A Neural Network Model For Detecting Intrusions Or Attacks On A Computer Network

Jon-Paul Boyd
School of Computer Science and Informatics
De Montfort University
United Kingdom

***Abstract*— This report presents an approach to the KDD Cup 1999 challenge (10% dataset) for classification of computer network activity as *bad*, covering intrusions and attacks, and *good* for normal connections. Typically, this capability is incorporated in Intrusion Detection System (IDS) for safeguarding the security, integrity and availability of networked systems and data by actively detecting anomalies and malicious activity. Data discovery first looks to understand the nature of the dataset and prepare it for optimal supervised machine learning. Single Layer Perceptron (SLP) and Multi-Layer Perceptron (MLP) neural network (NN) architectures are modelled to make binary (*good*, *bad*) and multiclass (specific attack category) predictions. Each proposed model is hyperparameter tuned to extract maximum performance, then evaluated with criteria including detection rate, false alarm rate, accuracy, complexity of topology and training runtime cost. A final evaluation is made, including comparison with the XGBoost classifier. The MLP (binary) model outperformed all other models, with a detection rate of 0.999%.**

Index – Neural Networks, Intrusion Detection System, Single Layer Perceptron, Multi-Layer Perceptron, Supervised Machine Learning, Keras, Python, RMSProp, ReLU, Sigmoid, XGBoost

I. INTRODUCTION

Intrusion Detection Systems (IDS) were first proposed in 1980 by James P. Anderson to "*improve the computer security auditing and surveillance capability of the customer's systems*" [1], and can be defined as "*software or hardware systems that automate the process of monitoring the events occurring in a computer system or network, analysing them for signs of security problems*" [2]. Today, they are an especially important component of computer network security architecture, given the increased exposure of business critical process, product, financial, customer and employee internal systems and data to the internet including via IoT (Internet of Things), mobile self-service applications, B2B (Business to Business) and B2C (Business to Consumer) integration. With network firewalls, inbound network traffic is filtered according to a set of predefined rules, meaning malicious traffic can reach systems if no rules exist that match and flag such traffic. Similarly, signature based detection systems rely on an inventory of signature definitions to enable flagging of activity as a threat. In both cases, the technique is reactive, with the firewall rules and signature inventory updated only after an attack type is known. Conversely, IDS is both proactive and able to deal with unknown threats and abnormal activity, providing an additional security layer by actively classifying internal and external traffic in real time as *good* or *bad* according to characteristics of the network data packets, and can respond appropriately by closing ports, blacklisting I.P. addresses or sending TCP reset packets to break connections [3].

In building an IDS for this study the Knowledge Discovery Databases (KDD) Cup 1999 (10%) dataset is used [4], a dump of local area network (LAN) network traces simulating air force base LAN activity and "*originally prepared by MIT Lincoln labs for the 1998 Defence Advanced Research Projects Agency (DARPA) Intrusion Detection Evaluation Program, with the objective of evaluating research in intrusion detection, and it has become a benchmark data set for the evaluation of IDSs*" [5]. This study focuses on 2 main NN types, SLP and MLP, to differentiate between *good* and *bad* connections. Furthermore, models making multiclass classification based on attack category are proposes and compared with their binary equivalents. This paper is organized as follows. Section II details the experimental setup that includes data analysis and pre-processing. Section III details and evaluates the proposed NN solutions. Section IV closes with final evaluation and thoughts.

II. EXPERIMENTAL SETUP

A. Software and Hardware Components

Given their accessibility and prevalence in machine learning and deep learning, the Python programming language with Keras NN API [6] is used, with further component specification as follows:

- **Software** Linux Ubuntu 18.04, Python 3.6, PyCharm Professional 2019.1
- **Hardware** Intel i7-5820K, Nvidia GeForce GTX 1080, 16GB RAM
- **Main Python Modules** Keras (2.2.4), numpy (1.16.2), pandas (0.24.2), sklearn (0), tensorflow-gpu (1.13.1), xgboost (0.82)

B. Initial Data Analysis

Initial data analysis is done to understand the size, quality and nature of information content. There are 494020 observations comprising 42 features (*Appendix A - Feature Distribution Statistical Analysis*), of which 3 are categorical and the remaining numerical. For this data discovery phase, categorical features *protocol_type*, *service* and *flag* are numerically encoded with sklearn's [7] *LabelEncoder*, and numerical features scaled with *StandardScaler*. Feature 42 "*label*" identifies each observation as either *normal* (*good*) or as 1 of 22 attack types such as *smurf* and *rootkit* (*bad*). New binary feature "*target*" was created as the target response or dependant feature, mapping *good* to 0 and *bad* to 1. The attack types are further grouped into 1 of 4 broader categories, which along with *normal* are used in this study as an added "*attack_category*" feature for multiclass analysis. The attack categories are as follows:

- **DoS (Denial of Service)** Floods target with requests, overloading resources thus denying legitimate requests.
- **Probe** Reconnaissance scanning the network for vulnerabilities and compromising information.
- **r2l (Remote to Local)** External attacks looking to gain local access
- **u2r (User to Root)** Attacks originating from normal user account, seeking to gain root (super user) privileges.

2 significant characteristics stand out. First, a huge 348439 duplicate records, summarised in Table I, primarily for observations labelled *smurf* (280149), *neptune* (55381) and *normal* (9446). Such extensive duplication can create machine learning bias towards the more heavily represented classes, of particular concern in this problem domain, as this risks infrequent yet potentially catastrophic *r2l/u2r* attacks proceeding undetected. Therefore, duplicates were removed, resulting in a total of 145581 remaining observations split ~ 60/40% by *good*/*bad* respectively. Table I documents the number and percentage of remaining records. The second issue is the extremely imbalanced distribution. *Normal* and *DoS* observations constitute 97.8% of the population, with minority class *probe* represented by only 1.463% (2130), *r2l* only 0.686% (999) and *u2r* just 0.036% (52). As these minority classes are severely underrepresented, proposed models will perform both binary and multiclass classification to compare accuracy in detecting minority attack types.

TABLE I
CATEGORISED, LABELLED OBSERVATION DUPLICATE STATS

Attack Cat	Label	Binary Class	# Dup.	# Recs Rem.	% Rem.
dos	back	bad	1235	968	0.664
dos	land	bad	2	19	0.013
dos	neptune	bad	55381	51820	35.595
dos	pod	bad	58	206	0.141
dos	smurf	bad	280149	641	0.440
dos	teardrop	bad	61	918	0.630
normal	normal	good	9446	87828	60.329
probe	ipsweep	bad	596	651	0.447
probe	nmap	bad	73	158	0.108
probe	portsweep	bad	624	415	0.285
probe	satan	bad	683	906	0.622
r2l	ftp_write	bad	0	8	0.005
r2l	guess_passwd	bad	0	53	0.036
r2l	imap	bad	0	12	0.008
r2l	multihop	bad	0	7	0.004
r2l	phf	bad	0	4	0.002
r2l	spy	bad	0	3	0.001
r2l	warezclient	bad	127	893	0.613
r2l	warezmaster	bad	0	20	0.013
u2r	buffer_overflow	bad	0	30	0.020
u2r	perl	bad	0	3	0.002
u2r	loadmodule	bad	0	9	0.006
u2r	rootkit	bad	0	10	0.006

Appendix A summarises feature distribution. *is_host_login* and *num_outbound_cmds* are constant with zero value therefore removed. 11 features only have between 2 and 10 distinct values. Most features are asymmetrically distributed and highly positively skewed (> +1), with 14 having zero value for over 90% of observations. One example, *src_bytes*, is extremely positively skewed (45.718) with 75th quantile (288) well below mean (3233.138), large std (107324.844) and high kurtosis (2143.580) indicating heavy tails (many 0, low and high values) relative to the rest of the distribution. Another example, *num_root*, has 99.60% of observations with 0 value, giving a high positive skew of 183.195. With significant kurtosis (37128.370) and max 306 over 75th quartile of 0 is indicative again of tail-heavy distribution. This suggests the need for suitable feature scaling.

C. *Sparse Features*

Evaluation of sparse features, meaning those with > 99% observations with zero value, highlighted several with non-zero values for minority attack categories (Table II). They will be preserved from any feature dropping to maximise minority class attack detection.

