Cross-datasets evaluation of machine learning models for intrusion detection systems

Said Al-Riyami, Alexei Lisitsa, and Frans Coenen

Department Of Computer Science, University of Liverpool, Liverpool, UK
{said.alriyami,lisitsa,coenen}@liverpool.ac.uk

Abstract. The conventional way to evaluate the performance of machine learning models intrusion detection systems (IDS) is by using the same dataset to train and test. This method might lead to the bias from the computer network where the traffic is generated. Because of that, the applicability of the learned models might not be adequately evaluated. We argued in [1] that a better way is to use cross-datasets evaluation, where we use two different datasets for training and testing. Both datasets should be generated from various networks. Using this method as it was shown in [1] may lead to a significant drop in the performance of the learned model. This indicates that the models learn very little knowledge about the intrusion, which would be transferable from one setting to another. The reasons for such behaviour were not fully understood in [1]. In this paper, we investigate the problem and show that the main reason is the different definitions of the same feature in both models. We propose the correction and further empirically investigate cross-datasets evaluation for various machine learning methods. Further, we explored cross-dataset evaluation in the multi-class classification of attacks, and we show for the most models that learning traffic normality is more robust than learning intrusions.

Keywords: Network Intrusion Detection System, Machine Learning, Model Evaluation, Network Security, Security and Privacy

1 INTRODUCTION

The number of cybersecurity attacks keeps increasing and changing every year. This makes the work to find them harder and makes it more challenging to evaluate any intrusion detection model obtained by machine learning methods. How can we make sure that the result obtained is not out of outfitting? And how can we measure the practicality of the models?

The paper [1] proposed a new cross-datasets evaluation method to address such a problem. The idea of the method is to use two different datasets generated in two different computer network settings. We use the first dataset in training of the machine learning models and use the other in testing. Then, we reverse the process by swapping the training and testing datasets. The method allows one to evaluate the quality of the detection models and the quality of the datasets

used. The paper [1] shows that training and testing by using the same dataset may produce F1 score of 99% (conventional way of evaluation), while the same machine learning models downgraded dramatically to around 36% score by using Cross-datasets Evaluation. The problem reported was a binary classification (Normal and Attack), and the datasets used were NSL-KDD [2] and gureKDD [3]. The paper [1] used Random forest [4], Artificial neural networks (ANN), and Long short-term memory (LSTM) [5] as machine learning algorithms.

In this paper, we will follow the same evaluation method and investigate the drop in accuracy when we change the datasets. We explain the results of [1] and propose a correction that leads to an increase in the accuracy of the machine learning models.

We also examine the cross-datasets evaluation methodology in multi-class classification for the same datasets. Since not all the attacks are common to both datasets, we map data labels into five groups (Normal Traffic, DoS, R2L, U2R, and Probe).

Further, we expand in the number of machine learning models to evaluate. These include: Decision tree learning, Random forest [4] (with a different number of trees), AdaBoost [6], Gradient boosting [7], Logistic regression, Naive Bayes, Linear Support Vector Machine (SVM) [8], K-Nearest Neighbours Algorithm (k-NN), Discriminant Analysis, Artificial neural networks (ANN), Convolutional neural network (CNN), Long short-term memory (RNN LSTM) [5] and Gated Recurrent Units (RNN GRU) [9].

The rest of the paper organized as follows. In section 2 we discuss different evaluation methodologies. Section 3 focuses on the binary classification and provides an explanation and a fix for performance drop from [1]. In section 4, we report the results on a multi-class classification problem and show the performance for each class. In section 5, we discuss the overall results, limitations, and future work.

2 Evaluation Methodologies

One of the main processes of machine learning is model evaluation. It measures how the model is performing. One of the goals of the evaluation is to have a good picture of the model performance during the deployment. We try we have a model that has less bias. This process consists of different dimensions:

- **Stages**: There are two main phases of creating the model: (1) training (where the model learns from part of the data) and (2) testing (where the model tries to predict from unseen data). Typically, we want to check how the model is performing during the testing phase.
- **Data Splitting:** Which part of the data used in training, validation (if any) and testing. There are two popular methods: Holdout method and K-fold cross-validation. We will discuss the splitting in more details later.
- **Scoring Metric**: used to measure the result (Accuracy, Confusion Matrix, Precision, Recall, Precision, F1 score, AUC, Loss, Root Mean Squared Error (RMSE), etc.).

