Neural Networks
Assignment 2 Report - Network Intrusion Detection

Dinis Marques Firmino
P13240786
Dr. Elizondo

Introduction

For this assignment, the task was to develop a neural network system that could detect intrusions on a computer network. This system needs to be able to classify good connections from bad ones such as dos attacks. Ideally, the system should also be able to classify each individual type of connection, i.e normal, dos, smurf, neptune…
The process will start from tool selection, then move onto data analysis and pre-processing, model building, assessment and finally showing the results.

Tools

Due to the assignment being the first machine learning problem I faced, the very first step was to research and come to a conclusion which tools would best be suited for the problem at hand. I wanted to be make the experience gained from this assignment valuable for similar problems in the future, so I opted to go the full route using Python and SAS for initial data exploration. I could have used SAS for the neural network modelling but I preferred the more hands on approach and variety of framework choices available for Python. Not to mention to speed difference of running the scripts.
The frameworks I will be using are Pandas for reading in datasets and providing handy utilities for manipulating datasets on the fly and Matplotlib for plotting graphs when needed. The machine learning library I opted for was Scikit learn.

The Data

One of the most crucial elements of a machine learning system is the data which is used to train and test the system's performance. Neural networks can have their performance significantly impacted if particular steps aren't taken into consideration during the initial data analysis and preprocessing stage.
In this section, I will explain how I went about analysing the dataset, choosing appropriate input variables, data normalisation, removing outliers and missing data.

Labelling the dataset

Firstly, I needed to understand the data. This meant getting to grips with certain networking concepts and understanding the metadata, i.e the labels and meaning of each column in a KDD database entry. As the .DAT file provided didn't include column labels, I inserted each name to the respective column as described on the KDD page using Excel. Having the column names would greatly facilitate further data exploration.

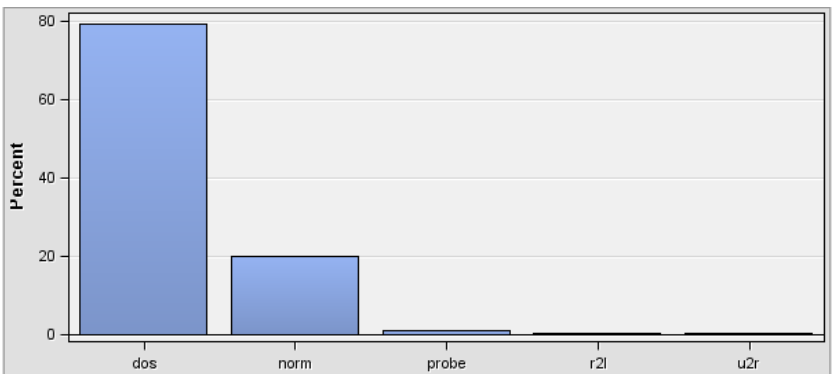
Lastly, I decided to add some more target class variables or y-variables to the dataset, so that it doesn't just contain a target column for every individual type of attack (smurf, neptune, nmap, ipsweep, back...), but also one for each type/category of connections (normal, dos, u2r, probe, r2l). A binary class column was also added to determine if the connection is normal (0) or malicious (1).
This provides the possibility to build 3 separate models, one for binary and two for multiclass classification. More importantly it helps reduce initial complexity in understanding the correlation between variables and target classes due to the problem becoming more general with fewer classes to predict.

Being on the data mining module and currently learning how to use various techniques using SAS enterprise miner to retrieve insights from datasets, I thought it would be a good opportunity to run this data set through the software in order to:

- Analyse the data using graphs such as histograms and box plots.
- Analyse shape of data distributions and statistics.
- Have a better idea of what kinds of input variables have high importance in relation to the target variable.
- Provides a visual way to see qualitative and quantitative data about each attack.

Graph data exploration

The first stage in should be to get a feeling for the proportions of each class in the dataset. Using a simple bar chart you can get a sense of how the data is spread out through the data set and gain some insights into possible problems that a predictive model such as a neural network might run into down the line.
Below is a bar chart representing the percentage of entries for each connection type class in the dataset.



