SIT719 Security and Privacy Issues in Analytics

Task 5.1 – End-to-End Project: Multiple Classifier Systems for Intrusion Detection

The use of machine learning to detect network security breaches is a promising and rapidly growing area of research. Malicious attacks against the availability, integrity and privacy of networked services and data incur high financial, social and political costs. Intrusion Detection Systems (IDS) have become a vital part of networked systems, providing offline and, increasingly online, real-time detection of a variety of attacks. While there is a wealth of real-time information available for monitoring networked systems – and therefore a very large number of features and observations to train machine learning models – the very size and variety of this information poses a difficult problem. In addition, there is a wide and ever-changing variety of attacks, each of which occurs with a different frequency – often inversely related to its severity or impact – and therefore there is a highly imbalanced set of target classes to be learnt. Reflecting these real-world difficulties, the NSL-KDD dataset is commonly used to research the application of machine learning to intrusion. We propose a static Multiple Classifier System (MCS) to address the complex feature space and imbalanced classes contained in the dataset. First, we explored a variety of hyperparameter optimizations before training a range of classifiers on the full KDTrain+ feature set. We then applied several naïve *'a posteriori'* [1][2] strategies to select an Ensemble of Classifiers (EoC), including heuristic, overall and per target class performance metric analysis to maximize diversity [3] and accuracy. Depending on the strategy this achieved between 82.75% and 85.09% accuracy on the KDDTest+ dataset. To gain a better understanding of the maximum theoretical (or 'oracle' [4]) accuracy of the MCS technique we applied the same strategies directly on the KDDTest+ set, which yielded an 85.71% accuracy.

1. Background

The NSL-KDD dataset was proposed as a replacement for the KDD'99 dataset in 2009 [5]. It addresses several problems in the original dataset, such as eliminating redundant records while maintaining diversity, and inverting the proportion of attack types in the original dataset to maximise the difficulty and better reflect real-world scenarios [6]. The dataset contains 125,973 training records and 22,544 test records. Each record is made up of 41 features plus a labelled attack type. *Table 1* lists these features and the type of values each field contains (adapted from [7]). Once the nominal and binary features are OneHot/Dummy encoded there are a total of 122 features. In addition, a tally of the number of correct predictions from 21 base test models is included in a *#success_pred* field (see). The 22 attack types are sorted into 5 target classes: Benign, DoS, Probe, R2L and U2R. The type, description, number and proportion of records is outlined in *Table 2*.

Type	Features
Nominal	Protocol_type (2), Service (3), Flag (4)
Binary	Land (7), logged_in (12), root_shell (14), su_attempted (15), is_host_login (21), is_guest_login (22)
Numeric	Duration (1), src_bytes (5), dst_bytes (6), wrong_fragment (8), urgent (9), hot (10), num_failed_logins (11), num_compromised (13), num_root (16), num_file_creations (17), num_shells (18), num_access_files (19), num_outbound_cmds (20), count (23) srv_count (24), serror_rate (25), srv_serror_rate (26), rerror_rate (27), srv_rerror_rate (28), same_srv_rate (29) diff_srv_rate (30), srv_diff_host_rate (31), dst_host_count (32), dst_host_srv_count (33), dst_host_same_srv_rate (34), dst_host_diff_srv_rate (35), dst_host_same_src_port_rate (36), dst_host_srv_diff_host_rate (37), dst_host_serror_rate (38), dst_host_srv_serror_rate (39), dst_host_rerror_rate (40), dst_host_srv_rerror_rate (41)

Table 1: Features and Value Type

Class	Description	Train Dataset		Test Dataset	
Benign	Normal behavior of the network	67343	53.46%	9711	43.08%
DoS	Denial of Service is when an attacker depletes a network resource so as to deny the legitimate availability of that resource	45927	36.46%	7636	33.87%
Probe	Probing, is surveillance or other information gathering attacks usually for the purposes of further exploitation	11656	9.25%	2423	10.75%
R2L	Remote-2-Local is when a remote attacker without a normal login exploits the network to take control of a local resource	995	0.79%	2574	11.42%
U2R	Unauthorised-2-Local is when an attacker with a normal login exploits a vulnerability to gain elevated or root privileges	52	0.04%	200	0.89%
All		125973	100.00%	22544	100.00%

Table 2 Attack Types and Proportion

When the dataset was compiled each of the records for an attack category was included based on an inverse proportion to the number of times it was predicted by 21 learning models: 7 classical machine learning algorithms – J48 ,Naïve Bayers, NB Tree, Random Forest, Multilayer Perceptron and SVM – with 3 models for each classifier trained on subsets of the entire KDD dataset. The average accuracy for those models (excluding SVM) on the

original KDDTest dataset ranged between 81.66% and 93.82%. *Figure 2* shows the accuracy of the tested models on the (then) newly formulated KDDTest+ dataset [6]. We regard these results as a reasonable lower bound benchmark for single classifier accuracy. A brief review of recent literature (see *Table 3*) reveals that there have recently been some significant gains in accuracy using ensemble and RNN methods, usually combined with careful feature extraction methods.

Model	Year	Feature Extractor	Features	Classifier	Train	Test
Our Method 'Train' EoC	2020	None	122	Static Multiple Classifier System (DTC, GDB, MLP)	99.88%	85.09%
Our Method 'Oracle' EoC	2020	None	122	Static Multiple Classifier System (DTC, LRG, GDB, ADB, RFC)	99.62%	85.71%
Adaptive Ensemble[8]	2019	CART	122 (17 for tree-based classifiers)	Voting (DTR, RF, kNN, DNN, MTR)	N/A	85.20%
CFS-BA-Ensemble[9]	2020	CFS-BA	10	Voting (C4.5,RF,ForestPA)	99.80%	87.37%
LSTM-RNN-GA[10]	2020	GA	99	LSTM-RNN	99.00%	93.88%

Table 3 NSL-KDD State of the Art

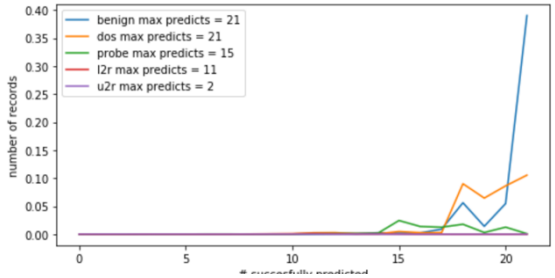


Figure 1 Successful Attack Predictions by 21 Classifier Models

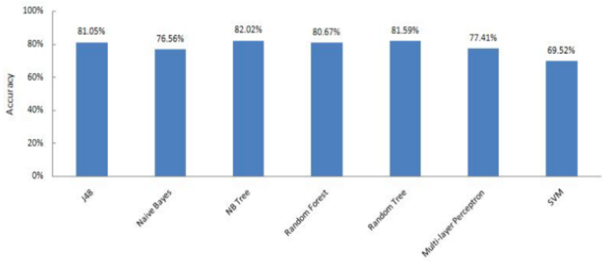


Figure 2 Original 21 Classifier Accuracy for KDDTest+

Inspired by [8] we plotted the first 3 PCA components against one another and labelled each instance by its true class in *Figure 3*. The difficulty inherent correctly identifying a network intrusion with a complex feature set and highly imbalanced target classes is readily apparent in this simplistic representation of the feature space and attack categories.