TABLE II
SPARSE FEATURES BY MINORITY ATTACK CATEGORY

Feature	# probe	# r2l	# u2r
urgent	0	2	1
num_failed_logins	0	52	1
num_compromised	0	4	23
root_shell	0	6	26
su_attempted	0	1	0
num_root	0	5	10
num_file_creations	0	9	23
num_shells	0	3	5
num_access_files	0	9	1
is_guest_login	0	314	0

D. *Manual Data Correction*

Of 20 non-zero observations for feature *land*, 19 were for attack category *land* and 1 for *normal*. Considered a data quality issue, this was rectified by setting *land=0* for category *normal*.

E. *Feature Correlation*

Fig. 1 presents a zero matrix illustrating the data density for features over all observations, with black representing non-zero value and white zero. It clearly shows features like duration (1st column) have very few non-zero values. Feature correlation is also suggested, for example between *serror_rate*, *srv_error_rate* and *diff_srv_rate* where blocks of black and white are symmetrical.

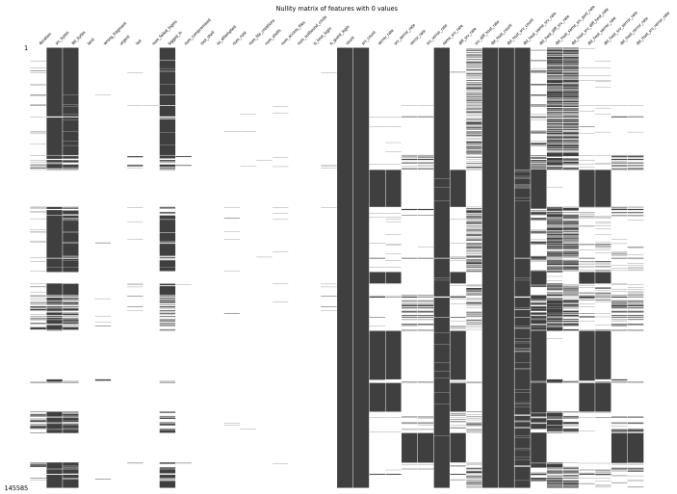

Fig. 1. Zero matrix of numerical features

The correlation heatmap (Fig. 2) shows higher feature correlation with reds and lower with blues, the value given in range -1 (low) to 1 (high). Features *dst_host_rerror_rate* and *rerror_rate* are highly correlated (0.98), whereas *count* and *same_srv_rate* are not (-0.84). Highly correlated features indicate redundancy. Aiming to reduce the final 2d tensor shape to support optimal performance, 1 feature in each highly correlated pair is dropped. "*Highly correlated*" is subjective to each problem domain. Not being a network security expert, correlation >= 80 is deemed reasonable and based on review of name pairs from the heatmap. This resulted in the following 8 features removed from the dataset: *same_srv_rate*, *srv_serror_rate*, *dst_host_same_srv_rate*, *dst_host_serror_rate*, *dst_host_rerror_rate*, *dst_host_srv_serror_rate*, *srv_rerror_rate* and *dst_host_srv_rerror_rate*.

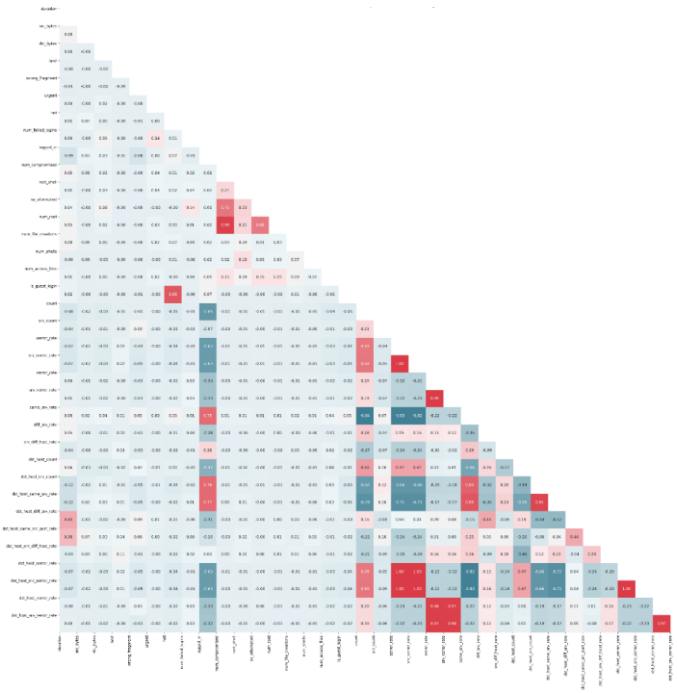

Fig. 2. Feature correlation heatmap

F. *Outliers*

Careful consideration was given to outliers which can introduce bias in model training. Testing several numerical feature value thresholds to determine observation removal supported concerns about dealing over-aggressively with outliers and risk important minority class information being removed. For example, the typical +/- 3STDEV rule resulted in deletion of several thousand observations. A more delicate approach was taken, removing observations where feature values were within 95% of max, provided those values were > 50 to prevent bulk deletion of narrow distributions. This resulted in 4 rows being dropped (*duration* > 55412, *src_bytes* > 658706858, *num_compromised* > 839, *num_root* > 814).

G. *Linear Separability*

To get a better feeling for this classification task it helps to understand if the problem is linear or non-linear. If linear, a less complex model may be sufficient. As this study includes binary classification, presence of linearity can be visualised using a scatter graph (Fig. 3). Data points from features *srv_diff_host_rate* and *srv_count* are plotted blue if the observation is labelled *good*, and red if *bad*. As shown, a straight line cannot clearly separate *good* from *bad*, thus this classification problem is non-linear. The convex hulls drawn to connect outermost points in each class provide further evidence of non-linearity. The blue convex hull is not able to include only blue points from class *good*, and with class *bad* red convex hull fully enclosing blue, the dataset is clearly non separable by any straight line.

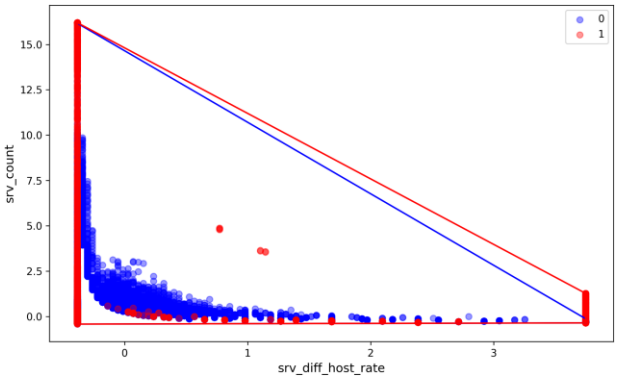

Fig. 3. Scatter with convex hull, *srv_diff_host_rate* vs *srv_count*

Using sklearn's Support Vector Classifier (SVC) with the Radial Basis Function (RBF) kernel, it is clearly visible the non-linear kernel performs better at separating the two classes using the same feature vectors, with class *bad* outlier values better separated.

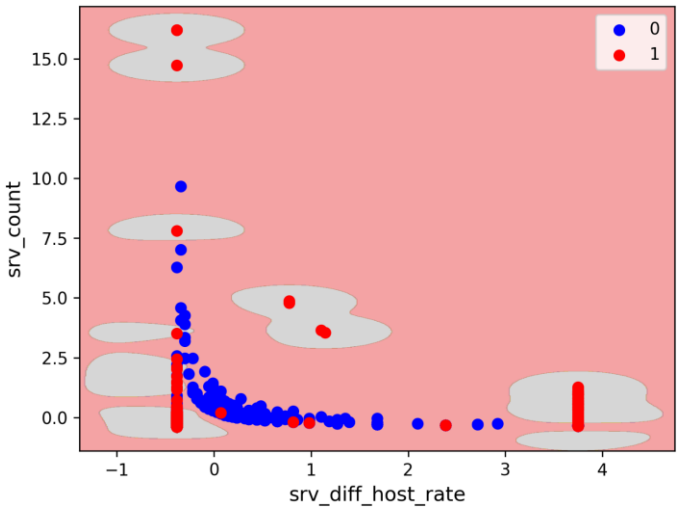

Fig. 4. Decision boundary with RBF kernal, *srv_diff_host_rate* vs *srv_count*

H. *Clustering*

Clustering is an unsupervised learning method that groups data based on a similarity measure without using target labels. It is a useful tool for discovering any distinct patterns present in the dataset. This study used *k*-means clustering, where *k* is number of clusters, and which looks to centre feature vectors around *k* centroids using Euclidean distance to select the closest centroid. The location of a centroid is adjusted to the average of vectors assigned to it. The aim is to maximize both inter-cluster homogeneity (variance within cluster) and intra-cluster separation (distance between clusters). *K* was tested in range 2 to 10. Fig. 5 shows *k*=2, with features *srv_diff_host_rate*, *srv_count* and *serror_rate*, to illustrate *k*-means in 3 dimensions. It is clear the clusters are not separable and further evidence of a non-linear problem.

