Peter Hogya

Executive Summary

The purpose of the second part of this project is to repeat the analytical, data driven approach to designing an Intrusion Detection System from part one with another machine learning algorithm to compare their efficacy in this particular problem. Specifically this project will use Weka's MLP, Multilayer Perceptron, algorithm, a type of neural network. A neural network is a computer algorithm that emulates the function of neurons in a brain. They receive inputs from multiple sources and the combination of those inputs determines whether or not the neuron generates a positive output. A perceptron is a data structure which imitates this process. A Multilayer Perceptron algorithm uses a series of perceptron layers, where each layer contains several perceptrons and directs their outputs into the next layer of perceptrons. Eventually these layers feed into a final output, classifying the data instance.

Using this algorithm somewhat inconsistent results were achieved. Several of the attacks had perfect classification and a few had precision and recall between sixty and seventy percent. Additionally the signature and anomaly sets, the ones that would actually be used with and IDS, were on the lower end. Changing parameters in the MLP infrequently yielded improvements and often runtime would increase with no change in classification efficacy.

Finally implementing the output of a MLP algorithm in an IDS tool like Snort is exceedingly difficult. It requires calculations beyond the capability of a normal machine to brute force and so instead requires a human to do a great deal of multivariable calculus. In a realistic attempt at creating a complete IDS the amount of data and types of attacks would render completing this task beyond any kind of reasonable time constraints.

Specification

As in the first part of the project both types of IDSs, signature and anomaly-based, were examined. Therefore the csv, comma separated value, files from the first part of the project were used again, specifically the ones outputted directly from Wireshark.

The same preprocessing program was used as before, with some modifications. Packets that did not use either the TCP or UDP were removed as Snort can only apply rules to those two protocols. As a result the HTTP Request Method feature was also removed. Finally several more csv files were also generated to examine the MLP algorithms ability to classify a binary outcome, a single labeled attack against normal traffic and other attacks labeled identically as not the that attack. Five additional csv files were generated, one for each attack type.

Next the csv files were modified into arff files. This just requires the labeling of data and features to allow Weka to easily interpret the file.

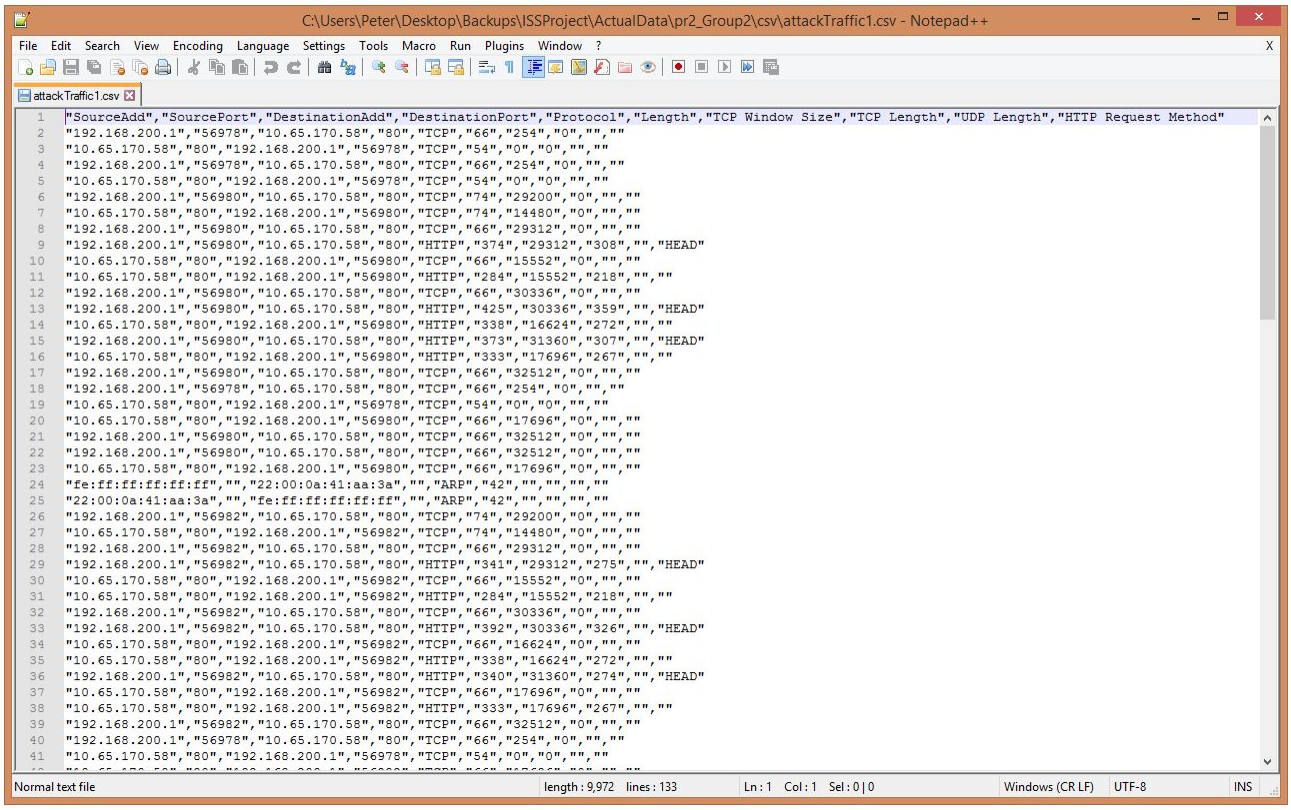
Once the data was preprocessed and converted into arff files it was fed into Weka's MLP algorithm. The algorithm was trained and tested with a 66% data split, as was specified in part one. Several different parameters were examined more closely, specifically the learning rate and the number of epochs trained through. In the results section there are several sets of outcomes for different values of these parameters. The learning rate was chosen as it has the

most influence over the rest of the process, assuming reasonable values for the other parameters. The number of epochs was chosen as improving accuracy often requires increasing runtime, either through increasing the size of the data set, increasing the number of features in the data set, or increasing certain parameters in the algorithm. By finding the minimum number of epochs to run while still achieving maximum accuracy for our data we have greater leeway to increase the complexity of our data set while still creating the MLP network in a reasonable amount of time. The results of the tests can be found in the Results section.

Methods and Techniques

The same csv files from part one were used.

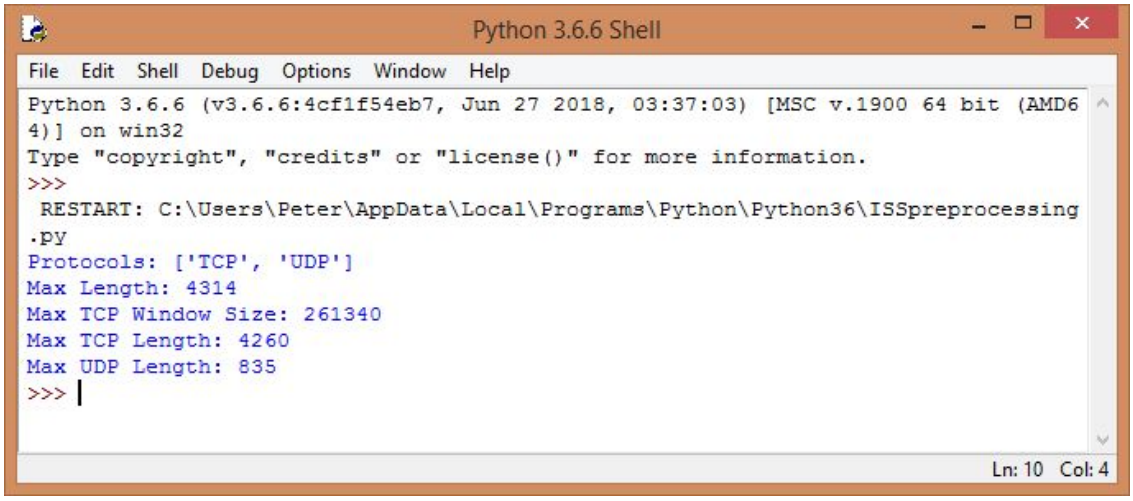
Raw csv:



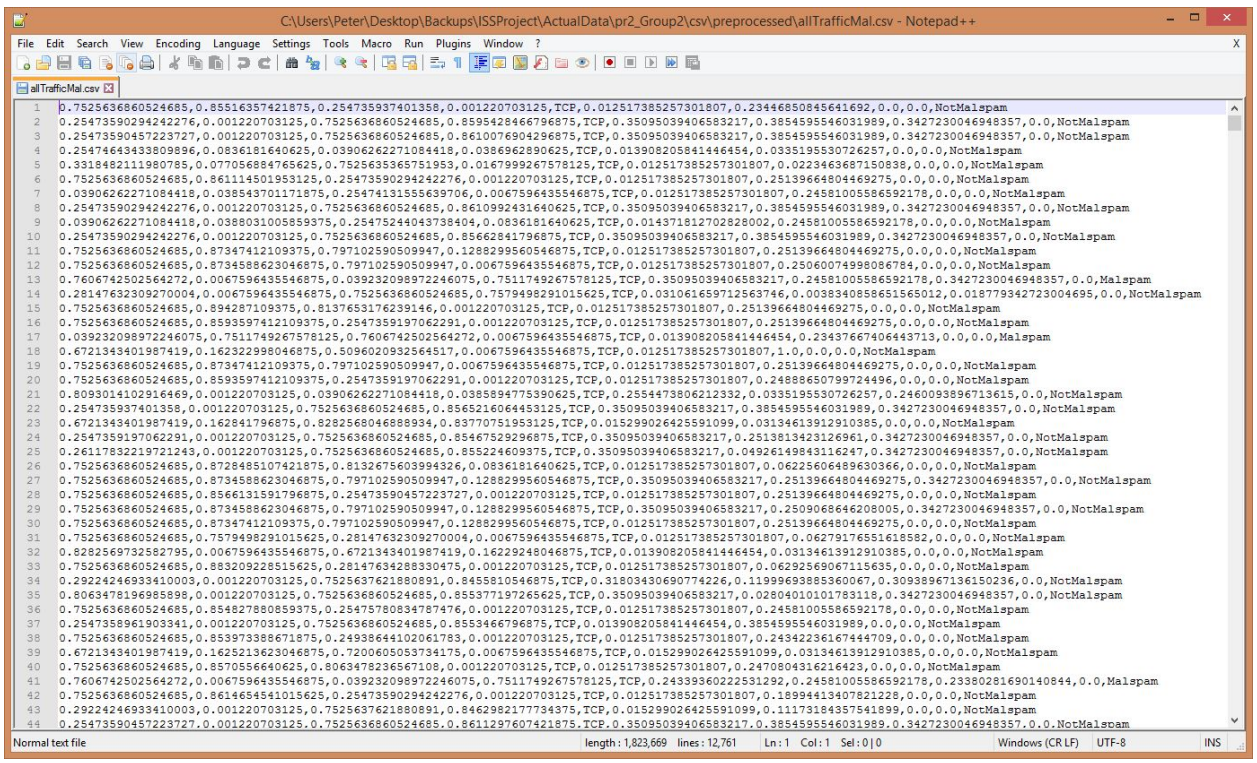

```
"SourceAdd","SourcePort","DestinationAdd","DestinationPort","Protocol","Length","TCP Window Size","TCP Length","UDP Length","HTTP Request Method"
"192.168.200.1","56978","10.65.170.58","80","TCP","66","254","0","",""
"10.65.170.58","80","192.168.200.1","56978","TCP","54","0","0","",""
"192.168.200.1","56978","10.65.170.58","80","TCP","66","254","0","",""
"10.65.170.58","80","192.168.200.1","56978","TCP","54","0","0","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","74","29200","0","",""
"10.65.170.58","80","192.168.200.1","56980","TCP","74","14480","0","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","66","29312","0","",""
"192.168.200.1","56980","10.65.170.58","80","HTTP","374","29312","308","","HEAD"
"10.65.170.58","80","192.168.200.1","56980","TCP","66","15552","0","",""
"10.65.170.58","80","192.168.200.1","56980","HTTP","284","15552","218","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","66","30336","0","",""
"192.168.200.1","56980","10.65.170.58","80","HTTP","425","30336","359","","HEAD"
"10.65.170.58","80","192.168.200.1","56980","HTTP","338","16624","272","",""
"192.168.200.1","56980","10.65.170.58","80","HTTP","373","31360","307","","HEAD"
"10.65.170.58","80","192.168.200.1","56980","HTTP","333","17696","267","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","66","32512","0","",""
"192.168.200.1","56978","10.65.170.58","80","TCP","66","254","0","",""
"10.65.170.58","80","192.168.200.1","56978","TCP","54","0","0","",""
"10.65.170.58","80","192.168.200.1","56980","TCP","66","17696","0","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","66","32512","0","",""
"192.168.200.1","56980","10.65.170.58","80","TCP","66","32512","0","",""
"10.65.170.58","80","192.168.200.1","56980","TCP","66","17696","0","",""
"fe:ff:ff:ff:ff:ff","","22:00:0a:41:aa:3a","","ARP","42","","","",""
"22:00:0a:41:aa:3a","","fe:ff:ff:ff:ff:ff","","ARP","42","","","",""
"192.168.200.1","56982","10.65.170.58","80","TCP","74","29200","0","",""
"10.65.170.58","80","192.168.200.1","56982","TCP","74","14480","0","",""
"192.168.200.1","56982","10.65.170.58","80","TCP","66","29312","0","",""
"192.168.200.1","56982","10.65.170.58","80","HTTP","341","29312","275","","HEAD"
"10.65.170.58","80","192.168.200.1","56982","TCP","66","15552","0","",""
"10.65.170.58","80","192.168.200.1","56982","HTTP","284","15552","218","",""
"192.168.200.1","56982","10.65.170.58","80","TCP","66","30336","0","",""
"192.168.200.1","56982","10.65.170.58","80","HTTP","392","30336","326","","HEAD"
"10.65.170.58","80","192.168.200.1","56982","TCP","66","16624","0","",""
"10.65.170.58","80","192.168.200.1","56982","HTTP","338","16624","272","",""
"192.168.200.1","56982","10.65.170.58","80","HTTP","340","31360","274","","HEAD"
"10.65.170.58","80","192.168.200.1","56982","TCP","66","17696","0","",""
"10.65.170.58","80","192.168.200.1","56982","HTTP","333","17696","267","",""
"192.168.200.1","56982","10.65.170.58","80","TCP","66","32512","0","",""
"192.168.200.1","56978","10.65.170.58","80","TCP","66","254","0","",""
"10.65.170.58","80","192.168.200.1","56978","TCP","54","0","0","",""
```

These were preprocessed using a slightly modified version of the Python preprocessing program from part one, ISSpreprocessing.py.

Sample Output:

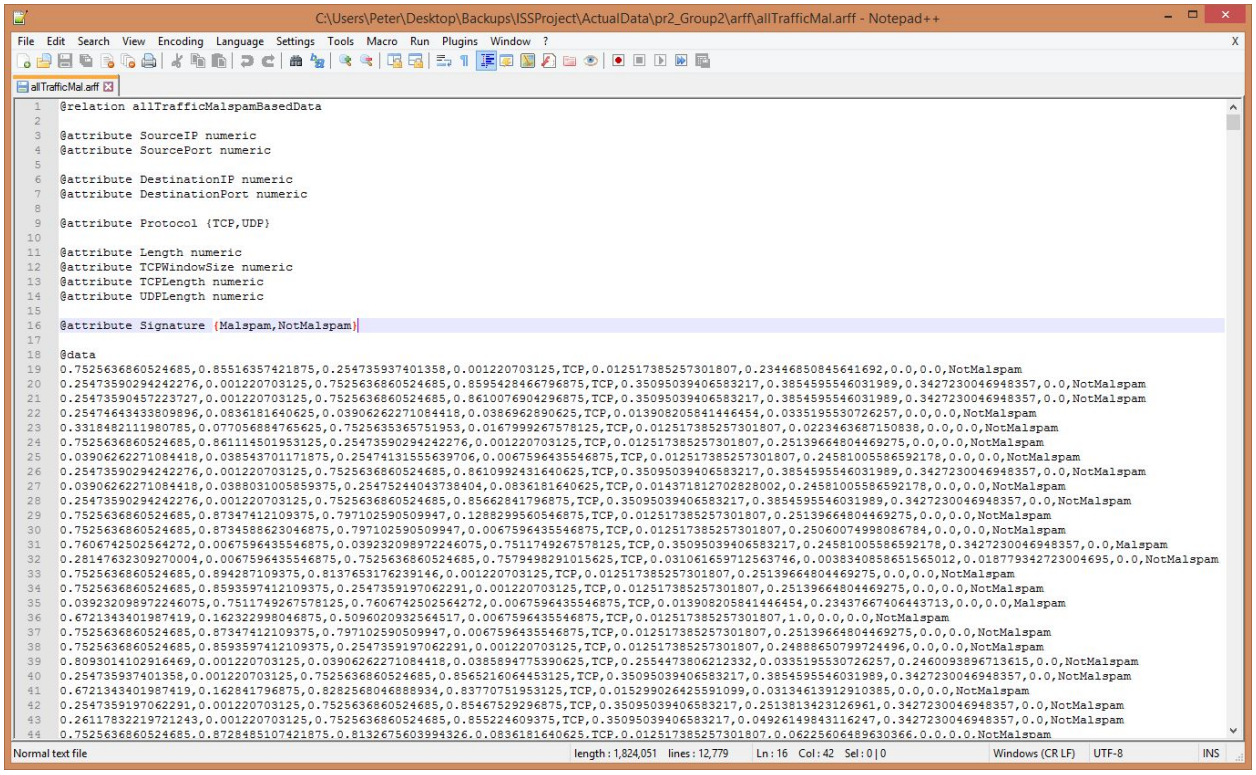


Preprocessed csv:



The preprocessed csv files were then converted into arff files

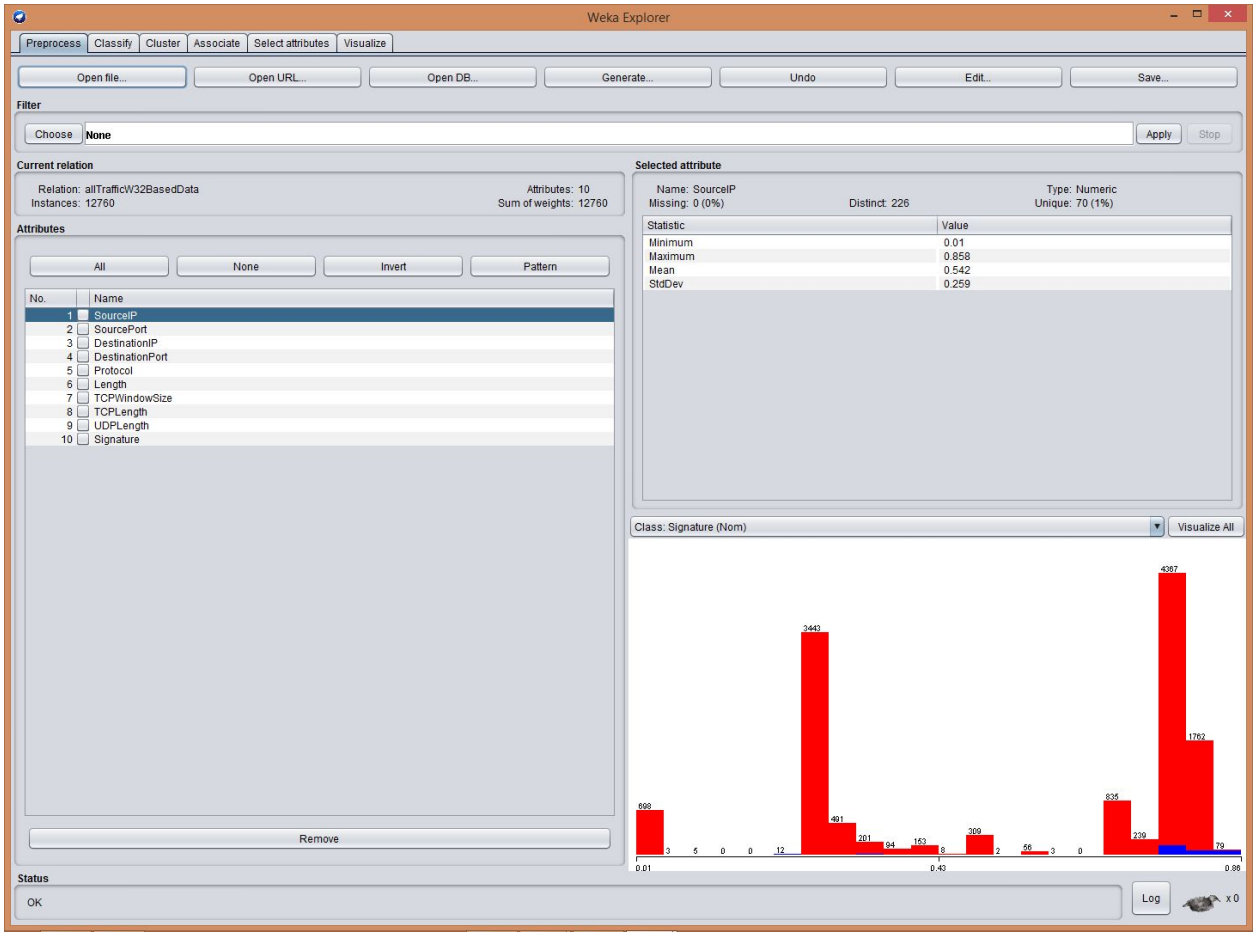
arff File:




```
@relation allTrafficMalspamBasedData

@attribute SourceIP numeric
@attribute SourcePort numeric

@attribute DestinationIP numeric
@attribute DestinationPort numeric

@attribute Protocol {TCP,UDP}

@attribute Length numeric
@attribute TCPWindowSize numeric
@attribute TCPLength numeric
@attribute UDPLength numeric

@attribute Signature {Malspam,NotMalspam}

@data
0.7525636860524685,0.85516357421875,0.254735937401358,0.001220703125,TCP,0.012517385257301807,0.23446850845641692,0.0,0.0,NotMalspam
0.25473590294242276,0.001220703125,0.7525636860524685,0.8595428466796875,TCP,0.35095039406583217,0.3854595546031989,0.3427230046948357,0.0,NotMalspam
0.25473590457223727,0.001220703125,0.7525636860524685,0.8610076904296875,TCP,0.35095039406583217,0.3854595546031989,0.3427230046948357,0.0,NotMalspam
0.25474643433809896,0.083618164062 5,0.03906262271084418,0.0386962890625,TCP,0.013908205841446454,0.0335195530726257,0.0,0.0,NotMalspam
0.3318482111980785,0.07705688476562 5,0.7525635365751953,0.0167999267578125,TCP,0.012517385257301807,0.0223463687150838,0.0,0.0,NotMalspam
0.7525636860524685,0.861114501953125,0.25473590294242276,0.001220703125,TCP,0.012517385257301807,0.2513966480446927 5,0.0,0.0,NotMalspam
0.03906262271084418,0.038543701171875,0.2547413155563970 6,0.0067596435546875,TCP,0.012517385257301807,0.24581005586592178,0.0,0.0,NotMalspam
0.25473590294242276,0.001220703125,0.7525636860524685,0.8610992431640625,TCP,0.35095039406583217,0.3854595546031989,0.3427230046948357,0.0,NotMalspam
0.03906262271084418,0.038803100859375,0.2547524404373840 4,0.0836181640625,TCP,0.014371812702828002,0.24581005586592178,0.0,0.0,NotMalspam
0.25473590294242276,0.001220703125,0.7525636860524685,0.8566284179687 5,TCP,0.35095039406583217,0.3854595546031989,0.3427230046948357,0.0,NotMalspam
0.7525636860524685,0.87347412109375,0.797102590509947,0.1288299560546875,TCP,0.012517385257301807,0.2513966480446927 5,0.0,0.0,NotMalspam
0.7525636860524685,0.8734588623046875,0.797102590509947,0.0067596435546875,TCP,0.012517385257301807,0.2506007499808678 4,0.0,0.0,NotMalspam
0.7606742502564272,0.0067596435546875,0.03923209897224607 5,0.7511749267578125,TCP,0.35095039406583217,0.24581005586592178,0.3427230046948357,0.0,Malspam
0.28147632309270004,0.0067596435546875,0.7525636860524685,0.7579498291015625,TCP,0.031061659712563746,0.00383408586515650 12,0.018779342723004695,0.0,NotMalspam
0.7525636860524685,0.894287109375,0.8137653176239146,0.001220703125,TCP,0.012517385257301807,0.2513966480446927 5,0.0,0.0,NotMalspam
0.7525636860524685,0.8593597412109375,0.25473591970622 91,0.001220703125,TCP,0.012517385257301807,0.2513966480446927 5,0.0,0.0,NotMalspam
0.03923209897224607 5,0.7511749267578125,0.7606742502564272,0.0067596435546875,TCP,0.013908205841446454,0.2343766740644371 3,0.0,0.0,Malspam
0.6721343401987419,0.162322998046875,0.5096020932564517,0.0067596435546875,TCP,0.012517385257301807,1.0,0.0,0.0,NotMalspam
0.7525636860524685,0.87347412109375,0.797102590509947,0.0067596435546875,TCP,0.012517385257301807,0.2513966480446927 5,0.0,0.0,NotMalspam
0.7525636860524685,0.8593597412109375,0.25473591970622 91,0.001220703125,TCP,0.012517385257301807,0.2488865079972449 6,0.0,0.0,NotMalspam
0.8093014102916469,0.001220703125,0.03906262271084418,0.0385894775390625,TCP,0.2554473806212332,0.0335195530726257,0.2460093896713615,0.0,NotMalspam
0.25473593740135 8,0.001220703125,0.7525636860524685,0.8565216064453125,TCP,0.35095039406583217,0.3854595546031989,0.3427230046948357,0.0,NotMalspam
0.6721343401987419,0.162841796875,0.8282568046888934,0.83770751953125,TCP,0.015299026425591099,0.0313461391291038 5,0.0,0.0,NotMalspam
0.25473591970622 91,0.001220703125,0.7525636860524685,0.8546752929687 5,TCP,0.35095039406583217,0.2513813423126961,0.3427230046948357,0.0,NotMalspam
0.26117832219721243,0.001220703125,0.7525636860524685,0.855224609375,TCP,0.35095039406583217,0.0492614984311624 7,0.3427230046948357,0.0,NotMalspam
0.7525636860524685,0.8728485107421875,0.8132675603994326,0.0836181640625,TCP,0.012517385257301807,0.0622560648963036 6,0.0,0.0,NotMalspam
```

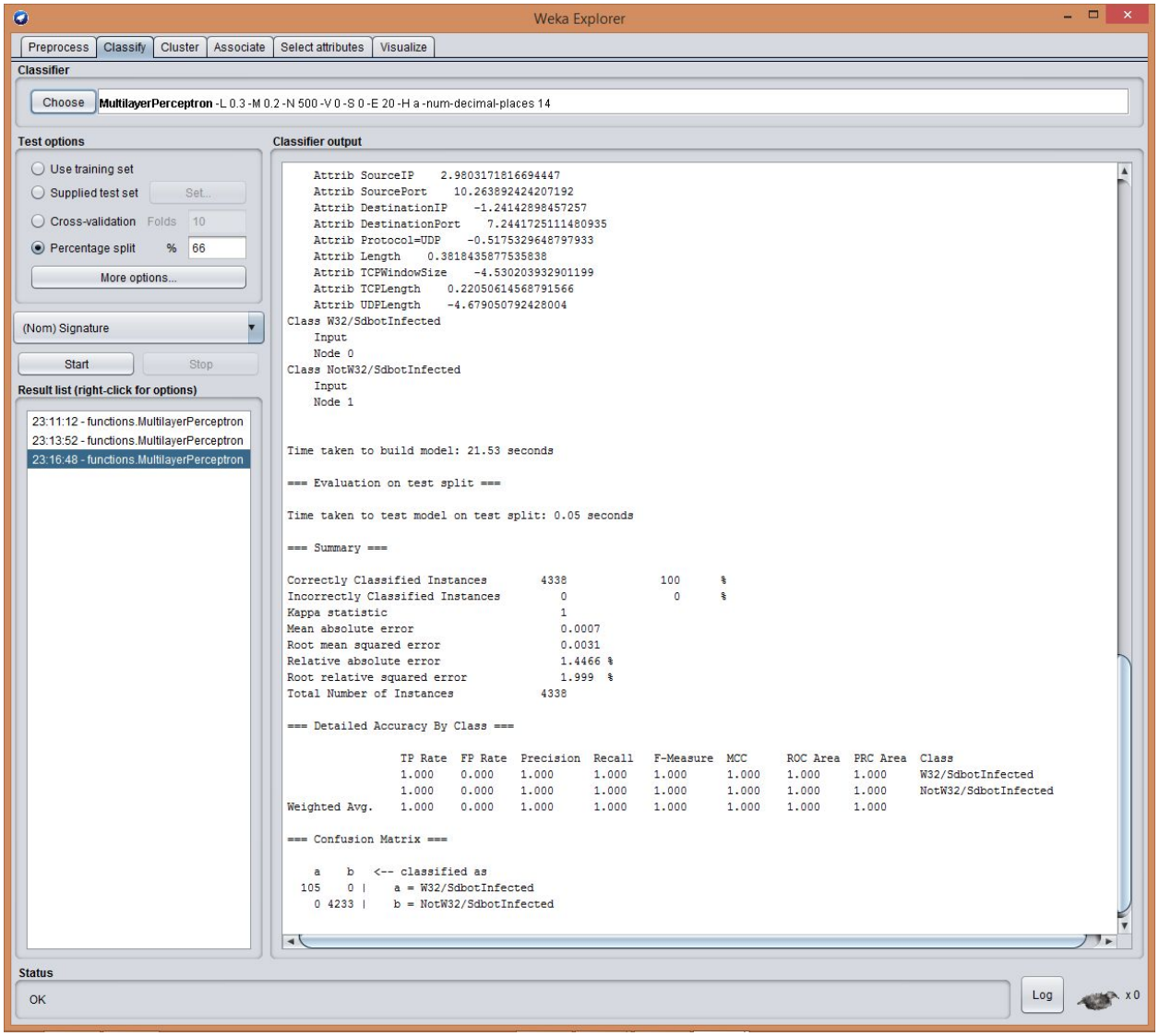
The arff files were then fed into Weka.

Weka Preprocess section:



Then the MLP algorithm was run with num-decimal-places set to 14 again to be able to utilize normalized IP addresses. The algorithm was run multiple times for each arff file with different values for the learning rate and number of epochs trained for.

Weka MLP Output Example:



Testing

Testing was done in several parts. Each attack was tested individually in a data set containing all the other attacks and the normal traffic, only the first attack is labeled accurately in these sets and the rest of the data is labeled not that attack. Then there are fully labeled data sets for both a signature based system and an anomaly based system. The signature based set has all attacks labeled as what attack they are and the normal traffic labeled as normal. The anomaly based set has all attacks labeled as abnormal and all normal traffic labeled as normal

Each data set was tested for two different values for the learning rate and the number of epochs trained for. The learning rate values were set to 0.3 and 0.8. The number epochs were either 500 and 1000 or 500 and 50 depending on whether the data set had perfect classification

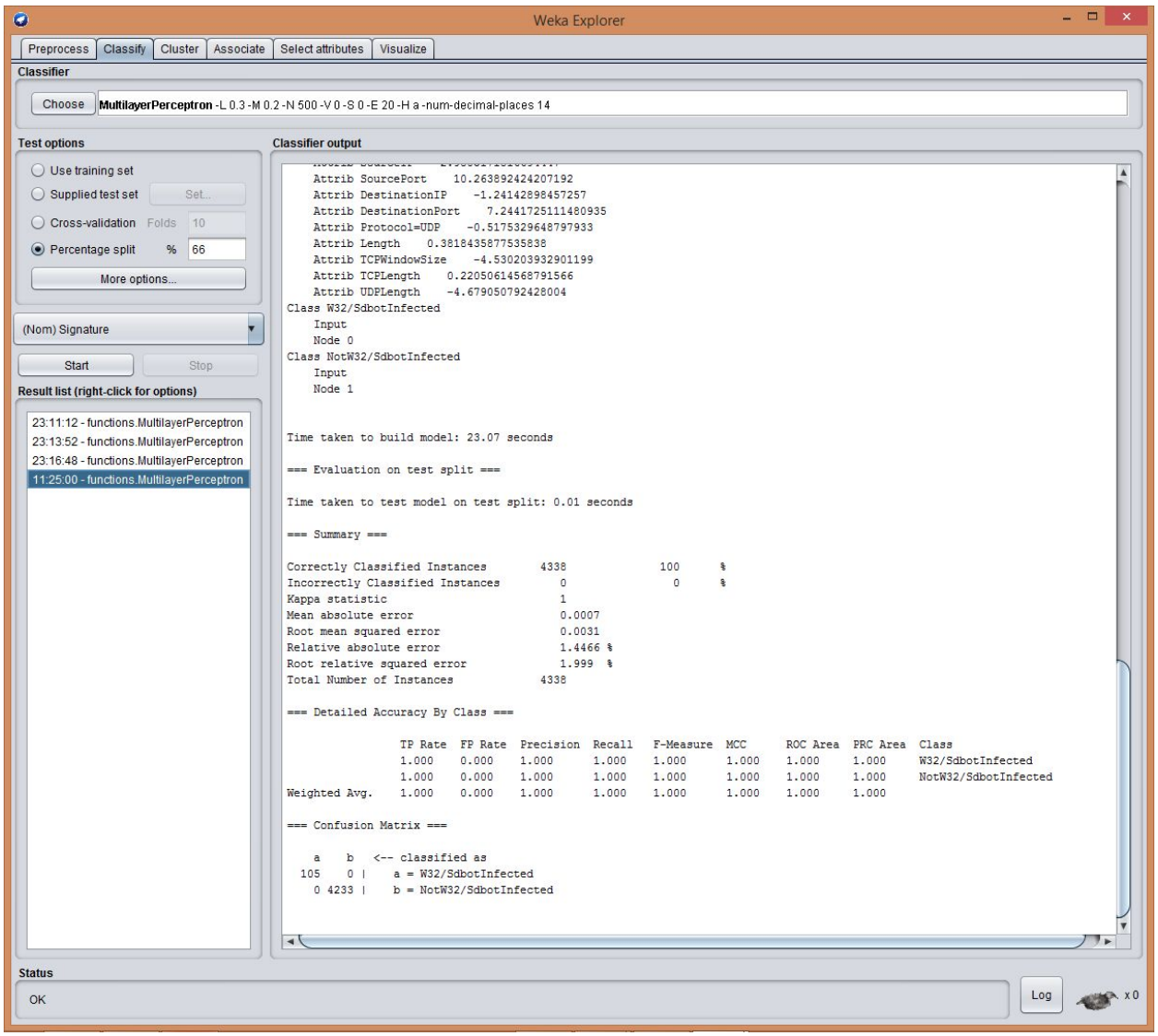
accuracy already. This was done to improve accuracy for sets that need it and to improve speed for sets that didn't.

The same equations as from part one were used in the results section to evaluate the data sets and neural networks classification efficacy. These equations were taken from the class slides and from https://en.wikipedia.org/wiki/Precision_and_recall.