2. Metrics and Model Evaluation

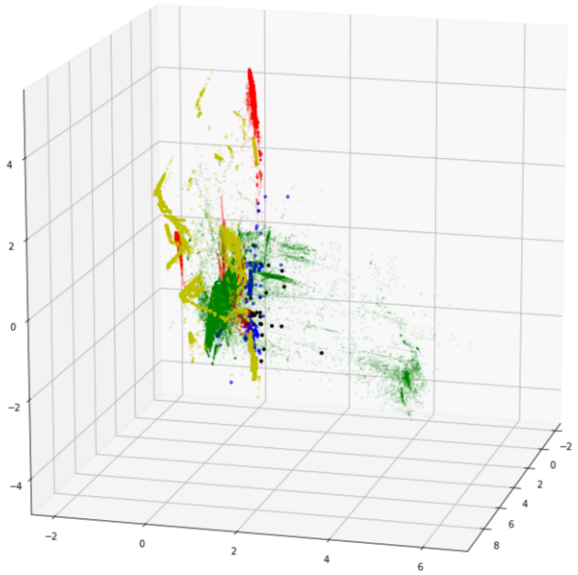


Figure 3 Analysis of First 3 PCA Components

The performance of classification models can be summarized by tabulating actual classes against predicted targets in a confusion matrix. For a binary classification problem, the confusion matrix looks like *Figure 4.* In our case, True Positives (TP) are the number of actual attack observations, correctly predicted as attacks; False Positives (FP) are the number of actual attack observations, incorrectly predicted as an attack; True Negatives (TN) are the number of observations that are normal, correctly predicted as normal; and False Negatives (FN) are the number of observations that are normal, incorrectly predicted as being an attack. In *Figure 5* we summarize the binary form of the metrics we used to evaluate each of our classifier models. The overall Accuracy of a classifier across all attack types remains the gold standard for evaluating its performance against the KDDTest+ dataset. However, with non-binary classification of highly imbalanced classes, Accuracy can be a misleading metric during hyperparameter tuning and validation of models against training data. It is particularly misleading for minority classes as even with only a few correctly predicted attacks the Accuracy will still be high since the majority classes will be labelled as TNs. Precision – also known as Detection Rate (DR) or True Positive Rate (TPR) is the percentage of predicted classes that are actually the target class. Recall – or Sensitivity – indicates the percentage of

observations being correctly classified with respect to the number of False Negatives for that class. The F1 score is the mean harmonic between Precision and Recall. The False Positive Rate indicates how often the classifier predicts an actual class as another, incorrect class.

We also used Weighted (WA) and Balanced Accuracy (BA) metrics to evaluate classifiers during hyper- parameter tuning and MCS selection. In contrast to Accuracy for multiclass classification – which is the average accuracy across all classes – WA is the mean of the individual class accuracy weighted by the number of instances of that class (also known as support). For binary classification problems, BA is the mean of the TPR and the True Negative Rate (TNR). For multiclass problems it is the 'macro average of recall scores per class or, equivalently, raw accuracy where each sample is weighted according to the inverse prevalence of its true class' [11]. If the dataset is balanced it is equivalent to the average (macro) Accuracy. However, when the dataset is imbalanced it provides a more even-handed idea of accuracy across classes with respect to their support.

In practice we found Balanced Accuracy, Recall and the False Positive Rate (both weighted and per class) to be the best indicators (during the training process) of the final classifier Accuracy. For EoC/MCS classifier selection we simply take the classifier with the lowest FPR for each attack class as this metric is independent from other class mis/classification (ie., it is the TP and FP across each row of the multiclass confusion matrix).

		Predicted Class	
		True	False
Actual Class	True	True Positive TP	False Negative TP
	False	False Positive TP	True Negative TP

Figure 4 Binary Classification Confusion Matrix

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

$$Precision\ (True\ Positive\ Rate) = \frac{TP}{TP + FP}$$

$$Recall\ (Sensitivity) = \frac{TP}{TP + FN}$$

$$F1 = \frac{2TP}{2TP + FN + FP}$$

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

$$Balanced\ Accuracy = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

Figure 5 Binary Classification Metrics

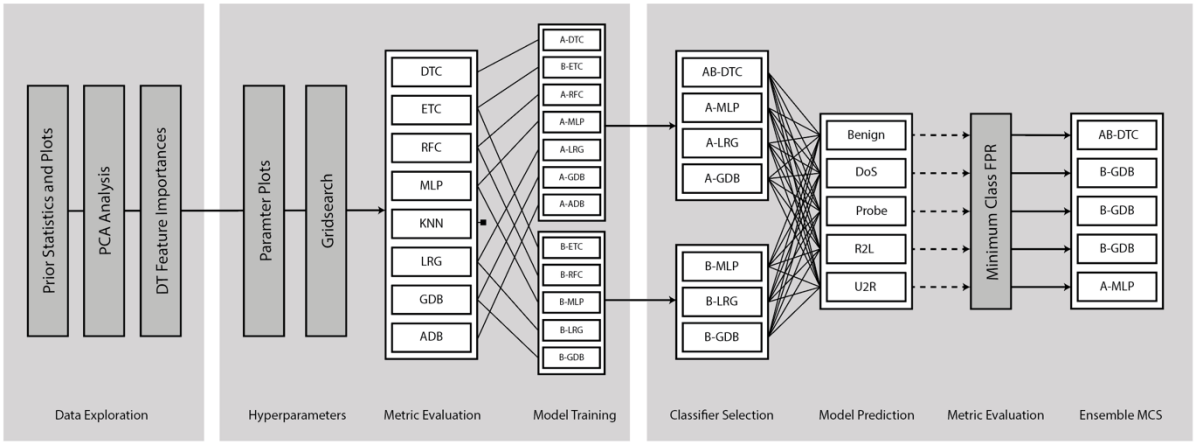


Figure 6 Pipeline for Ensemble MCS Selection using KDDTrain+

3. Methodology

Our methodology is summarized in *Figure 6*. We started with a statistical analysis of classification difficulty and visualization of the complex feature space using PCA. We briefly explored feature reduction using PCA and Decision Tree feature 'importances', with little success.