– **External Evaluation**: all other measures like the processing time (How long it takes to train, re-train and deploy the model), and size of the model (and how many parameters contains).

2.1 Data Splitting Methods

The idea of splitting the data is to assess how the model will perform in unseen data after training in part of the data. There are two main ways to split the data: (1) Holdout cross validation, and (2) K-fold cross-validation.

Holdout cross validation In this method, we split the data into different parts, each having its own purpose. Figure 1 shows simple illustration of this method. The following is the common split [10, 11]:

– **Training set**: Part of the data that the model will use to learn from during training phase.
– **Testing set**: The data that will be used to test the performance of the model after the training phase.
– **Dev (development) set**: also known as validation set or holdout cross-validation set. This part is used during the training phase to tune the hyper-parameters of the model. The distribution of in the validation set should be the same as the distribution of the testing set [12]. This set helps in avoiding overfitting while hyperparameters tuning by giving feedback on how the change affects the performance. We stop training phase after we reach to performance of training set as good as validation set performance. In another words, stop the training when the error of the validation set starts to increase [11].

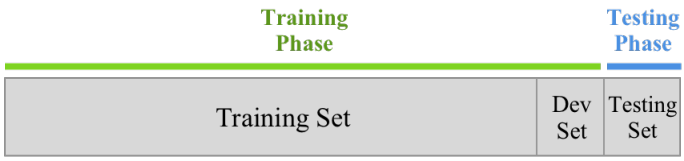

Fig. 1. Simple illustration of Holdout method

K-fold cross validation The following are the steps in how it works:

1. Divide the dataset into k number of fold or group.
2. Make one group a validation group and the rest of the groups a training set. Calculate the performance score.
3. Repeat step 2 until all group used as a validation set.
4. The result is the average of all scores.

2.2 Scoring Metrics

There are many scoring metrics to measure performance. For our work, the following four metrics are relevant:

- **Accuracy**: this might be the most used metric. The accuracy is calculating the ratio of the total number of correct predictions by the total number of predictions. Equation (1) shows how it is calculated.
- **Precision**: also known as positive predictive value. Precision metric try to answer the question: what proportion of positive identifications was actually correct?[13] Equation (2) shows how it is calculated.
- **Recall**: also known as sensitivity. Recall metric try to answer the question: what proportion of actual positives was identified correctly?[13] Equation (3) shows how it is calculated.
- **F1 Score**: also known as F-measure or balanced F-score. It is a harmonic mean of precision and recall. Equation (4) shows how it is calculated.

For a multiclass classification problem, there are different ways to calculate the average of the score [14]:

- **Micro Average**: Calculate metrics globally by counting the total true positives, false negatives and false positives.
- **Macro Average**: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.
- **Weighted Average**: Calculate metrics for each label, and find their average weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance; it can result in an F-score that is not between precision and recall.

$$Accuracy = \frac{TruePositive + TrueNegative}{TotalPrediction} \tag{1}$$

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3}$$

$$F1 = 2.\frac{Precision.Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \tag{4}$$

2.3 Conventional way to evaluate ML models in intrusion detection

Most of the publications on machine learning in intrusion detection follow the same procedure when evaluating the performance:

- Select an available public dataset;

– Use the dataset and follow one of the method mentions in data splitting methods. This includes holdout or k-fold cross-validation.
– Report the performance result by using one or more scoring metrics.

The drawback of this setting is we don't know how much the model learns about the intrusion compare to how much about the particular computer network environments. It might be just a result of the overfitting of a particular dataset. This may make the reported results about the accuracy of the intrusion detection overly optimistic and suspiciously looking. Most of the reported results reach 99% precision [1]. To address this problem, we proposed in [1] the Cross-datasets Evaluation method.

