Florian Neziri

Birkbeck Student ID: 13172387

MSc Data Science

Applied Machine Learning – Individual Practical Project

Word Count: 2141

Abstract

The aim of this paper is to build a machine learning classifier capable of distinguishing between intrusive traffic (attacks) and good (normal) traffic from the Aegean WiFi Intrusion/threat Dataset (AWID). Additional features to the dataset were created and added using a Stacked Autoencoder (SAE), and then 3 selection techniques were applied to the larger dataset: Mutual Information (MI) - a filter method; Feature Importance (FI) using an Extra Trees classifier – an embedded method; and Recursive Feature Elimination (RFE) – a wrapper method.

The top 5 features were selected using these methods and then used on the candidate algorithms: Logistic Regression (LR), Classification and Regression Trees or Decision Trees (CART), and Multilayer Perception Neural Network (MLP). Each algorithm was then tuned to find the optimal hyperparameters, before being evaluated using metrics such as Accuracy, Precision, Detection Rate/Recall, False Negative Rate (FNR), False Alarm Rate (FAR), F1 Score, and Matthews Correlation Coefficient (MCC) to determine the most accurate combination of algorithm and feature selection technique. This was found to be the MLP neural network with FI feature selection applied to the dataset.

Introduction and Dataset

The task of cyber threat detector learning is to build a predictive model (i.e. a machine-learning classifier) capable of distinguishing between “intrusive” traffic, called threats or attacks, and “good” normal traffic.

The Aegean WiFi Intrusion/threat Dataset (AWID) project was prepared and managed by George Mason University and University of the Aegean. The AWID dataset categorises the attacks according to the methodology of execution. Attacks that have similar patterns of expression fall under one of the groups: a. injection attacks, b. flooding attacks, c. impersonation attacks, d. passive attacks [1].

We will use the reduced CLS portion of the AWID dataset because it serves as a useful starting point for preliminary research, and affords a baseline against which the new built models can be compared with the state-of-the-art Deep Feature Extraction and Selection (DFES) method, and other methods using the same reduced CLS portion.

The first row of each dataset gives variable numbers. The original dataset has 154 input variables and 1 target variable however two of the input variables numbered 4 and 7 have been removed from both datasets as they provide temporal information which may cause unfair prediction. The training set has 97044 observations while testing set has 40158 observations.

Constructing and Selecting Features

A. Feature Creation

A stacked autoencoder (SAE) consists of multiple autoencoders connected from each layer to the subsequent layer. The output of the previous encoder is the input of the next encoder and from this structure, higher representations (such as features) of the input data can be found.

Adding more layers helps the autoencoder learn more complex coding [2]. We must be careful not to make the autoencoder too powerful, otherwise the autoencoder will simply learn to copy its inputs to the output, without learning any meaningful representation [3].

