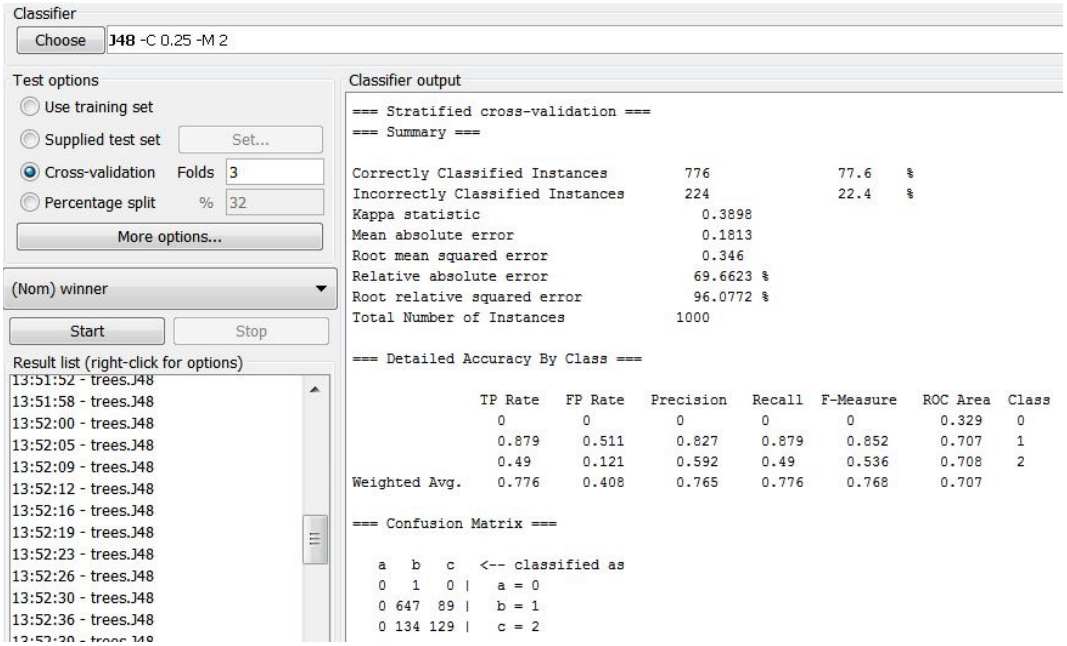
When increasing the hold out percentage to 30% the accuracy jumped to 77.1%. Again the hold out percentage was increased by 10% to see if the accuracy would continue to increase. Shown below is the screenshot of the 40% hold out test.



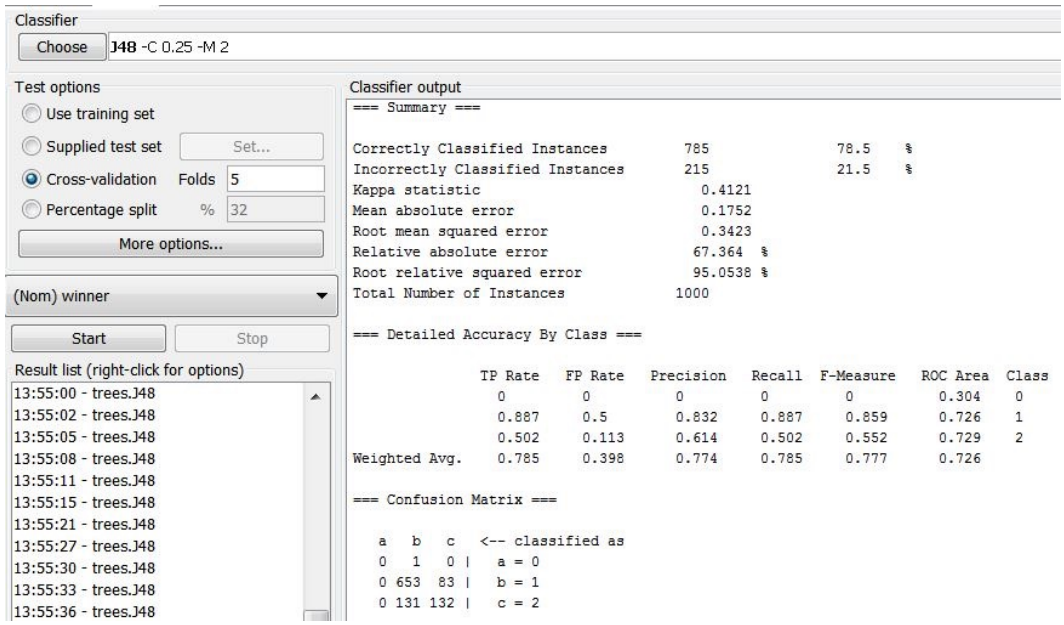
As shown by the data, the accuracy of the decision tree at 40% hold out decreases from the accuracy at 30%. This means that the optimal amount to hold out is somewhere between 30% and 40%. The highest accuracy was achieved at 32% holdout with an accuracy of 77.4%. Below is the screenshot of the 32% hold out test.