We then chose 4 classes of classifier and 2 forms of boosting, for a total of 8 classifiers: Decision Tree (DTC), Extra Tree (ETC), Random Forest (RFC), Multilayer Perceptron (MLP), Logistic Regression (LRG), Gradient Boost (GDB) with the LRG models as base classifiers and AdaBoost (ADB) with the Decision Tree model as base classifier.

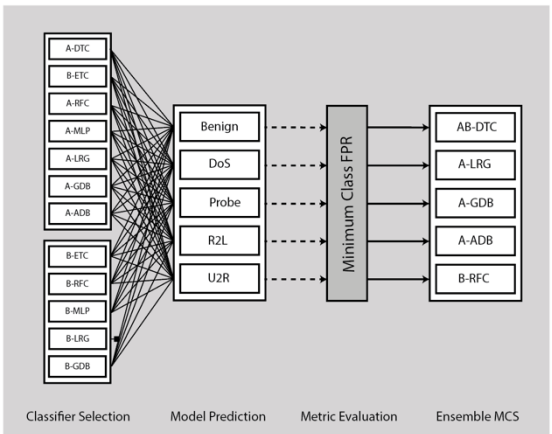


Figure 7 Estimated Oracle MCS Selection Using KDDTest+

During hyperparameter tuning we recorded and ranked each set of parameters according to mean Accuracy, Precision, Recall, F1 and Balanced Accuracy scores (as well as Weighted Accuracy, Train and Predict times). Based on the discussion above regarding metrics for imbalanced classes, we used incremental parameter and Gridsearch, 5-fold cross-validation techniques to find two sets of optimal parameters for each classifier based on:

A. the highest mean ranked Accuracy and/or Recall, and
B. the highest mean ranked Balanced Accuracy.

In almost all cases mean Accuracy and Recall scores resulted in identical ranks during cross-validation, however for many classifiers the model parameters were quite different when ranked by Balanced Accuracy. Where these ranks produced suitably divergent hyperparameters we trained two models, and where they coincided, we trained a single model. We denote classifiers trained using parameters derived from a) with an A- prefix, and those using b) with a B- prefix, eg., a Logistic Regression classifier using a set of parameters tuned for best Accuracy/Recall is denoted by A-LRG; and tuned for best Balanced Accuracy as B-LRG.

One of the key ways to address imbalanced datasets is to use decision thresholds, usually specified as a set of class weights (or an internal weighting algorithm). For each class you specify a number that is (usually) inversely proportional to the class size/support. Where appropriate we tested 3 hand-tailored sets of class weights plus whatever internal algorithms the classifier supported to determine the best decision thresholds in terms of our highest mean rank metrics. Our 3 hand-tailored class weights for each class were:

Method/Class	Benign	DoS	Probe	R2L	U2R	Formulation
1 `(cWeight1MnTR)`	0.47	0.64	0.91	0.99	1.00	1 – class size/total records
2 `(cWeight1InvTR)`	1.87	2.74	10.81	126.60	2422.56	1 / (class size/total records)
3 `(cWeightHvySI)`	1	2	20	40	200	based on [8]

In all bar one case, (1) performed the best across all supported classifiers – often for both A and B rankings.

In all, this yielded 12 models based on the 8 classifiers. We trained these using the KDDTrain+ dataset split into training and validation sets using stratified sampling, averaged the results across 10 iterations recording both the average and per class results along with a confusion matrix for each model. We chose this approach because it maintained stratified sampling and allowed us to record per class metrics (scikit-learns cross-validation method only supports single value scoring functions). Since resampling to match the imbalanced class ratios in KDDTest+ might introduce data-leakage and therefore overfitting to the test set, we chose to train our validation models using a train-to-validation dataset ratio of [1 – (KDDTest+ dataset size / KDDTrain+ dataset set size)].

This works out to 17.9% train (22,544) and 82.1% validate (103,429) ie., our validation training set is the same size as the total KDDTest+ dataset size. We tried several other training-to-validation size ratio's (including 50/50, 70/30 and inverting the proposed ratio) but this formula yielded the best results for selecting and predicting the performance of our EoC/MCS classifiers. We believe this is because limiting the training dataset size leads to more underfit (as opposed to overfit) models, whilst retaining a larger number of minority class records for our per class performance analysis.

Once we had collated all the performance metrics for each model, we then set out to formulate a process for selecting the most optimal model for predicting each attack class ie., a Multiple Classifier System. While a full review of the literature is beyond the scope of this project there are several objective multiple classifier system selection methods for regression, but for classification tasks it remains an open question [1][3][4]. There are many proposed methods, including static and dynamic selection, maximum diversity, *a priori* (without relying on training performance data) and *a posteriori* systems (using training performance data) [2][12]. However, prior to this project we were unaware of this area of research and arrived at our method of classifier selection via a purely experimental approach. Further research would in all likelihood yield better results.

Finally, we used a two-tier approach to selecting the most optimal classifier for predicting each attack class: pre-select a pool of classifiers based on objective and/or heuristic criteria, and then iterate the remaining models to find the minimum FPR for each class. We summarize the best results obtained using this method in *Table 5* **MCS Ensemble Selection Methods and Results**.

4. Classifiers

4.1. Decision Tree Classifier (DTC)

Decision tree classifiers partition feature space by iteratively (binary) splitting training instances from the top down into majority classes. At each iteration a node is formed representing the feature that maximises a majority class, thereby resulting in a splitting of the node into leaves. Information gain is the criterion for the binary splits and is usually measured by either Gini or Entropy algorithms. Gini measures the impurity of a node with respect to the number of training instances that would be incorrectly labelled if those instances were randomly labelled. Entropy is a measure of impurity with respect to the disorder or randomness in the training instances at each node. Splitting of the tree continues until either the Gini or Entropy index reaches zero for a particular leaf node, or until a user specified maximum depth is reached (`max_depth`). The number of features used at each split can also be specified (`max_features`). Class weights (as discussed above) can also be used when constructing the tree (`class_weight`). One advantage of decision tree classifiers is that the tree built during the training process provides a concrete, human understandable basis for the classification decision as is evident in *Figure 8*. They are also extremely fast to train and predict, although prone to overfitting on the training data.