Class	Percentage	Frequency
Normal	19.69%	391458
Dos	79.24%	97278
Probe	0.831%	4107
R2L	0.228%	1126
U2R	0.010%	52

With such a simple chart, we can already get a pretty big insight into the dataset, which is the imbalance of the classes. The normal and dos connections make up the entire dataset or 98.93% to be more precise. Based on the amount of training samples available, the neural network models are more likely to predict the classes with the highest amount of samples (DOS and NORMAL), and struggle the most with U2R connections in a multi class classification problem.

Clustering

With the hope of finding clear distinctions between 1 class and the other, I attempted to perform some unsupervised learning techniques such as k-means clustering in SAS.
This turned out to be quite ineffective on the KDD dataset. I expected to find somewhat cleanish ‘cuts’ between data in 1 cluster and the other, but as it turns it there were too many clusters generated to obtain any clear distinctions between 1 attack and the other due to the clustering algorithm considering many different attacks to be in the same cluster, even though they have differences.
This was due to the sheer number of individual attacks that we are trying to classify, the high dimensionality of the problem and high correlation between the data. I concluded this because most clusters were very close to each other in terms of distance.

Data Pre-Processing

Before creating and training our neural network models, the imported KDD dataset needs to go through a series of pre-processing stages to ensure the data is in the correct format to be fed into the neural network model. This is important because neural networks are sensitive to the data fed to them and if not in the right format can break the network, increase training times, or make the algorithms such as gradient descent less effective which in turn can result in worse model performance.

Encoding categorical variables

Neural networks only take numerical values as input. Most variables in the KDD dataset are either continuous or discrete, however nominal/categorical variables have to be encoded in a way that each category is represented by a number.

The LabelEncoder class in SciKit Learn was used to fit the categorical variables in the dataset and create numerical representations of nominal values. **Refer to appendix (A)** for values. OneHotEncoding could have been a more suitable method because the format would be more friendly for the network as it will be in the form of a vector of 0's and single 1, whereas a number could be interpreted as a quantity by the network.

Outliers

Outliers in data can drastically affect the mean of continuous and discrete variables in a dataset. They are seen as anomalies in a dataset, and typically originate from human error when recording data or hardware failure such as sensors.
For this assignment I have decided to keep the original dataset with outliers and use a z-scoring technique to create a separate filtered dataset. This was so that I could compare model performance using both data sets.
The filtration was achieved using a z-score technique such as the measure of standard deviations from the mean of continuous or discrete variables. I used a measure of 3 standard deviations to remove rows with outliers.

Feature selection

Feature selection is a crucial stage in the data pre-processing stage. The KDD dataset contains 41 variables, and out of those we only want to select the features that contribute most to the target variable.
This is because feeding a high number of input variables which are irrelevant can decrease the accuracy of the neural network models. Feature selection allows the models to benefit from a reduced chance of overfitting due to the decreased noise in the data, and reduced training time as the algorithms are processing less data.

There are many selection techniques but in this project I started off by performing a simple univariate selection method called chi squared test to produce a score for each variable that represents a correlation to that target, variables with the lowest scores are removed.

While the aim of such techniques is to reduce dimensionality, they differ from other methods such as PCA in which the goal is to combine and create new variables/components which didn't exist in the dataset.

Scaling

One of the most important pre-processing steps is to scale every continuous and discrete variable in the dataset. The reason for doing this is because the ranges of values vary greatly from one variable to the other, i.e one ranging from 0-10 and another from 0-30000.
Scaling the data makes sure that all the values have an equal importance to the network with all values ranging from 0-1.
In python this was achieved using the MinMaxScaler from the Scikit learn library which scales the variables in relation to the min and max value in the column.

The remaining feature columns in the dataset which contained categorical variables were

excluded from the scaling process as it would render those columns useless to the network due to losing all meaning of individual categories.