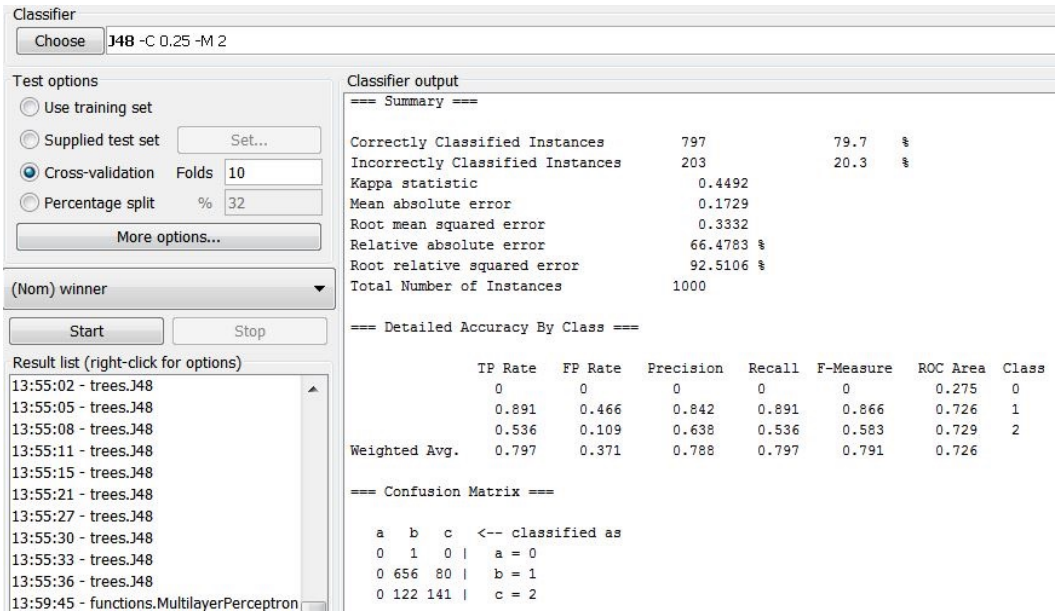
After the best hold out percentage was determined the next step was to find the most accurate model using K-fold cross validation. To determine the best number of folds. First, a low number of folds was used. Then, the number of folds was increased until the accuracy began to decrease. Below is the screenshot of the summary of a three fold decision tree.



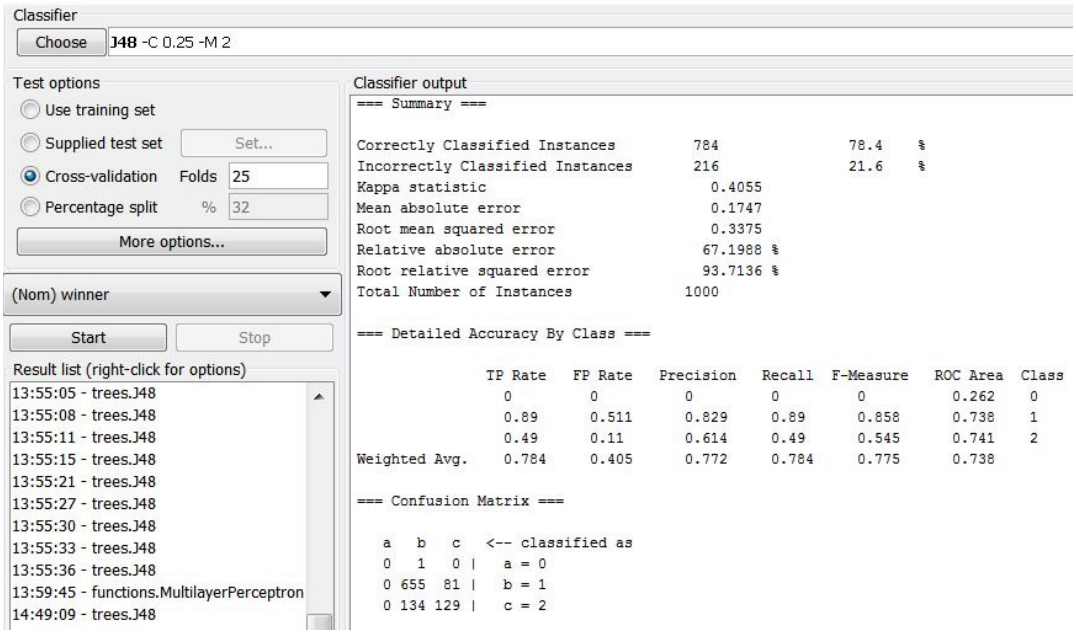
With three folds the decision tree was 77.6% accurate. When the number of folds was increased to 5, the model became even more accurate. Below is the screenshot of the model with 5 folds.



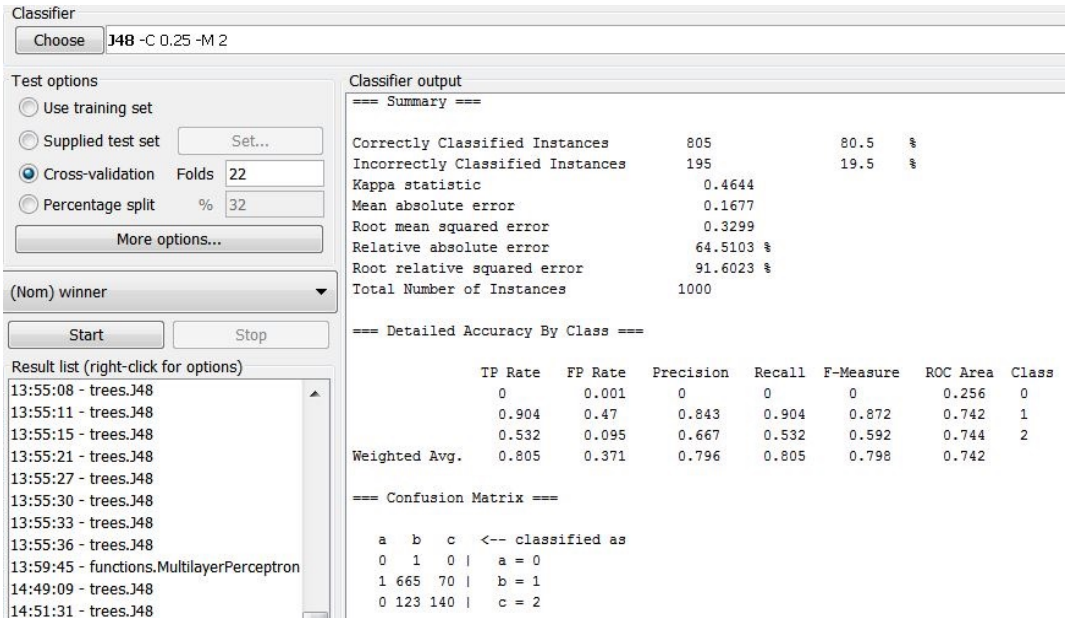
As shown, the accuracy of the decision tree's predictions increased to 78.5%. This showed that as the number of folds increased, so did the accuracy of the model. The next step was increasing the number of folds to 10. Below is the screenshot of the 10 fold decision tree analysis.