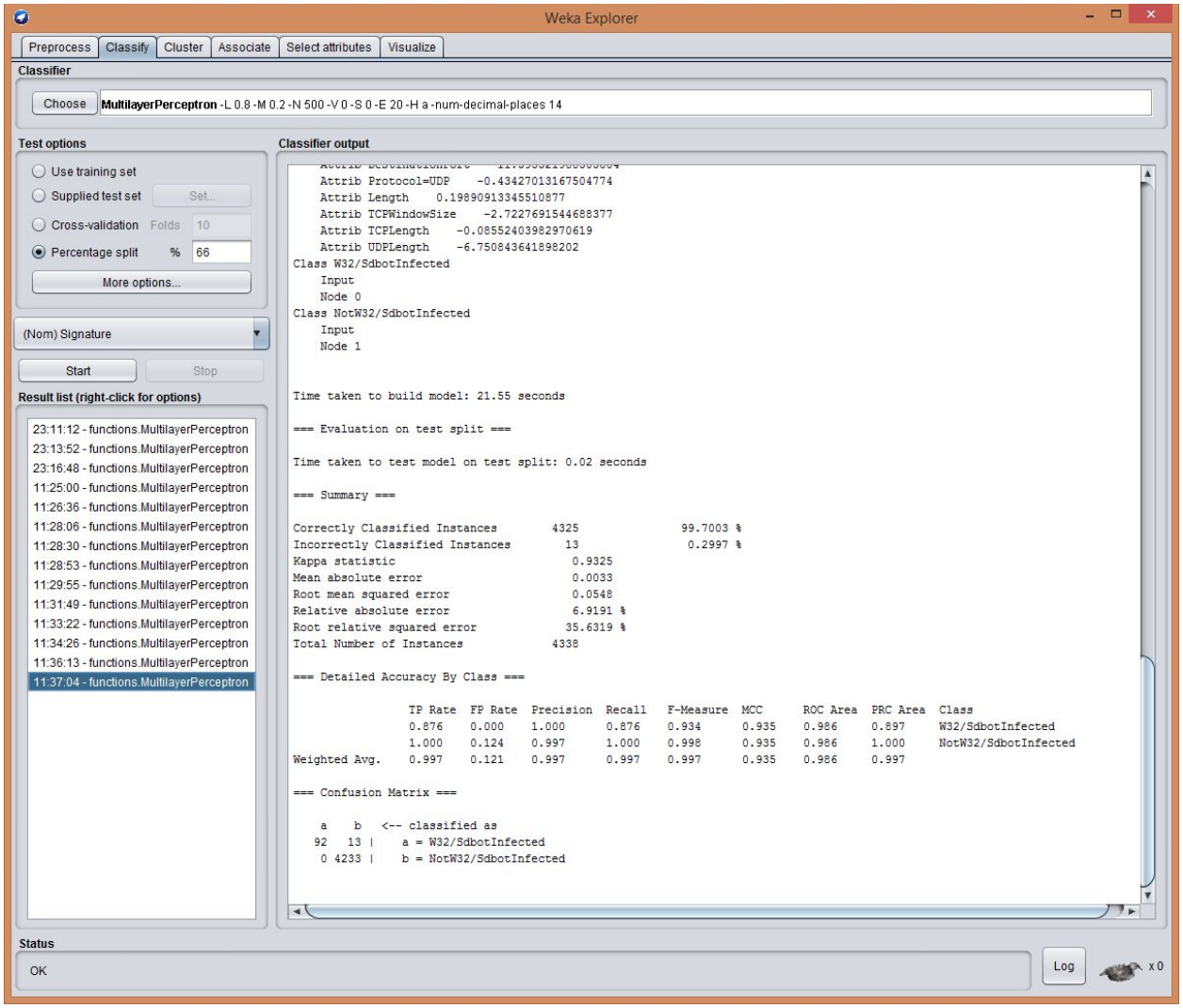
Results

W32/Sdbot

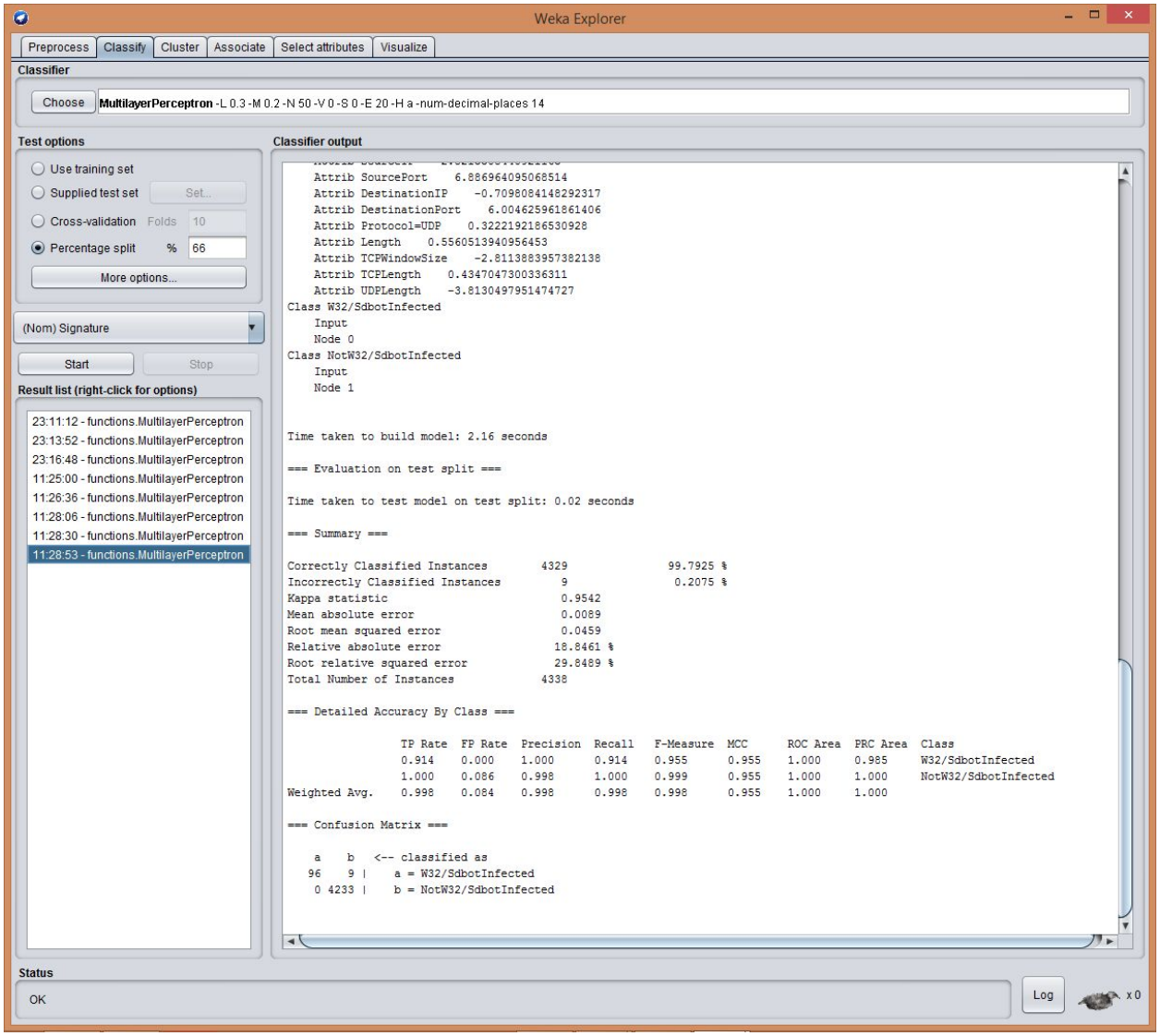
Weka run with the W32/Sdbot Infected labeled set with learning rate 0.3 and trained for 500 epochs:



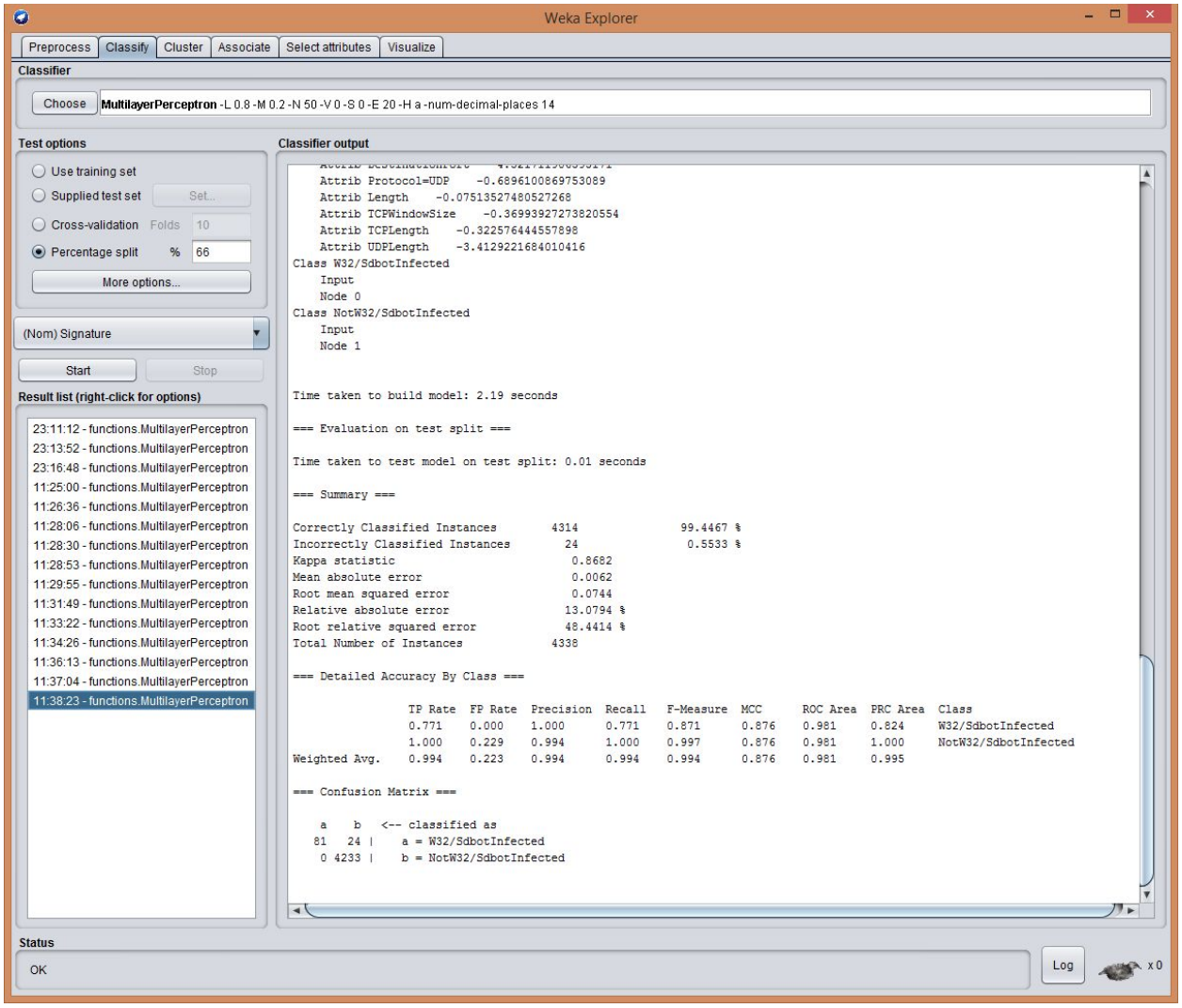
Weka run with the W32/Sdbot Infected labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the W32/Sdbot Infected labeled set with learning rate 0.3 and trained for 50 epochs:



Weka run with the W32/Sdbot Infected labeled set with learning rate 0.8 and trained for 50 epochs:



The best run was with the learning rate at 0.3 and the number of epochs at 500. Increasing the learning rate to 0.8 decreased the number of correctly identified attacks. It took a very small number of epochs to decrease classification efficacy, somewhere between 50 and 100 epochs it became less effective.

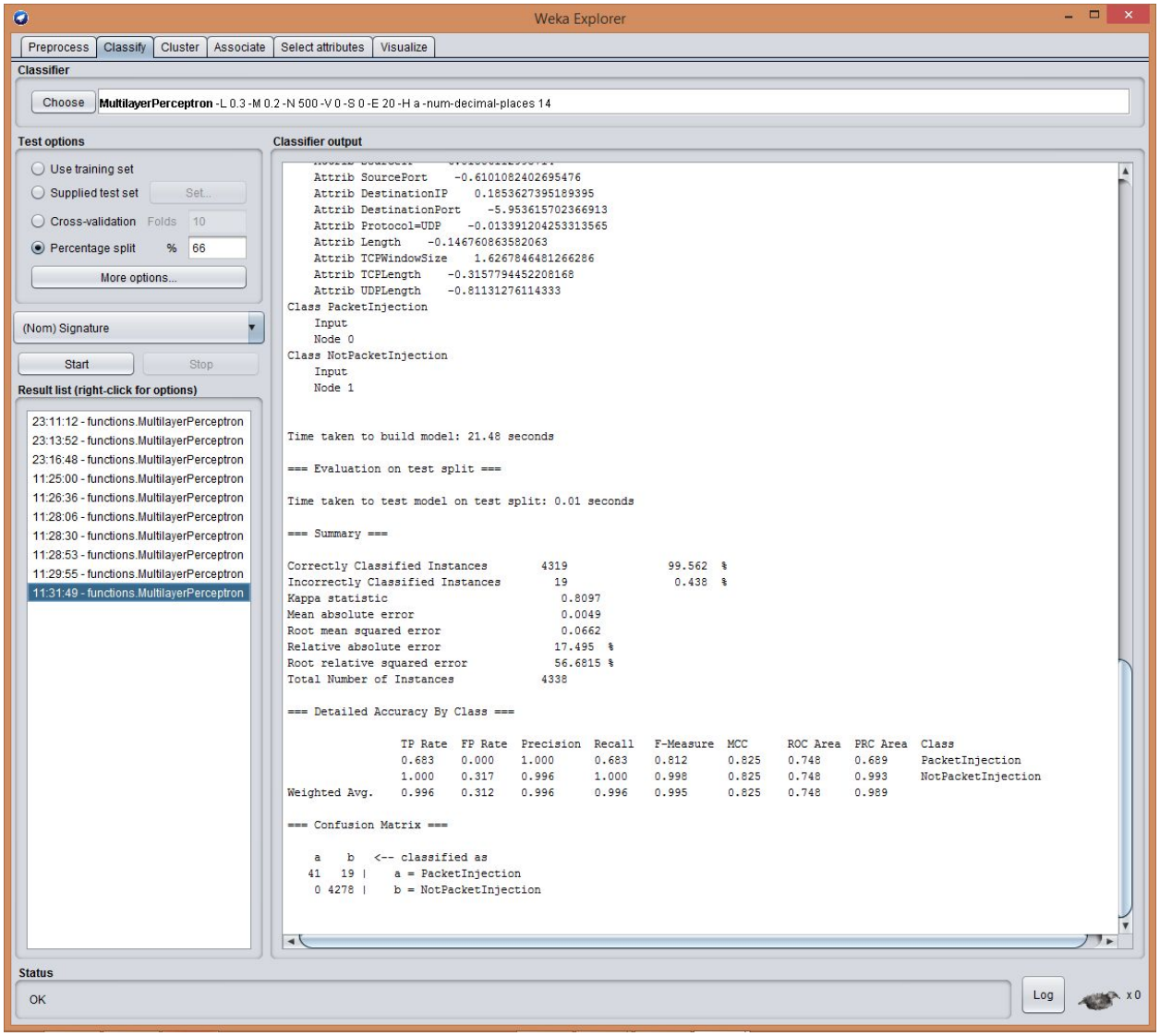
For the best run, learning rate at 0.3 and number of epochs at 500:

$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

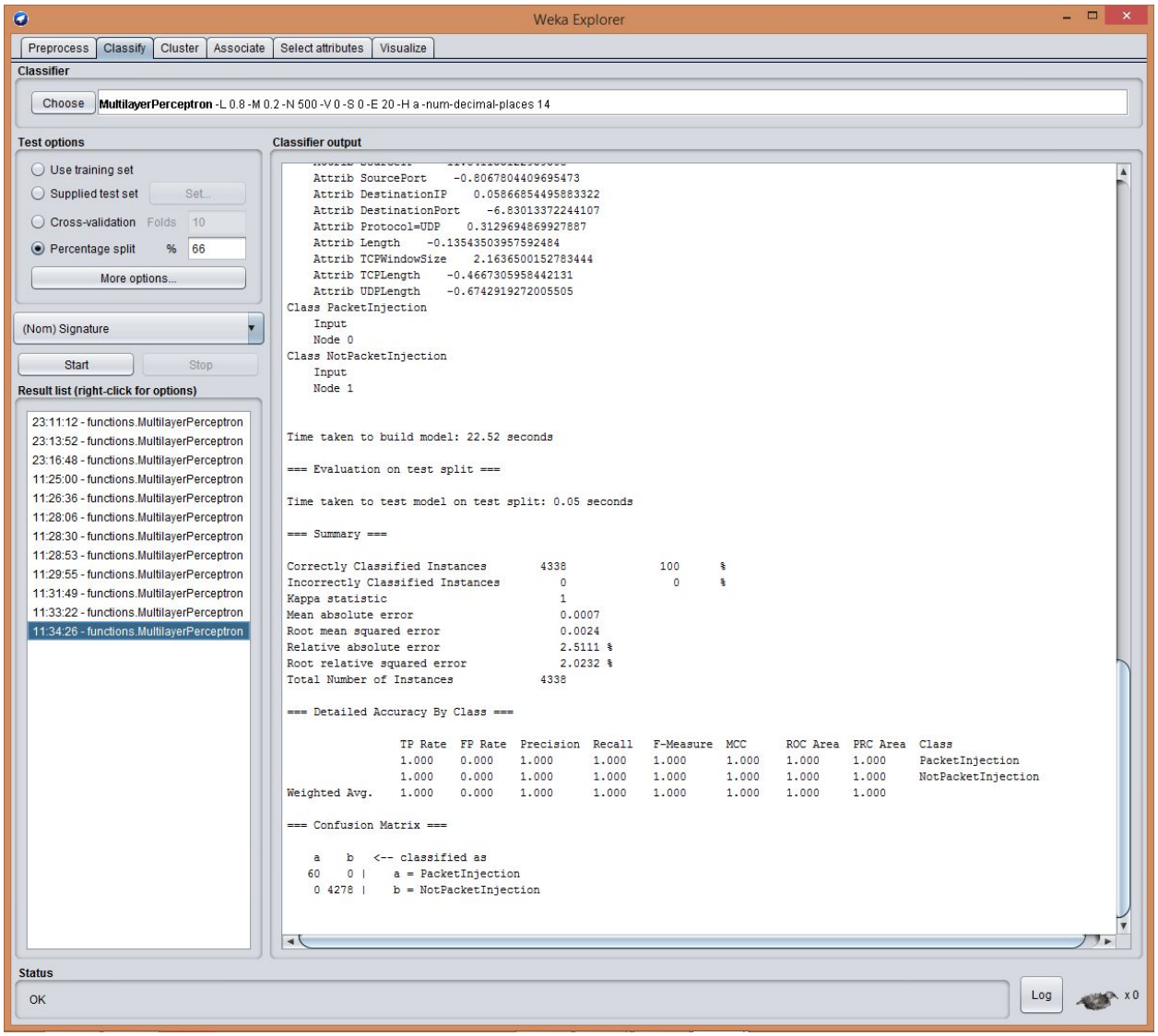
$$precision = \frac{105}{105+0} = 1$$
$$recall = \frac{105}{105+0} = 1$$
$$accuracy = \frac{105+4233}{105+4233+0+0} = 1$$