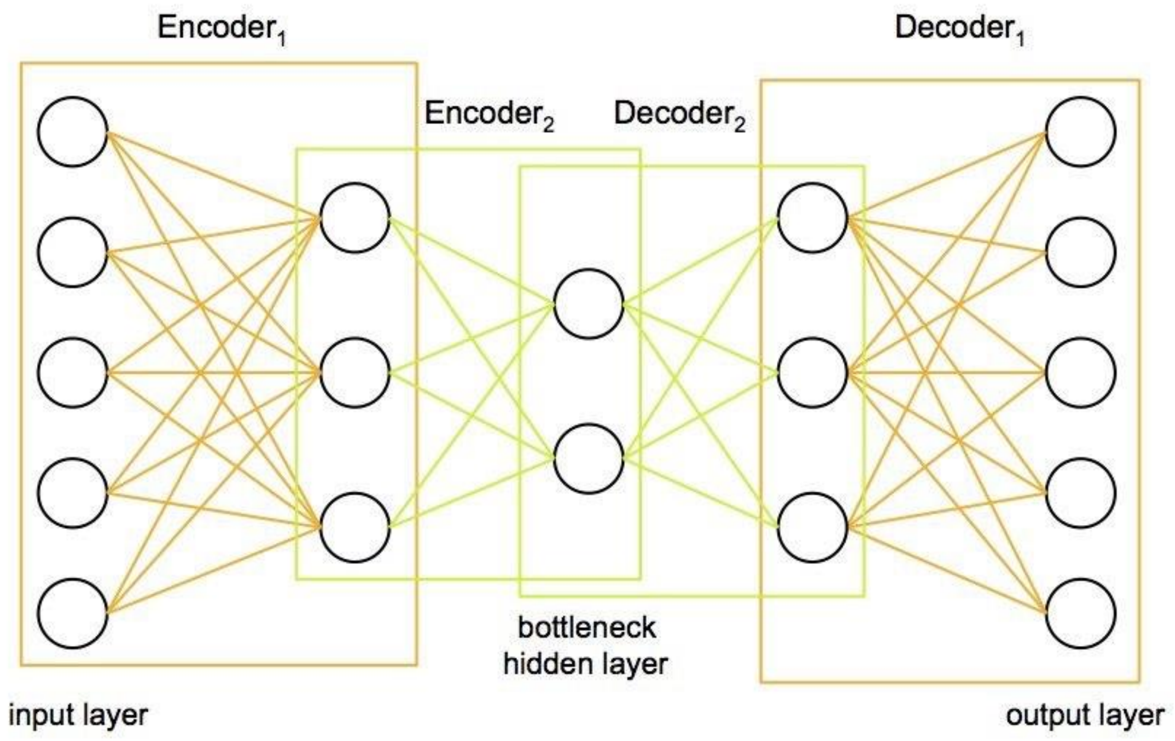

Figure 1: Diagram of a stacked autoencoder [4]

An SAE was chosen because a single AE behaves too greedily and important information for accurate classification of the target class could be discarded. The SAE prevents this by gradually refining the neurons in the hidden layers. The SAE learns a better representation of the input data than a single AE. However, as more encoders need to be trained, the training time and complexity of model are increased. [5]

For the number of hidden neurons for two encoder layers, 100 and 50 were chosen respectively, as these were found to be optimal for the WID impersonation dataset according to Aminanto et al. [6].

B. Feature Selection

It is important to apply feature selection to the data features that are used to train the machine learning models selected. By employing feature selection techniques, we aim to select those features in the data that contribute most to the output which we are in interested in – in this case detecting normal traffic as opposed to intrusive traffic (attacks).

Having the irrelevant features in the data can decrease the accuracy of the ML algorithms. The 3 main benefits of performing selection before modelling the data are:

1. Reduce overfitting: led redundant data means less opportunity to make decision based on noise
2. Improves accuracy: modelling accuracy improves as we have less misleading data
3. Reduces training time: less data means our chosen algorithms train faster.

The SAE created 50 new features which were combined with the original 152, to give us 202 features. Different feature selection techniques were then applied to the larger dataset: Feature Importance (FI) using the Extra Trees classifier, Mutual Information (MI), and Recursive Feature Elimination (RFE).

MI is a quantity that measures the mutual dependence between two random variables; how much information one random variable has about another. In other words, it is the indication of the reduction in uncertainty of one random variable when given the knowledge about another [5]. It is an example of a filter selection technique.

An Extra Trees classifier, in FI, implements an estimator that fits a number of randomised decision trees (in this case extra trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control overfitting [7]. It is an example of an embedded feature selection technique.

The goal of RFE is to select features by recursively considering smaller and smaller sets of features and building a model on the attributes that remain [8]. RFE uses the model accuracy to identify which attributes/combination of attributes contribute the most to predicting the target attribute (attacks/intrusive traffic). It is an example of a wrapper selection technique.

When the MI selection technique was run on the 202 variables (152 from the original dataset + 50 created by the SAE), 152 out of the 202 features had non-zero MI values – including 47 out 50 created variables – and the 5 variables with the highest MI values were: 8, 9, 38, 167, 195. Using the FI, 111 out of the 202 features had non-zero FI values – including 41 out of 50 created features - and the 5 variables with the highest importance were: 51, 67, 71, 170, 197. Lastly, for the RFE, the top 5 variables were: 107, 163, 170, 179, 196.

Building ML Algorithms

The chosen algorithms used were: Logistic Regression, CART (Classification and Regression Trees or Decision Trees), and a Multilayer Perception Neural Network (MLP). These algorithms were then tuned using both the grid and randomised search methods.

Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid. Random search is an approach to parameter tuning that will sample algorithm parameters from a random distribution (i.e. uniform) for a fixed number of iterations. A model is constructed and evaluated for each combination of parameters chosen [9].