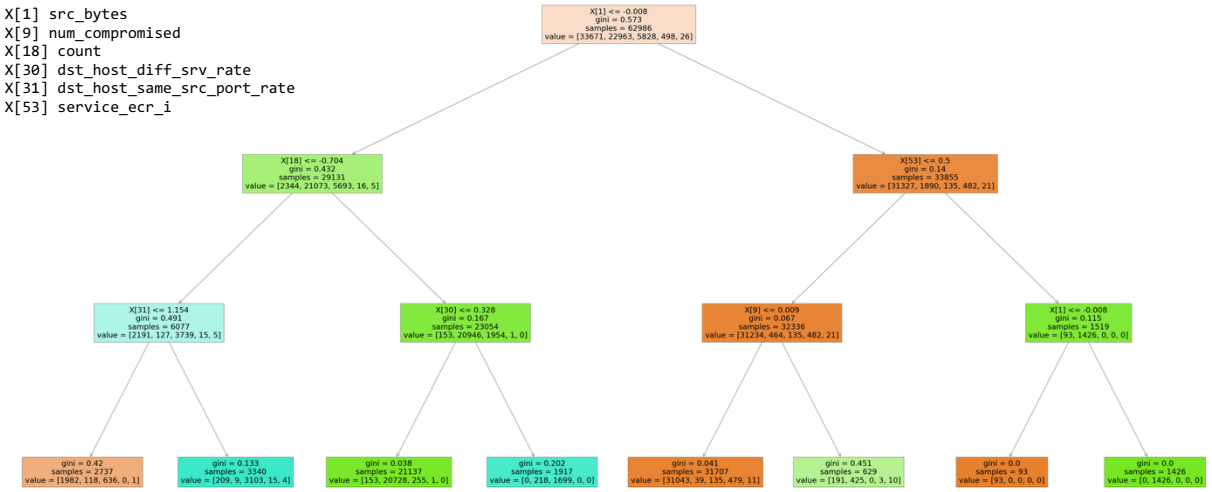


Figure 8 Decision Tree with Max Depth 4 for KDDTrain+

4.2. Ensemble Trees – Random Forest Classifier (RFC) & Extra Tree Classifier (ETC)

Extra Tree and Random Forest classifiers are ensemble classifiers made up of multiple decision trees (`n_estimators`). Each tree is trained using a random 'bootstrapped' subsample of training instances. Each node is split using either all features or a random subset (`max_features`). This introduces two sources of randomness when training the ensemble with the aim of reducing decision tree variance, at the cost of (usually low) increased bias. Training halts when impurity reaches zero or when a maximum depth is reached (`max_dpeth`). Classification is based on either the majority or mean vote of the decision trees in the forest. Extreme Randomized Trees introduce an extra source of randomness to RFCs by randomly selecting the decision threshold for each of the assigned feature subsets – again this generally reduces variance at the cost of an increased bias. As with DTC, class weights can be supplied to the decision thresholds (`class_weight`).

4.3. Logistic Regression (LRG)

Despite the name, Logistic Regression is a classification algorithm capable of solving non-linear problems. Unlike Linear Regression the logistic function does not approximate the weights needed to be applied directly to ***x*** to predict *y*, but instead models the probabilities – or log of odds – that a particular instance belongs to a particular class using a logistic or sigmoid threshold function. Just as in Linear Regression, an L1, L2 or ElasticNet regularization penalty (`penalty`) can be applied to the weights ***w*** for each of the features ***x*** to help keep the weights balanced. L1 regularization encourages weights to approach 0; L2 encourages weights to not grow large and ElasticNet is a ratio of L1 and L2 regularization. The type (`penalty`) and amount of regularization is controlled by a regularization strength parameter (C which is specified as the inverse of the strength in scikit-learns implementation). Several different algorithms can be used to solve the optimization problem for finding the minimum log of odds (`solver`). The LRG classifier also accepts class weights to be applied to the probability decision thresholds (`class_weight`).

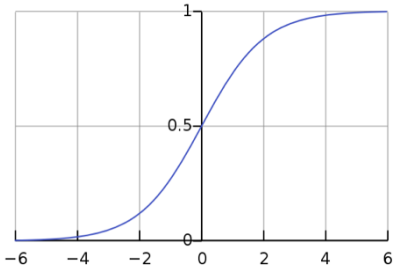


Figure 9 Sigmoid Function

4.4. Multilayer Perceptron (MLP)

The multilayer perceptron is a supervised learning algorithm that given a set of input features and output targets, can learn a set of weights to approximate a function to predict the targets. When only a single layer of inputs and outputs are used it is essentially the same as logistic regression. However, by adding a series of hidden layers (`hidden_layer_sizes`) it can approximate optimal non-linear functions. On the right we see a simple single hidden layer MLP [13]. The leftmost input layer consists of a set of 'neurons' representing the input features. The outputs of each succesive layer are fed to the next layer and are transformed using a weighted linear summation followed by an activation function (activation). The weights are iteratively updated to minimize the error of each at each layer using one of several algorithms (`solvers`). The solver continues to minimize the error until either the either a minimum variation (`tol`) or maximum number of iterations (`max_iter`) is reached.

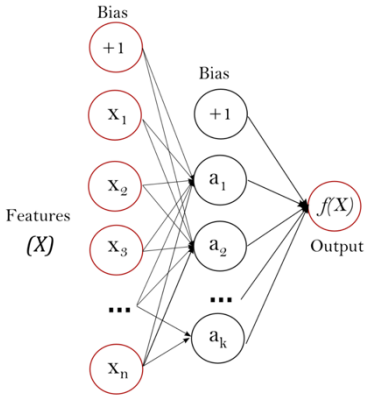


Figure 10 Single Hidden Layer MLP

4.5. K-Nearest Neighbours (KNN)

The K-Nearest Neighbours classifier does not *per se* learn from the training data. Instead it is a class of algorithms known as *instance-based* or *non-generalized learners*. KNN simply stores a copy of the features and targets during the training process and actually does all the work of classification during the prediction phase. When an unseen target instance is fed to the algorithm, KNN searches the training feature space to find the K-nearest instances based on a distance function (`metric`), eg., Euclidean or Minkowski distance. Rather than simple voting, the target instance is usually classified by the sum of the weighted distances to its K- nearest neighbours. As such class weights can be supplied to further attenuate the decision threshold (`class_weight`). The value of K is not learnt from the data, but provided as a hyperparameter (K) – there is a trade-off between variation and bias depending on the value of K – low values are prone to bias, while high numbers are prone to variation). Since classification essentially all happens during prediction, KNN can takes only a small time to train but a long time to predict – generally too long for an IDS given the large number of features and need for swift prediction.

4.6. Boosting – Gradient Boost (GDB) & AdaBoost (ADB)

AdaBoost fits a number (`n_estimators`) of weak learners to re-weighted subsamples of the feature set. Weak learners are models that are generally only slightly better than random guessing, such as shallow decision trees (`base_estimator`). At each iteration, each of the subsampled data instances is given a weight – if the instance is misclassified its weight is increased and, if correctly classified, it is reduced. This results in difficult to classify instances having increasing influence over the final trained model as the weak classifiers spend more and more time concentrating on the difficult to classify instances. The predictions from all the learners are then combined using a simple weighted majority vote or sum. The classifier can use two different algorithms (`solver`) and also provides a learning rate parameter (`learning_rate`) to control the contribution of each learner. Gradient Boosting follows a similar pattern of training a set of weak learners, but solves for an optimized differentiable loss function. It is possible to supply class weights to base estimators, but not to the boosting algorithms themselves (despite there being a bug in scikit-learn where it is necessary to supply the class weights without named targets). Both classifiers have a `warm_start` parameter that allows for adding extra learners without re-training.