Packet Injection

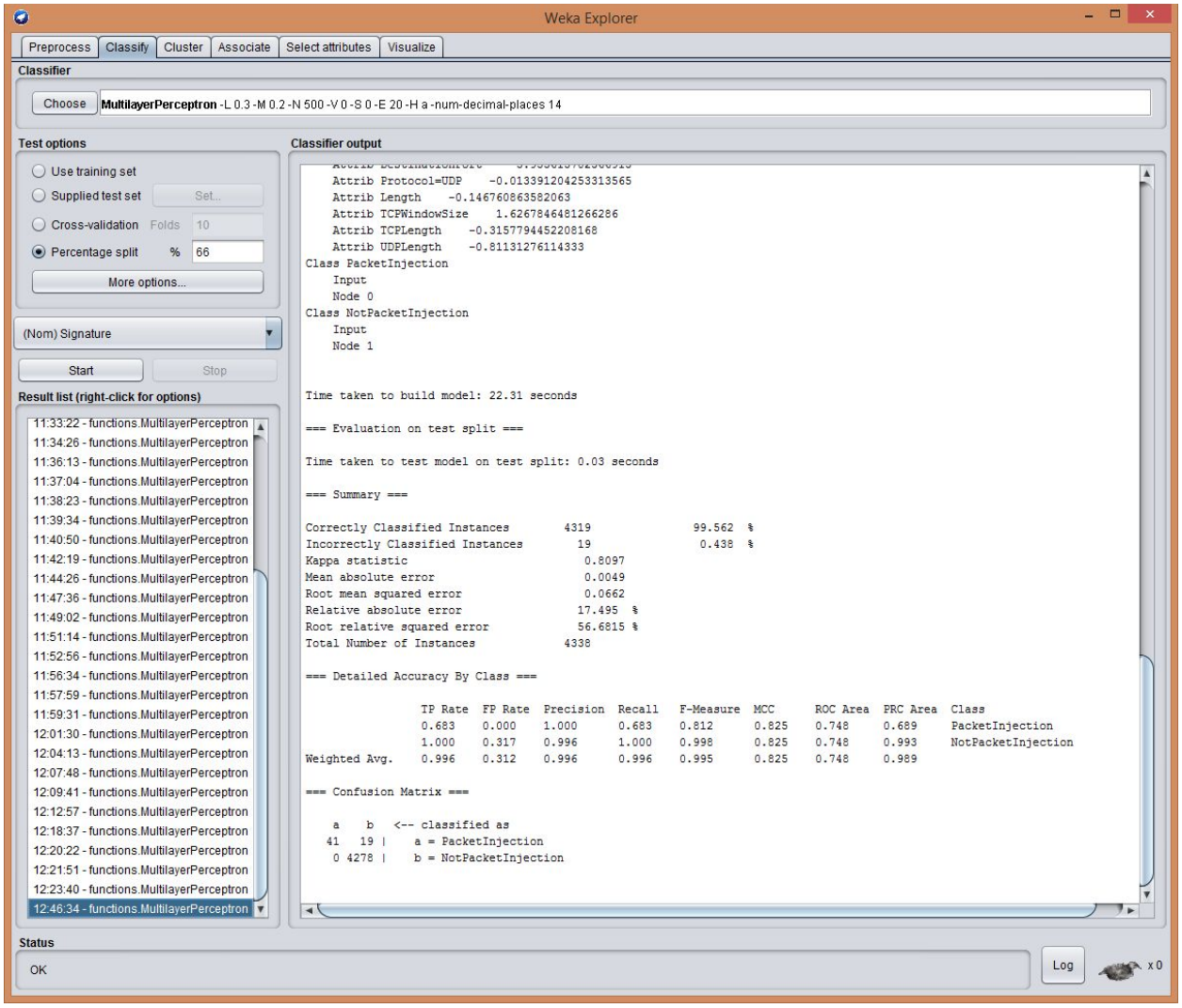
Weka run with the Packet Injection labeled set with learning rate 0.3 and trained for 500 epochs:



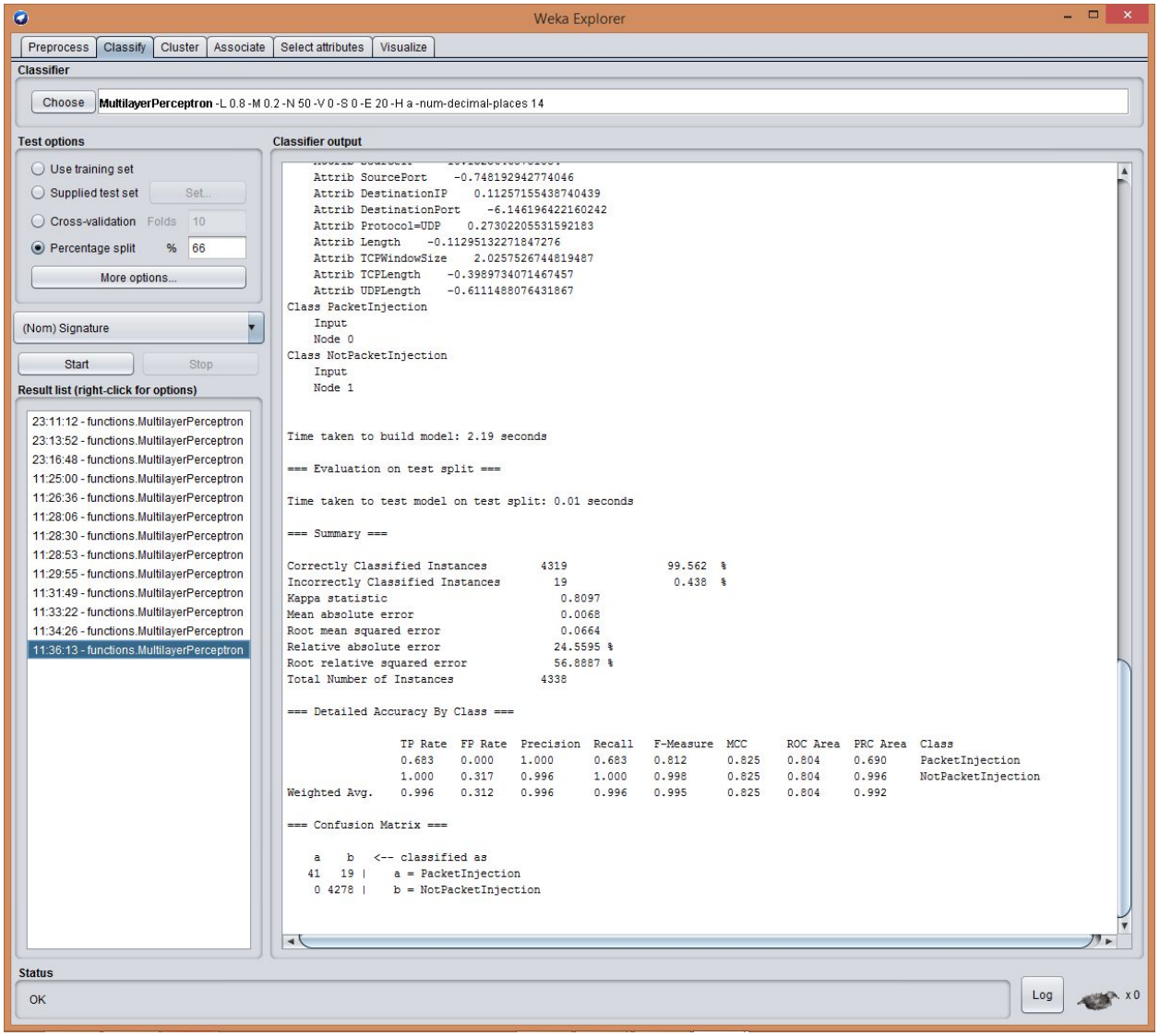
Weka run with the Packet Injection labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the Packet Injection labeled set with learning rate 0.3 and trained for 50 epochs:



Weka run with the Packet Injection labeled set with learning rate 0.8 and trained for 50 epochs:



The best run was with the learning rate at 0.8 and the number of epochs at 500. Increasing the learning rate increased classification efficacy from somewhat poor to perfect classification. Decreasing the number of epochs decreased efficacy much more quickly than the W32/Sdbot runs.

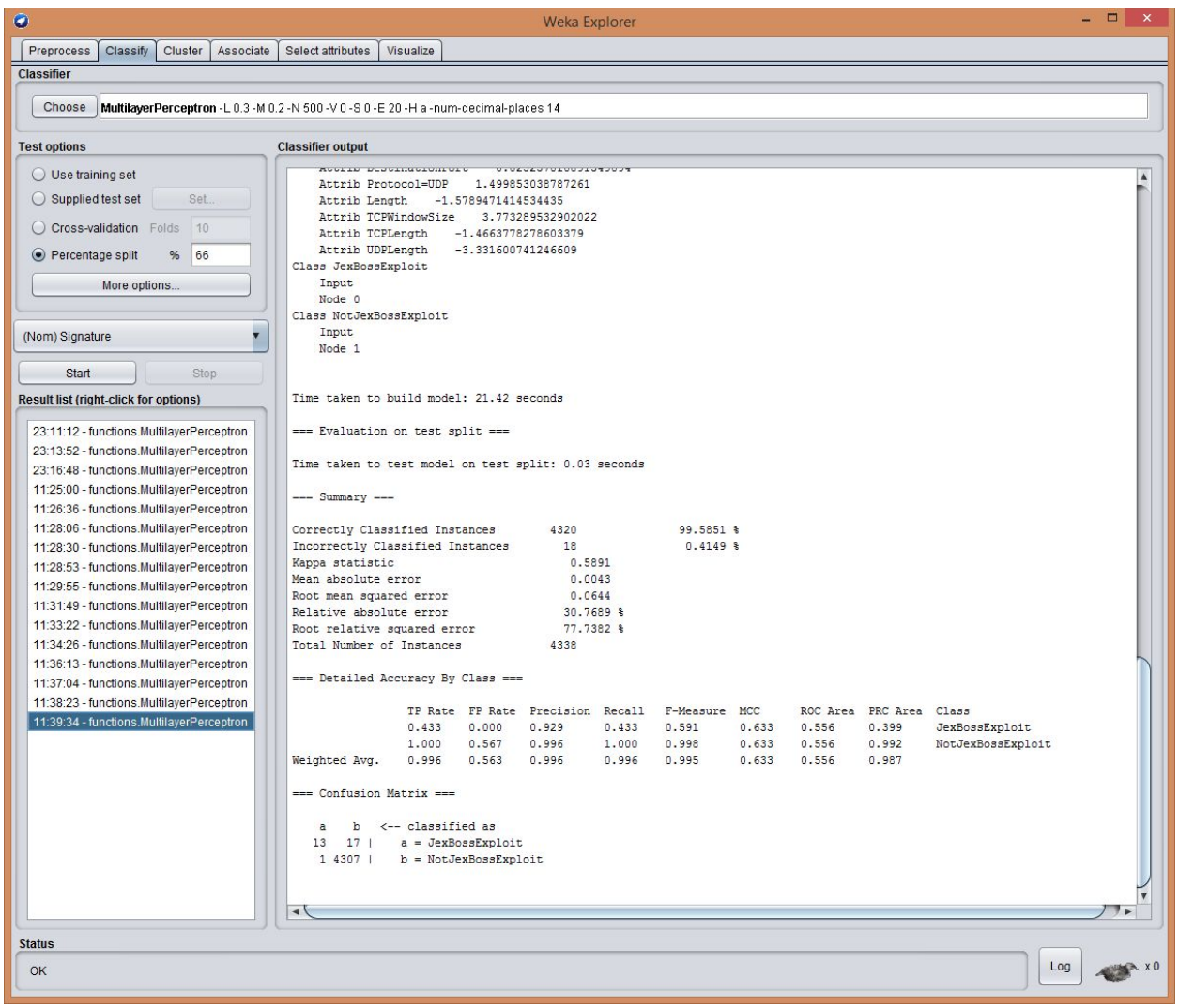
For the best run, learning rate at 0.8 and number of epochs at 500:

$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

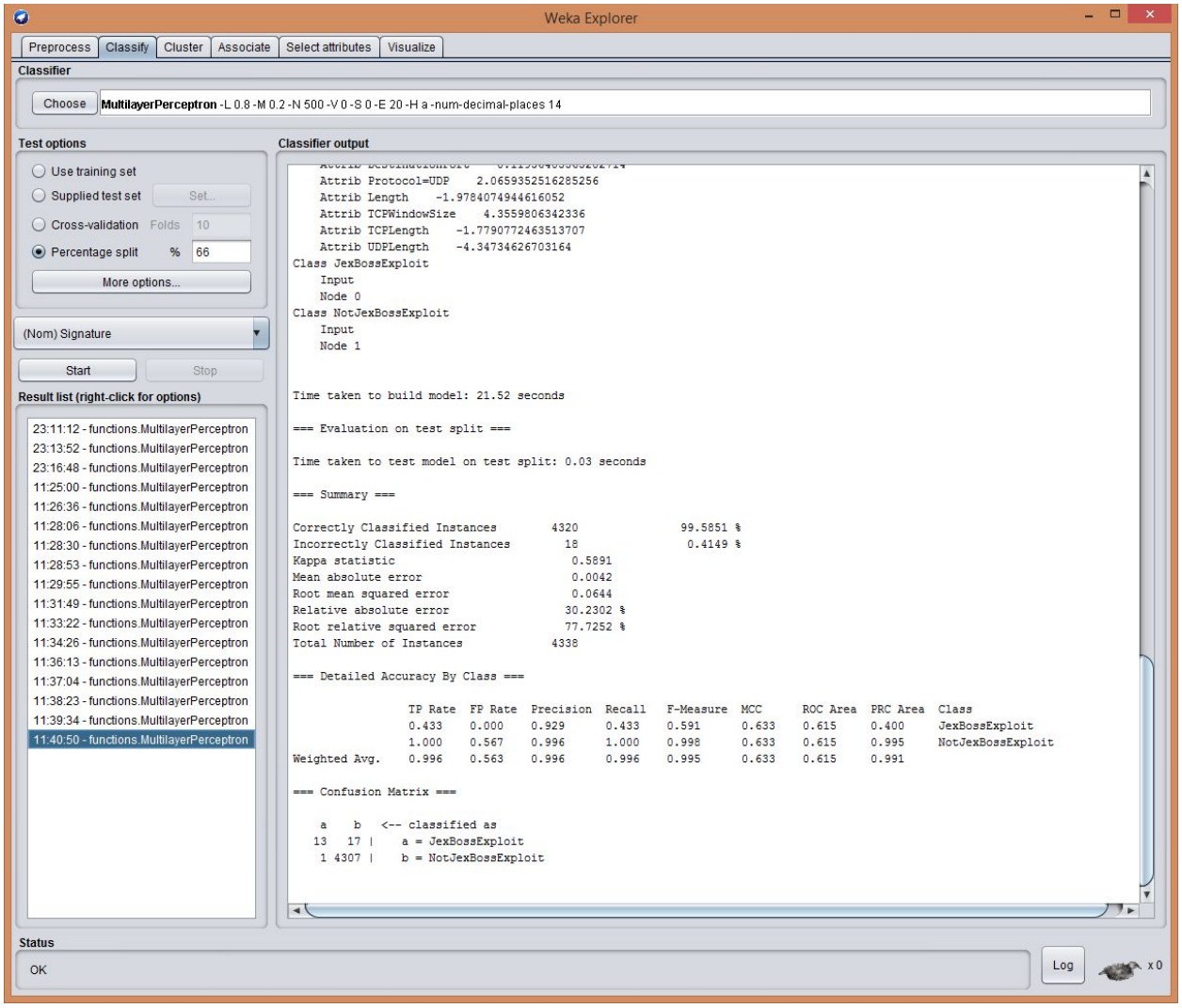
$$precision = \frac{60}{60+0} = 1$$
$$recall = \frac{60}{60+0} = 1$$
$$accuracy = \frac{60+4278}{60+4278+0+0} = 1$$