2.4 Cross-datasets Evaluation

Cross-datasets Evaluation method tries to make the evaluation of ML-based NIDS more practical. The idea is to use two different datasets generated from two different computer networks. We train in one dataset and test the model in the other dataset (and swap between the datasets). This gives a better understanding of the actual performance of the machine learning models when the computer network changes. That will allow us to see how our model can adapt to the change in network architecture. By using this method, we also gain insight into the quality of the datasets used in the training. If we get a high performing, this means that the dataset used in training contains more information about the intrusions. Figure 2 shows an illustration of the method.

We use F1 score 4 as a metric for evaluate the performance. This will help to avoid the accuracy paradox that might be there because of the imbalance of the data [15].

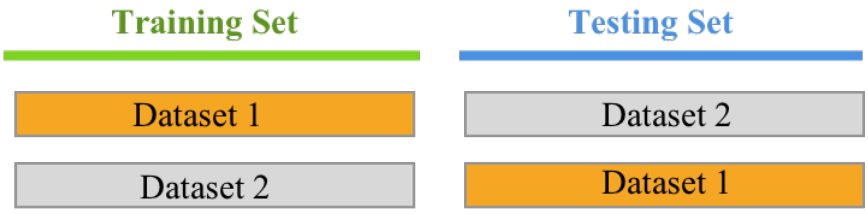

Fig. 2. Cross-dataset validation

2.5 Datasets

In this work, we will use two datasets: NSL-KDD [2] and gureKDD [3]. Both datasets generated from two different networks.

NSL-KDD is a refined version from a famous KDD cup 1999 dataset, which reported having some problems [16][17]. NSL-KDD dataset comprises 148,517 connections with 40 distinct attacks.

The gureKDDdatasets dataset created from scratch followed the specifications of KDD cup 1999 dataset. The gureKDDdatasets contains 2,759,494 connections with 28 distinct attacks. NSL-KDD dataset contains more distinct attacks, but less number of connections.

2.6 Machine learning models

In this work, we tried to include a wider range of machine learning models. This will allow us to compare the performance of each model. The machine learning models that we used are: Decision tree learning, Random Forest (RF) [4] (with a different number of trees), AdaBoost [6], Gradient Boosting Algorithm (GBM) [7] (deviance and exponential), Logistic regression (LR), Naive Bayes (Gaussian -GNB- and Complement -CNB-), Linear Support Vector Machine (SVM) [8], K-Nearest Neighbours Algorithm (k-NN), Quadratic Discriminant Analysis (QDA), Linear Discriminant Analysis (LDA), Artificial neural networks (ANN), Convolutional neural network (CNN), Long short-term memory (RNN LSTM) [5] and Gated Recurrent Units (RNN GRU) [9].

3 Binary classification

3.1 Problem overview

The binary classification problem is when we have two classes of labels. In our scenario, all labels of both datasets mapped to Normal and Attack.

The result of binary classification reported in [1] shows that the machine learning models reach high accuracy and F1 score by using holdout cross validation method of the same dataset to around 99%. When we change the evaluation method of the same machine learning algorithms to cross-datasets evaluation, all models fail. While the accuracy metric reaches up to 97.65% (highest case), the highest F1 score reported is 36.08%. And the majority of the F1 score cases are below 10%. The result here shows that the machine learning models did not learn anything about the intrusion since the performance is less than 50%. Random forest model performs slightly better than an artificial neural network (ANN) and Long short-term memory (LSTM) models.

3.2 Investigation and correction

We used tree-based classification methods to investigate the problem of low F1 score. We added AdaBoost Classifier, Gradient Boosting Classifier beside the use of Random Forest Classifier.

The initial results of each model by using cross-datasets evaluation shown in table 1. GradientBoost result is better than other models reported in [1], but

Table 1. Result before the fix (Train → Test).