5. Hyperparameter Tuning

In general, where we have indicated a (`hyperparameter`) above we carried out some kind of hyperparameter optimization either by iterative incremental graphing of the parameter vs accuracy or via Gridsearch cross-validation (see ipynb code). Due to time constraints and computational complexity many of our models were not optimally tuned. For the tree-based learners it was more feasible to apply incremental, isolated parameter tuning since they are relatively fast to fit and predict. We often had to opt for narrow parameter range Gridsearch runs to try and identify a direction for trying the next set of parameters (eg., trying 20 and 200 estimators and then 100, 150 200 etc), thereby not being able to get a more fine-grained picture of parameter importance. For example, we would have liked to have had the time to tune the `n_estimator` parameter for GDB and ADB; `hidden_layer_sizes`, `tol` and `max_iter` for MLP, etc. There were many other parameters we ran out of time to tune, and we note that throughout the literature the same classifiers have achieved significantly better performance – so there's plenty of room to improve – though we also note that it is frustratingly rare for authors to actually discuss or publish these parameters which makes reproducing their results difficult, if not impossible.

Table 4 summarizes the final hyperparameters we supplied to models – in some cases we tried tuning parameters not listed here but found the defaults to work best, so we omit them for clarity (the ipynb file includes these as commented out parameter searches in the Gridsearch Hyperparameter Tuning section).

Final Model Hyperparameters	
AB-DTC	`{'class_weight': cWeight1MnTR, 'max_depth': 20,'max_features': 118}`
A-ETC	`{'class_weight': cWeight1MnTR, 'max_depth': 80,'max_features': 60, 'n_estimators': 90}`
B-ETC	`{'class_weight': cWeight1MnTR, 'max_depth': 50,'max_features': 30, 'n_estimators': 90}`
A-RFC	`{'class_weight': cWeight1MnTR, 'max_depth': 35,'max_features': 63, 'n_estimators': 60}`
B-RFC	`{'class_weight': cWeight1MnTR, 'max_depth': 35,'max_features': 107, 'n_estimators': 10}`
A-MLP	`{'alpha': 0.0001, 'hidden_layer_sizes': (256, 128, 64, 32)}`
B-MLP	`{'alpha': 0.001, 'hidden_layer_sizes': (256, 128, 64, 32)}`
A-LRG	`{'class_weight': cWeight1MnTR, 'C': 20}`
B-LRG	`{'class_weight': 'balanced', 'C': 20}`
A-GDB	`{'base_estimator': LogisticRegression (B-LRG), 'n_estimators': 100}`
B-GDB	`{'base_estimator': LogisticRegression (B-LRG), 'n_estimators': 200}`
AB-ADB	`{'base_estimator': DecisionTreeClassifier (AB-DTC), 'n_estimators': 150}`
AB-KNN	`{'n_neighbors': 2, 'weights': 'distance'}`

Table 4 Final Hyperparameters for all Models

6. Results

Tabulated summaries of Accuracy, Precision, Recall, F1, Weighted and Balanced Accuracy scores along with Train and Predict Times for each classifier are contained in **Appendix C**Error! Reference source not found.. The same metrics are given for each classifier by attack class in **Appendix D.** Confusion matrices for all classifiers are contained in **Appendix B**.

Name	Pre-selection Criteria	MCS Ensemble	KDDTrain+			KDDTest++			
			AA	FPR	BA	AV	FPR	BA	PRT
L-MCS	Use all except AB-KNN since its prediction time is too long for real-time intrusion detection	AB-DTC A-RFC A-RFC AB-ADB A-ETC	99.89	0.09	92.05	**82.75**	12.02	72.09	2.81
H-MCS	Drop AB-KNN, A-ETC, B-ETC, A-RFC, B-RFC and AB-ADB as AB-DTC has the highest Balanced Accuracy of all tree-based classifiers	AB-DTC B-GDB B-GDB B-GDB B-MLP	99.88	0.08	92.99	**85.09**	9.88	75.62	2.96
-	Use all except AB-KNN	B-LRG A-LRG A-GDB A-ADB B-RFC	99.26	0.65	95.19	**83.56**	11.54	72.28	2.83
O-MCS	Drop AB-KNN and B-LRG since in all other circumstances AB-DTC has best performance for the benign class	AB-DTC A-LRG A-GDB A-ADB B-RFC	99.62	0.23	93.45	**85.71**	9.91	76.4	2.66

AA = Average Accuracy FPR = False Positive Rate BA = Balanced Accuracy PRT = Prediction Time

Table 5 MCS Ensemble Selection Methods and Results

Our methods of classifier pre-selection are outlined in *Table 5,* along with summary performance metrics. As can be seen in *Figure 12* and *Table 7,* our MCS model far outperform any single classifier.

7. Discussion

One of the main challenges in selecting an EoC to make up a MCS is that most learning rate performance metrics do not well predict test performance. This is clearly seen in the plots in **Appendix A**, where it is obvious that Accuracy, Precision, Recall and F1 do not well predict final performance. As proposed in **Metrics and Model Evaluation** and validated by *Figure 11*, Balanced Accuracy (BA) is the only summary metric that adequately predicts final test performance across all classifiers. All other summary metrics fail with one or another classifier to indicate the final performance of the model (B-LRG is an outlier across all metrics, with an often, inverse train-to-test performance ratio). The only per attack class metric that provides any indication of train-to-test

performance is FPR, however for minority classes this is often still inaccurate as there are few samples and therefore high variance.

The difficulty of not being able to predict final performance from learning rate metrics is in general a challenge for MCS EoC selection, especially for classification problems [1][2][3][4]. We tried several different naïve algorithms – including brute force permutation search and trying to sort algorithms based on every summary and per attack class metric – and still we were unable to find a consistent, simple indicator that would guide selection of the classifiers to make up the MCS. In all selection methods (above, *Table 5*) it is possible that the discovery of a more performant ensemble is influenced by chance due to per attack class FPR variance and/or bias. In most cases we had to also apply heuristic pre-selection methods – which are somewhat arbitrary and open to criticism as over-fitting to the KDDTest+ dataset – to arrive at a high accuracy MCS.