Reducing the feature space to just 3 dimensions with principal component analysis (PCA) and clustering with *k*=5 shows improved inter-cluster homogeneity, especially along the x and y axis, and intra-cluster separation (Fig. 6). Given there are 5 attack categories, this result supports the testing of PCA transformed 2d tensor input for classification with proposed neural network models.

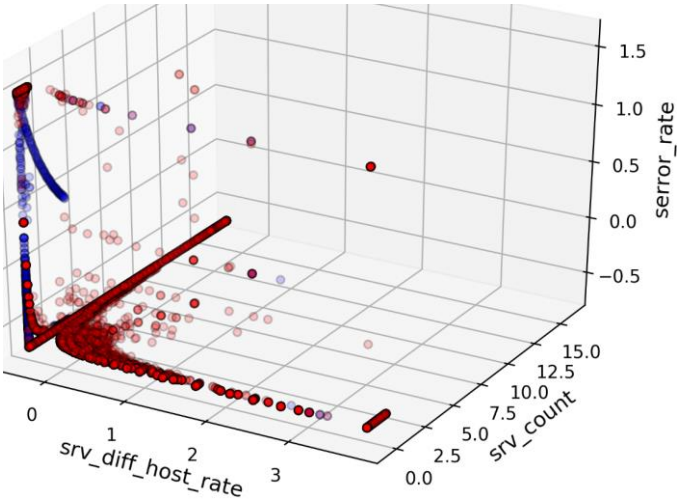

Fig. 5. *k*=2 clustering of 3 dimensions

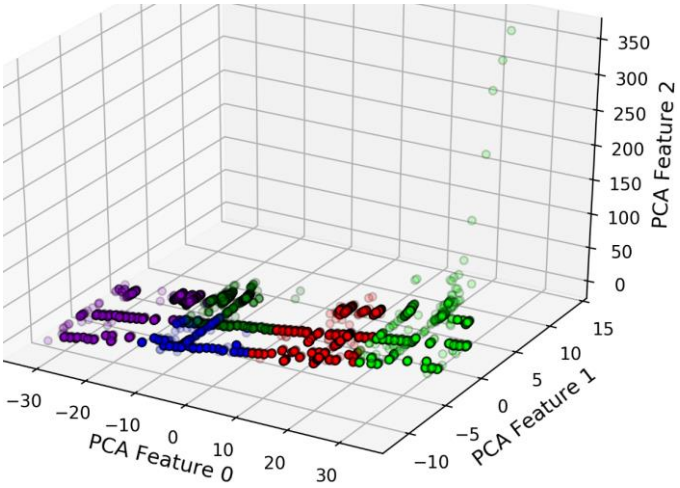

Fig. 6. PCA with 3 dimensions, *k*-means clustered with *k*=5

I. *Feature Selection*

Sklearn feature selection techniques were tested to rank features in their importance to making a correct prediction, including a univariate selector based on SelectKBest with chi-squared as the scoring function, and the Random Forest classifier (RFC) which exposes a feature importance list after class prediction. Using a subset of ranked features when training classification models can increase their ability to generalize whilst reducing dimensionality, complexity and training time. The top 20 features proposed by these 2 selectors, as listed in *Appendix B*, were used to subset the dataset which was then scored for accuracy using 10-fold cross validation with the XGBoost classifier making both binary and multiclass predictions.

Without feature selection, using all 31 features, XGBoost took 155s to complete binary classification with an accuracy of 99%, and multiclass (5-class) 701s runtime with accuracy 99.4%. Univariate feature selection of the top 20 features returned an accuracy of 99.12%, taking 115s to make a binary classification, and for multiclass an accuracy of 99.57% taking 500s. Random Forest top 20 for binary classification achieved accuracy of 99.14% taking 121s, with multiclass accuracy of 99.32% taking 531s. Acknowledging accuracy is very similar across all tests, feature sub-setting with RFC delivers best scores for both binary and multiclass, suggesting improved generalization capacity. Irrespective of number of classes, the computation time is substantially reduced. Higher *XGBoost* accuracy in multiclass prediction supports multiclass testing with neural networks. The final feature subset for NN classification will be the top 20 proposed by RFC (same for binary/multiclass), plus sparse minority class features in table II, all of which didn't make the top 20, but will be retained to maximise opportunity for minority class detection.

J. *Feature Scaling*

As highlighted earlier, most features are not normally distributed, exhibiting large positive skew and tail heaviness. The top 2 ranked features from RFC selection, *count* and *diff_srv_rate*, are clearly measured in different quantitative scale and magnitude, with maximum 511/1 and stdev 100/0.120 respectively. The Kernel Density Estimation (KDE) in Fig. 7 plots the data density for all selected features before any scaling. Features *src_bytes* and *dst_bytes*, UoM in bytes maxing at 500,000, clearly dominate overall scale, with many other features in an extremely narrow band centred around 0.

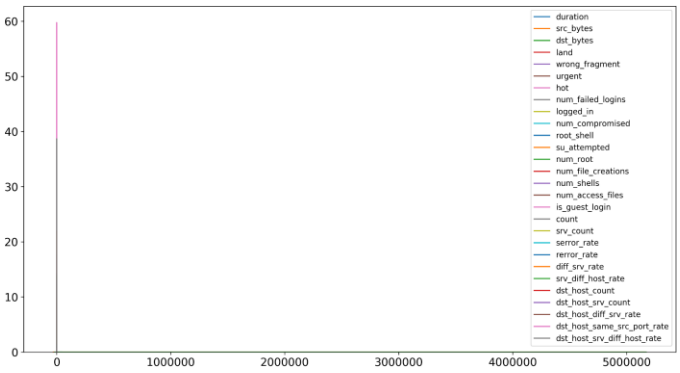

Fig. 7. Feature distribution before scaling

To manage dimension variance the data can be rescaled, with sklearn scalers including *StandardScaler*, *MinMaxScaler*, *QuantileTransformer* and *PowerTransformer* tested. Scoring accuracy with *XGBoost* after scaling achieved comparable results ranging 98.12% - 99.09% (binary) and 98.43% - 99.74% (multiclass). Given the heavily skewed, multi-scale distribution, with many features exhibiting mean close to 0, the *MinMaxScaler* will be used to rescale to fixed range 0-1. As Fig. 8 shows, after transformation the original data shape remains, preserving characteristics including outliers. Features now distributed within range 0-1 should better support model training by minimising the weights necessary to reduce loss function error between predicted and actual class, and therefore move layer output in the right direction of the activation function decision boundary according to the target label. Furthermore, "*Convergence is usually faster if the average of each input variable over the training set is close to zero*" [8].