The accuracy of the model increased again to 79.7%, The number of folds was again increased, this time to 25, the plan was to increase the number of folds until the accuracy of the model started to decrease. Shown below is the screenshot of the 25 fold decision tree analysis.



When the number of folds was increased to 25 the accuracy decreased to 78.4%. This meant that the ideal number of folds was between 10 and 25. After experimenting the ideal amount of folds was determined to be 22. Below is the screenshot of the 22 fold decision tree analysis.

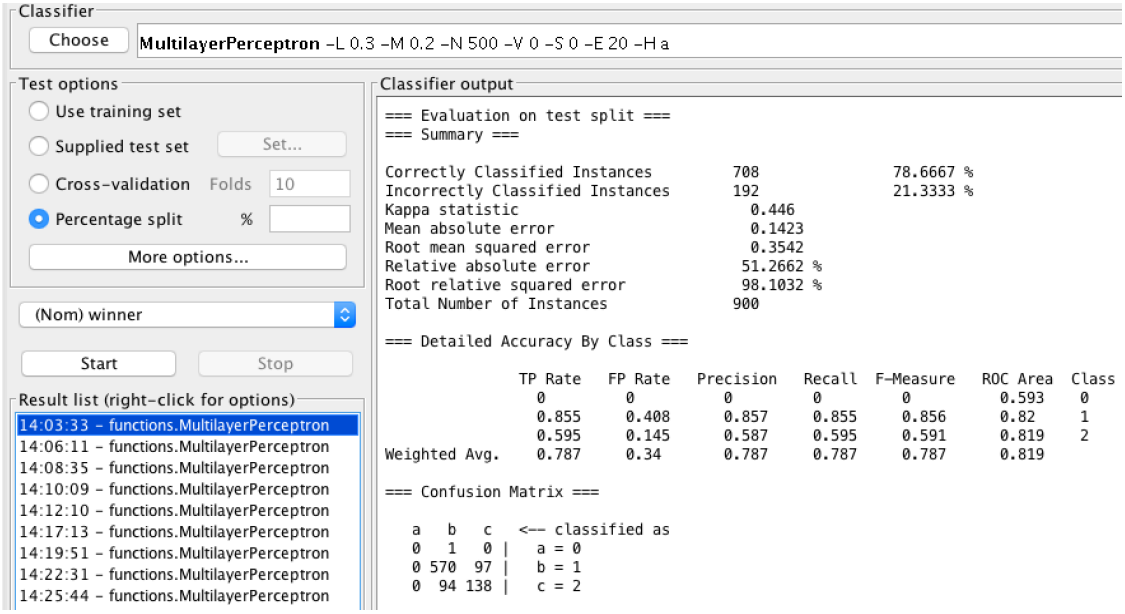


The maximum accuracy from holding out a percent of the data was only 77.4% whereas the maximum accuracy from K-fold cross validation was 80.5%. Based on the data K-fold cross validation is the best method of cross validation to use for creating a decision tree. The best decision tree we could create was cross validated with 22 folds and we achieved an accuracy of 80.5%

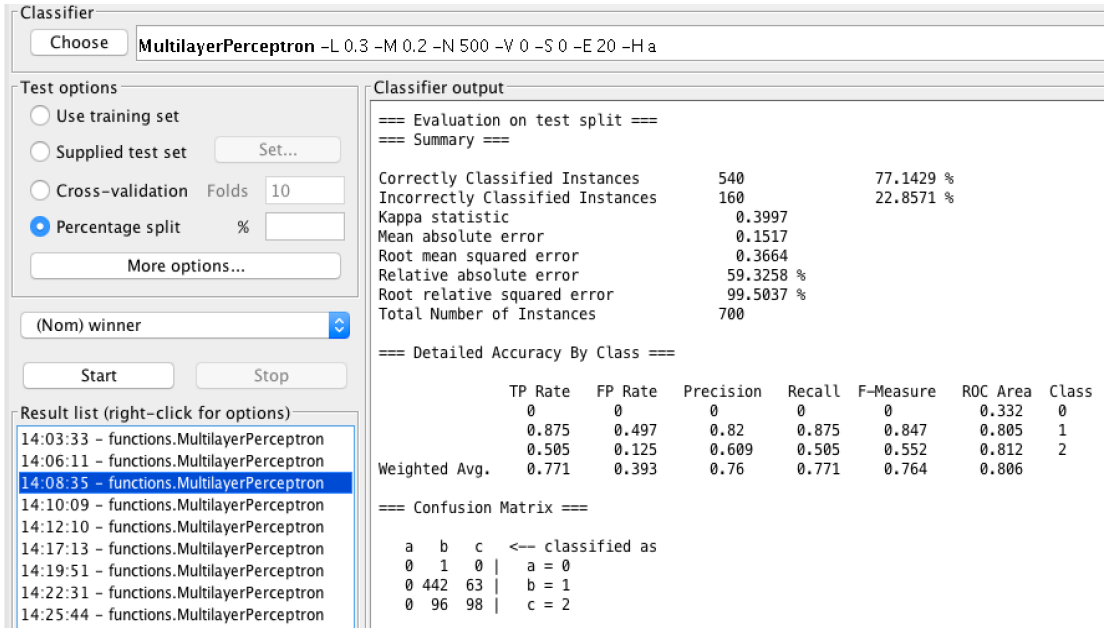
Neural Net Testing

For our neural network testing, it was first planned to run percentage split tests on 10%, 20%, 30% and 40% holdouts. Not finding a conclusive “best” holdout percentage from these tests, we continued and tested on 50% through 90% holdout at 10% intervals. Some of these tests are shown here to provide the readier with a good understanding of the data. The rest are shown in Appendix A.