	NSL-KDD → gureKDD		gureKDD ->NSL-KDD	
	Accuracy	F1	Accuracy	F1
Random Forest	0.9687	0.2969	0.5273	0.0362
AdaBoost	0.7460	0.1378	0.5340	0.0673
GradientBoost	0.9727	0.3755	0.9727	0.3757

still, it falls when considering F1 score. All F1 score is less than 48%. This means that the models still do not learn about the intrusion.

We use feature importance as a method to check the features as following:

- Run 5000 estimators/trees in Random Forest Classifier for both the datasets as training. So, we have 2 Random Forest Classifier and each one has 5000 estimators/trees.
- Check the performance estimators/trees inside Random Forest Classifier independently in the testing set by using F1 score.
- report the top features each of best and worst performing estimators. The top feature is the sum of importance value of each feature that estimators group.
- analysis each feature in both datasets.

After running the experiment and analysing the features, we found that there is a mismatch between both datasets on *service* feature naming. NSL-KDD contains 70 distinct labels of service, while gureKDD contains 24 labels. For example, the HTTP protocol service label as 'http' in gureKDD, but it has 4 different labels in NSL-KDD based on the used port number (http, http_2784, http_443, http_8001). We conjectured that this was the reason for the dramatic drop in performance in cross-datasets validation results reported in [1]. To check the conjecture we modified the datasets to have a compatible service feature naming.

The service feature was labelled differently in both datasets. There is no standard to label the service feature. To match the label of this feature, we need the port number (sending and responding) and the protocol used in the transport layer. Since that pieces of information are not available in NSL-KDD, we decided to match the gureKDD to the one in NSL-KDD. Each service needs to be mapped carefully since there is no documentation from original KDD on the meaning of each label. So service is more straightforward to change based on the specifications in [18], other labels need more understanding of the connection to decide on the meaning.

3.3 New results

After the fix, that is uniform relabelling, the performance rose dramatically in both datasets. The result of this fix shows in Table 2 with different machine learning models.

Table 2. Result of testing of all models after the fix in binary classification (Train → Test).

	NSL-KDD → gureKDD		**gureKDD → NSL-KDD**		**Average**	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
Decision Tree	0.979329544	0.98254637	0.89352734	0.88096654	0.9364	0.9317
RF (50)	0.992732726	0.993862766	0.937441505	0.930948116	**0.9651**	0.9624
RF (100)	0.992117395	0.993347618	0.933441963	0.92603982	0.9627	0.9596
RF (300)	**0.994688157**	**0.995507977**	0.923591239	0.914079772	0.9591	0.9547
AdaBoost(100)	0.98604708	0.98829732	0.940659992	0.936725038	0.9633	**0.9625**
GBM(50 dev)	0.990704093	0.992165347	0.891588168	0.875542054	0.9411	0.9338
GBM(50 exp)	0.992828758	0.993945005	0.893608139	0.877039804	0.9432	0.9354
LR(lbfgs)	0.959154831	0.966101389	0.794737303	0.732922737	0.8769	0.8495
GNB	0.424640532	0.051422805	0.792764465	0.753124248	0.6087	0.4022
CNB	0.316152527	0.00119196	0.791491883	0.808072043	0.5538	0.4046
Linear SVC	0.91628429	0.931950544	0.861470404	0.839516381	0.8888	0.8857
k-NN	0.95732406	0.964246723	0.897311419	0.885911563	0.9273	0.9251
QDA	0.45256268	0.170958267	0.640916528	0.405144336	0.5467	0.2881
LDA	0.920531445	0.93568379	0.852218938	0.833302953	0.8863	0.8844
ANN Model	0.983080231	0.985854019	0.919948558	0.911263537	0.9515	0.9485
CNN Model	0.951199387	0.959975533	**0.95345314**	**0.951141078**	0.9523	0.9555
LSTM Model	0.958949358	0.966230262	0.894799922	0.87976174	0.9268	0.9229
GRU Model	0.990871515	0.992313945	0.907519005	0.897989498	0.9491	0.9451
Best Result	**0.994688157**	**0.995507977**	**0.95345314**	**0.951141078**	**0.9651**	**0.9625**

3.4 Analysis

Fixing the Service label shows how important this feature in the learning process. The previously reported result [1] mostly did not reach over 10% of the F1 score with Cross-datasets evaluation method. Now we can gain more insight into the models and the datasets used.