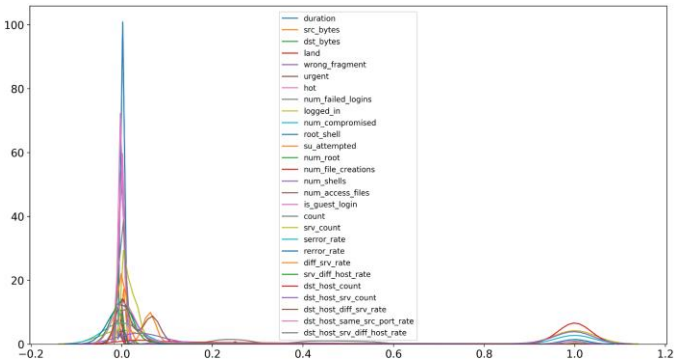

Fig. 8. Feature distribution after scaling with sklearn *MinMaxScaler* (0,1)

K. *Sampling*

Minority class attack categories *probe*, *r2l* and *u2r* combined represent just 2.1% of the population. Training any classifier on such imbalanced representation risks insensitivity to these rare classes, with misclassification of attacks as normal risking real world high cost. Several resampling methods were tested, including random oversampling and ADASYN (Adaptive Synthetic), however best results were obtained with imbalanced-learn's SMOTE (Synthetic Minority Over-sampling Technique) implementation [9] when scoring accuracy with *XGBoost* (98.81% binary, 99.22% multiclass). As proposed by [10], this method uses an "*over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement*" by "*taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the k minority class nearest neighbors*". The result is an increase to 439140 observations (from 145581), with all 5 attack categories equally represented.

L. *Categorical Feature Encoding*

NNs only work with numerical features, therefore *protocol_type*, *service* and *flag* are one-hot encoded, generating additional features with binary value for each original categorical value.

M. Network Evaluation

Success measures evaluating NN performance include accuracy (ACC), detection rate (DR) and false alarm rate (FAR). DR and FAR are commonly used for benchmarking IDS [11] [12] [13]. ACC is the ratio of correct predictions to total number of predictions. DR is the ratio of correctly predicted attacks to total number of attacks. FAR is the ratio of falsely predicted attacks to total number of non-attacks (normal). Formulas are defined below, with TP (true positive) representing correctly classified attacks, TN (true negative) normal activity correctly classified, FP (false positive) normal activity incorrectly classified as attack, and FN (false negative) attacks classified as normal.

$$ACC = (TP+TN) / (TP+TN+FP+FN)$$
$$DR = TP / (TP+FN)$$
$$FAR = FP / (TN+FP)$$

Implementing robust model evaluation, the dataset is first split 70/30 into train and test sets. Stratified *k*-fold cross validation with *k*=5 divides the training set into 5 subsets, each containing an equal percentage of samples from each target class to ensure balanced representation. Models will be validated against each *k*-fold after training on the combined 4 other folds, with learning from iterating *n* passes (epoch) over each 4-fold training set. Metrics scoring model training and validation on loss (measuring distance between predicted and actual target), ACC, DR and FAR are collected for each epoch/*k*-fold, with the mean for each epoch plotted as indicators of performance. Finally, the model is evaluated on the unseen 30% test set. A primary objective is understanding how models perform in detecting previously unseen attacks, therefore test set benchmarking of the model is what really matters as a guide to the ability of the network to generalise. Model optimisation, including configuration of epochs, layers, hidden units, activation functions, optimisers and regularization will only be based on results of training data to avoid information leak into unseen validation and test sets, thereby ensuring generalisation is maximised.

III. NN Solutions

A neural network is inspired by a biological model that combines multiple inputs from senses like eyes and ears to generate output signals. NNs are widely used in the field of machine learning, which author Tom M. Mitchell defines as "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E*" [14]. Evaluation of SLP/MLP types in supervised learning from the labelled, pre-processed KDD dataset and tasked with classifying network activity as *good* or *bad* follows.

A. *Single Layer Perceptron*

Dr Frank Rosenblatt of Cornell University is credited with defining the perceptron in 1957, demonstrating via mathematics and computer simulation that neural networks with variable weight connections could be trained to classify spatial patterns into prespecified categories [15]. It made sense to first benchmark NN performance with a simple, shallow architecture, hence the first proposed model is a single layer perceptron (SLP) with topology as illustrated in Fig. 9.

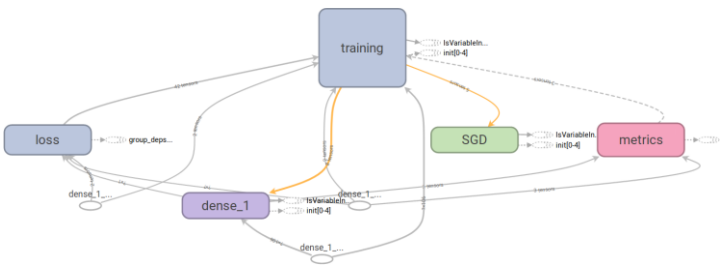

Fig. 9. Keras model for Single Layer Perceptron

An SLP is a single layer linear summation of all input values for binary classification and thus effectively a linear regression. For each training iteration (epoch) a random (stochastic) batch of 100 samples from the training data subset are fed into the network. All 106 features are connected to the output layer (*dense_1*) which for binary classification comprises a single neuron. Each feature has an associated weight coefficient, initialised to a random value. The neuron first sums the input value-weight dot product and adds an additional bias. In the proposed model a sigmoid activation function (Fig. 10) is assigned. If the input sum gives a sigmoid output of 0.7 the observation has a 70 percent probability of belonging to positive class 1 *bad*. It is this s-curve characteristic of sigmoid that supports the network in addressing our non-linear network attack problem.

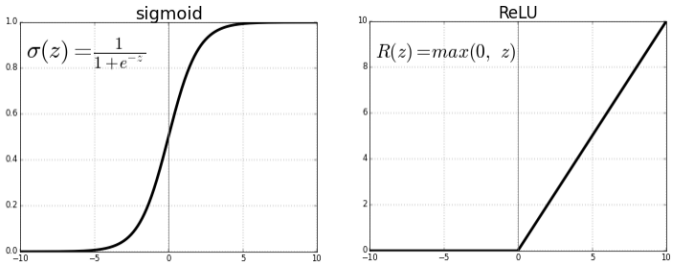

Fig. 10. Sigmoid and ReLU activation functions

Typical for binary classification and used here is the binary cross entropy loss function, which measures the difference between predicted and actual class. The predicted probability is derived from the activation function, and the ground truth from the training batch target label. For each sample, loss is calculated as the negative log of the predicted probability to actual class, therefore the loss "*penalty*" for an incorrect prediction rises exponentially the further the prediction is from the truth, with the actual class supervising the calculation of loss.

The proposed SLP used a stochastic gradient descent (SGD) optimizer, which takes the mean loss for all samples to calculate the gradient for the loss relative to the network's current weights. The weights are adjusted in the opposite direction from the gradient then propagated back up the network. In addition to the model calculating the standard metrics ACC and loss for each epoch, custom callback functions provide DR and FAR. Fig. 11 left graph shows loss reducing with each epoch as the predicted probability moves closer to the target class, with the network learning from mistakes, adjusting weights to compensate. The right graph shows DR. The green trace indicates slight overfitting (memorizing) of the data during training, especially in earlier epochs.

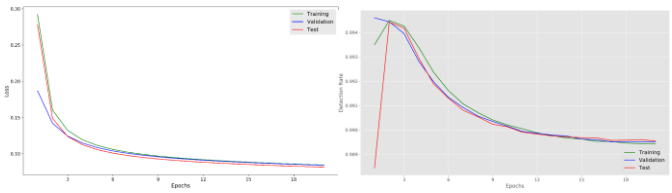
Fig. 11. SLP loss and detection rate

Both graphs suggest 20 epochs is sufficient, after which loss reduction tails off (finding global minimum) and detection rate maximised. Given the simplicity of the model it does a very good job, returning a detection rate of 0.99% on the unseen test dataset (Table III).

TABLE III
SLP PERFORMANCE METRICS

Metric	Train	Validation	Test
DR	0.990	0.990	0.990
FAR	0.146	0.131	0.142
ACC	0.963	0.966	0.963
Runtime (s)	799		208

However, the confusion matrix (Fig. 12) tells us that 1120 observations were classified *good* when in fact they were *bad*, with false alarms for 3257 *good* observations misclassified *bad*.