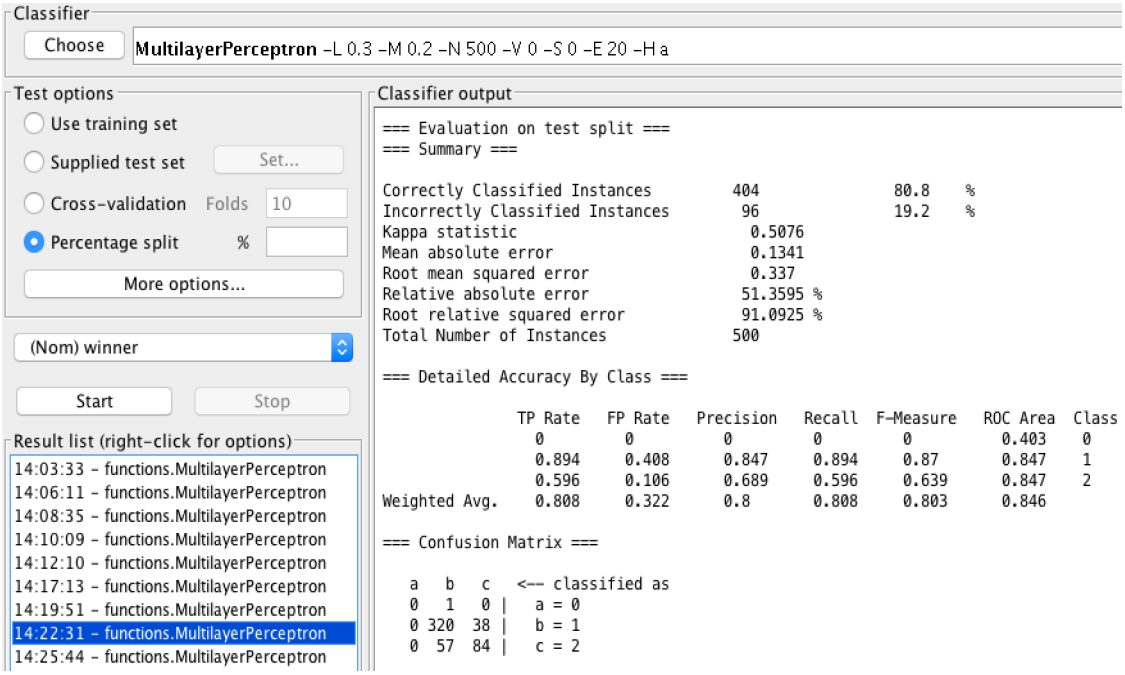
10% Holdout:



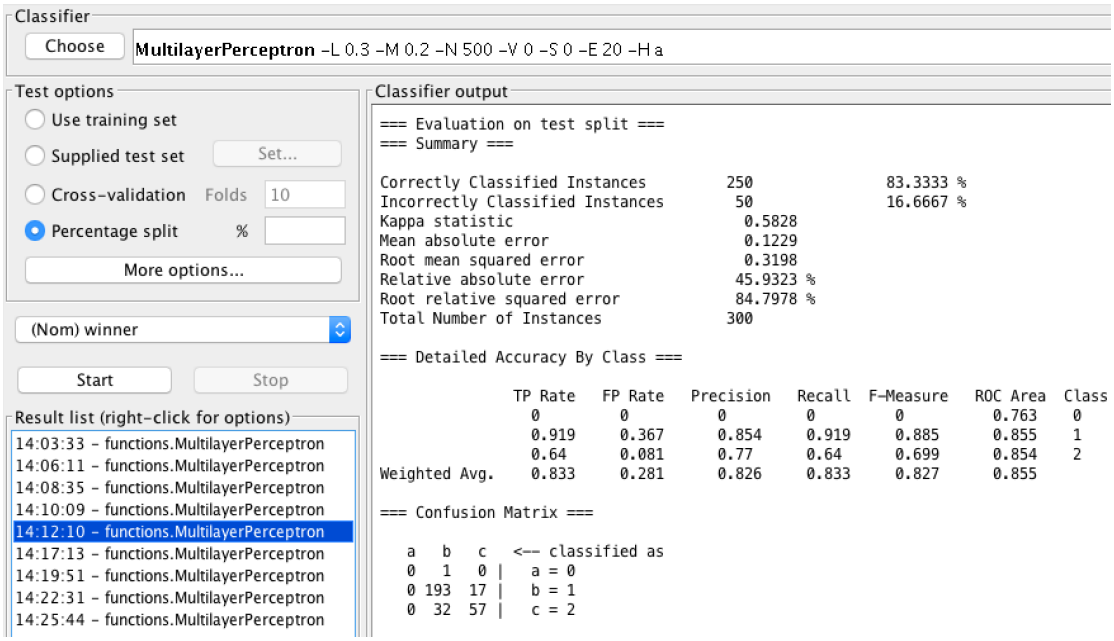
30% Holdout:



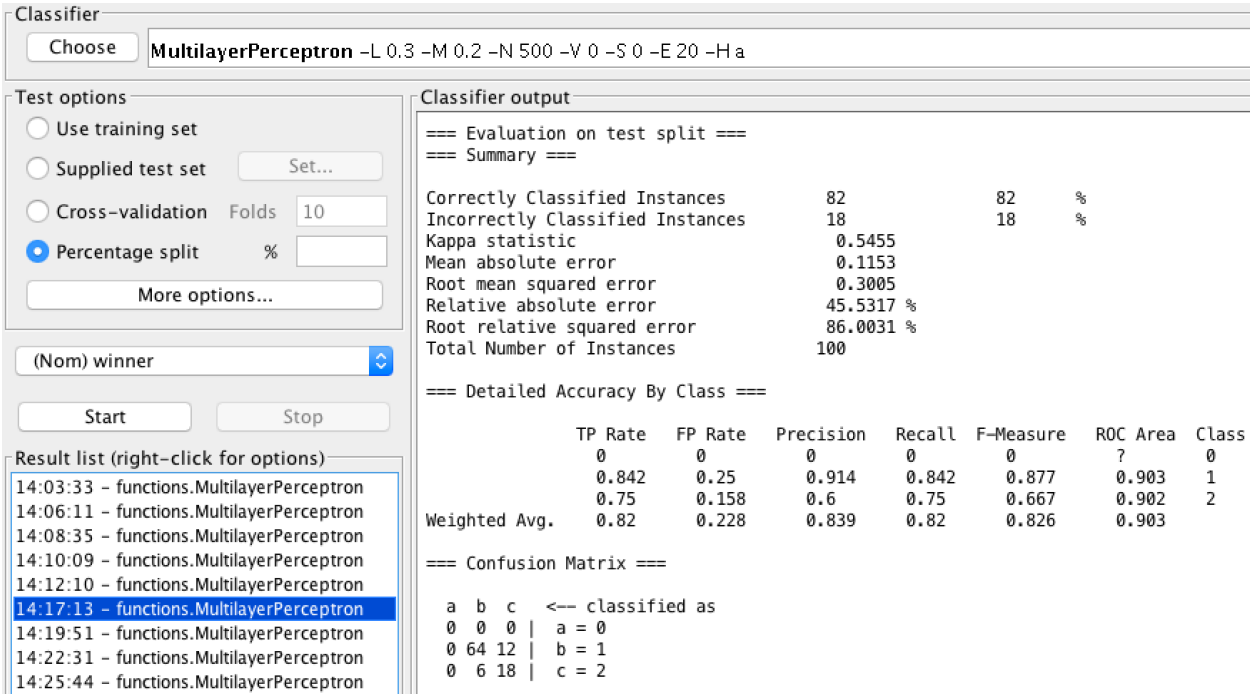
50% holdout:



70% Holdout:



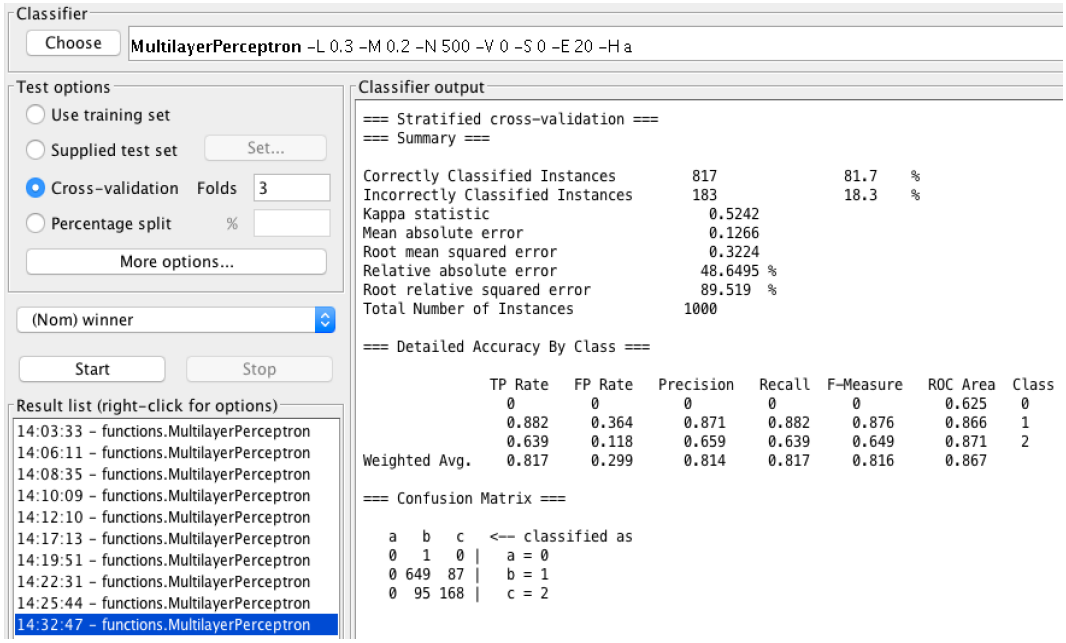
90% Holdout:



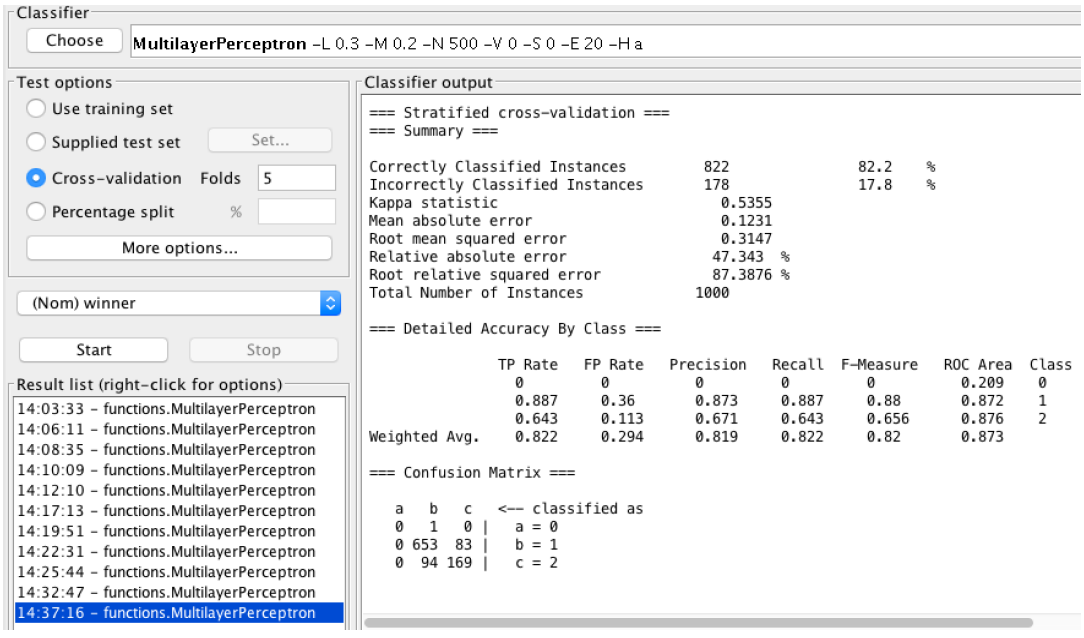
The highest percentage of correctly classified instances was 83.33% correct, which occurred at 70% holdout percent. This was a surprise, as it needed only 300 data point of the provided 1000 to train a very good neural network. This does make some sense though, as the 300 data point may provide less overfitting that could cause problems when attempting to correctly tune a neural network.