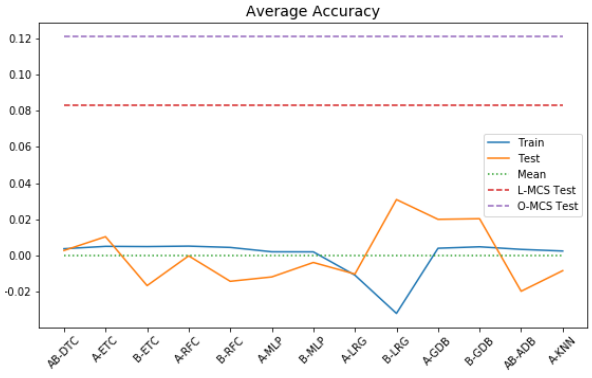


Figure 12 Average Accuracy for All Classifiers (with MCS)

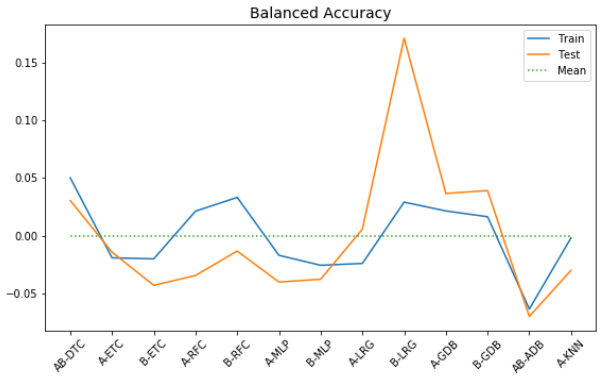
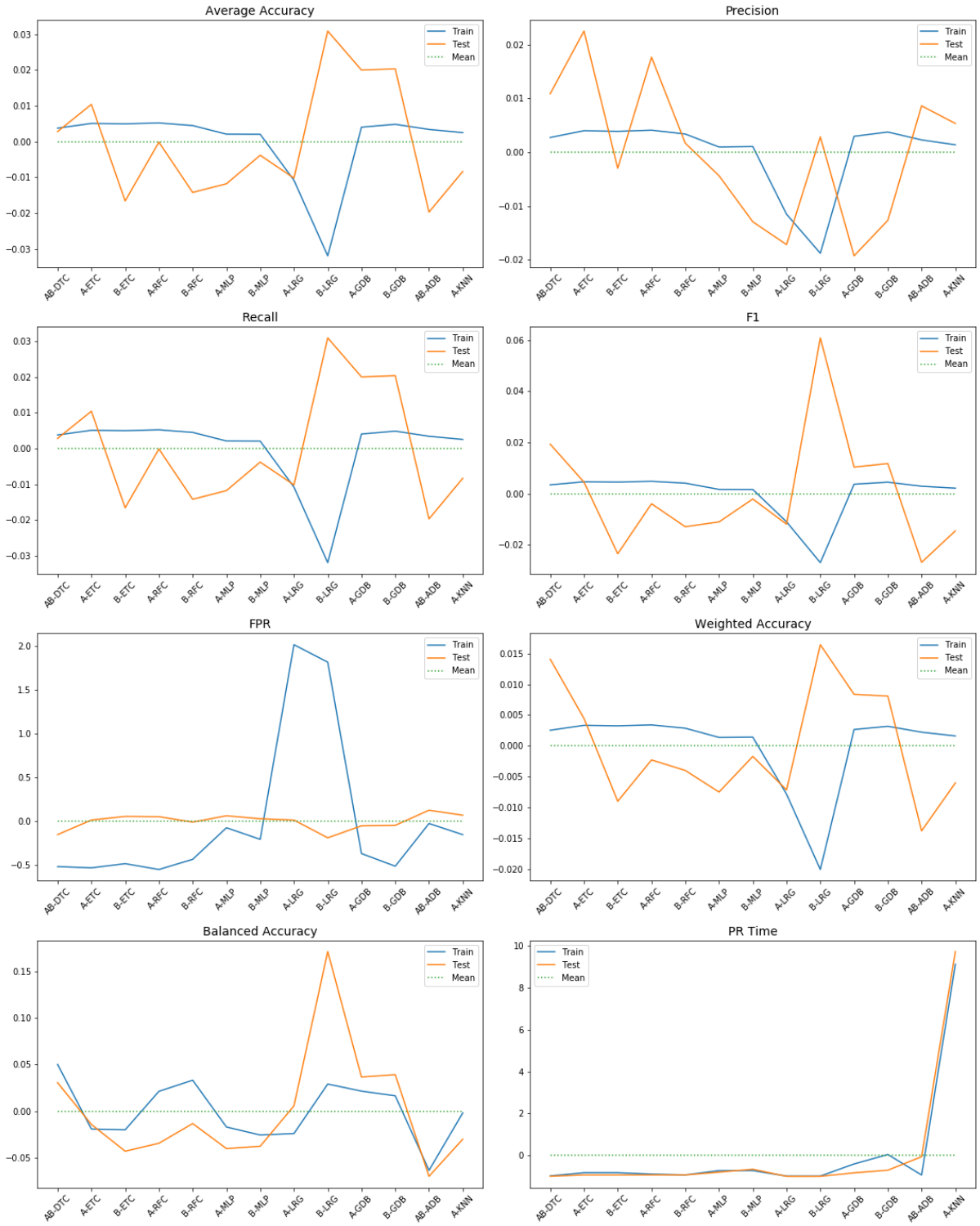