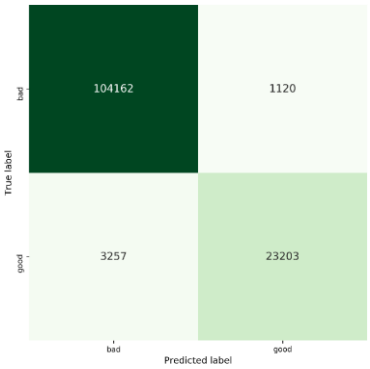

Fig. 12. SLP confusion matrix for binary classification

B. Multi-Layer Perceptron

Greater network complexity is now proposed as a means to increasing predictive performance. The multi-layer perceptron (MLP) is a feed forward network using back-propagation as described earlier, however consists of at least 3 distinct layers: an input layer, hidden layer(s) and an output layer. The topology for the Keras MLP model optimised for the binary classification task is shown in Fig. 13. It consists of an input layer (dense_1), 3 hidden layers (dense 2 – 4) and an output layer (dense_5). *Dense* is Keras terminology signifying fully connected layers where any single neuron of one layer is fully connected to all neurons of the following layer. As with the SLP, 1 neuron in the output layer is sufficient for binary classification.

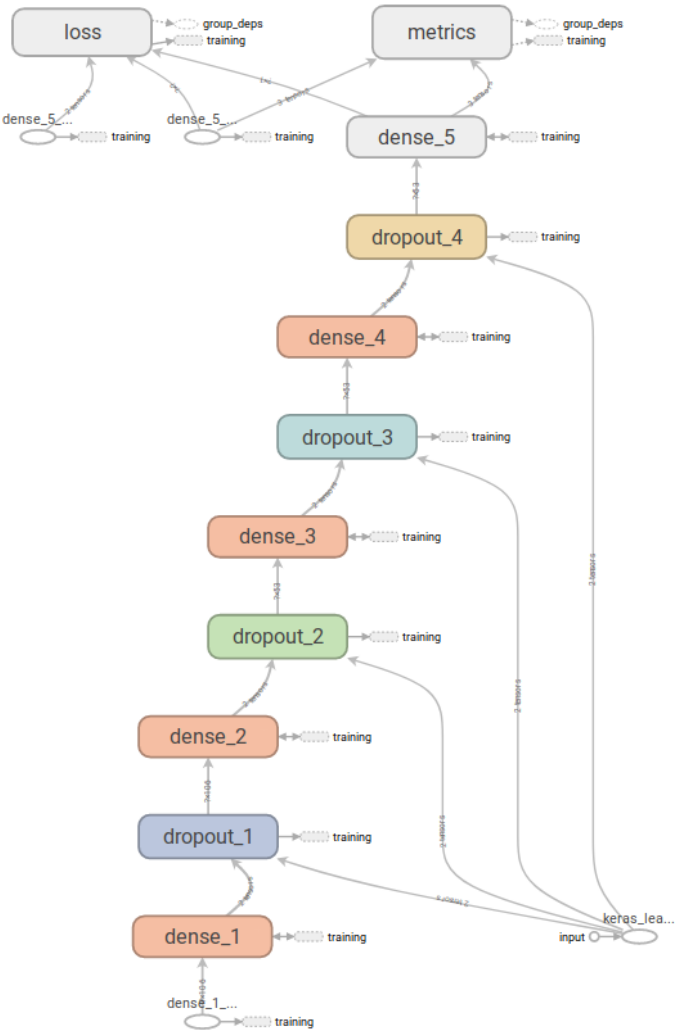

Fig. 13. Keras model for Multi Layer Perceptron

NNs are highly configurable, with myriad settings known as hyperparameters that can be changed to define both architecture and behaviour. It can be extremely time-consuming to manually configure, run and analyse each parameter one by one. To support the search for optimal network configuration, the python module Talos [16] is used, which can search a defined parameter space and score the network's

binary and multiclass classification performance on each permutation with DR, FAR and ACC metrics. The parameter boundaries consisted of the following: learning rate (0.0005, 0.005, 0.001), number of neurons in first layer (50% of input features, 80% of input features, all input features), number of neurons in hidden layers (50% of input features, 80% of input features, all input features), number of hidden layers (1, 2, 3), batch size (100, 500, 1000), dropout (0, 0.2, 0.5), optimizer (SGD, RMSprop). Hidden layer activation functions were fixed with ReLU. Output layer activation function for binary classifcation was fixed to use sigmoid, and softmax for multiclass. The generated search grid contained 8100 permutations, downsampled to a more reasonable random 81, the results logged with performance metrics in a .csv for later analysis. Epochs were fixed at 20 to avoid an additional permuation, and later manually tuned after review.

The heatmap in Fig. 14 shows correlation both between the hyperparameters themselves and validation DR following an 81 permutation parameter scan for MLP binary classification. *Dropout* has the strongest influence on metric val DR, and *batch size* the least. *Dropout* also has the largest negative correlation with *learning rate*. *Learning rate* has a strong positive correlation with number of neurons in *hidden layers.*

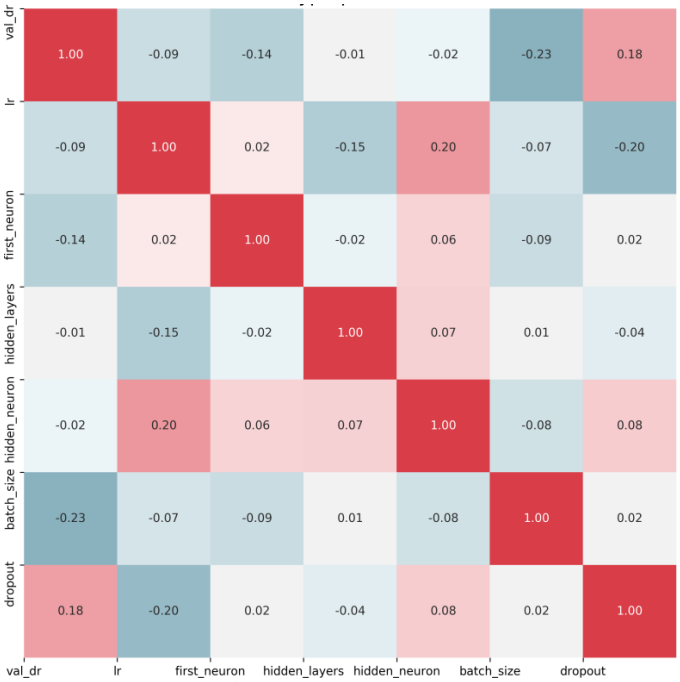

Fig. 14. MLP binary classification hyperparameter & validation DR correlation

For univariate analysis of hyperparameters as a function of val DR, the sklearn logistic regression classifier was used to define the baseline val DR of 0.983. This is subtracted from each hyperparameter permutation val DR to indicate val DR change in the following box plots. The left box plot Fig. 15 shows various learning rates as a function of val DR. Learning rate in combination with gradient, computed as a result of error loss, is used to set the adjusted weights. The plot indicates that defining any learning rate is more effective than default 0 and configured 0.001 rates, suggesting computed gradient is getting stuck in local minima. With increased learning rates gradient calculation is able to take larger steps and overcome local "*bumps*" in the direction towards the global minimum loss. However, with the larger learning rate steps 0.004/0.005, the weight adjustments are too much and cause the gradient to continually overshoot the global minimum, impacting val DR. A learning rate of ~0.003 appears optimal.

The right box plot (Fig. 15) illustrates batch size as a function of val DR. Batch size is the number of random samples drawn from the training set that are propagated through the network. During 1 epoch iteration, multiple batches will be propagated until the full training set is used. A batch size of 1000 has a detrimental effect on val DR, likely due to overfitting, supported by the authors of [17] who claim "*it has been observed in practice that when using a larger batch there is a degradation in the quality of the model, as measured by its ability to generalize*". Reduced batch sizes of 100 and 500 similarly influence the network with small positive effect.