Jexboss Exploit

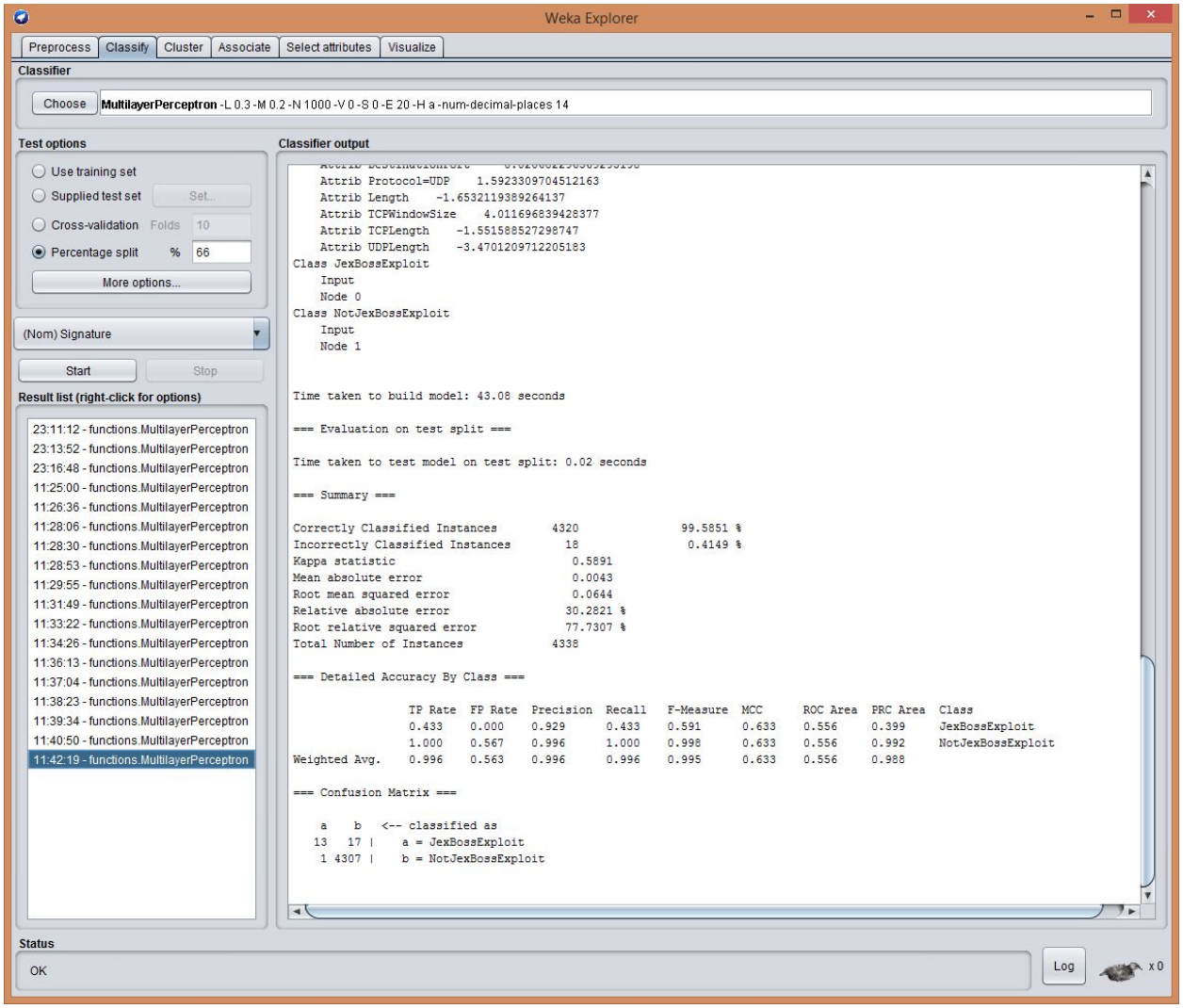
Weka run with the JexBoss Exploit labeled set with learning rate 0.3 and trained for 500 epochs:



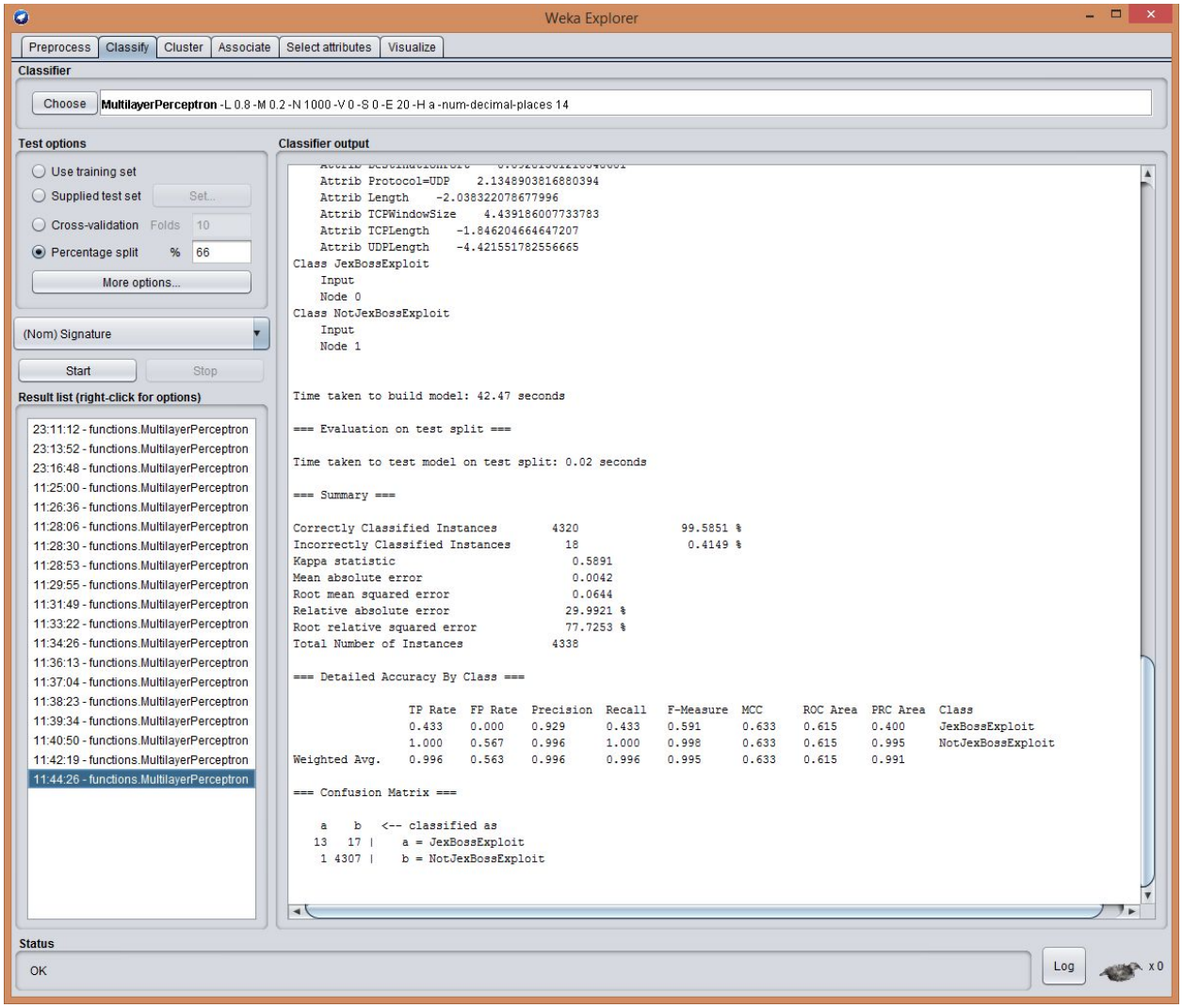
Weka run with the JexBoss Exploit labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the JexBoss Exploit labeled set with learning rate 0.3 and trained for 1000 epochs:



Weka run with the JexBoss Exploit labeled set with learning rate 0.8 and trained for 1000 epochs:



All of the runs were the same. Increasing the number of epochs did not increase efficacy and changing the learning rate had no effect. This is potentially a result of having a smaller sample size than the other attacks.

For the best run, learning rate at 0.3 and number of epochs at 500:

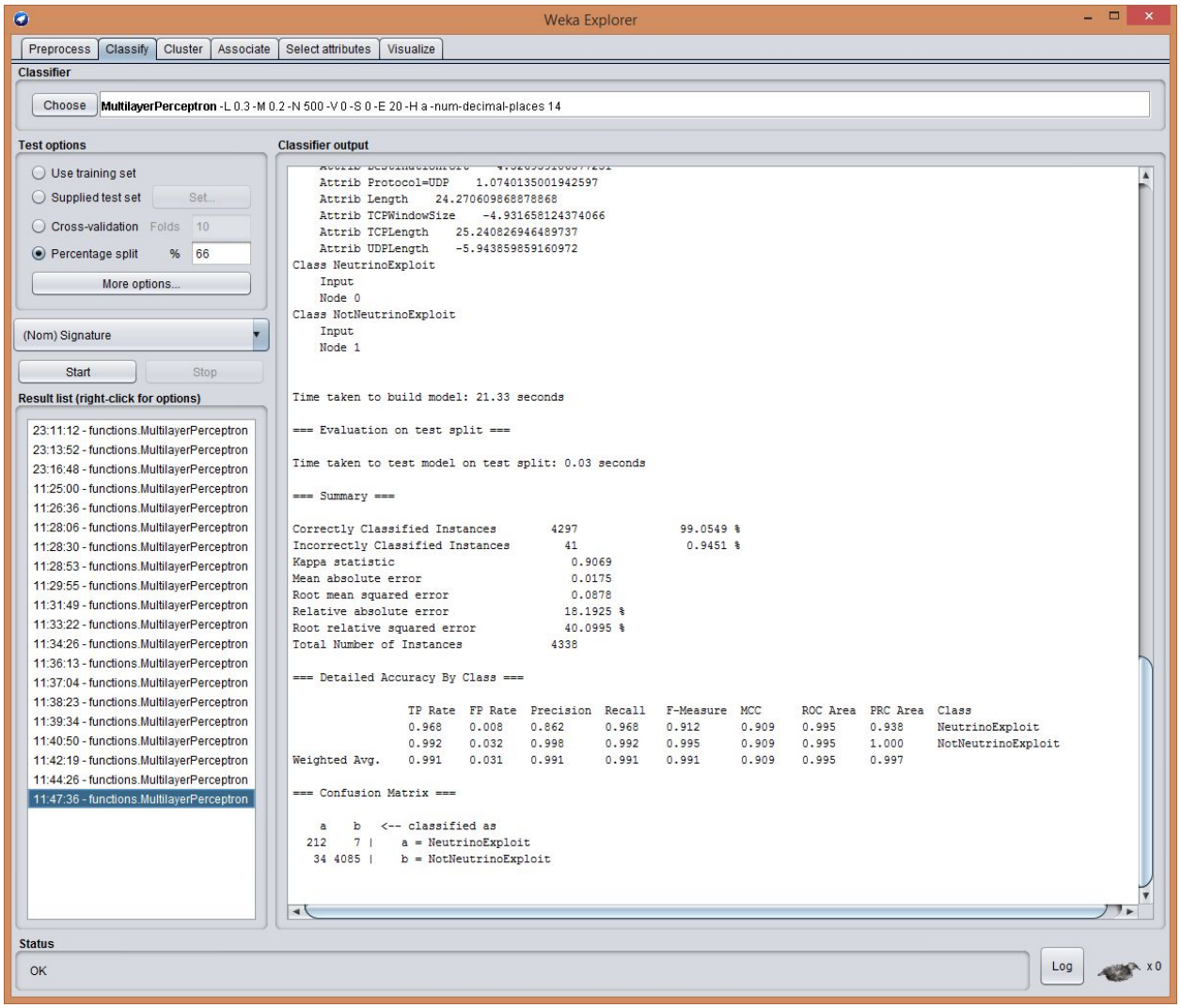
$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

$$precision = \frac{13}{13+1} = 0.929$$
$$recall = \frac{13}{13+17} = 0.433$$

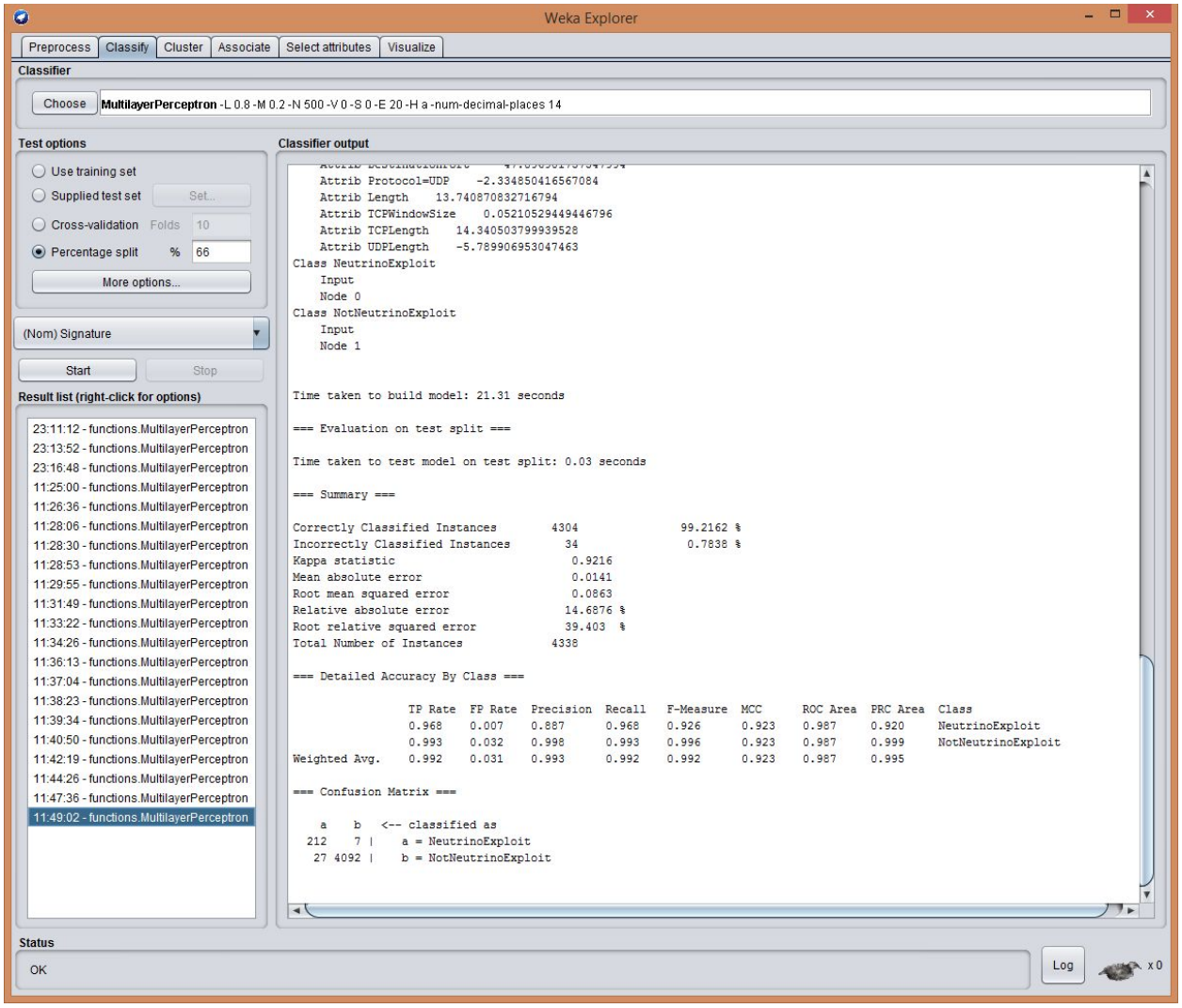
$$accuracy = \frac{13+4307}{13+4307+17+1} = 0.996$$

Neutrino Exploit

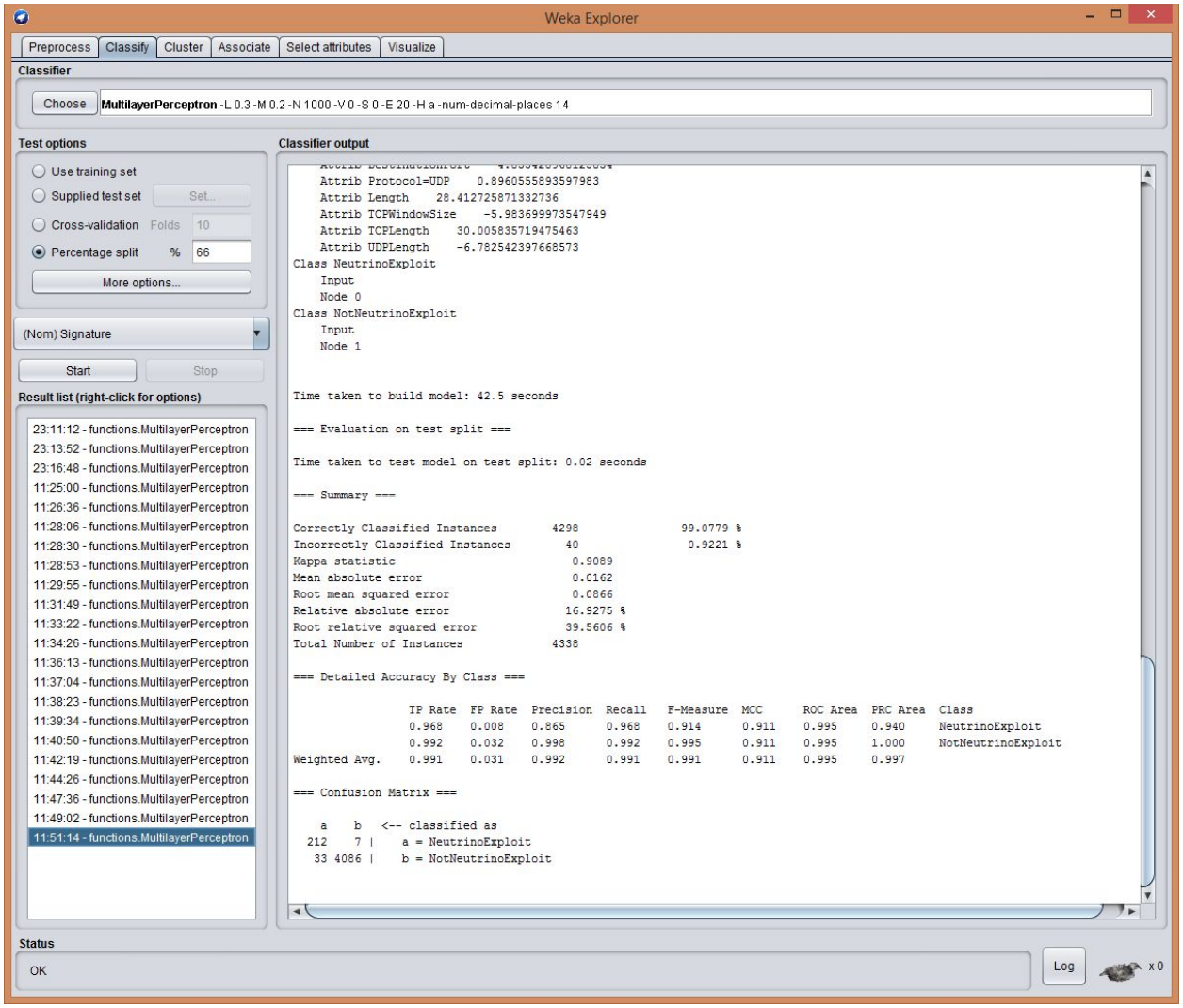
Weka run with the Neutrino Exploit labeled set with learning rate 0.3 and trained for 500 epochs:



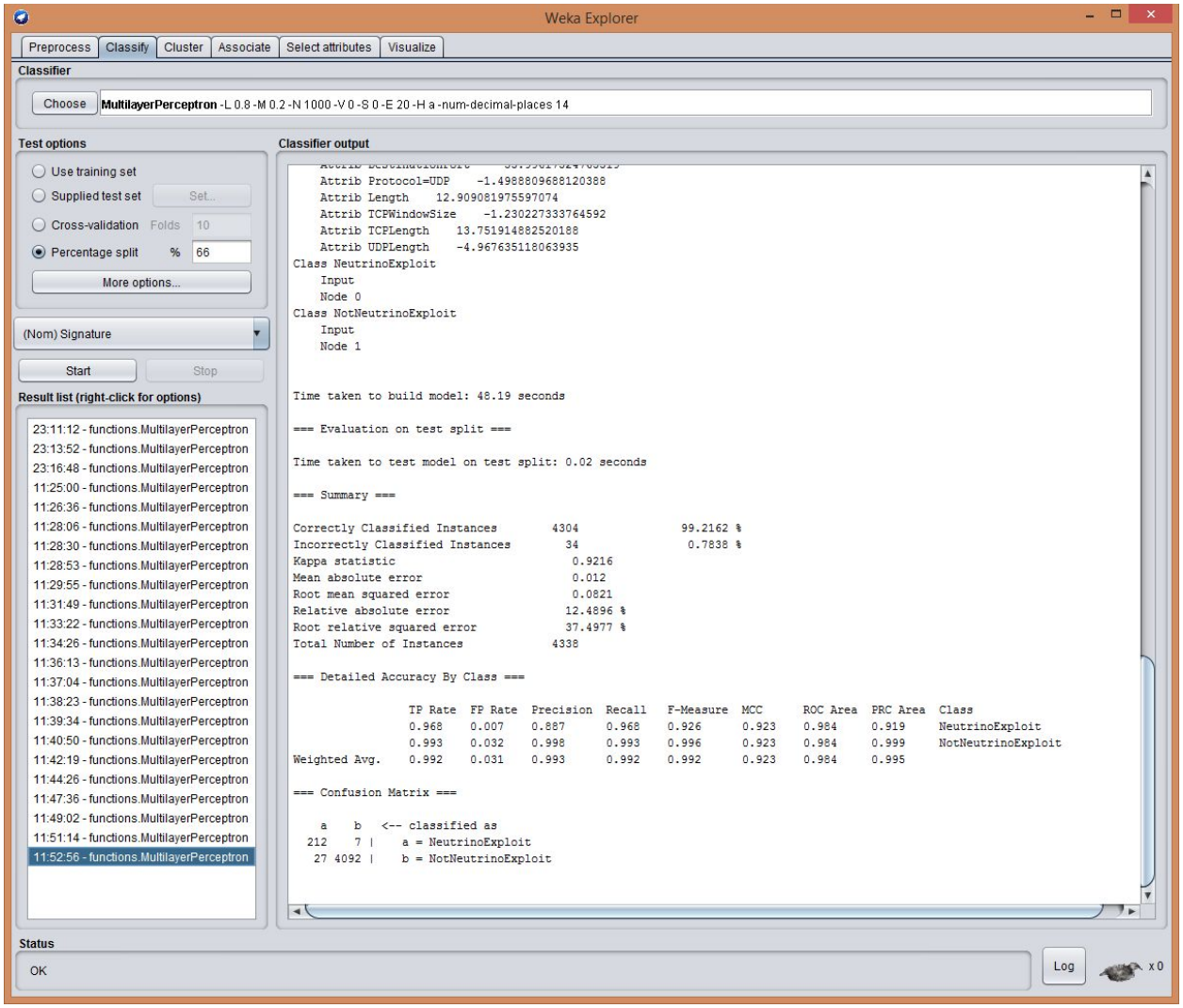
Weka run with the Neutrino Exploit labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the Neutrino Exploit labeled set with learning rate 0.3 and trained for 1000 epochs:



Weka run with the Neutrino Exploit labeled set with learning rate 0.8 and trained for 1000 epochs:



The best runs were with the learning rate at 0.8, increasing the number of epochs to 1000 did improve efficacy in that case but it did slightly improve efficacy when the learning rate was at 0.3.

For the best run, learning rate at 0.8 and number of epochs at 500:

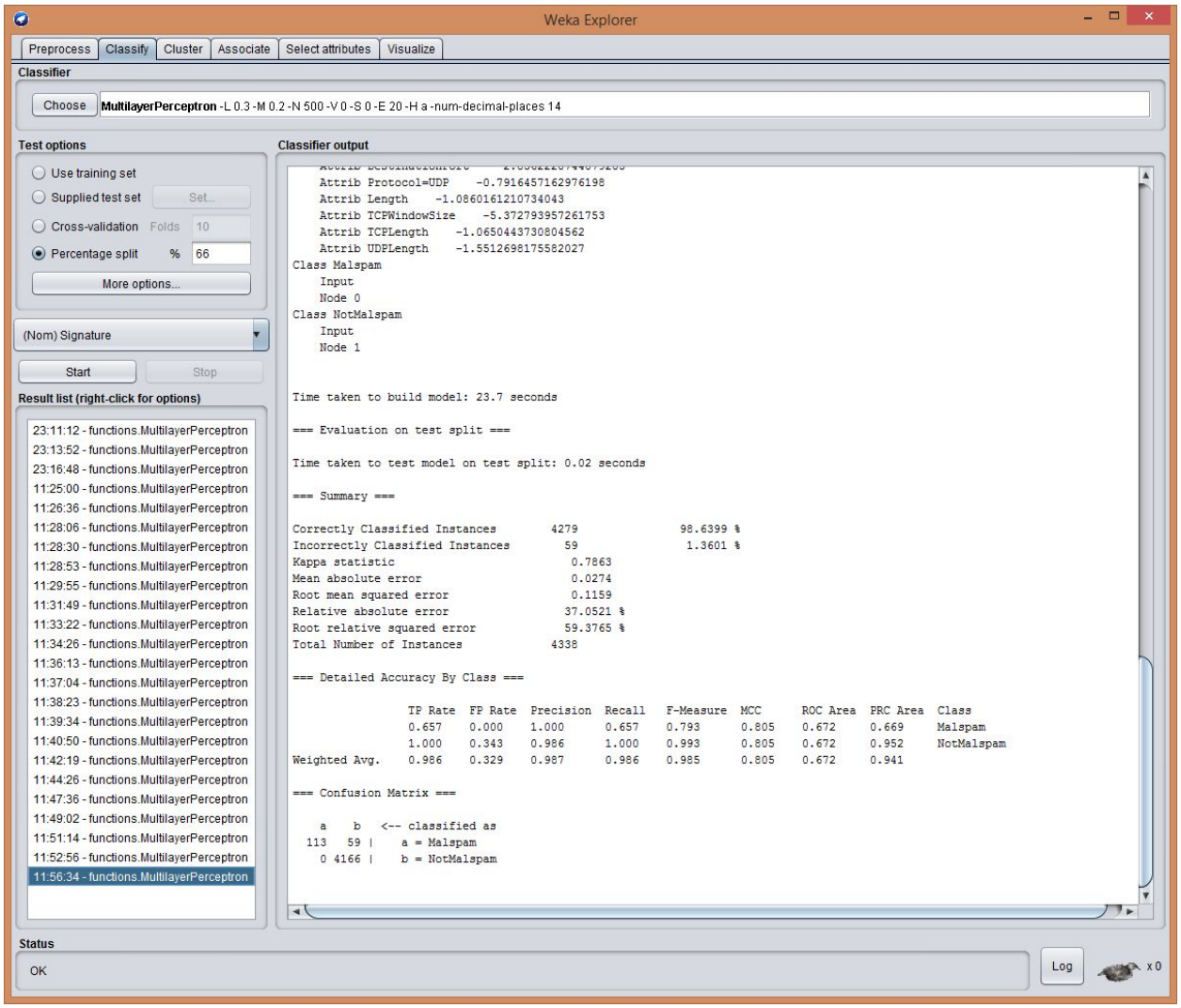
$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

$$precision = \frac{212}{212+27} = 0.89$$
$$recall = \frac{212}{212+7} = 0.968$$

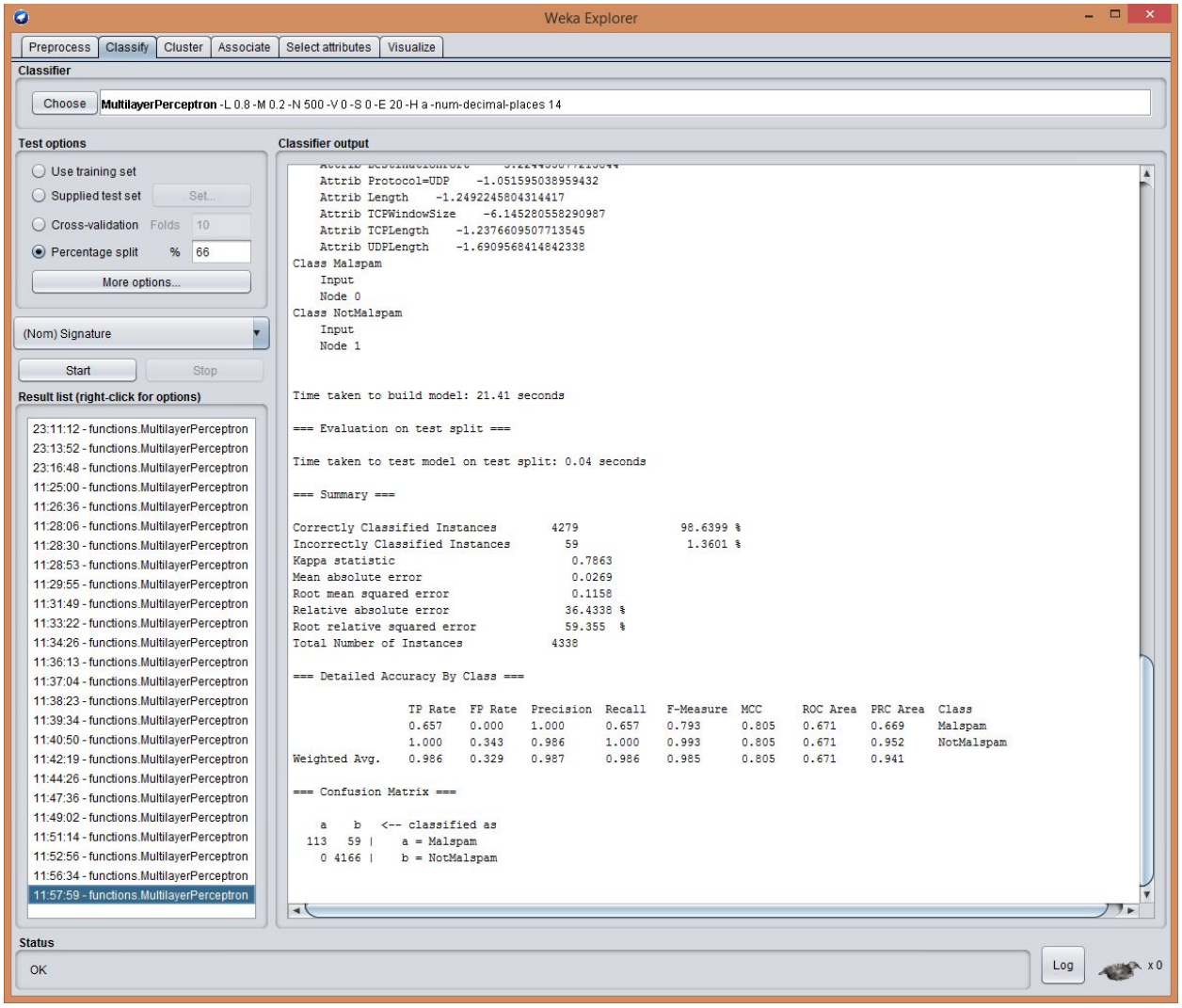
$$accuracy = \frac{212+4092}{212+4092+7+27} = 0.992$$

Malspam

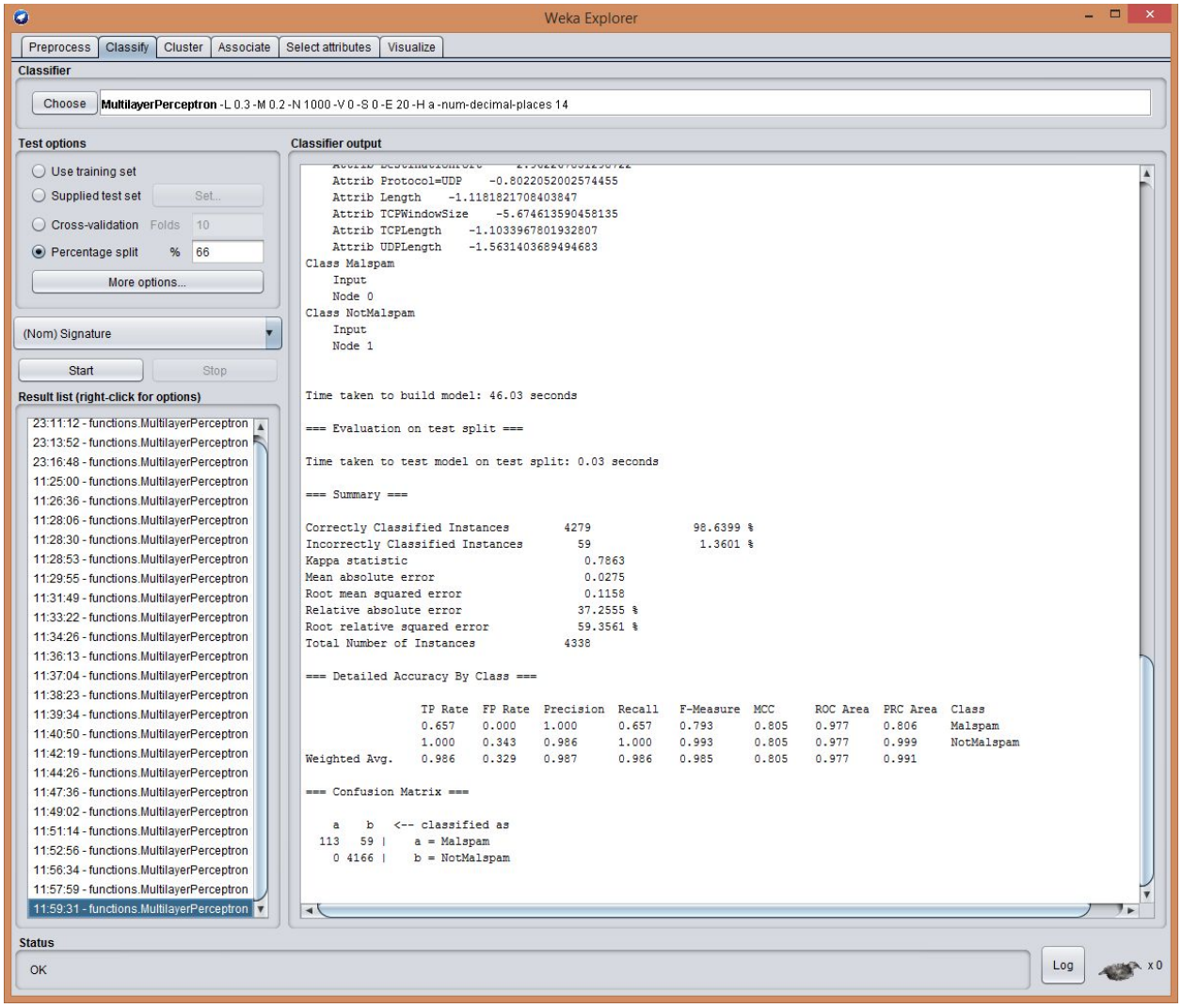
Weka run with the Malspam labeled set with learning rate 0.3 and trained for 500 epochs:



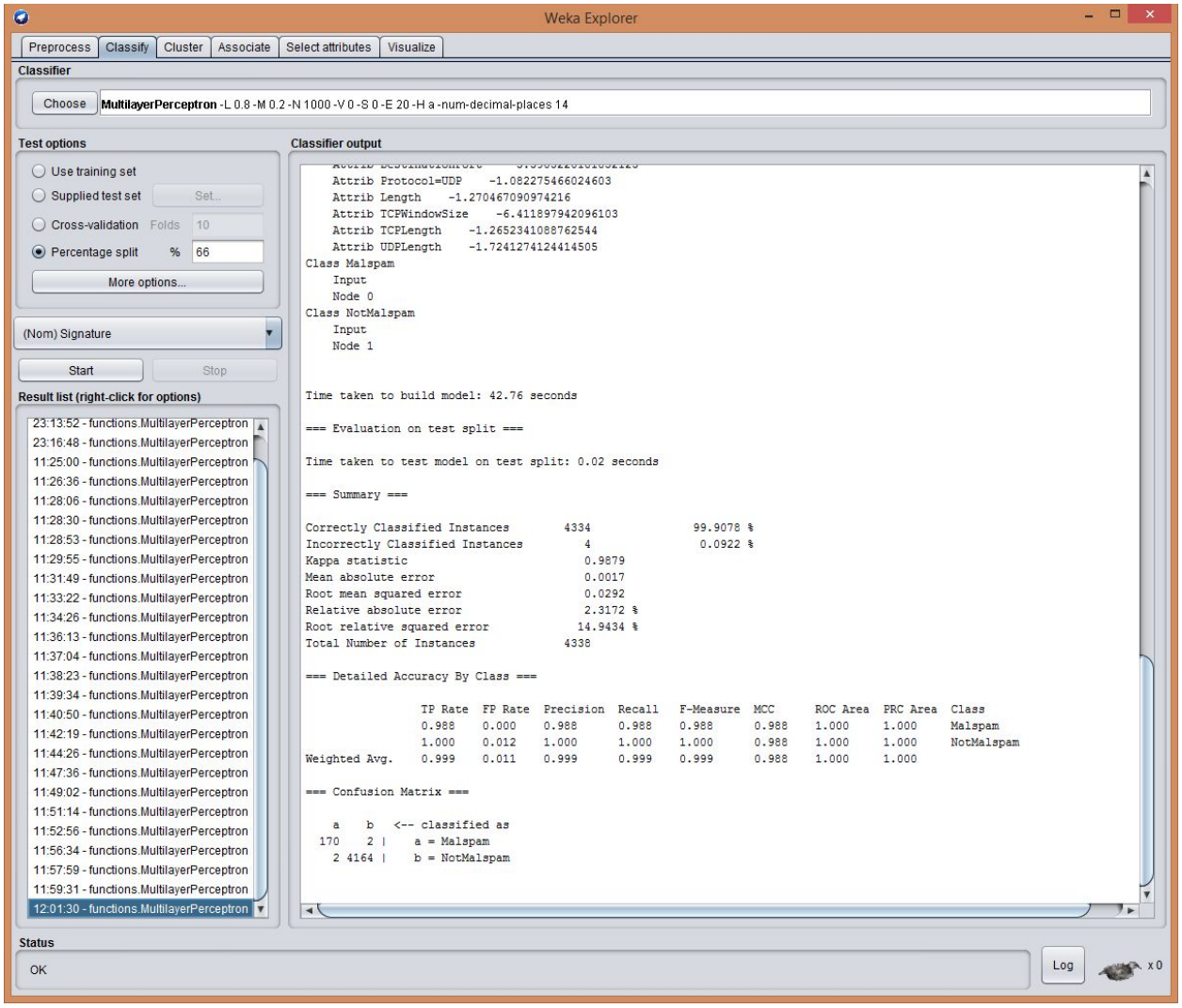
Weka run with the Malspam labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the Malspam labeled set with learning rate 0.3 and trained for 1000 epochs:



Weka run with the Malspam labeled set with learning rate 0.8 and trained for 1000 epochs:



The best run was with the learning rate at 0.8 and the number of epochs at 1000. Increasing the learning rate or the number of epochs did not improve efficacy but increasing both in the same run increased efficacy greatly.