When we use NSL-KDD for the training, most of the models getting over 95% of F1 score in the testing. 15 classifiers out of 18 get over 90%. The best model here is Random Forest classifier with 300 trees, which get 99.55% of F1 score.

When we use gureKDD for the training, the result differs from each classifier. Only 6 models out of 18 get over 90% of testing F1 score. The best classifier in the scenario is the Convolutional neural network (CNN) with an F1 score of 95.35%.

The best average model from both scenarios was AdaBoost (96.25% of F1 score in testing). Most of the times based Decision classification methods (Decision tree, Random Forest Classifier, AdaBoost Classifier, Gradient Boosting Classifier) perform better in both scenarios.

As to the datasets, the NSL-KDD is a better dataset to learn compared with gureKDD. Interestingly, gureKDD contains 18 times the number of connections of NSL-KDD, but a smaller number of distinct attacks.

For the feature importance, we notice that src_bytes shows before the fix and after the fix as one of the top 10 important features.

4 Multiclass Classification

4.1 Problem Overview

In this section we investigate cross-datasets evaluation for multiclass classification for intrusion detection tasks.

The labels on NSL-KDD and gureKDD are for several type attacks. The problem is that those labels are not matching. There are several different attacks in both datasets. We can remove data that have no matching label in the other datasets. But, we prefer not to lose any data, so we can compare the results with the results from binary classification.

Another option we have is to map the attacks label to the category of the attacks. Since the original KDD datasets give a specification of grouping the attacks, we decided to follow the same grouping/categories.

4.2 Mapping into 5 categories

Based on the original KDD datasets [19], there are 5 groups (Normal and 4 group of attacks) as following:

- **Normal**: Normal traffic.
- **DoS**: Denial of Service attacks, e.g. syn flood
- **R2L** R2L: unauthorized access from a remote machine, e.g. guessing password;
- **U2R** U2R: unauthorized access to local superuser (root) privileges, e.g., various "buffer overflow" attacks;
- **Probing** probing: surveillance and other probing, e.g., port scanning.

The grouping process was not straightforward since mismatch or lack of the documentation. So we needed to understand each attack before categorised it. The final list of groups is as following:

- **DoS**:'back', 'land', 'neptune', 'pod', 'smurf', 'teardrop', 'syslog', 'mailbomb', 'udpstorm', 'apache2', 'processtable'.
- **R2L**:'ftp_write', 'ftp-write', 'guess_passwd', 'imap', 'multihop', 'phf', 'spy', 'warezclient', 'warezmaster', 'warez', 'sendmail', 'named', 'snmpgetattack', 'snmpguess', 'xlock', 'xsnoop', 'worm', 'dict', 'dict_simple', 'guest'.
- **U2R**: 'buffer_overflow', 'loadmodule', 'perl', 'ps', 'perl_clear', 'perlmagic', 'rootkit', 'httptunnel', 'sqlattack', 'xterm', 'eject', 'eject-fail', 'ffb', 'ffb_clear', 'format', 'format-fail', 'format_clear', 'load_clear'.
- **Probing**: 'ipsweep', 'nmap', 'portsweep', 'satan', 'mscan', 'saint'.

4.3 Results

Table 3 and 4 summarise the results of testing F1 score for NSL-KDD, respectively, gureKDD used as training sets. The result of F1 score are in terms of Micro Average, Macro Average and Weighted Average.

Learning normality of the traffic getting the highest result on both sides. followed by the DoS attacks. The rest of the groups (R2L, U2R, and Probe) was difficult for all models to detect. The worst label to detect is Probe, even though the Probe label is third largest sample in both datasets (After normal and DoS labels).

Tree-based models perform well in this task most of the time. If we consider macro average and weighted average as a metric, then tree-based models will win here. k-NN model was not in the top of the list for the binary classification, but in this task performs better.