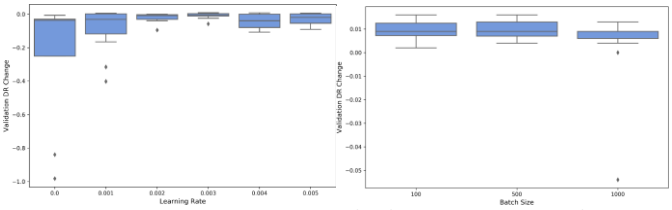

Fig. 15. Learning Rate and batch size as fn of val DR, binary classification

Layers are the fundamental component of networks, which typically consist of stacked layers through which data is progressively filtered into more meaningful form. The difference in how 1, 2 or 3 hidden layers can contribute to val DR is marginal as illustrated in Fig. 16 left boxplot. In hyperparameter permutation testing the best 3 hidden layer network scores val DR of 0.999, whereas the best 2 hidden layer scored 0.998. In many other problem domains 2 hidden layers may be considered sufficent for the sake of reduced model compelxity. However in IDS this small performance difference can make the difference between missed and detected malicious activity.

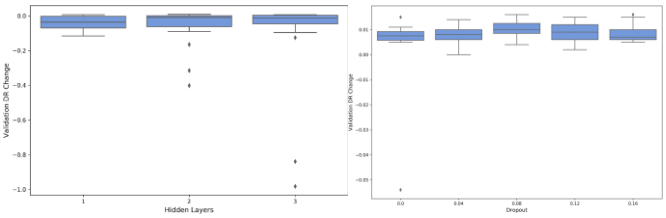

Fig. 16. # Hidden layers and dropout as fn of val DR, binary classification

The right box plot (Fig. 16) shows the impact different dropout configurations have on val DR. Invented by Hinton et al., [18], dropout is described as "*The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. This significantly reduces overfitting and gives major improvements over other regularization methods"*. In Keras dropout can be added as additional layers between fully connected (dense) layers. A dropout rate in range 0-1 is specified, being the fraction of features dropped. No dropout negatively impacts val DR, suggesting overfitting when using the full 106 features. Dropping 8% of features increases DR with improved generalisation. Raising the drop rate further to 0.16 (or 16% of features) lowers val DR, due to a adverse loss of feature information.

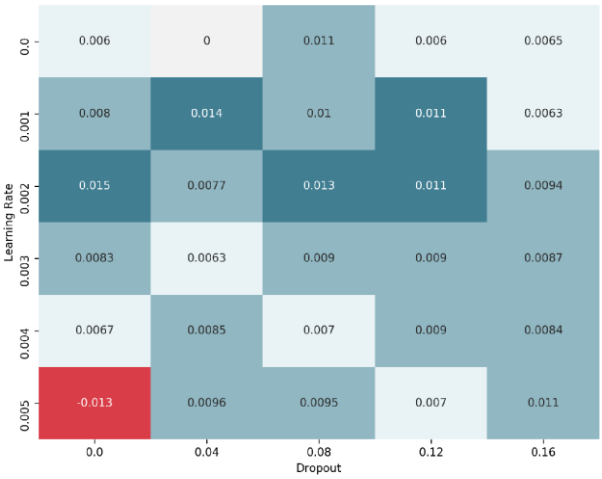

Fig. 17. Dropout and Learning Rate correlation as a fn of val DR

The search results show dropout and learning rate as important parameters with positive effect on val DR. However, parameters do not persist in isolation but influence network performance as a combined whole. The heatmap in Fig. 17 presents the correlation between dropout and learning rate as a function of val DR, suggesting several permutations are able to provide improved network performance.

DR for training, validation and test after each epoch is plotted (Fig. 18, left). Training is relatively stable until 17 epochs then declines. However, supported by dropout rates, the model is generalizing well on unseen validation and test data, with DR remaining above training results. FAR (Fig. 18, right) for validation and test remain mostly below

training, further indication of good generalisation, but again suggests stopping training at 17 epochs.

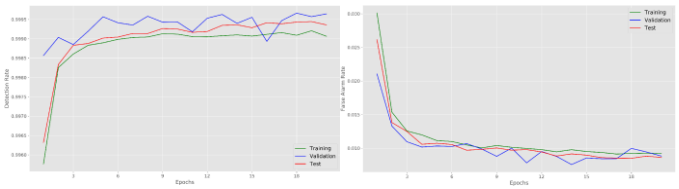

Fig. 18. MLP DR and FAR metrics after epoch iterations, binary classification

As with SLP, the MLP binary classification model uses the binary cross-entropy loss function. However here, in place SGD the RMSProp optimizer is used. Proposed by the renowned Geoffrey Hinton, RMSProp "*divides the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight*" [19] which helps to combat overshooting minima. In search space testing that included both SGD and RMSProp permutations. RMSProp featured in the top 18 solutions ranked by val DR. The ReLU (Recitifed Linear Unit, Fig. 10) activation function, which zeroes out negative input values and commonly used in NN's, is assigned to the input and all hidden layers.

Table IV presents final performance metrics for the proposed MLP binary classification NN, configured as follows: input layer using all 106 features, 3 hidden layers of 53 neuron units with the ReLU activation function and dropout rate of 0.08, and an output layer of 1 neuron unit with the sigmoid activation function, using the RMSProp optimizer with a binary cross-entropy loss function and learning rate of 0.0023. An impressive DR of 0.999% on the unseen 30% test set is achieved, 0.009 higher than the SLP. Unsurprisingly, runtime for both train/validation and test is almost double that of SLP.

Despite stratified k-fold cross-validation for class-balanced, randomized data drawn from the same preprocessed, scaled dataset as train and test, oscillation of the validation set is visible in both graphs (Fig. 18). It should also be noted that for all 3 data subsets the loss metric cumulatively increases after 8 epochs. For example, training rises from ~0.0125 up to ~0.0180 at epoch 19. Further study should include varying number of *k*-folds to adjust validation fold size, testing of batch normalisation to regulate weight updates between batches, and dynamically adjusting the dropout rate, learning rate and momentum over epochs via callback functions to try addressing these problems.

TABLE IV
MLP PERFORMANCE METRICS – BINARY CLASSIFICATION

Metric	Train	Validation	Test
DR	0.998	0.999	0.999
FAR	0.011	0.010	0.010
ACC	0.996	0.998	0.997
Runtime (s)	1423		355

The confusion matrix (Fig. 19) shows the MLP misclassifying 219 *good* network connections as *bad*, and 21 *bad* connections as *good*. This signifcantly improves upon SLP perfromance, where critically, 1120 *bad* connections were misclassified *good*. This clearly emphasises the value of extracting maximum performance from networks despite complexity cost, especially in this domain where the quantity of false negatives really does matter in best protection of systems and data.

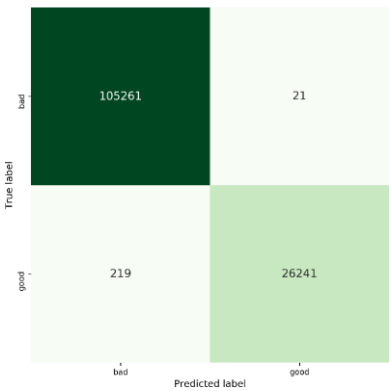

Fig. 19. MLP confusion matrix for binary classification

C. Multi-Layer Perceptron Multiclass Classification

With the dataset specification characterising each *bad* observation as 1 of 4 attack categories, it was of research interest to build an MLP variant for multiclass classification. Comparing performance with its binary equivalent would provide insight into any "*hard to predict*" classes. The same methodology was used to help derive optimal architecture for the multiclass model by using Talos to search and score a hyperparameter permutation specification. This resulted in a different configuration. The input layer now uses only 84 of the 106 available features. There are 2 hidden layers instead of 3 but with the same 53 neuron units with ReLU activation function and increased dropout rate of 0.12. Being a multiclass problem, the output layer contains 5 neuron units using softmax activation, with the neuron predicting the highest probability determining the winning class. The same RMSProp optimizer is used, but now with a categorical cross-entropy loss function and learning rate increased to 0.0032.

The heatmap (Fig. 20) shows correlation between hyperparameters and validation DR for MLP multiclass classification. Whereas *dropout* was most highly correlated with val DR for binary classification, *learning rate* exerts most influence for mutliclass, followed by *dropout*. This evidence supports the increased rates for both parameters in the final model as specified above. It also suggests that for most effective multiclass classification even greater generalisation is required, as implemented by increased dropout rate and reduction of hidden layers to 2. An increased learning rate proposes additional support is required to help avoid loss being trapped in incorrect local minima.