For the best run, learning rate at 0.8 and number of epochs at 1000:

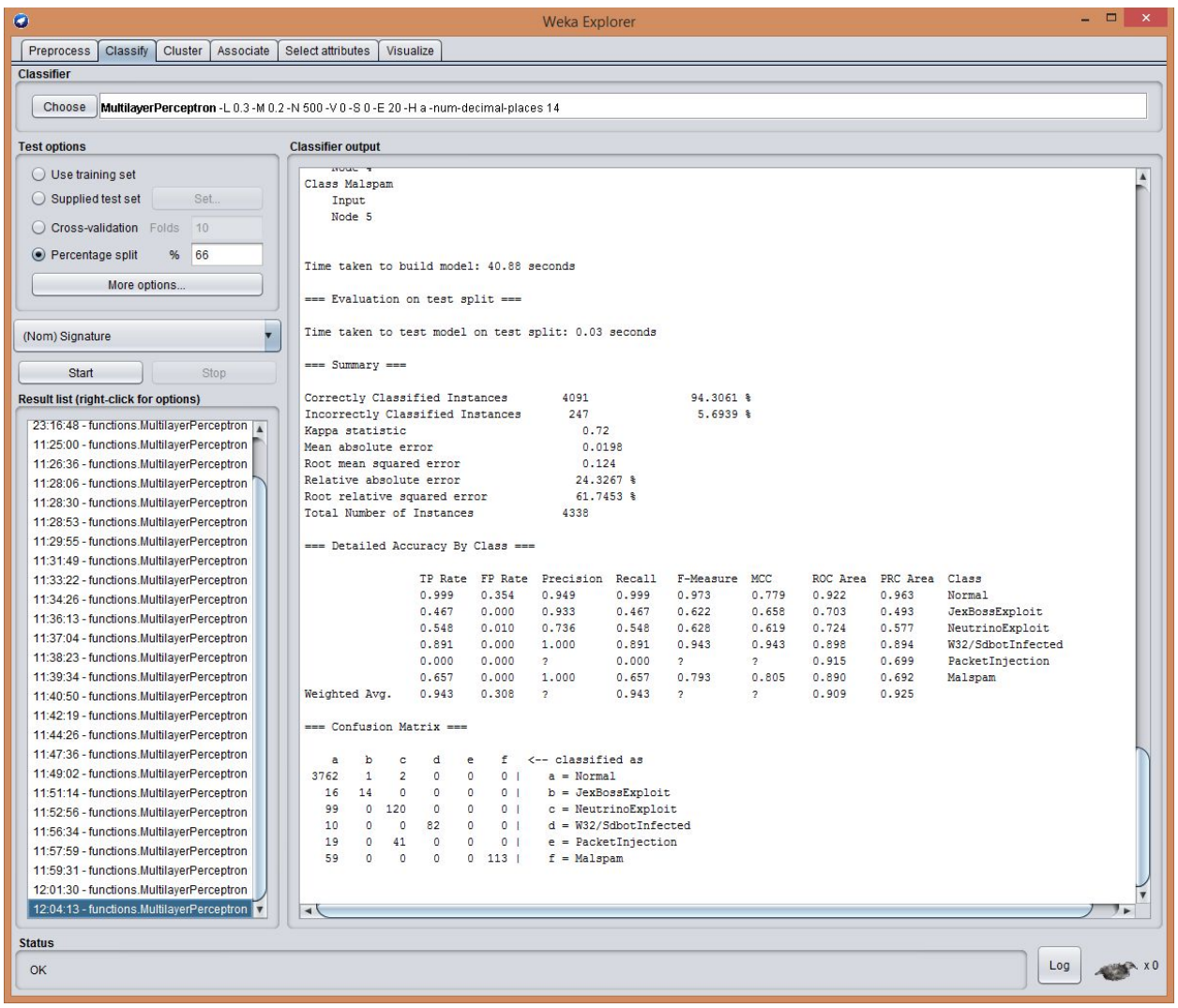
$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

$$precision = \frac{170}{170+2} = 0.988$$
$$recall = \frac{170}{170+2} = 0.988$$

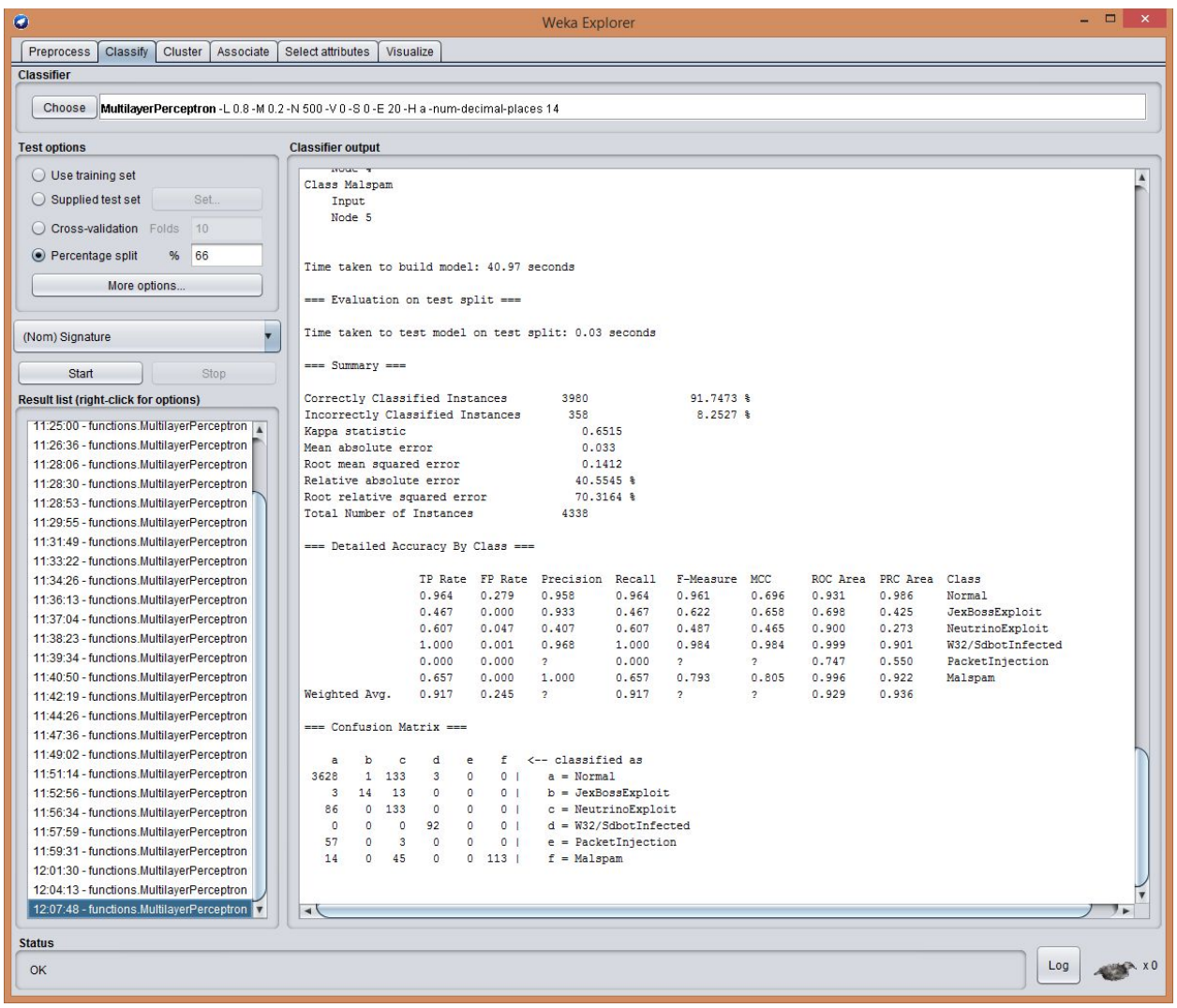
$$accuracy = \frac{170+4164}{170+4164+2+2} = 0.999$$

All Signatures

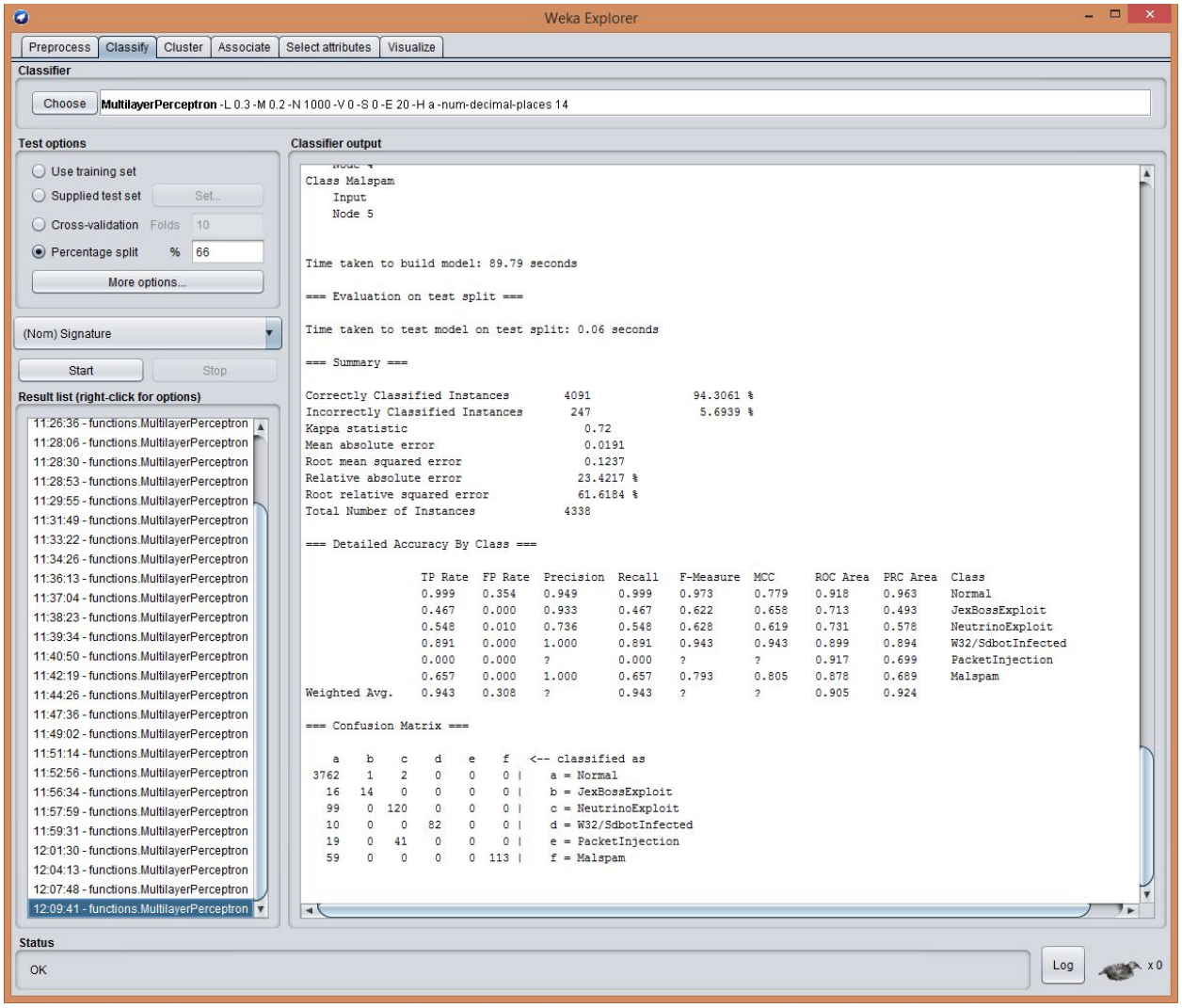
Weka run with the all Signatures labeled set with learning rate 0.3 and trained for 500 epochs:



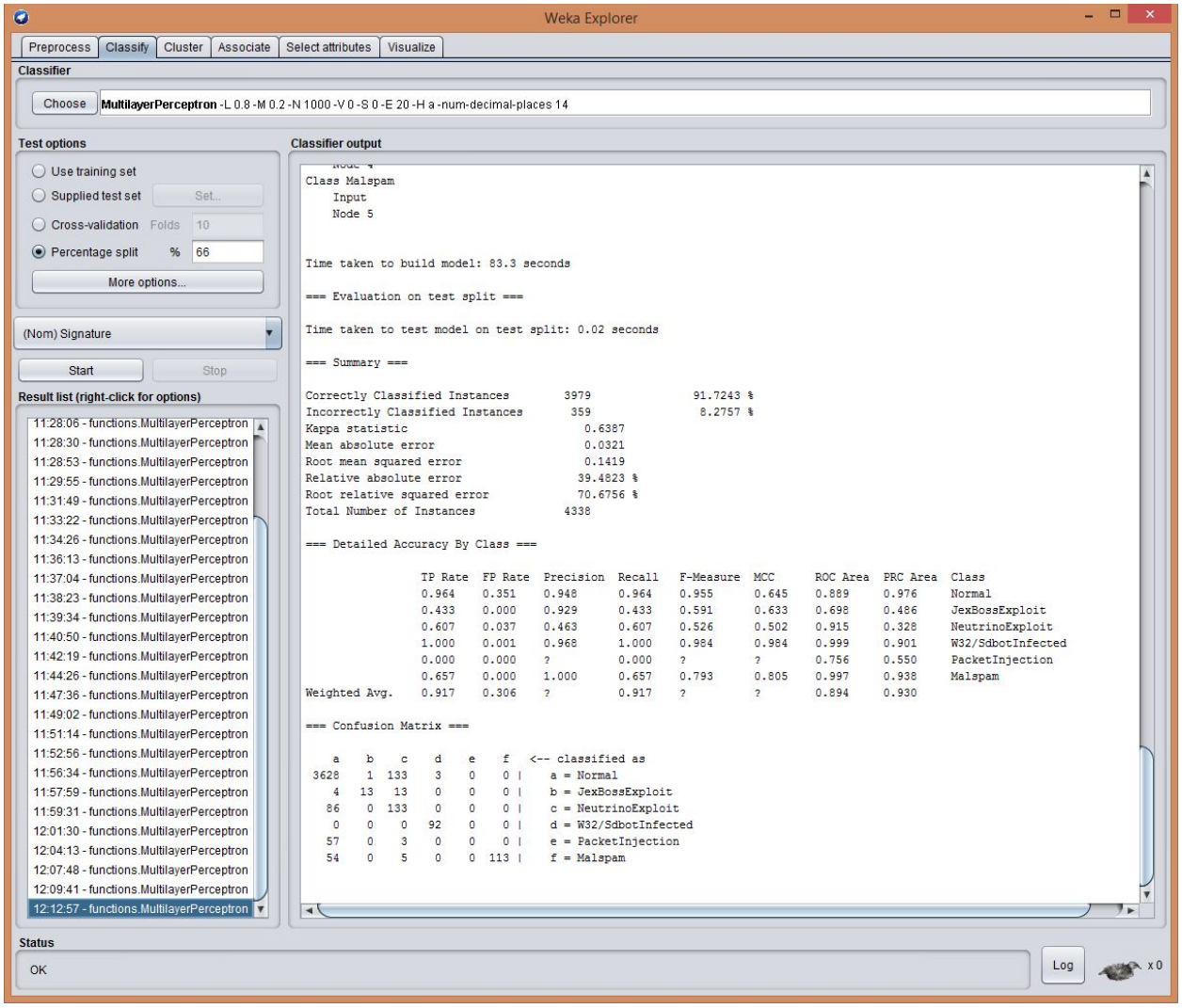
Weka run with the all Signatures labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the all Signatures labeled set with learning rate 0.3 and trained for 1000 epochs:



Weka run with the all Signatures labeled set with learning rate 0.8 and trained for 1000 epochs:



The best runs were with the learning rate at 0.3. Increasing the number of epochs did not increase efficacy with either learning rate. Increasing the learning rate decreased efficacy.

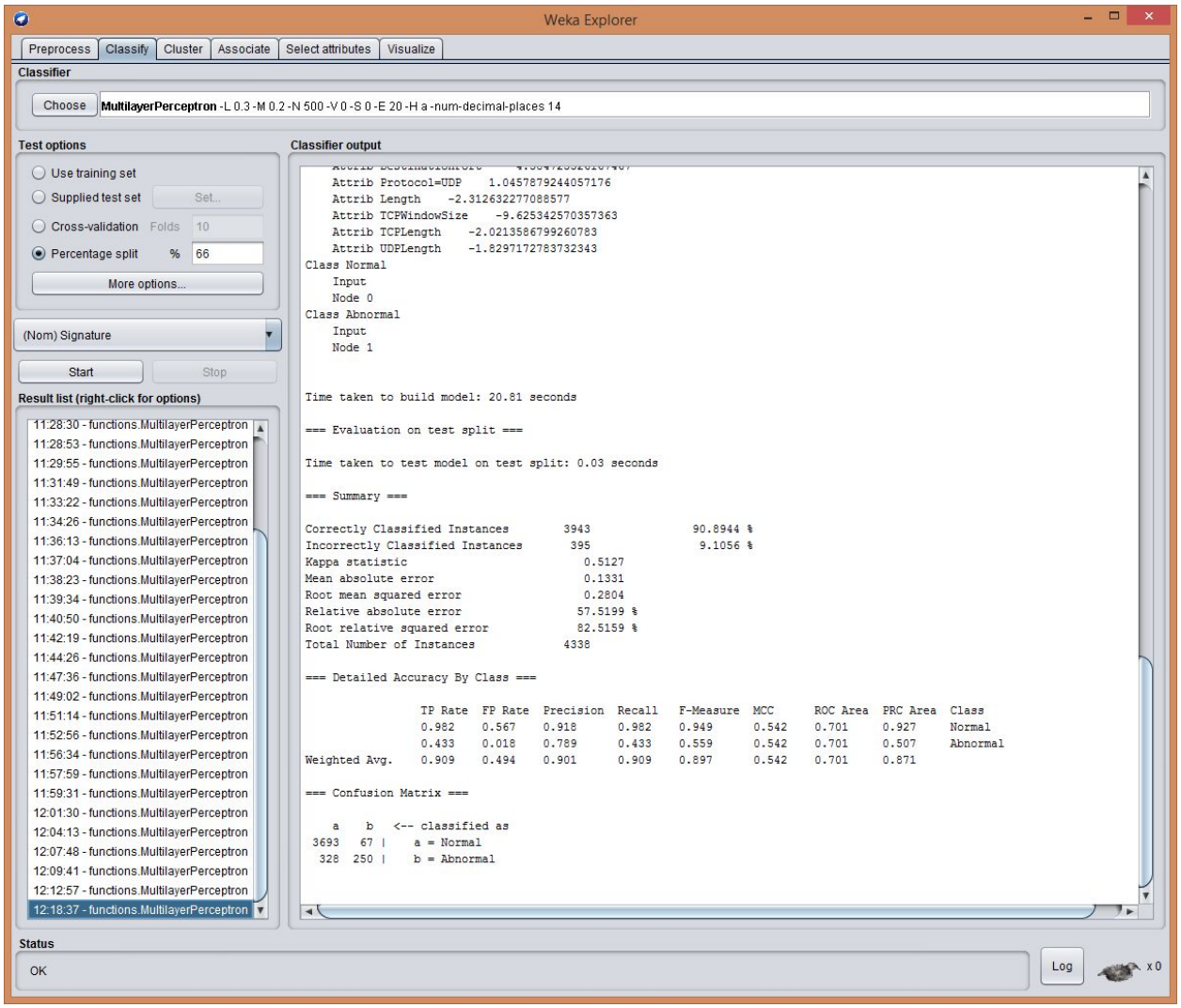
For the best run, learning rate at 0.3 and number of epochs at 500:

$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

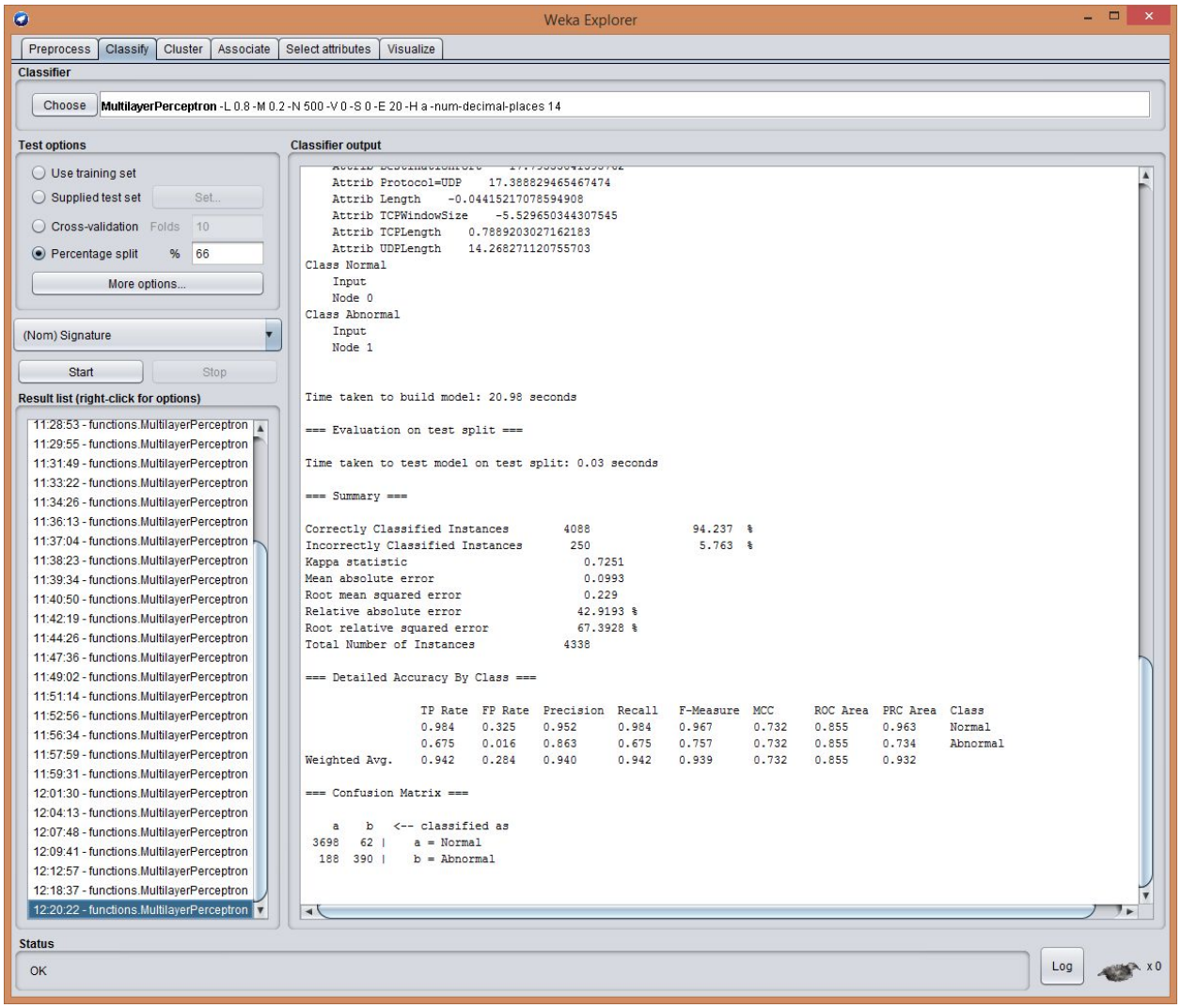
$$precision = \frac{(14+120+82+113)}{(14+120+82+113)+(1+2+41)} = 0.882$$
$$recall = \frac{(14+120+82+113)}{(14+120+82+113)+(1+133+3)} = 0.706$$
$$accuracy = \frac{(14+120+82+113)+3428}{(14+120+82+113)+3428+(1+2+41)+(1+133+3)} = 0.954$$

<u>Anomaly</u>

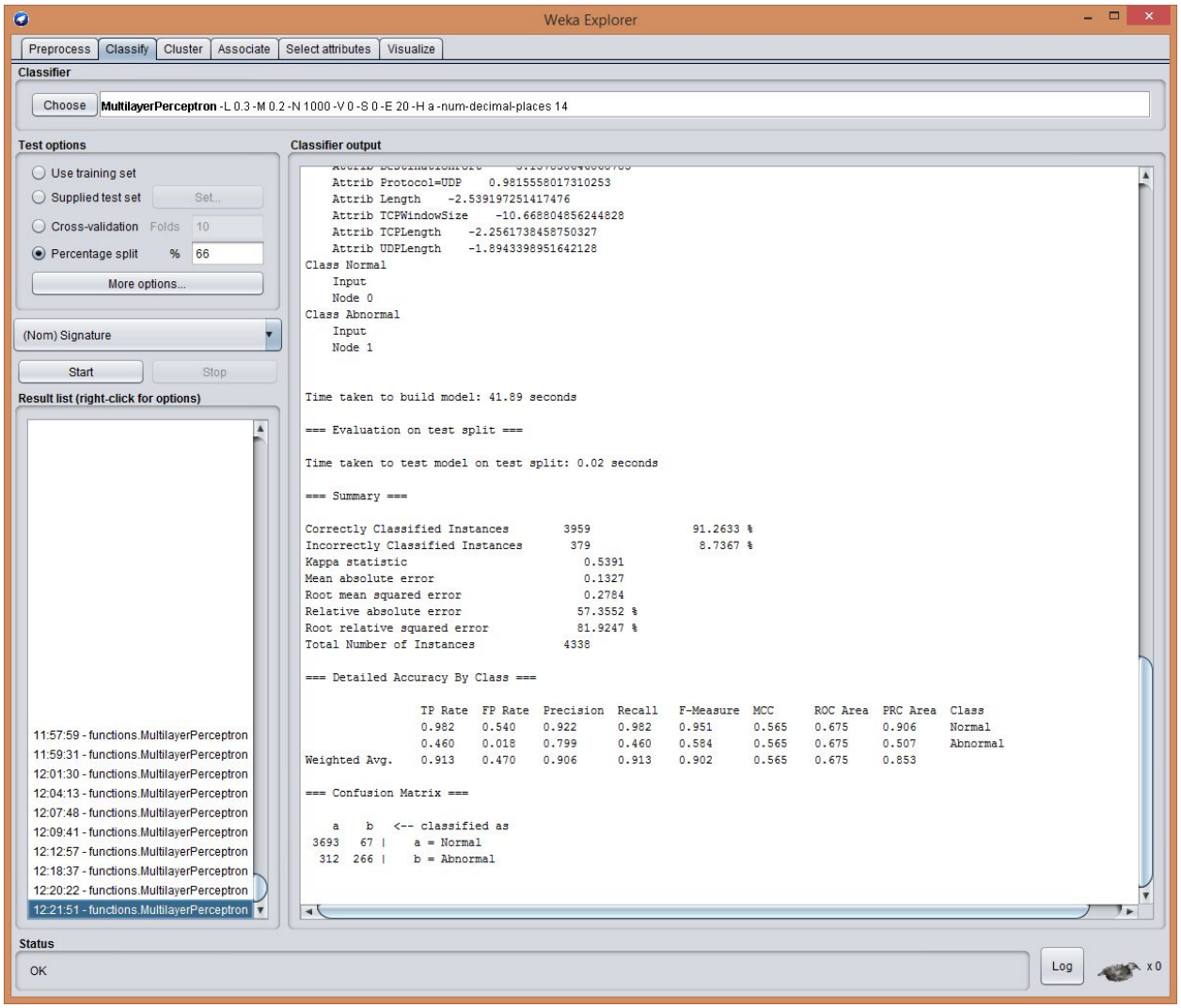
Weka run with the Anomaly labeled set with learning rate 0.3 and trained for 500 epochs:



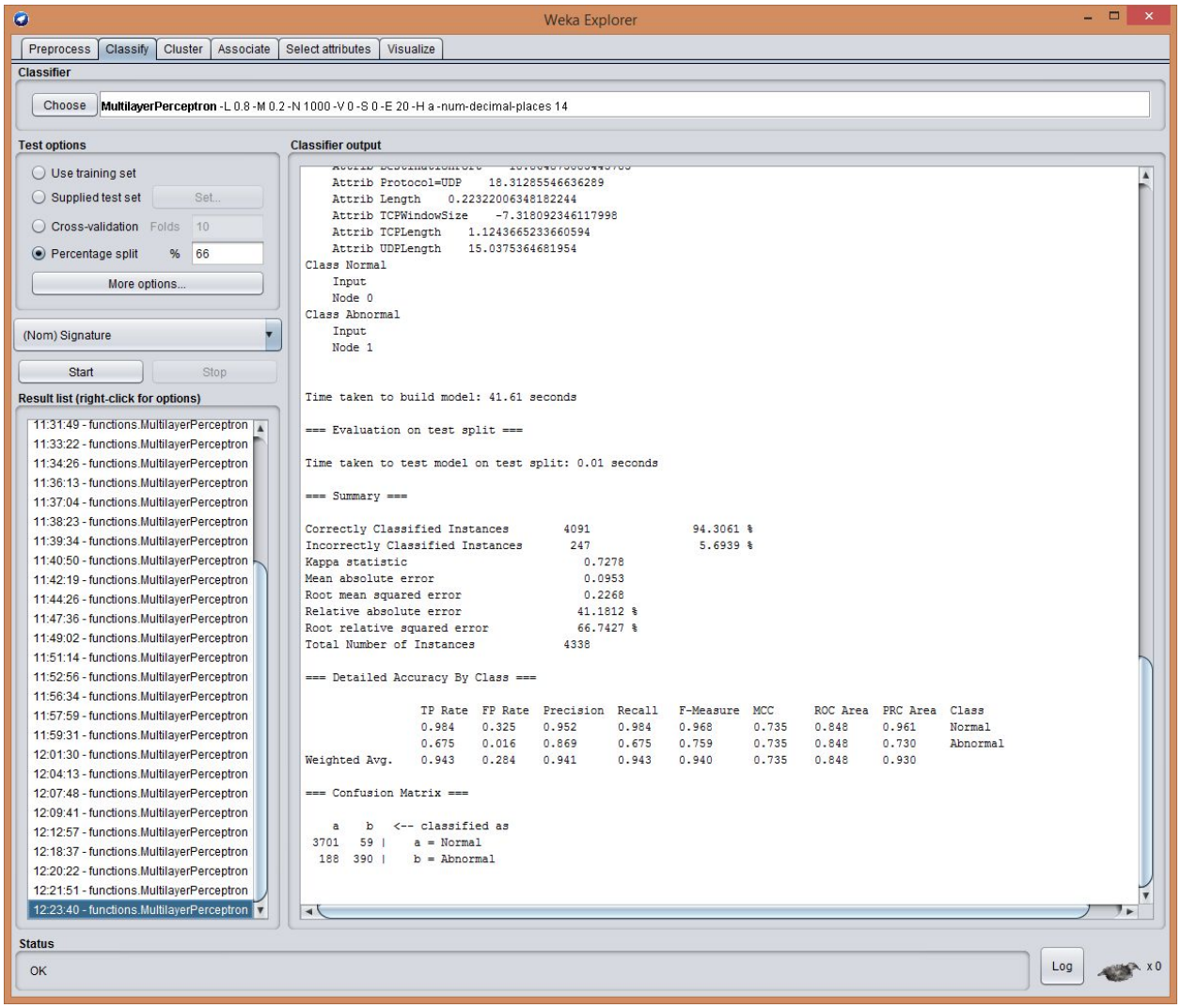
Weka run with the Anomaly labeled set with learning rate 0.8 and trained for 500 epochs:



Weka run with the Anomaly labeled set with learning rate 0.3 and trained for 1000 epochs:



Weka run with the Anomaly labeled set with learning rate 0.8 and trained for 1000 epochs:



The best run was with the learning rate at 0.8 and the number of epochs at 1000. Increasing the learning rate increased efficacy significantly however increasing the number of epochs only increased the efficacy very slightly.

$$precision = \frac{TP}{TP+FP}$$
$$recall = \frac{TP}{TP+FN}$$
$$accuracy = \frac{TP+TN}{ALL}$$

$$precision = \frac{390}{390+59} = 0.869$$
$$recall = \frac{390}{390+188} = 0.675$$
$$accuracy = \frac{390+3701}{390+3701+59+188} = 0.943$$

Conclusion

The results of the MLP algorithm proved inconsistent across the data sets. While some had perfect classification efficacy others had very low efficacy. The important data sets, the full signature based set and the anomaly based set, were among the weaker sets. Adjusting the selected parameters often did not yield any changes at all. Some of these problems may have come from the extra data trimming in the data preprocessing step. One attempt not in the results section using cross fold validation yielded slightly better classification efficacy but significantly higher runtimes. Based on the results a MLP does not appear to be very effective at first but several limitations may have hindered it.

The reason these neural networks weren't implemented in Snort is because the process is very complicated. Each perceptron in the network is part of a larger equation that maps the output across n-dimensional space, where each dimension is a feature. Using the equation to generate rules that can be implemented in Snort would require finding the n-dimensional areas of attacks then using the ranges of those areas to create ranges in packet parameters to base the rules on. This process would be exceedingly time consuming as there are many attack spaces across the n-dimensional space. Additionally the problem cannot be brute forced as this would require searching a space of the range of all IP addresses squared, as there is a host and destination IP, times the rest of the ranges of the features. That range is magnitudes larger than 4.6 billion squared, the IP addresses range. If Snort had greater functionality in its rules this process could be circumvented by doing the calculation for each data point in real time as the packet comes in. This would not take a significant amount of time as normally there are potentially thousands of rules to compare the packet against and a single, albeit large, calculation is unlikely to compare to the time taken by that process. The calculation may even render many rules redundant which makes it possible that using an equation developed in this way to differentiate between attacks and normal traffic more time efficient than using rules alone. Ultimately, since computer security is constantly changing and the neural network needs to be update very frequently, the process of manually recalculating ranges from the equation output from the neural network is untenably time consuming and it would very likely be easier to just write an entirely new tool like Snort or modify an existing tool to support this type of packet discrimination directly.