Logistic regression assumes a Gaussian distribution for the numeric input variables and is used to estimate the probability that an instance belongs to a particular class e.g. is this good (normal) traffic, or is it intrusive traffic (an attack). If the estimated probability is greater than 50%, then the model predicts that the instance belongs to the positive class (intrusive traffic/attacks) labelled 1, and otherwise it predicts that that it belongs to the negative class (normal traffic) labelled 0. This makes it a binary classifier [10].

The logistic regression function is made up of several hyperparameters which we can tune to find the optimal parameters for the logistic regression classifier. The results of this tuning is shown in table 1 in the appendix. The parameters chosen for tuning were: **Penalty** – which are used to regularise the logistic regression model; **C** - which controls the penalty (regularisation) strength of the logistic regression model (the higher the value of C, the less the model is regularised); and **Solver** - the algorithm to use in the optimisation problem [11].

CART constructs a binary tree from the training data. The algorithm works by first splitting the training set into two subsets, using a single feature and a threshold value evaluated on that feature. The feature and threshold are chosen as the pair that minimise a cost function (thus producing the purest subsets). Once the CART algorithm has successfully split the training data in two, it splits the subsets using the same logic, then the sub-subsets produced by this split, and so on, recursively.

It stops recursing once it reaches maximum depth or if it cannot find a split that will reduce impurity. Maximum depth is a hyperparameter of the CART algorithm, defined as **max_depth** and is one of the hyperparameters that we have tuned in table 2, along with **min_samples_split** (the minimum number of samples a node must have before it can be split) and **min_samples_leaf** (the minimum number of samples a leaf node must have) [12].

A Multilayer Perceptron is a class of artificial neural network (ANN) that is composed of one input layer, one or more hidden layers, and a final output layer. The input layer contains of a set of neurons representing the input features, which we can specify with the input_dim argument in the Sequential model from the Keras API. We can then add layers to the input layer using the Dense class in Keras until we are satisfied with the network structure. In this paper, a fully-connected network structure with three layers was used.

We can specify the number of neurons or nodes in the layer as the first argument, and specify the activation function using the activation argument. We will use the rectified linear unit activation function referred (ReLU) on the first two layers and the sigmoid function in the output layer to ensure our network output is between 0 and 1.

After defining the model, it needs to be complied. When compiling, we must specify the loss function to use to evaluate a set of weights and the optimizer which is used to search through different weights for the network. In this case, we will use cross entropy as the loss argument. This loss is for a binary classification problems and is defined in Keras as **binary_crossentropy**. We will define the optimiser as the gradient descent algorithm **adam**. This is a popular version of gradient descent because it automatically tunes itself and gives good results in a wide range of problems [13].

Once the model is defined and compiled, we are ready to execute the model on some data. The training process will run for a fixed number of iterations through the dataset called epochs, that we must specify using the epochs argument. We must also set the number of dataset rows that are considered before the model weights are updated within each **epoch**, called the batch size and set using the **batch_size** argument.

The optimizer model design component, as well as the batch size and epoch hyperparameters were tuned using grid and random search, with the results shown in table 3.

Evaluation and Conclusion

Several measures were used to evaluate the machine learning models discussed in this paper. These were: detection accuracy (Accuracy); Detection Rate/Recall (DR), which is the ratio of correctly detected attacks to total number of attacks; Precision, which measures the number of correctly identified attacks; the F1 Score, which can be interpreted as a weighted average of the precision and recall; the Matthews correlation coefficient (MCC), which is the coefficient between detected and observed behaviours; the False Alarm Rate (FAR), which is the ratio of the number of normal instances misclassified as attacks against the total number of normal instances; and the False Negative Rate (FNR), which represents the number of undetected attacks.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

$$DR(recall) = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$FAR = \frac{FP}{TN + FP}$$

$$FNR = \frac{FN}{FN + TP}$$

$$F_1 = \frac{2TP}{2TP + FP + FN}$$

$$Mcc = \frac{(TP * TN) - (FP * FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Figure 2: Formulas for the various methods to evaluate the machine learning classifiers [14].