Learning from NSL-KDD gives better results than gureKDD in all metric of best results. This might be because the NSL-KDD contains more distinct number of attacks compared with gureKDD, even though that gureKDD is larger dataset (2,759,485 data points compare with 148,517 in NSL-KDD).

Table 3. Result of testing F1 score - NSL-KDD as training set

ML Models	Normal	DoS	R2L	U2R	Probe	Micro Avg	Macro Avg	W. Avg
Decision Tree	0.9268	0.8830	0.0973	0.0055	0.2140	0.8507	0.4253	0.8858
RF(50)	0.9936	0.9779	0.0037	0.0484	0.6159	**0.9718**	0.5279	**0.9756**
RF(100)	0.9814	0.9474	0.0000	0.0508	0.4664	0.9399	0.4892	0.9501
RF(300)	**0.9941**	0.9479	0.0000	0.0541	0.4161	0.9403	0.4824	0.9545
AdaBoost(100)	0.8913	0.5627	0.1455	0.0938	0.0927	0.5996	0.3572	0.6868
GBM(50 dev)	0.9923	0.9279	**0.4595**	0.0339	0.3482	0.9190	0.5524	0.9413
GNB	0.0690	0.7488	0.0094	0.0002	0.1937	0.5843	0.2042	0.4578
CNB	0.7221	0.9678	0.0499	0.0000	0.0016	0.8025	0.3483	0.8456
Linear SVC	0.8935	0.7016	0.0369	0.0000	0.1061	0.6619	0.3476	0.7667
k-NN	0.9473	**0.9830**	0.0749	**0.3441**	**0.7471**	0.9527	**0.6193**	0.9624
QDA	0.5798	0.0906	0.0136	0.0011	0.1825	0.4223	0.1735	0.2928
LDA	0.8718	0.0454	0.3271	0.0284	0.0625	0.3547	0.2670	0.3844
ANN	0.9268	0.8830	0.0973	0.0055	0.2140	0.8507	0.4253	0.8858
CNN	0.8990	0.1330	0.1547	0.0230	0.0472	0.5403	0.2514	0.4448
LSTM	0.7531	0.2167	0.1638	0.0624	0.0864	0.4915	0.2565	0.4335
GRU	0.9885	0.8841	0.2361	0.0711	0.1598	0.8730	0.4679	0.9107
Best Result	**0.9941**	**0.9830**	**0.4595**	**0.3441**	**0.7471**	**0.9718**	**0.6193**	**0.9756**

5 Discussion

The applicability of the machine learning-based intrusion detection systems should be given more attention in the research done in this area. Using cross-datasets evaluation is one approach to measure the practicality of detection classifiers.

Using two different datasets that have been generated from two different computer network should give us a clearer picture of how much should we trust the model. And since we want to understand the behaviour of the attack, different attacks in both datasets will pose a more challenging task.

The result from both classification problems (binary and multiclass) tend to give a higher result for detecting the normality. This might give an insight for a better intrusion detection system based on normality detection (as a variant of anomaly-based intrusion detection based on machine learning).