Figure 11 Balanced Accuracy for all Classifiers

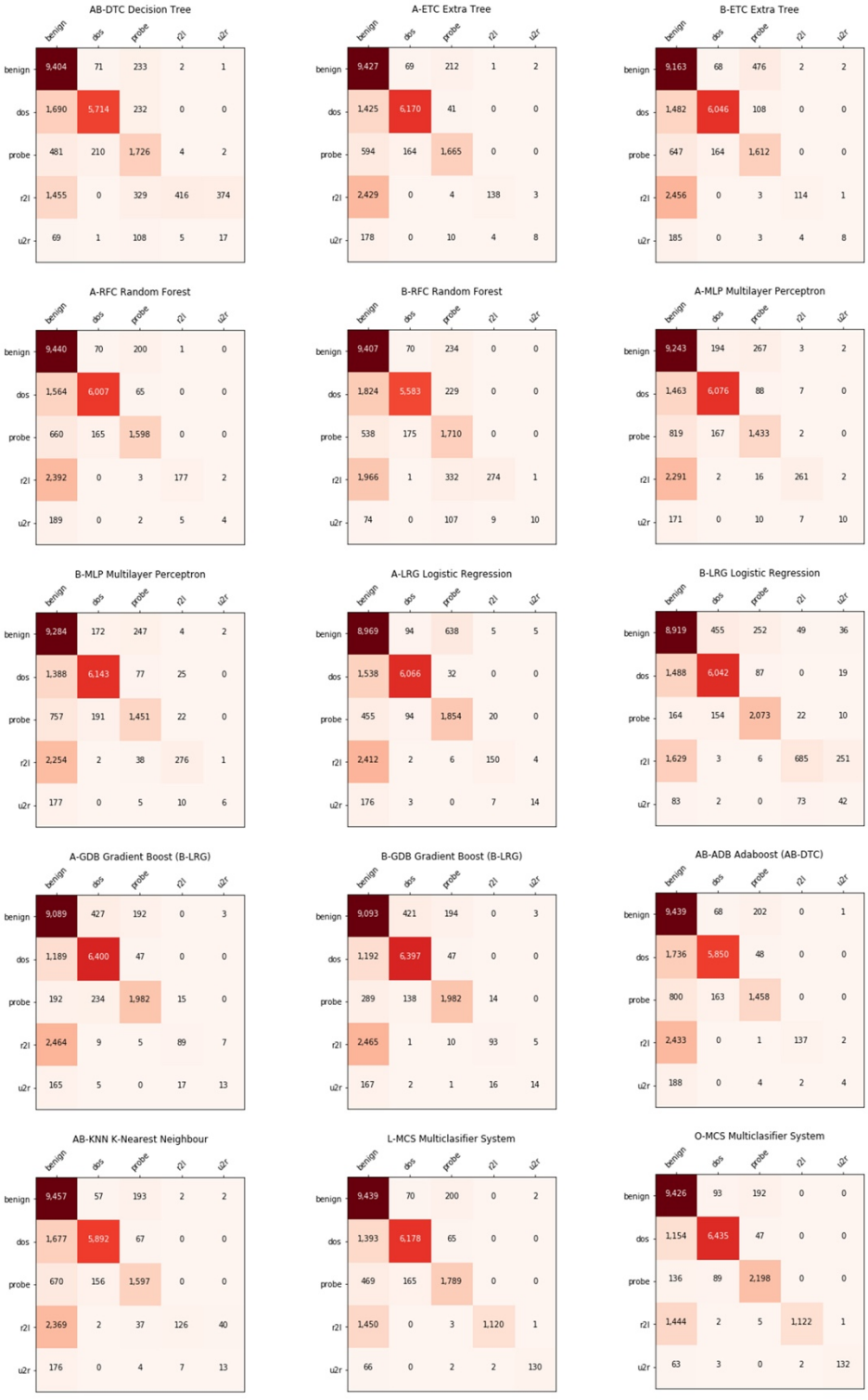
8. Conclusion

Static and dynamic classifier selection for MCS is an ongoing area of theoretical and practical research and though it has been superficially discussed in relation to network intrusion [14][15], it is far less researched than RNN/DNN, bagging, boosting, voting or stacking ensembles. In particular dynamic ensemble construction could be applied to both (*a priori*) feature selection and (*a posteriori*) classifier selection, thus adapting the system in real-time to different attack categories. While our methods are open to some criticism, we believe our results demonstrate that MCS for network IDS warrants further research – since even with non-optimally trained classifiers, no feature reduction and naïve static EoC selection –we obtained competitive, if not better accuracy than many other ensemble methods.

Appendix A. Comparison Plots of All Classifiers and Metrics



Appendix B. Confusion Matrices for all Classifiers



Appendix C. Summary Classifier Validation and Test Results

KDDTrain+ Validation Results									
	Accuracy	**Precision**	**Recall**	**F1**	**FPR**	**TR Time**	**PR Time**	**Weighted**	**Balanced**
AB-DTC	99.62	99.62	99.62	99.62	0.21	**0.23**	0.09	99.77	**88.33**
A-ETC	**99.77**	**99.77**	**99.77**	**99.76**	**0.19**	1.59	0.54	**99.86**	83.74
B-ETC	99.76	99.76	99.76	99.75	0.21	1.00	0.54	99.85	84.41
A-RFC	99.76	99.76	99.76	**99.76**	0.20	1.01	0.43	99.85	86.86
B-RFC	99.73	99.73	99.73	99.72	0.19	0.39	0.23	99.83	88.07
A-MLP	99.48	99.48	99.48	99.48	0.34	44.36	0.76	99.67	83.96
B-MLP	99.47	99.47	99.47	99.47	0.34	39.96	0.77	99.67	84.23
A-LRG	98.19	98.23	98.19	98.21	1.20	3.34	**0.06**	98.75	83.52
B-LRG	96.10	97.51	96.10	96.63	1.12	3.73	**0.06**	97.53	88.07
A-GDB	99.67	99.67	99.67	99.67	0.25	31.49	1.87	99.79	87.42
B-GDB	99.75	99.75	99.75	99.75	0.20	57.80	3.29	99.84	86.99
AB-ADB	99.62	99.61	99.62	99.60	0.38	0.46	0.23	99.76	80.17
AB-KNN	99.52	99.51	99.52	99.51	0.34	0.76	30.63	99.69	85.42
MCS-L	99.89	99.89	99.89	99.88	0.09	3.15	6.10	99.93	92.05
MCS-O	99.62	99.62	99.62	99.62	0.23	35.91	7.06	99.70	93.45

Table 6 KDDTrain+ Validation Results

KDDTest+ Test Results									
	Accuracy	**Precision**	**Recall**	**F1**	**FPR**	**TR Time**	**PR Time**	**Weight**	**Balanced**
AB-DTC	76.64	81.42	76.64	**74.48**	13.55	**2.58**	0.03	87.16	53.51
A-ETC	77.22	**82.36**	77.22	73.39	16.20	26.39	0.19	86.33	51.19
B-ETC	75.16	80.31	75.16	71.35	16.86	14.12	0.25	85.18	49.70
A-RFC	76.40	81.96	76.40	72.77	16.81	16.20	0.14	85.75	50.14
B-RFC	75.34	80.68	75.34	72.12	15.82	5.84	0.15	85.61	51.24
A-MLP	76.06	80.01	76.06	72.79	16.65	288.20	0.32	85.68	50.30
B-MLP	76.37	80.08	76.37	73.09	16.19	413.67	0.55	85.94	50.72
A-LRG	75.64	79.17	75.64	72.19	16.20	22.27	**0.02**	85.34	52.23
B-LRG	**78.78**	80.78	**78.78**	77.51	**12.97**	22.95	**0.02**	**87.36**	**60.83**
A-GDB	77.97	79.01	77.97	73.82	15.15	264.97	0.40	86.67	53.85
B-GDB	77.98	79.53	77.98	73.92	15.23	415.60	0.56	86.64	53.96
AB-ADB	74.92	81.24	74.92	71.10	17.99	244.85	1.77	84.76	48.30
AB-KNN	75.79	80.98	75.79	72.00	17.08	29.36	20.12	85.43	50.37
MCS-L	82.75	85.96	82.75	82.08	12.02	290.02	2.80	89.31	72.09
MCS-O	85.71	88.11	85.71	84.99	9.91	540.51	2.65	91.08	76.40

Table 7 KDDTest+ Test Results

Appendix D. Classifier Test Results by Attack Class

Decision Tree Classifier AB-DTC					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	82.25	71.79	96.84	82.46	28.79
dos	90.22	95.30	74.83	83.83	1.89
probe	92.91	65.68	71.23	68.34	4.48
r2l	90.38	97.42	16.16	27.72	0.06
u2r	97.52	4.31	8.50	5.72	1.69