As can be seen in table 4b, logistic regression performed best on the MI selected top 5 features, recording the best scores for accuracy, DR, F1, MCC and FAR, with the scores for precision and FNR close to the highest scores recorded across the datasets. For the CART algorithm, in table 5b, the algorithm once again performed best on the MI selected top 5 features with best scores in every measure other than FAR (which was marginally worse than the best performing dataset). For the MLP neural network, in table 6b, the algorithm performed best on the FI selected top 5 features on all measure other than FAR by quite some margin (despite performing the worst of the 4 datasets on FAR, it is still a very low false alarm rate).

The datasets the different algorithms performed best on were then combined into table 7b so we could compare which of the algorithms performed best overall. Based on this comparison, we can conclude that the most effective algorithm and dataset combination was the MLP neural network with Feature Importance selection technique, as it outperformed the other two best performing combinations in all measures other than detection rate and false negative rate.

Appendix

Table 1 - Logistic Regression hyperparameter tuning on 3 different datasets utilising different feature selection techniques: Feature Importance, Mutual Information, Recursive Feature Elimination

	Grid Search			Random Search		
	FI	MI	RFE	FI	MI	RFE
C	0.0001	0.01	100	0.0001	0.01	4.5
Penalty	l2	l2	l1	l2	l2	l1
Solver	saga	saga	liblinear	saga	saga	liblinear
Best Score	0.9622	0.9349	0.9689	0.9636	0.9362	0.9685

Table 2 - CART hyperparameter tuning on 3 different datasets utilising different feature selection techniques: Feature Importance, Mutual Information, Recursive Feature Elimination

	Grid Search			Random Search		
	FI	MI	RFE	FI	MI	RFE
Criteria	gini	entropy	entropy	gini	entropy	entropy
max_depth	6	8	4	7	8	4
min_samples_split	6	2	2	6	4	8
min_samples_leaf	3	3	1	4	2	1
Best Score	0.9623	0.9003	0.9812	0.9622	0.9003	0.9812

Table 3 – MLP Neural Network hyperparameter tuning on 3 different datasets utilising different feature selection techniques: Feature Importance, Mutual Information, Recursive Feature Elimination

	Grid Search			Random Search		
	FI	MI	RFE	FI	MI	RFE
Optimizer	adam	sgd	adam	adam	sgd	adam
Epochs	100	50	100	121	81	109
Batch Size	100	500	500	192	483	499
Best Score	0.9393	0.8892	0.9492	0.9403	0.8646	0.9484

The following tables display evaluation metrics on the 3 machine learning classifiers used, on 4 different datasets: the larger dataset which includes SAE created features (labelled as Full), and the top 5 features from the Feature Importance (FI), Mutual Information (MI), and Recursive Feature Elimination (RFE) feature selection techniques.

Table 4 – Logistic Regression

4a: Confusion Matrices

		Predicted Class							
		Full		MI		FI		RFE	
		Normal	Attack	Normal	Attack	Normal	Attack	Normal	Attack
Actual Class	Normal	19995	84	18095	1984	19555	524	19986	93
	Attack	18643	1436	1449	18630	11381	8698	19155	924

4b: Evaluation Metrics

	Evaluation Metric						
	Accuracy	DR	Precision	F1	MCC	FAR	FNR
Full	0.5337	0.0715	**0.9447**	0.1330	0.1764	**0.0042**	0.9285
MI	**0.9145**	**0.9278**	0.9037	**0.9156**	**0.8293**	0.0988	**0.0722**
FI	0.7035	0.4332	0.9432	0.5937	0.4839	0.0261	0.5668
RFE	0.5207	0.0460	0.9086	0.0876	0.1317	0.0046	0.9540

Table 5 – CART

5a: Confusion Matrices

		Predicted Class							
		Full		MI		FI		RFE	
		Normal	Attack	Normal	Attack	Normal	Attack	Normal	Attack
Actual Class	Normal	13918	6161	12785	7294	19873	206	19751	328
	Attack	13143	6936	8	20071	19922	157	20043	36

5b: Evaluation Metrics

	Evaluation Metric						
	Accuracy	DR	Precision	F1	MCC	FAR	FNR
Full	0.5193	0.3454	0.5296	0.4181	0.0412	0.3068	0.6546
MI	**0.8181**	**0.9996**	**0.7335**	**0.8461**	**0.6829**	0.3633	**0.0004**
FI	0.4988	0.0078	0.4325	0.0154	-0.0129	**0.0103**	0.9921
RFE	0.4927	0.0018	0.0989	0.0035	-0.0767	0.0163	0.9982