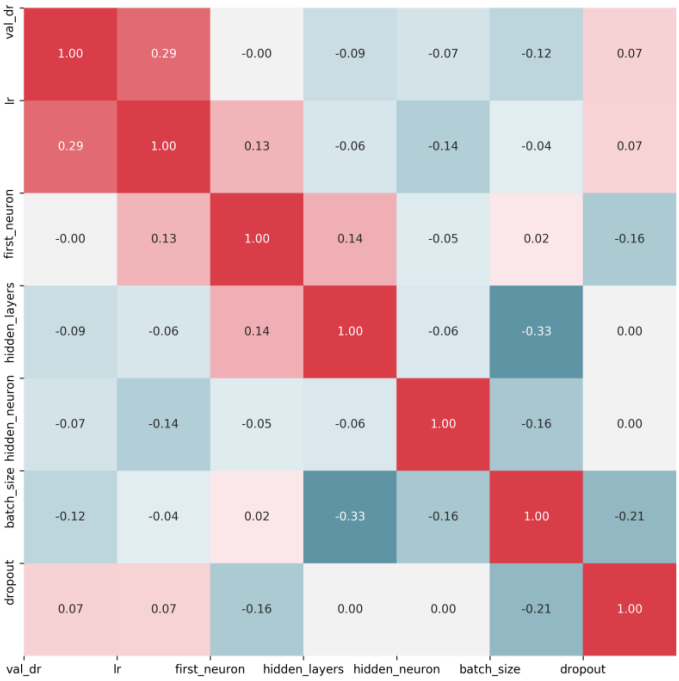

Fig. 20. MLP multiclass hyperparameter & validation DR correlation

DR and FAR performance plots for multiclass are shown in Fig. 21. With validation and test the right side of training in both, the model can be said to generalise well with no overfitting. Note for binary classification 19 epochs were required to achieve optimal results, whereas only 10 epochs were necessary for multiclass. It is proposed attributable to a combination of greater separation between classes and higher learning rate used, enabling faster convergence. No significant, repeated oscillation of validation is observable, unlike for binary classication, despite using the same 5 fold cross-validation.

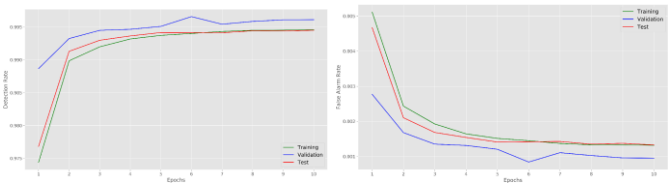

Fig. 21. MLP DR and FAR metrics after epoch iterations, multiclass

Table V presents final performance metrics for the MLP multiclass variant. Compared with SLP, DR is slightly worse across the board while FAR slightly improved. Runtime is significantly less than the binary equivalent, and below SLP runtime, indicating it's more sophisticated architecture supports convergence in fewer epochs.

TABLE V
MLP PERFORMANCE METRICS – MULTICLASS CLASSIFICATION

Metric	Train	Validation	Test
DR	0.991	0.994	0.992
FAR	0.001	0.001	0.001
ACC	0.991	0.994	0.992
Runtime (s)	592		155

The MLP multiclass confusion matrix (Fig. 22) provides a clearer view of the scale of hard to predict or misclassified classes. Category *r2l* is misclassified 25 times, most notably as *normal* 11 times. *Probe* is incorrectly classified 31 times, 15 times as *normal*. *Dos* claims the highest error rate amongst attack categories, misclassified 35 times, 19 times as *normal*. *U2r* is 100% correctly classified. Conversely, *normal* is misclassified as *r2l* a significant 169 times. Overall, *good* network connections were misclassified 219 times by the binary model and 235 times by the multiclass variant. *Bad* connections were misclassified as *good* 21 times with binary, and 45 times with multiclass.

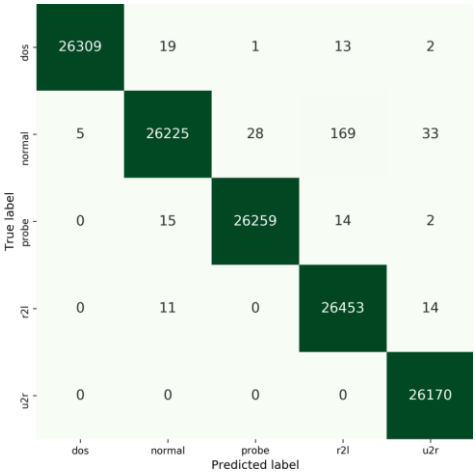

Fig. 22. MLP confusion matrix for binary and multiclass classification

IV. CONCLUSION

Tasked with the problem of classifying network activity as *good* or *bad*, extensive data preprocessing supported a standard logistic regression model in scoring 0.983%, and XGBoost 0.995%. However, with the proposed MLP for binary classification, a higher detection rate of 0.999% was achieved, made possible with extensive hyperparameter analysis and tuning to combat overfitting and maximise generalisation for optimal performance. Models were proven to be stable during training, performing exceptionally well on an unseen 30% test data subset. The effectiveness of NNs in dealing with a non-linear problem has been demonstrated. The simplest SLP resulted in a DR of 0.990%, showing even the most basic of networks are able to approximate any continuous function well. The best PCA feature-reduced dataset variant with 26 features managed a DR of 0.997%.

A well known criticism of NNs is that they can be a black box, making it difficult to explain the arrival of a given conclusion. Other models such as decision trees are more transparent. Despite that, in this problem domain where utmost performance in detecting malicious attacks is critical to protect systems and data, this price is worth paying.

There still remains some scepticism regarding the quality of the original KDD dataset. This is justified by the huge number of duplicate records, imbalanced nature of minority attack classes and possible mislabelling of *r2l* as *normal*. The author can only hope for a newer, more diligently verified and balanced dataset with latest attack types to be made available so IDS research continues better equiped.

Study solution https://github.com/jonpaulboyd/KDDCup1999

REFERENCES

[1] J. P. Anderson, "Computer Security Threat Monitoring And Surveillance," Fort Washington, 1980.

[2] G. Liu and X. Wang, "An integrated intrusion detection system by using multiple neural networks," in *2008 IEEE Conference on Cybernetics and Intelligent Systems*, Chengdu, China, 2008.

[3] S. Behera, A. Pradhan and R. Dash, "Deep Neural Network Architecture for Anomaly Based Intrusion Detection System," in *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, Noida, India, 2018.

[4] U. K. Archive, "KDD Cup 1999 Data," 1999. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[5] S. Songma, W. Chimphlee, K. Maichalernnukul and P. Sanguansat, "Classification via k-means clustering and distance-based outlier detection," in *2012 Tenth International Conference on ICT and Knowledge Engineering*, Bangkok, Thailand, 2012.

[6] Keras, "GitHub - keras-team/keras," [Online]. Available: https://github.com/keras-team/keras.

[7] "Scikit-learn,org," [Online]. Available: http://scikit-learn.org/stable/.

[8] Y. A. LeCun, L. Bottou, G. B. Orr and K.-R. Müller, "Efficient Backprop," in *Neural Networks: Tricks of the Trade*, Springer, 1998, pp. 9-48.

[9] imbalanced-learn, "imblearn.over_sampling.SMOTE," [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html.

[10] N. V. Chawla, K. W. Bowyer, L. O. Hall and P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research,* vol. 16, p. 321–357, 2002.

[11] Y.-F. Zhang, Z.-Y. Xiong and X.-Q. Wang, "Distributed Intrusion Detection Based On Clustering," in *Proceedings of the Fourth International Conference on Machine Learning and Cybernetics*, Guangzhou, 2005.

[12] N. Gao, L. Gao, Q. Gao and H. Wang, "An Intrusion Detection Model Based on Deep Belief Networks," in *2014 Second International Conference on Advanced Cloud and Big Data*, Huangshan, China, 2014.

[13] J. Kim, J. Kim, H. L. T. Thu and H. Kim, "Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection," in *2016 International Conference on Platform Technology and Service (PlatCon)*, Jeju, South Korea, 2016.

[14] T. M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[15] G. Nagy, "Neural networks-then and now," *IEEE Transactions on Neural Networks,* vol. 2, no. 2, pp. 316 - 318, 1991.