We then tested a neural network using K-Folding, with K-folds of 3, 5, 10 and 25.

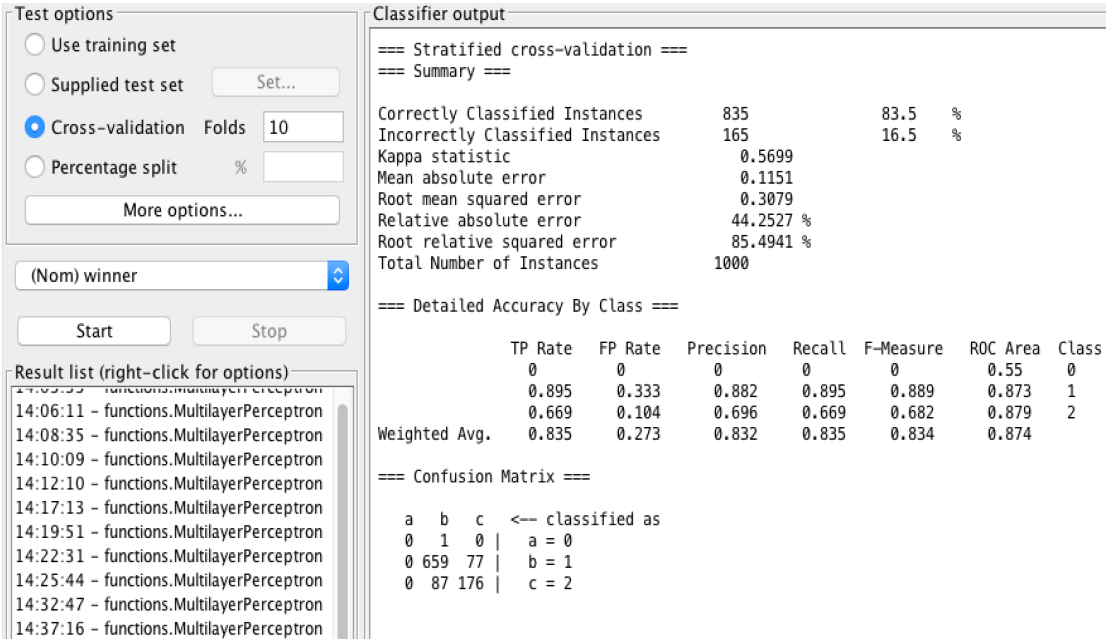
K = 3:


Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
3
Percentage split
%
More options...
(Nom) winner
Start
Stop
Result list (right-click for options)
14:03:33 - functions.MultilayerPerceptron
14:06:11 - functions.MultilayerPerceptron
14:08:35 - functions.MultilayerPerceptron
14:10:09 - functions.MultilayerPerceptron
14:12:10 - functions.MultilayerPerceptron
14:17:13 - functions.MultilayerPerceptron
14:19:51 - functions.MultilayerPerceptron
14:22:31 - functions.MultilayerPerceptron
14:25:44 - functions.MultilayerPerceptron
14:32:47 - functions.MultilayerPerceptron
Classifier output
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 817 81.7 %
Incorrectly Classified Instances 183 18.3 %
Kappa statistic 0.5242
Mean absolute error 0.1266
Root mean squared error 0.3224
Relative absolute error 48.6495 %
Root relative squared error 89.519 %
Total Number of Instances 1000
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
0 0 0 0 0 0.625 0
0.882 0.364 0.871 0.882 0.876 0.866 1
0.639 0.118 0.659 0.639 0.649 0.871 2
Weighted Avg. 0.817 0.299 0.814 0.817 0.816 0.867
=== Confusion Matrix ===
a b c <-- classified as
0 1 0 | a = 0
0 649 87 | b = 1
0 95 168 | c = 2