Extra Tree Classifier A-ETC					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	77.73	66.57	96.99	78.95	36.85
dos	91.73	96.42	78.51	86.55	1.50
probe	95.84	87.31	71.81	78.79	1.26
r2l	89.31	97.05	6.56	12.26	0.03
u2r	99.13	64.67	3.60	6.82	0.02

Extra Tree Classifier B-ETC					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	76.41	65.76	94.36	77.51	37.17
dos	91.92	96.30	79.18	86.91	1.56
probe	93.79	73.21	66.53	69.71	2.93
r2l	89.06	95.00	4.43	8.46	0.03
u2r	99.14	72.73	4.00	7.58	0.01

Random Forest Classifier A-RFC					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	77.48	66.27	97.21	78.81	37.44
dos	91.73	96.24	78.67	86.57	1.58
probe	95.14	85.55	65.95	74.48	1.34
r2l	89.34	96.72	6.88	12.84	0.03
u2r	99.12	66.67	2.00	3.88	0.01

Random Forest Classifier B-RFC					
benign	79.13	68.12	96.87	79.99	34.30
dos	89.80	95.78	73.11	82.93	1.65
probe	92.84	65.47	70.57	67.92	4.48
r2l	89.76	96.82	10.64	19.18	0.05
u2r	99.15	90.91	5.00	9.48	0.00
benign	79.13	68.12	96.87	79.99	34.30

Logistic Regression Classifier A-LRG					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	76.39	66.19	92.36	77.12	35.70
dos	92.18	96.92	79.44	87.31	1.29
probe	94.48	73.28	76.52	74.86	3.36
r2l	89.11	82.42	5.83	10.89	0.16
u2r	99.14	60.87	7.00	12.56	0.04

Logistic Regression Classifier B-LRG					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	81.56	72.61	91.84	81.10	26.21
dos	90.21	90.78	79.13	84.55	4.12
probe	96.92	85.73	85.56	85.64	1.71
r2l	90.98	82.63	26.61	40.26	0.72
u2r	97.90	11.73	21.00	15.05	1.41

Multilayer Perceptron A-MLP					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	76.88	66.11	95.19	78.01	36.98
dos	91.47	94.44	79.58	86.33	2.44
probe	93.91	79.16	59.16	67.46	1.90
r2l	89.65	93.29	10.14	18.23	0.10
u2r	99.14	64.53	5.15	9.45	0.02

Multilayer Perceptron B-MLP					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	77.80	67.00	95.61	78.77	35.67
dos	91.75	94.43	80.46	86.86	2.46
probe	94.05	80.03	59.90	68.37	1.84
r2l	89.53	84.49	10.75	18.92	0.31
u2r	99.12	46.64	3.15	NaN	0.02

Gradient Boost (B-LRG Base) A-GDB					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	79.45	69.39	93.59	79.69	31.25
dos	91.52	90.46	83.81	87.01	4.53
probe	96.96	89.04	81.80	85.27	1.21
r2l	88.84	73.55	3.46	6.60	0.16
u2r	99.13	56.52	6.50	11.66	0.04

Gradient Boost (B-LRG Base) B-GDB					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	79.01	68.86	93.64	79.36	32.05
dos	92.01	91.92	83.77	87.66	3.77
probe	96.93	88.72	81.80	85.12	1.25
r2l	88.86	75.61	3.61	6.90	0.15
u2r	99.14	63.64	7.00	12.61	0.04

AdaBoost (AB-DTC Base) AB-ADB					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	75.90	64.66	97.20	77.66	40.21
dos	91.05	96.18	76.62	85.29	1.56
probe	94.59	85.06	60.20	70.50	1.27
r2l	89.18	97.81	5.32	10.08	0.01
u2r	99.12	56.47	2.15	4.14	0.01

AdaBoost (AB-DTC Base) AB-ADB					
Attack Type	Accuracy	Precision	Recall	F1	FPR
benign	77.17	65.91	97.38	78.61	38.12
dos	91.31	96.48	77.16	85.75	1.44
probe	95.00	84.14	65.91	73.92	1.50
r2l	89.10	93.33	4.90	9.30	0.05
u2r	98.98	23.64	6.50	10.20	0.19

[1] Didaci, L., Giacinto, G., Roli, F., & Marcialis, G. L. (2005). A study on the performances of dynamic classifier selection based on local accuracy estimation. *Pattern recognition*, *38*(11), 2188-2191.

[2] Woods, K., Kegelmeyer, W. P., & Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE transactions on pattern analysis and machine intelligence*, *19*(4), 405-410.

[3] Brown, G., Wyatt, J., Harris, R., & Yao, X. (2005). Diversity creation methods: a survey and categorisation. *Information Fusion*, *6*(1), 5-20.

[4] Ko, A. H., Sabourin, R., & Britto Jr, A. S. (2008). From dynamic classifier selection to dynamic ensemble selection. *Pattern recognition*, *41*(5), 1718-1731.

[5] Lee, W., Stolfo, S. J., & Mok, K. W. (1999, May). A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No. 99CB36344)* (pp. 120-132).

[6]Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009, July). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1-6).

[7] Dhanabal, L., & Shantharajah, S. P. (2015). A study on NSL-KDD dataset for intrusion detection system based on classification algorithms. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(6), 446-452.

[8] Gao, X., Shan, C., Hu, C., Niu, Z., & Liu, Z. (2019). An adaptive ensemble machine learning model for intrusion detection. *IEEE Access*, *7*, 82512-82521.

[9] Zhou, Y., Cheng, G., Jiang, S., & Dai, M. (2020). Building an efficient intrusion detection system based on feature selection and ensemble classifier. *Computer Networks*, 107247.

[10] Muhuri, P. S., Chatterjee, P., Yuan, X., Roy, K., & Esterline, A. (2020). Using a Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) to Classify Network Attacks. *Information*, *11*(5), 243.

[11] scikit-learn.org. (n.d.). *3.3.2.3. Balanced accuracy score*. [online] Available at: https://scikit-learn.org/stable/modules/model_evaluation.html.

[12] Lysiak, R., Kurzynski, M., & Woloszynski, T. (2014). Optimal selection of ensemble classifiers using measures of competence and diversity of base classifiers. *Neurocomputing*, *126*, 29-35.

[13] scikit-learn.org. (n.d.). *1.17.1. Multi-layer Perceptron*. [online] Available at: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[14] Tama, B. A., & Rhee, K. H. (2017). An extensive empirical evaluation of classifier ensembles for intrusion detection task. *Computer Systems Science and Engineering*, *32*(2), 149-158.

[15] Kumar, Y. V., & Kamatchi, K. Anomaly Based Network Intrusion Detection Using Ensemble Machine Learning Technique.