Table 6 – MLP

6a: Confusion Matrices

		Predicted Class							
		Full		MI		FI		RFE	
		Normal	Attack	Normal	Attack	Normal	Attack	Normal	Attack
Actual Class	Normal	19958	121	19885	194	19595	484	19996	83
	Attack	18600	1479	18663	1416	2623	17456	20061	18

6b: Evaluation Metrics

	Evaluation Metric						
	Accuracy	DR	Precision	F1	MCC	FAR	FNR
Full	0.5338	0.0737	0.9244	0.1364	0.1729	**0.0060**	0.9263
MI	0.5304	0.0705	0.8795	0.1306	0.1551	0.0097	0.9295
FI	**0.9226**	**0.8694**	**0.9730**	**0.9183**	**0.8501**	0.0241	**0.1306**
RFE	0.4984	0.0009	0.1782	0.0018	-0.0323	0.0041	0.9991

Table 7 – Combined best results

7a: Confusion Matrices

		Predicted Class					
		LR MI		CART MI		MLP FI	
		Normal	Attack	Normal	Attack	Normal	Attack
Actual Class	Normal	18095	1984	12785	7294	19595	484
	Attack	1449	18630	8	20071	2623	17456

7b: Evaluation Metrics

	Evaluation Metric						
	Accuracy	DR	Precision	F1	MCC	FAR	FNR
LR MI	0.9145	0.9278	0.9037	0.9156	0.8293	0.0988	0.0722
CART MI	0.8181	**0.9996**	0.7335	0.8461	0.6829	0.3633	**0.0004**
MLP FI	**0.9226**	0.8694	**0.9730**	**0.9183**	**0.8501**	**0.0241**	0.1306

Bibliography

[1] C. Kolias, G. Kambourakis, A. Stavrou and S. Gritzalis, "Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 184-208, Firstquarter 2016, doi: 10.1109/COMST.2015.2402161.

[2] A Geron 'Representation Learning and Generative Learning Using Autoencoders and GANs' in *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow,* 2nd Edition 2019, ch. 17 pp. 572

[3] A. Dertat, 'Applied Deep Learning - Part 3: Autoencoders', towardsdatascience.com, https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798#a265 (accessed January 10, 2021)

[4] Briot, Jean-Pierre & HADJERES, Gaëtan & Pachet, Francois. (2019). Deep Learning Techniques for Music Generation - A Survey.

[5] S. J. Lee et al., "IMPACT: Impersonation Attack Detection via Edge Computing Using Deep Autoencoder and Feature Abstraction," in IEEE Access, vol. 8, pp. 65520-65529, 2020, doi: 10.1109/ACCESS.2020.2985089.

[6] M. E. Aminanto, R. Choi, H. C. Tanuwidjaja, P. D. Yoo, and K. Kim, "Deep abstraction and weighted feature selection forWi-Fi impersonation detection," IEEE Trans. Inf. Forensics Security, vol. 13, no. 3, pp. 621-636, Mar. 2018.

[7] Scikit Learn, 'sklearn.ensemble.ExtraTreesClassifier', scikitlearn.org, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html (accessed January 10, 2021)

[8] Scikit Learn, 'sklearn.feature_selection.RFE', scikitlearn.org, https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html#sklearn.feature_selection.R (accessed January 10, 2021)

[9] J. Brownlee, 'How to Tune Algorithm Parameters with Scikit-Learn', machinelearningmastery.com, https://machinelearningmastery.com/how-to-tune-algorithm-parameters-with-scikit-learn/ (accessed January 10, 2021)

[10] A Geron, 'Training Models' in *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow,* 2nd Edition 2019, ch. 4 pp. 142-147

[11] Scikit Learn, 'sklearn.linear_model.LogisticRegression', scikitlearn.org, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (accessed January 10, 2021)

[12]] A Geron, 'Decision Trees' in *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow,* 2nd Edition 2019, ch. 6 pp. 179-182

[13] J. Brownlee, ‘Your First Deep Learning Project in Python with Keras Step-By-Step’, machinelearningmastery.com, https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (accessed January 10, 2021)

[14] DEMISe: Interpretable Deep Extraction and Mutual Information Selection Techniques for IoT Intrusion Detection