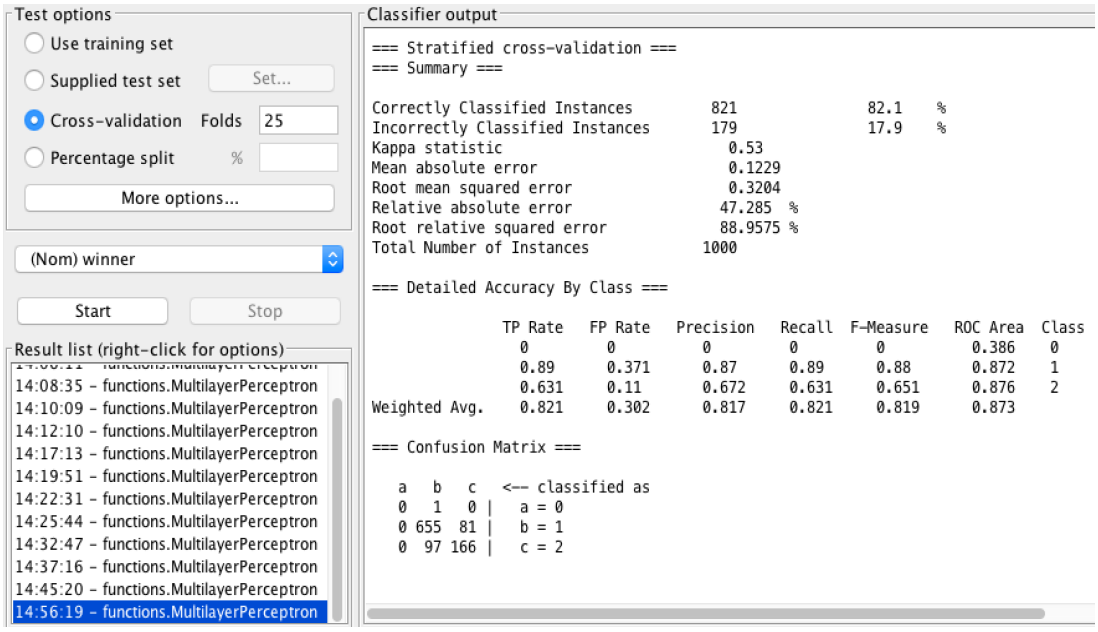
K = 5:


Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
5
Percentage split
%
More options...
(Nom) winner
Start
Stop
Result list (right-click for options)
14:03:33 - functions.MultilayerPerceptron
14:06:11 - functions.MultilayerPerceptron
14:08:35 - functions.MultilayerPerceptron
14:10:09 - functions.MultilayerPerceptron
14:12:10 - functions.MultilayerPerceptron
14:17:13 - functions.MultilayerPerceptron
14:19:51 - functions.MultilayerPerceptron
14:22:31 - functions.MultilayerPerceptron
14:25:44 - functions.MultilayerPerceptron
14:32:47 - functions.MultilayerPerceptron
14:37:16 - functions.MultilayerPerceptron
Classifier output
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 822 82.2 %
Incorrectly Classified Instances 178 17.8 %
Kappa statistic 0.5355
Mean absolute error 0.1231
Root mean squared error 0.3147
Relative absolute error 47.343 %
Root relative squared error 87.3876 %
Total Number of Instances 1000
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
0 0 0 0 0 0.209 0
0.887 0.36 0.873 0.887 0.88 0.872 1
0.643 0.113 0.671 0.643 0.656 0.876 2
Weighted Avg. 0.822 0.294 0.819 0.822 0.82 0.873
=== Confusion Matrix ===
a b c <-- classified as
0 1 0 | a = 0
0 653 83 | b = 1
0 94 169 | c = 2

K = 10:



K = 25:

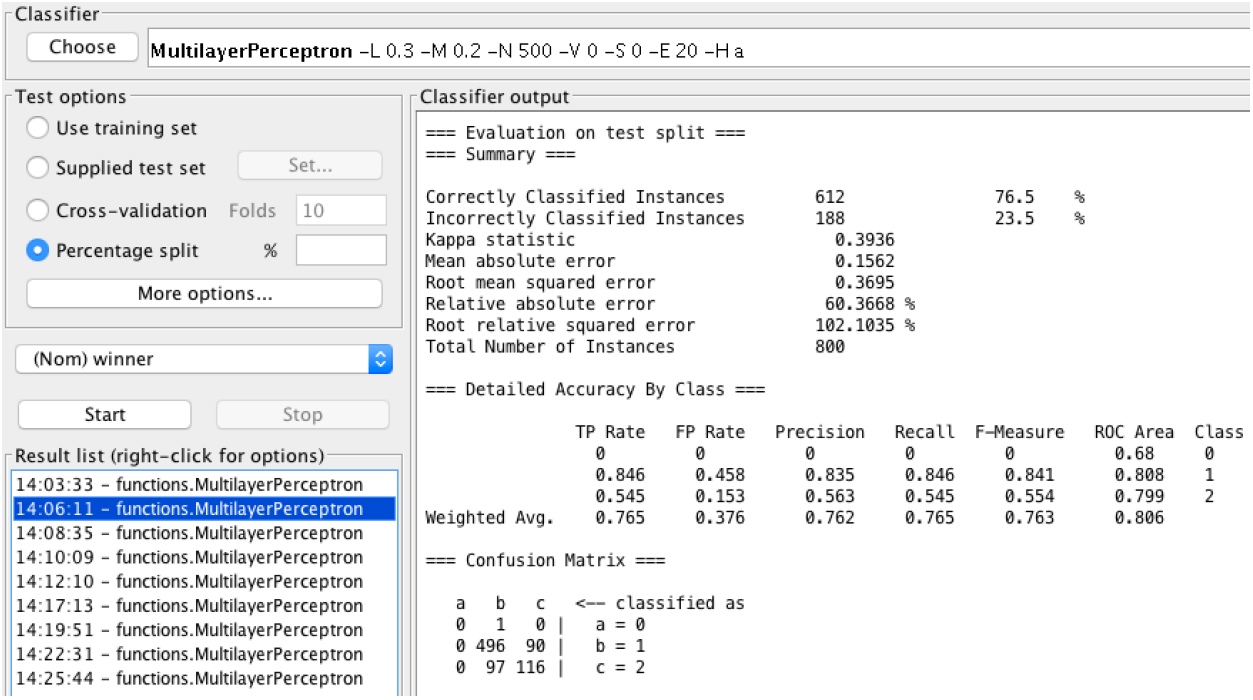


The best number of folds we found was k = 10, which trained the neural network to correctly clarify instances in 83.5% of cases.