Table 4. Result of testing F1 score - gureKDD as training set

ML Models	Normal	DoS	R2L	U2R	Probe	Micro Avg	Macro Avg	W. Avg
Decision Tree	0.9199	0.9210	**0.4266**	**0.1789**	0.7552	0.8955	**0.6403**	**0.8910**
RF(50)	0.9242	0.9183	0.0064	0.0000	**0.7739**	**0.8980**	0.5246	0.8831
RF(100)	0.9220	0.9151	0.0043	0.0000	0.7290	0.8926	0.5141	0.8765
RF(300)	0.9169	0.9164	0.0027	0.0000	0.6763	0.8870	0.5025	0.8693
AdaBoost(100)	0.8194	0.8730	0.2023	0.0000	0.4140	0.7785	0.4618	0.7833
GB(50 dev)	0.9178	0.8821	0.2308	0.0127	0.4894	0.8635	0.5066	0.8455
GNB	0.7394	0.1034	0.0608	0.0713	0.2166	0.4175	0.2383	0.4430
CNB	0.7814	0.7445	0.1198	0.0000	0.0000	0.6973	0.3291	0.6761
Linear SVC	0.9019	0.9092	0.0594	0.0087	0.2367	0.8490	0.4232	0.8187
k-NN	0.9067	0.8708	0.2719	0.1600	0.6493	0.8601	0.5717	0.8521
QDA	0.8146	0.0175	0.2216	0.0098	0.2050	0.5235	0.2537	0.4540
LDA	0.8956	0.8790	0.2012	0.0114	0.0644	0.8242	0.4103	0.7918
ANN	**0.9256**	0.8220	0.0000	0.0000	0.0000	0.8282	0.3495	0.7757
CNN	0.9228	**0.9221**	0.0418	0.0000	0.1978	0.8670	0.4169	0.8300
LSTM	0.9151	0.8874	0.1420	0.0000	0.5715	0.8639	0.5032	0.8515
GRU	0.9079	0.8818	0.1499	0.0000	0.4645	0.8562	0.4808	0.8358
Best Result	**0.9256**	**0.9221**	**0.4266**	**0.1789**	**0.7739**	**0.8980**	**0.6403**	**0.8910**

Our result indicates that when we use two different datasets, the dataset that contains more distinct attacks will perform better than more connections. The observation is that the NSL-KDD provides more attacks but fewer connections than gureKDD. When we use NSL-KDD as a training set, it will give a better result on the testing set across most models. This reflects the quality of leaning set. We speculate that more attacks in the dataset will provide better learning for the model, but this observation needs further investigation.

Based on our work here, deep learning models don't give an advantage when compared with classical machine learning models like tree-based. And by using tree-based models we have the advantage of easier interpretation of the learned model.

6 Conclusions

In this work, we demonstrate the results of the cross-datasets evaluation of different machine learning models after adjusting the datasets to reduce difference in the interpretation of the same feature names.

We have argued that Cross-datasets evaluation is a better evaluation strategy for the intrusion detection system to test practicality. We provided the explanation of the previously reported results that the models tested by this method fail to learn anything about the intrusion. We investigated the issue and provided a correction for the datasets. We have also examined many machine learning algorithms and reported their performance based on the cross-datasets evaluation. We show the performance of the different models in multiclass classification, after mapping the label of both datasets into 5 groups.

References

1. S. Al-Riyami, F. Coenen, and A. Lisitsa, "A re-evaluation of intrusion detection accuracy: Alternative evaluation strategy," in *Proceedings of the 2018 ACM SIGSAC*

Conference on Computer and Communications Security. ACM, 2018, pp. 2195–2197.
2. M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 2009, pp. 1–6.
3. I. Perona, I. Gurrutxaga, O. Arbelaitz, J. I. Martín, J. Muguerza, and J. M. Pérez, "Service-independent payload analysis to improve intrusion detection in network traffic," in *Proceedings of the 7th Australasian Data Mining Conference-Volume 87*. Australian Computer Society, Inc., 2008, pp. 171–178.
4. T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.
5. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
6. Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.
7. J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.
8. V. N. Vapnik, "The nature of statistical learning," *Theory*, 1995.
9. K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
10. N. Hjort, *Pattern recognition and neural networks*. Cambridge university press, 1996.
11. C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.
12. A. Ng, "Machine learning yearning: Technical strategy for ai engineers in the era of deep learning," *Retrieved online at https://www. mlyearning. org*, 2019.
13. G. M. L. C. Course. (2020) Classification: Precision and recall. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall
14. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
15. Wikipedia. (2018) Accuracy paradox. Accessed: 2018-04-17. [Online]. Available: https://en.wikipedia.org/wiki/Accuracy_paradox
16. J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.
17. M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2003, pp. 220–237.
18. M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry." *RFC*, vol. 6335, pp. 1–33, 2011.
19. S. Stolfo, W. Fan, W. Lee *et al.*, "Kdd-cup-99 task description," 1999.