Re-balancing the dataset
I wrote a small script (Appendix - B) that retrieved some samples of U2R and R2L class from the full KDD dataset and combines them with the original dataset. This boosted the number of classes slightly, just enough to reduce the MSE by a fraction for the final 23 class model.

Neural Network Models

After the data had been analysed and pre processed, the actual building of the neural network models could commence. I decided to take the approach of incrementally stepping up the classification task complexity by building 3 separate neural network models, one for binary classification, another for 5 classes and the final one for **t**he 23 individual attack types.

Binary Classifier
Inputs:
Chi2 Test = Count, Srv_serror_rate, protocol, logged_in, service, dst_host_same_src_port_rate, dst_host_diff_srv_rate
Outputs: Not attack (0) or Attack (1)

5 Class Classifier
Inputs:
W/ Chi2 Test = Duration, Src_Bytes, Dst_Bytes, Count, Srv_Count, **1 missing**
Outputs: Dos(0), Normal(1), Probe(2), R2L(3), U2R(4)

23 Class Classifier
Input:
W/ Chi2 Test = Src_Bytes, Dst_Bytes, Count, Srv_Count, Serror_rate, dst_host_srv_diff_host_rate
Outputs: Refer to appendix() for output attack names.

Number of hidden layers and units
The number of hidden layers chosen was a maximum of 2. There was no need to go over that as performance wouldn't necessarily increase but training times would. The number of hidden units for each layers was a bit of a manual selection process, varying from low numbers to high numbers. The sweet spot seemed to be around 8 hidden units in the first layer and 6-7 in the second. However, as discussed later on, more methods could have been applied to better tune these parameters.

Activation function
All the network models use a logistic activation function. This is usually the standard option for most multi layer perceptron networks and is reported to be able to perform well on most multi-class classification problems.

$$f(x) = 1/(1 + exp(-x)).$$

Optimizer
Stochastic gradient descent.

Learning rate & Momentum
Typically when assessing neural network models these parameters will either selected manually, or an algorithm is used to determine the most optimal parameters in a process called hyper-parameter optimization. For this assignment, I used Scikit learn's default values of 0.1 for learning rate and 0.9 momentum, in order to reduce burden when evaluating the performance of the models.

Model Performance Assessment

Cross validation
The importance of having a separate holdout test set containing samples of data that the network has never seen during fitting/training is crucial to the performance of any machine learning algorithm. There are many problems that may arise, the most common being that the network will overfit the data and memorise the outputs.

Initially, the dataset was split into a single training and test sets of size 70% and 30% of the respectively. This proved to be a good approach and could have worked for the rest of the assignment, but after doing research, various sources seemed to suggest that a more robust evaluation strategy would be to use cross validation.
More specifically I decided to use Stratified K-Fold algorithm for splitting the data into K number of folds containing a train and test set. The stratification is necessary in this problem because of the diverse set of classes and to preserve the balance of each class within each fold. This differs from the randomised approach used in normal splitting.

The functions that were used were **Crossvalscore()** and **Crossvalpredict()**. The first function returns the mean accuracy that was achieved by the model when predicting the holdout test set of each fold. The latter returns an array of the predicted value of Y for each sample in the dataset, when that particular sample was in one of the test sets.

Metrics
There were a range of useful metrics that were used to evaluate the performance of the models. The prediction accuracy metric turned out to be quite unrepresentative and uninformative in relation to the true model performance. This was due to the class imbalance where 99% of the dataset are Normal and Dos connections, every other attack fell in the 1%. Due to the number of samples of these classes, the overall prediction accuracy is bound to be high. So I decided to use the classification report feature of Sci-Kit learn to output the prediction accuracy, recall and f1-score for each individual class.
This way we can see exactly which classes the network is predicting less and are holding the overall performance back.

The coefficient of determination or r2, was also used to measure how well future samples are likely to be predicted by the model. This metric is more accurate than the above metrics for measuring overall model performance and would equal 0 if the model is overfitting by constantly predicting the value of y.
The mean squared error is also used to measure the regression loss.

Confusion Matrices
Confusion matrices were a great model assessment tool due to their power to express the amount of true positives, true negatives, false positives and false negatives for each class classified by the system.

Results

The results show a series of tables consisting of the hyperparameters chosen for learning rate (**LR**), momentum (**MM**),epochs (**EP**) and units per hidden layer **(L1,L2 #U).** The dataset used is also included in the (**DS)** column to provide a comparison of results. The datasets tested on are Original (**OG**), Duplicates removed (**DR**), Original merged with Extra samples (**OM**)...
Note: The outlier filtered dataset was not included in the results due to time constraints!

The metrics included in the table are R2 and MSE, a full report on the classification accuracy for each class and confusion matrix for the best model will also be included in the appendices.

Binary Classifier
Refer to appendix (C) for the confusion matrix.
Refer to appendix (C) for full report on the classification accuracy.

5 Class Classifier
Refer to appendix (F) for the results table.
Refer to appendix (H) for the confusion matrix.
Refer to appendix (H) for full report on the classification accuracy.

23 Class Classifier
Refer to appendix (G) for the results table.
Refer to appendix (D) for the confusion matrix.
Refer to appendix (E) for full report on the classification accuracy.

Future Improvements

Due to the time constraints and the nature of neural network problem solving, there was simply no time to try some of the more advanced techniques in order to determine the best overall model.

Hyperparameter selection and tuning

The manual selection of hyperparameters proved to be a tedious task as there were simply too many possible combinations of parameters to try. For each of those, a new set of results would need to be produced to evaluate the new model performance, even if minor.
A more convenient and effective method would be to use Grid search cross validation to generate many classifiers from a grid of possible parameter values. The model achieving the best performance would be picked as the winner.

More dimensionality reduction techniques

The usage of a few more dimensionality reduction techniques would have been helpful to produce more comparisons in how the model performances would be affected. In particular, principal component analysis would be a strong candidate along with recursive feature elimination of features using regression algorithms.

Conclusion

To conclude, the assignment has been a very pleasant experience as an introduction to the practical side of machine learning by solving a real world problem. From the point of first contact with the data, to learning a range of techniques for pre-processing and evaluating the performance of neural network models, the end result, while nowhere near perfect, was an eye opener to the almost infinite ways of solving a single problem.
The main obstacle was the imbalance of data which really showed through in the last model a lot of the attacks being misclassified 100% of the time due to few samples as seen in the support columns of the classification reports in the appendices.
With more time and experience the ability to deal with these problems will increase.

References

http://scikit-learn.org/stable/documentation.html

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

http://scikit-learn.org/stable/modules/cross_validation.html

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Scikit_Learn_Cheat_Sheet_Python.pdf

http://qingkaikong.blogspot.co.uk/2016/11/machine-learning-6-artificial-neural.html

http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

https://datascience.stackexchange.com/questions/410/choosing-a-learning-rate

Feature selection
http://machinelearningmastery.com/an-introduction-to-feature-selection/
http://machinelearningmastery.com/feature-selection-machine-learning-python/
http://scikit-learn.org/stable/modules/feature_selection.html#univariate-feature-selection

Evaluation
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
https://github.com/edublancas/sklearn-evaluation#

https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet#gs.kcRxK8s

Pre-processing data for neural networks - Department of Computing
http://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/

Appendices

Appendix (A) - Categorical variable numerical representations

Protocol	Service	Flag	Attack Name	Attack Type
TCP = 1	HTTP = 22		NORMAL = 11	DOS = 0
ICMP = 0	SMTP = 50		SMURF = 18	NORMAL = 1

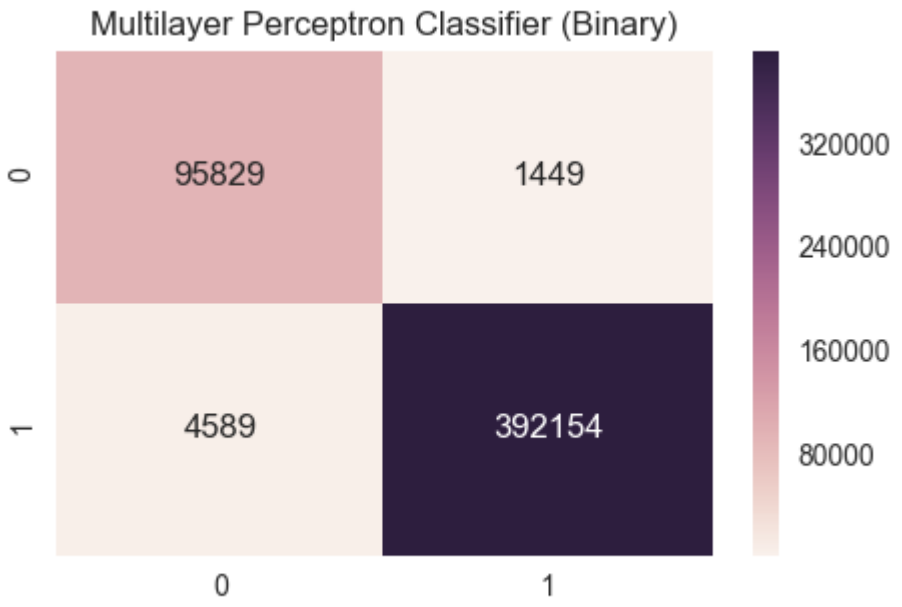
UDP = 2	**FTP_DATA = 19**		**NEPTUNE = 9**	**PROBE = 2**
	FINGER = 17		**BACK = 0**	**R2L = 3**
	TELNET = 56		**BUFFER_OVERFLOW = 1**	**U2R = 4**
			IPSWEEP = 5	
			LOAD_MODULE = 7	
			FTP_WRITE = 2	
			IMAP = 4	
			N_MAP = 10	
			PHF = 13	
			MULTIHOP = 8	
			LAND = 6	
			PERL = 12	
			SPY = 19	
			TEARDROP = 20	
			POD = 14	
			PORTSWEEP = 15	
			ROOTKIT = 16	
			SATAN = 17	
			WAREZCLIENT = 21	
			WAREMASTER = 22	

Appendix (B) - Merge datasets script

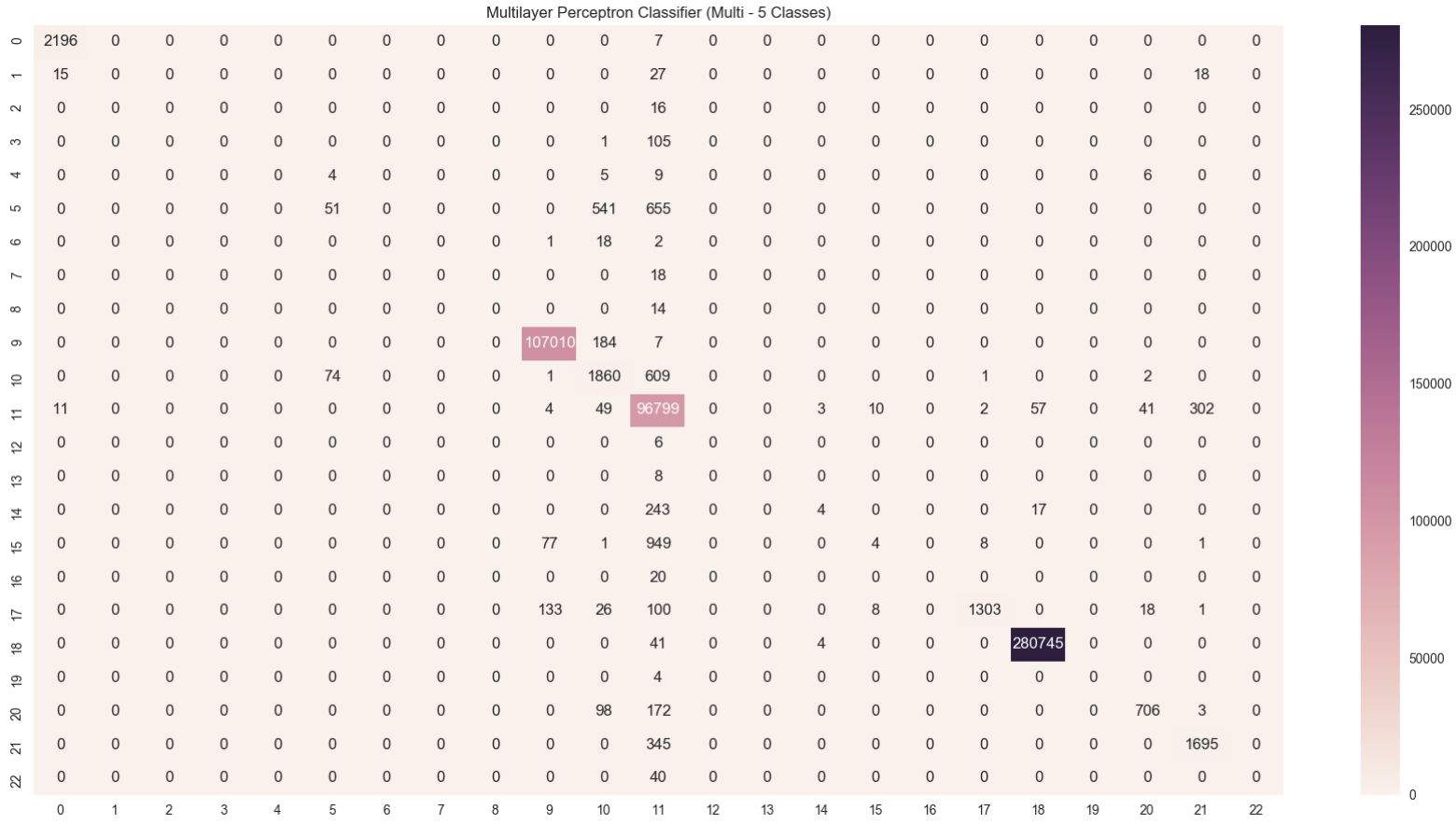
```
import pandas as pd
import numpy as np

## Column names
names = ['duration','protocol','service','flag','src_bytes','dst_bytes','land','wrng_frags','urgen
         'su_attempt','num_root','file_creations','num_shells','num_access_files','num_out_cmds','
         'diff_srv_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same
         'dst_host_srv_serror_rate','dst_host_rerror_rate','dst_host_srv_rerror_rate','attack_name

## Read data from .dat file and insert header row
data = pd.read_table("kddcup_full.data", sep=',', header=None, names=names)
print('KDD Dataset imported...\n')
print('Dataframe header\n')
print(data.head())
print("Dataframe shape: ", data.shape)

## Import


data = data.loc[data['attack_name'].isin(['buffer_overflow.','ftp_write.','guess_passwd.',
                                          'imap.','loadmodule.','multihop.','nmap.','perl.',
                                          'phf.','rootkit.','spy.','warezclient.','warezmaster.'])]
## EXCLUDE PROBE ATTACKS FOR NOW'ipsweep.','portsweep.','satan.'
print("Dataframe shape after selection: ", data.shape)

## Obtain sample numbers
##print(data.groupby('attack_name').count())

# Create a Pandas Excel writer using XlsxWriter as the engine.
writer = pd.ExcelWriter('Extra_Samples.xlsx', engine='xlsxwriter')

# Convert the dataframe to an XlsxWriter Excel object.
data.to_excel(writer, sheet_name='Sheet1')

# Close the Pandas Excel writer and output the Excel file.
writer.save()
```

Appendix (C) - Binary Classification Report + Confusion Matrix
0 = Not Attack, 1 = Attack

```
              precision    recall  f1-score   support

          0    0.95430   0.98510   0.96946     97278
          1    0.99632   0.98843   0.99236    396743

avg / total    0.98804   0.98778   0.98785    494021

MLP Classifier R2:  0.922711547264
MLP Classifier MSE:  0.0122221524996
```



Appendix (D) - 23 Class Classifier Confusion Matrix
Refer to the categorical table for numeric class representations(My apologies didn't have time for converting the variable back to strings!)



Appendix (E) - 23 Class Classifier Classification Report

	precision	recall	f1-score	support
0	0.98830	0.99682	0.99254	2203
1	0.00000	0.00000	0.00000	60
2	0.00000	0.00000	0.00000	16
3	0.00000	0.00000	0.00000	106
4	0.00000	0.00000	0.00000	24
5	0.39535	0.04090	0.07413	1247
6	0.00000	0.00000	0.00000	21
7	0.00000	0.00000	0.00000	18
8	0.00000	0.00000	0.00000	14
9	0.99799	0.99822	0.99810	107201
10	0.66834	0.73027	0.69794	2547
11	0.96610	0.99508	0.98037	97278
12	0.00000	0.00000	0.00000	6
13	0.00000	0.00000	0.00000	8
14	0.36364	0.01515	0.02909	264
15	0.18182	0.00385	0.00753	1040
16	0.00000	0.00000	0.00000	20
17	0.99163	0.82001	0.89769	1589
18	0.99974	0.99984	0.99979	280790
19	0.00000	0.00000	0.00000	4
20	0.91332	0.72114	0.80594	979
21	0.83911	0.83088	0.83498	2040
22	0.00000	0.00000	0.00000	40
avg / total	0.98594	0.98966	0.98708	497515

MLP Classifier R2: 0.976936311168
MLP Classifier MSE: 0.401575831884

Appendix (F) - 5 Class Classifier Results

DS	EP	MM	LR	L1 #U	L2 #U	R2	MSE
OG	300	0.9	0.1	8	5	0.9430	0.0116
OG	1000	0.9	0.1	8	5	0.9418	0.0119
OG	1000	0.9	0.1	8	7	0.9641	0.0073
OG	500	0.9	0.1	8	6	0.9619	0.0077
OG	1000	0.9	0.1	8	6	0.9630	0.0075
OG	500	0.9	0.1	10	0	0.9235	0.0156
OG	500	0.9	0.1	10	5	0.9428	0.0117
OG	500	0.9	0.1	5	3	0.9221	0.0159
DR	300	0.9	0.1	8	5	0.8925	0.0338
DR	1000	0.9	0.1	8	5	0.8992	0.0317
DR	500	0.9	0.1	5	3	0.8360	0.0516
DR	500	0.9	0.1	10	0	0.8633	0.0430
DR	500	0.9	0.1	10	5	0.8703	0.0401
DR	500	0.9	0.1	10	10	0.8912	0.0343
DR	1000	0.9	0.1	10	10	0.9076	0.0291
OM	300	0.9	0.1	8	5	0.948289	0.012186
OM	1000	0.9	0.1	8	5	0.94875	0.01207
OM	1000	0.9	0.1	8	7	0.9422761	0.0136036

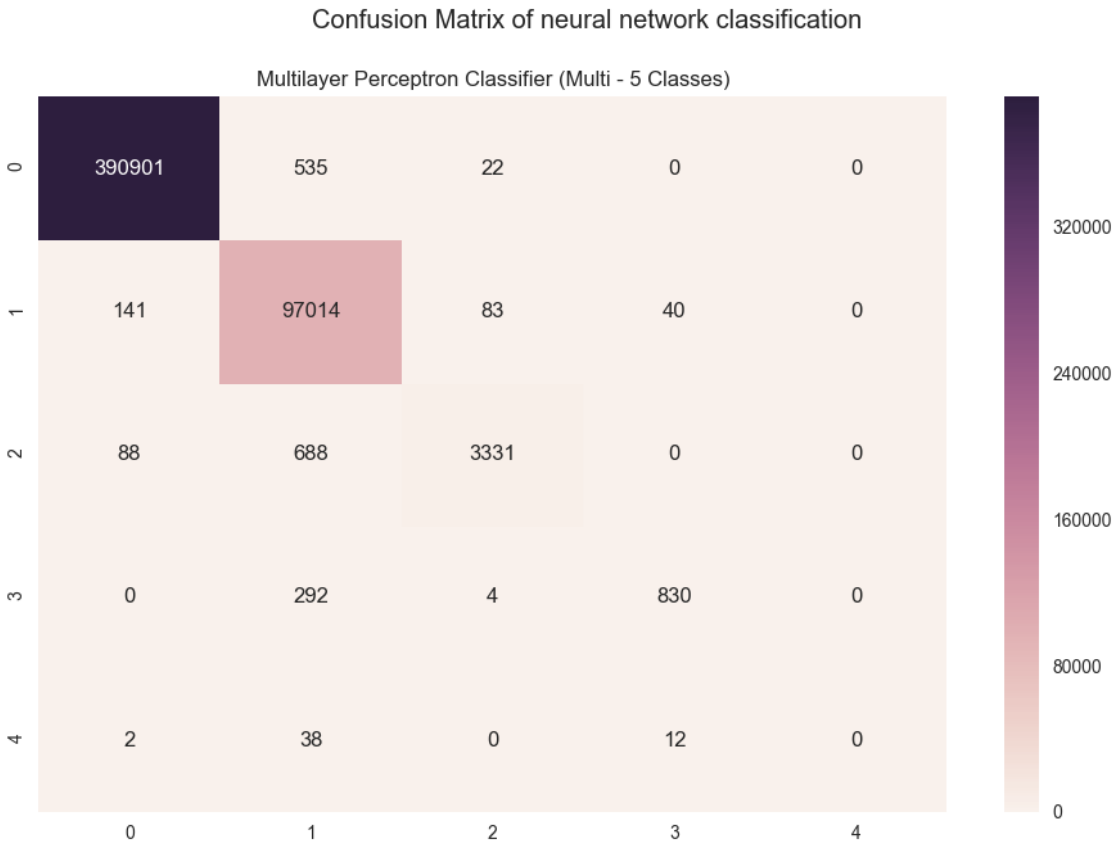
OM	500	0.9	0.1	8	6	**0.951811**	0.011356
OM	1000	0.9	0.1	8	6	0.950370094	0.0116961

Appendix (G) - 23 Class Classifier Results

DS	**EP**	**MM**	**LR**	**L1 #U**	**L2 #U**	**R2**	**MSE**
OG	500	0.9	0.1	8	5	0.97568	0.42102
OG	1000	0.9	0.1	8	5	0.97572	0.42042
OG	1000	0.9	0.1	8	7	0.97669	0.4035071
OG	500	0.9	0.1	8	6	0.976330	0.40988
OG	1000	0.9	0.1	8	6	0.975648	0.42170
OG	500	0.9	0.1	10	5	0.971406	0.495159
OG	500	0.9	0.1	5	3	0.9736278	0.4566911

OM	300	0.9	0.1	8	5	0.9755217	0.426206
OM	1000	0.9	0.1	8	7	0.9718814	0.489589
OM	1000	0.9	0.1	8	6	0.976766	0.404539
OM	2000	0.9	0.1	8	6	0.9772557	0.39601419
OM	5000	0.9	0.1	8	6	0.9771803	.39601419

Appendix (H) - 5 Class Classification Report + Confusion Matrix



```
             precision    recall  f1-score   support

          0    0.99941   0.99858   0.99899    391458
          1    0.98424   0.99729   0.99072     97278
          2    0.96831   0.81105   0.88273      4107
          3    0.94104   0.73712   0.82669      1126
          4    0.00000   0.00000   0.00000        52

avg / total    0.99593   0.99606   0.99590    494021

MLP Classifier R2:  0.964160728878
MLP Classifier MSE:  0.00729726064277
```