[16] pypi.org, "talos - PyPi," [Online]. Available: https://pypi.org/project/talos/.

[17] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima," in *International Conference on Learning Representations*, 2017.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[19] G. Hinton, *Neural Networks For Machine Learning - Lecture 6a - Overview Of Mini-Batch Gradient Descent.*

[20] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Communications Society,* vol. 18, no. 2, pp. 1153 - 1176, 2015.

APPENDIX A – FEATURE DISTRIBUTION STATISTICAL ANALYSIS

Column	Type	0 Count	Distinct	Min	Mean	25%	50%	75%	Max	Std	Skew	Kurt
duration	int64	134974 (92.71%)	2492	0	131.424	0	0	0	42448	1213.402	15.256	313.112
protocol_type	String	0	3									
service	String	0	66									
flag	String	0	11									
src_bytes	Int64	57872 (39.75%)	3299	0	3233.138	0	147	288	5135678	107324.844	45.718	2143.580
dst_bytes	Int64	67662 (46.48%)	10723	0	2856.367	0	105	1164	5155468	60803.256	74.364	6005.699
land	Int64	145561 (99.99%	2	0	0	0	0	0	1	0.012	85.301	7274.300
wrong_fragment	Int64	144460 (99.23%)	3	0	0.020	0	0	0	3	0.239	12.157	147.517
urgent	Int64	145577 (99.99%)	4	0	0	0	0	0	3	0.010	243.003	64053.110
hot	Int64	143625 (98.66%)	22	0	0.100	0	0	0	30	1.427	18.076	339.822
num_failed_logins	Int64	145518 (99.96%)	6	0	0.001	0	0	0	5	0.029	87.284	10967.415
logged_in	Int64	74030 (50.85%)	2	0	0.491	0	0	1	1	0.500	0.034	-1.999
num_compromised	Int64	144593 (99.32%)	21	0	0.015	0	0	0	281	1.248	203.667	43258.090
root_shell	Int64	145528 (99.96%)	2	0	0	0	0	0	1	0.019	52.382	2741.906
su_attempted	Int64	145571 (99.99%)	3	0	0	0	0	0	2	0.012	140.499	21051.007
num_root	Int64	144998 (99.60%)	18	0	0.026	0	0	0	306	1.389	183.195	37128.370
num_file_creations	Int64	145316 (99.82%)	18	0	0.004	0	0	0	28	0.178	104.413	12843.416
num_shells	Int64	145530 (99.96%)	3	0	0	0	0	0	2	0.020	59.085	3877.566
num_access_files	Int64	145130 (99.69%)	5	0	0.003	0	0	0	4	0.062	21.439	578.087
num_outbound_cmds	Int64	145581 (100.00%)	1	0	0	0	0	0	0	0	0	0
is_host_login	Int64	145581 (100.00%)	1	0	0	0	0	0	0	0	0	0
is_guest_login	Int64	144896 (99.53%)	2	0	0.005	0	0	0	1	0.068	14.475	207.539
count	Int64	2 (0.00%)	490	0	74.388	2.000	12.000	132.000	511	100.337	1.316	1.099
srv_count	Int64	2 (0.00%	470	0	13.012	2.000	8.000	15.000	511	30.737	10.869	147.591
serror_rate	Float64	101809 (69.93%)	92	0	0.291	0	0	1	1	0.453	0.921	-1.147
srv_serror_rate	Float64	102051 (70.10%)	51	0	0.292	0	0	1	1	0.454	0.920	-1.152
rerror_rate	Float64	129218 (88.76%)	77	0	0.108	0	0	0	1	0.309	2.517	4.355
srv_rerror_rate	Float64	128818 (88.49%)	51	0	0.108	0	0	0	1	0.309	2.527	4.410
same_srv_rate	Float64	1717 (1.18%)	99	0	0.655	0.080	1	1	1	0.447	-0.552	-1.659
diff_srv_rate	Float64	90035 (61.85%)	78	0	0.041	0	0	0.060	1	0.120	6.413	44.654
srv_diff_host_rate	Float64	112128 (77.02%)	64	0	0.093	0	0	0	1	0.242	2.988	7.913
dst_host_count	Int64	3 (0.00%)	256	0	181.470	78.000	255	255	255	99.097	-0.794	-1.142
dst_host_srv_count	Int64	3 (0.00%)	256	0	129.937	12.000	117.000	255	255	114.715	0.039	-1.885
dst_host_same_srv_rate	Float64	5847 (4.02%)	101	0	0.553	0.050	0.770	1	1	0.456	-0.155	-1.890
dst_host_diff_srv_rate	Float64	63316 (43.49%)	101	0	0.061	0	0.020	0.070	1	0.147	4.562	21.406
dst_host_same_src_port_rate	Float64	80614 (55.37%)	101	0	0.093	0	0	0.030	1	0.241	3.031	8.002
dst_host_srv_diff_host_rate	Float64	94790 (65.11%)	65	0	0.019	0	0	0.020	1	0.060	8.677	107.165
dst_host_serror_rate	Float64	97042 (66.66%)	100	0	0.292	0	0	1	1	0.452	0.923	-1.144
dst_host_srv_serror_rate	Float64	97586 (67.03%)	72	0	0.291	0	0	1	1	0.453	0.923	-1.147
dst_host_rerror_rate	Float64	124249 (85.35%)	101	0	0.110	0	0	0	1	0.306	2.490	4.286
dst_host_srv_rerror_rate	Float64	125252 (86.04%)	101	0	0.108	0	0	0	1	0.304	2.527	4.447
Label	String	0	23									
Target	Int64	87828 (60.33%)	2	0	0.397	0	0	1	1	0.489	0.422	-1.822

APPENDIX B – RANKED FEATURES BY UNIVARIATE AND RANDOM FOREST SELECTION

	Univariate		**Random Forest**	
Rank	**attack_category**	**label**	**attack_category**	**label**
XGBoost Acc. (cv=10)	99.57%	99.12%	99.72%	99.14%
XGBoost Runtime (s)	500s	115s	531s	121s
Rank				
1	service	duration	count	count
2	flag	protocol_type	diff_srv_rate	diff_srv_rate
3	src_bytes	service	src_bytes	src_bytes
4	wrong_fragment	flag	dst_host_srv_count	dst_host_srv_count
5	hot	src_bytes	flag	flag
6	num_failed_logins	wrong_fragment	dst_bytes	dst_bytes
7	logged_in	hot	dst_host_diff_srv_rate	serror_rate
8	root_shell	logged_in	serror_rate	dst_host_diff_srv_rate
9	num_shells	num_access_files	service	service
10	is_guest_login	count	dst_host_count	dst_host_count
11	count	srv_count	dst_host_srv_diff_host_rate	dst_host_srv_diff_host_rate
12	serror_rate	serror_rate	logged_in	logged_in
13	rerror_rate	rerror_rate	dst_host_same_src_port_rate	protocol_type
14	diff_srv_rate	diff_srv_rate	protocol_type	dst_host_same_src_port_rate
15	srv_diff_host_rate	srv_diff_host_rate	srv_count	hot
16	dst_host_count	dst_host_count	wrong_fragment	srv_count
17	dst_host_srv_count	dst_host_srv_count	hot	wrong_fragment
18	dst_host_diff_srv_rate	dst_host_diff_srv_rate	num_compromised	num_compromised
19	dst_host_same_src_port_rate	dst_host_same_src_port_rate	rerror_rate	rerror_rate
20	dst_host_srv_diff_host_rate	dst_host_srv_diff_host_rate	srv_diff_host_rate	srv_diff_host_rate
21	dst_bytes	dst_bytes	num_access_files	num_access_files
22	num_access_files	num_shells	num_shells	num_shells
23	su_attempted	num_file_creations	num_file_creations	num_file_creations
24	num_file_creations	su_attempted	su_attempted	su_attempted
25	protocol_type	num_compromised	duration	duration
26	duration	urgent	urgent	urgent
27	urgent	root_shell	root_shell	root_shell
28	num_root	num_root	num_root	num_root
29	land	land	land	land
30	srv_count	is_guest_login	is_guest_login	is_guest_login
31	num_compromised	num_failed_logins	num_failed_logins	num_failed_logins