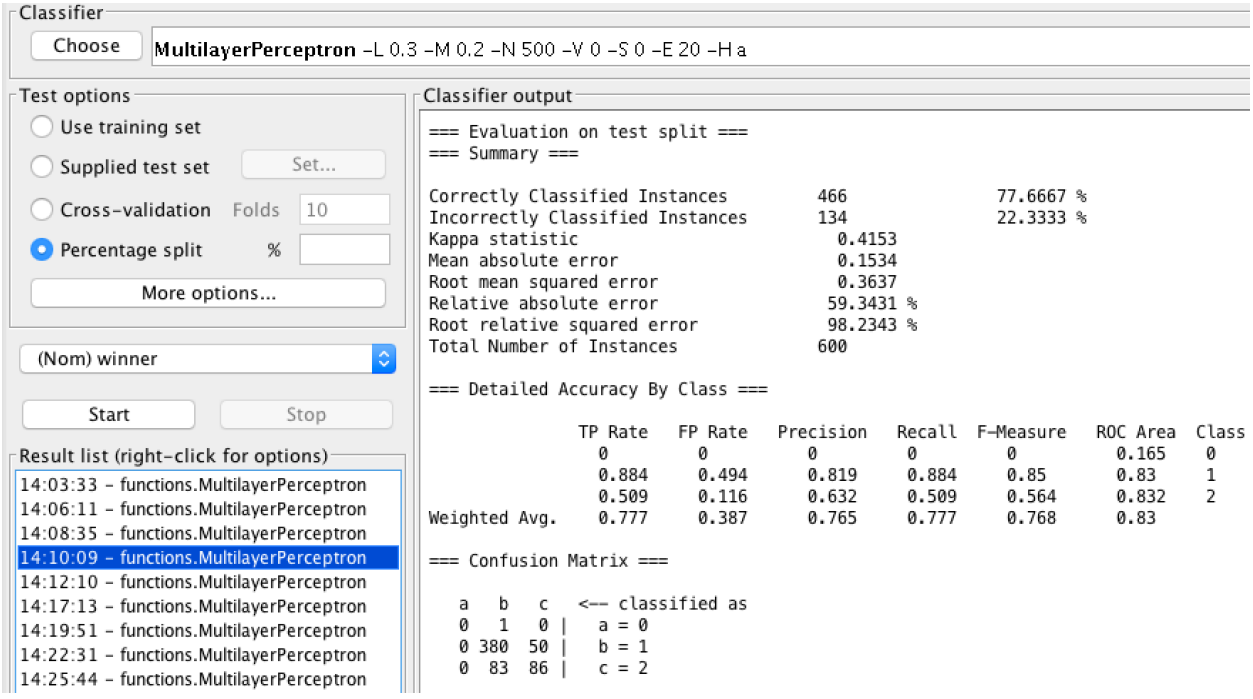
K-folding was much more consistent in percentage classified correctly than holdout percentages when looking across a wide range of different holdout percentages or k-folds. K-folding was on average a better method, too. K-folding and holdout percentages both were able to produce almost exactly the same level of correctly classified instances at peak, however. Because of this, we are unable to draw any strong conclusions as to which is a better method for training a neural network.

Appendix A:

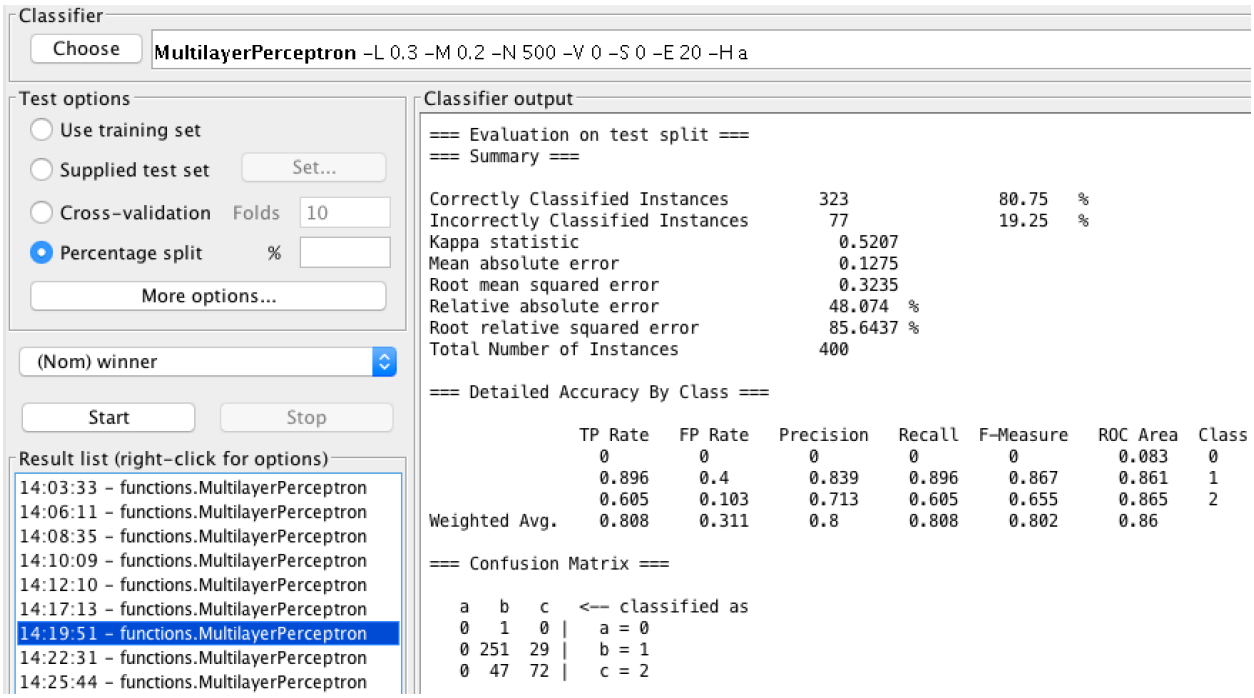
20% Holdout:



40% Holdout:



60% Holdout:



80% Holdout: