Evaluation of the scikit-learn library for anomaly detection in network data

Leon Lukas
Munich University of Applied Sciences
Department of Computer Science and Mathematics
München, Deutschland
l.lukas@hm.edu

***Abstract*—The importance of Machine Learning is steadily increasing in many fields including Anomaly-based Network Intrusion Detection Systems (A-NIDS). There is not much information on how well, standardized models work in the area of A-NIDS. In this paper, the performance of the library scikit-learn is evaluated on different datasets. This is done by applying commonly used models on the well knows datasets KDD CUP 99, NSL-KDD as well as Kyoto2006+. Also, combinations of single models were evaluated. The results show the efficacy of the scikit-learn library and discuss which models can be applied for specific use cases.**

***Index Terms*—Anomaly Detrection, Network, KDD CUP 99, NSL-KDD, Kyoto2006+, Evaluation, Scikit-learn, Computational Effort**

I. Introduction

Monitoring network connections is a well-known technique in the cyber-security industry. It can detect many types of attacks, including but not limited to malware and viruses. These attacks can not only impact the host computers but also the whole network. Because network traffic is changing and growing at a fast pace there is a need for new techniques in the area of Anomaly-based Network Intrusion Detection Systems (A-NIDS). Conventional network anomaly detection follows a signature-based approach. These systems need to be updated as new signatures become available to be abler to detect new threats. This might be a problem if real-time detection is required. The area of machine learning has become popular in recent years due to better hardware, an increase in computational power, and emerging of new technologies. Even though machine learning for finding anomalies has been around for some time in a wide range of applications such as network intrusion, credit card fraud, crowd surveillance and healthcare, finding ways to effectively use it in real-time in networks is still a challenge. There have been many different approaches including specially designed neuronal networks as well as more conventional techniques like SVM and K-Means. These approaches are all fairly complicated since they mostly rely on "hand-crafted" models. Only some papers take advantage of existing libraries like scikit-learn to build intrusion detection systems. To evaluate the capabilities, this work tests six different scikit-learn machine learning algorithms on the three well-known datasets.

II. Related work

As both Ciber-Security and Machine learning are emerging topics, a lot of research is being done in this area. While some projects focus on anomaly detection taking network data as their use case, others take Machine learning as a tool to solve security-related issues.

A. FENCE GAN Towards Better Anomaly Detection

Generative Adversarial Networks(GANs) provide a state-of-the-art tool to detect anomalies because of their ability to model high-dimensional data distributions. The Loss-function of the standard GAN is not perfectly aligned with the objective of finding anomalies as it encourages the model to generate data which is overlapping with the real data. The authors of [1] propose to modify the loss-function in such a way, that the generated data lies at the boundaries of the data instead of overlapping with it. They test their model on the MNIST, CIFAR10 and KDD99 data-sets, showing improved accuracy over other state-of-the-art methods.

B. An Adaptive Diversity-Based Ensemble Method for Binary Classification

The authors of [2] propose a ensemble model which is capable of performing adaptive selection of the best base model for each unknown instance instead, of linearly combining them. The resulting models yields 18.5% better results (F1 Score) compared to the base model on the Repeat Buyers Prediction dataset.

C. Evaluation of Machine Learning Techniques for Network Intrusion Detection

[3] provides an overview of different techniques in the area of network intrusion detection. For this they survey 18 different papers. Additionally they compare the performance of six different models as well as an ensemble model which combines the six models. Instead of the commonly used KDD'99 data set they use the Kyoto2006+ data set which eliminates several of the drawbacks of the KDD'99 set. In their findings the RBF algorithm performs best followed by the ensemble model. As a result they identify ensemble techniques having potential and see further work in this area as worth-pursuing.

D. Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives

The authors of [4] analyse several disadvantages of the KDD CUP 99 dataset and compare it to different datasets regarding size and redundancy, features, skewedness as well as non-stationarity. Additionally, they use oversampling and undersampling to achieve uniform distribution of pattern responses. The modified datasets are used to train various scikit-learn classifiers to evaluate the effect of different properties of the dataset on the performance of the classifiers. Even though a wide range of classifiers are tested, no boosting techniques are used. Their results show how imbalances can hamper the performance of classifiers on the smallest classes. Interestingly reducing the imbalances can lead to a decrease in performance on the bigger classes as the model can not concentrate on the bigger classes and is forced to underfit compared to the base dataset.

III. Background

Machine learning is divided into two main types: supervised and unsupervised learning [5]. Other types can be deduced from the two main types. Supervised learning has two subcategories of its own – classification and regression. While the output of classification is a discrete class, regression assigns a continuous value like the predicted house price to a given input. In this paper, all experiments fall into the category of classification. More specifically binary classification, a subset where only two possible labels can be assigned to a data point. In the case A-NIDS normal and anormal.

Methodically, a classifier is a function which maps instances of data to given classes. During the training of a model the machine learning algorithm estimates the parameters of the function to fit the training data. The prediction of a binary classifier can be of two types: A discrete class label (positive or negative) or a probabilistic score which indicates the confidence of a data point belonging to a class.

A. Ensemble models

Ensemble learning helps improve the predictive performance of models by combining several weak models into one big strong model. Generally there are two main types of ensemble models: Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting. It is usually applied to decision-tree methods. Bagging is a special case of the model averaging approach. There multiple classifiers are trained on different subsets of the data. Each classifier returns a prediction for an unknown sample. The bagging classifier then returns the class with the most votes [6]. Boosting on the other hand trains multiple weak models in series and tries to correct the earlier models faults in the later predictors.

B. Model evaluation

A broadly used way to visualise and interpreted the performance of a classifier on a dataset is the confusion matrix [7]. It is a nxn matrix (n represents the number of classes) constructed from the Cartesian product of actual classes and predicted classes. In the case of binary classification n is equal to 2 and represents the positive and negative classes. The confusion matrix contains the four basic-metrics from which more advanced metrics can be derived:

- True Positives (TP): represents the number of normal instances correctly classified or predicted as normal
- False Negatives (FN): represents the number of normal instances incorrectly classified or as malicious
- False Positives (FP): represents the number of malicious instances incorrectly classified or predicted as normal
- True Negatives (TN): represents the number of malicious instances correctly classified

To better measure and compare the performance of classifiers advanced metrics can be calculated using the basic performance metrics. As the goal of any A-NIDS should be to maximise TP and TN while minimizing FP and FN the metrics used to compare the performance will be:

- Precision: TP/(TP + FP)
  The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The score can be misleading if the negative class is underrepresented in the dataset[8].
- Recall: TP/ (TP + FN)
  The recall is intuitively the ability of the classifier to find all the positive samples.

Because it is somewhat difficult to compare models on two metrics, this work will additionally use the ROC (the Receiver Operating Curve) metric to compare the results of different algorithms[8]. To keep comparability to other papers the F1-score will also be used to rank the performance of the classifiers. F1-score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

C. Data-sets used

The choice of dataset is key to any machine learning task. To deliver comparable results this work uses the well-known datasets KDD CUP 99, NSL-KDD and Kyoto 2006+.

1) KDD CUP 99: The KDD CUP 99 is a modification of the DARPA98 dataset originally an IDS program conducted at MIT's Lincoln Laboratory, which was evaluated first in 1998 and again in 1999. The DARPA98 dataset was subsequently filtered for use in the International Knowledge Discovery and Data Mining Tools Competition [9], resulting in what we recognize as the KDD CUP 99 dataset [10, 4]. The KDD CUP 99 is widely used in research[11] despite its flaws found by[12, 4]. Shortcomings of the dataset are its unrealistic network architecture, overt synthesis of data, high tolerance for false alarms, and questionable evaluation methodology. The authors of [13] provide addition insight to KDD 99's idiosyncrasies as

TABLE I
KDD CUP 1999 TRAIN AND TEST DATA DISTRIBUTION[4]

Class	Training Set	Percentage	Test Set	Percentage
Normal	812,814	75.661%	60,593	19.481%
DoS	247,267	23.002%	229,853	73.901%
Probe	13,860	1.289%	4,166	1.339%
R2L	999	0.093%	16,189	5.205%
U2R	52	0.005%	228	0.073%
Total	1,074,992	100%	311,029	100%

well as proposing a improved version called NSL-KDD. The summarized pertinent aspects of the KDD 99 dataset taken from [4] are the following:

1) *Genesis:* This dataset originates from the TCPdump data of simulated network traffic captured in 1998 at Lincoln Labs. Seven weeks (five million connection records) are used as a training set , a further two weeks as a test set with two million examples.
2) *Target classes:* KDD-99 has five classes of patterns: Normal, DoS (Denial of Service), U2R (User to Root), R2L (Remote to Local) and Probe (Probing Attack). Table I lists the distribution of patterns into target classes.
3) *Size and redundancy:* KDD's training and test datasets are highly redundant.
4) *Skewedness:* The skewed nature of the dataset is evident from Table I 98.61% of the data belongs to either the Normal or DoS categories.

2) NSL-KDD: This dataset was chosen for additional comparability as it provides an improvement over KDD CUP 99, while still containing the same features. The dataset originates from an adaption of the KDD CUP 99 dataset done by [13] to overcome some of the drawbacks of the original dataset. Improvements over the original dataset include the reduced skewedness and better distribution of data between the training and test set. To only have unique datapoints the dataset is also greatly reduced in size, thus being less computationally expensive to use for training machine learning models [4].

3) Kyoto 2006+: To eliminate the drawbacks of the KDD CUP 99 dataset Song et al. [14] created a new dataset called Kyoto2006+ containing 24 features with 14 being identical to KDD CUP 99. The six out of the 10 extra features contain information on connections which were omitted in the original dataset but deemed relevant later by researchers working in the field [15]. The data was collected using 348 honeypots in the Kyoto University. The labelling of the traffic was done by the three security-software SNS7160 IDS, Calm Antivirus and Ashula.

IV. Methods

A. Hardware used

All experiments were conducted using the Python programming language on a Jupyter-Hub-server running Ubuntu 18.04.6 LTS. The server has two Intel(R) Xeon(R) E5-2650 v3 processors with 10 cores each as well as 256GB of RAM.

B. Prepossessing

Even though the used datasets contain already processed data, some additional steps are necessary to prepare the data for use with the machine learning models.

1) Feature scaling: Scaling is a commonly used technique in Machine Learning. There continuous features are transformed to a given range (0 to 1 in this case). Scaling was applied to all continuous features in all three datasets.

2) Integer encoding: Categorial features need to be encoded before they can be processed by a model. Integer Encoding takes categorial features such as Protocol and maps all unique values to an integer. Protocol: *tcp* would be mapped to 0, *udp* to 1 and so on. Integer Encoding was used specially needed for the scikit-learn[16] *HistGradientBoostingClassifier* as it takes integer-encoded values opposed to One-Hot encoded data commonly used by other models.

3) One-Hot encoding: Most models need one-hot encoding for categorical variables. This means creating a binary column for each category in a feature and setting the present feature to one while setting all the other columns to zero. This method was applied for all models except the *HistGradientBoosting-Classifier*.

4) Binarization : For all three datasets the target values where binarized to reduce the complexity of the analysis as well as keeping the three datasets comparable. In the case of KDD CUP 99 and NSL-KDD the targets were mapped from the specific attack type to a binary target indicating whether the sample was normal or anormal. For the Kyoto2006+ dataset the two anormal labels *known attack* and *unknown attack* were combined.

5) Feature selection: KDD CUP 99 and NSL-KDD were used with all available features. For Kyoto2006+ the features *IDS detection* ,*Malware detection* and *Ashula detection* where omitted as they provide to much information on the class of the sample. Additionally the feature *IP-Adress* was left out as the encoding of IP-Addresses is a research topic on its own.

C. Dataset selection

As the NSL-KDD dataset is already reduced in size, the whole training and test set were used for developing the models.

From the KDD CUP 99 test data all rows with the feature *service: icmp* were removed as it didn't show up in the training data and seems to be an error in the data. For our training purposes the first 1.000.000 rows were selected. After dropping all duplicate rows 582542 training and 78835 testing datapoints remained.

From the Kyoto2006+ records first a base dataset from 03.-06.2015 containing 10812247 samples was chosen. Subsequently duplicates inside the dataset were dropped resulting in a reduced set of size of 4994457. This set was randomly split into training and test data.

D. Models

For this work the following five scikit-learn 1.01[16] models as well as the separate XGBoost-Classifier, which seemed promising for the use case of A-NIDS were evaluated:

TABLE II
DISTRIBUTION OF CLASSES FOR KYOTO2006+

Dataset	training	training	testing	testing
Anomal	4164568	90.6%	362166	90.6 %
Normal	430332	9.4%	37391	9.4%
Total	4594900	100%	399557	100%

TABLE III
DISTRIBUTION OF CLASSES FOR NSL-KDD

Dataset	training	training	testing	testing
Anomal	67307	53.5%	12832	56.9%
Normal	58615	46.5%	9711	43.1%
Total	125922	100%	22543	100%

- **HistGradientBoostingClassifier**: This implementation of a boosting-tree algorithm is specially designed for datasets with more than 10.000 samples, It is inspired by the LightGBM model developed by Microsoft[16].
- **BaggingClassifier**: A Bagging classifier is an ensemble meta-estimator, that fits base classifiers, in this case a decisiontree each on random subsets of the original dataset. Subsequently, it aggregates their individual predictions (either by voting or by averaging) to form a final prediction [16].
- **RandomForestClassifier**: A RandomForestClassifier is a special case of a BaggingClassifier, were only averaging is used on decisiontrees.
- **XGBClassifier**:The Extremegardientgoostingclassifier optimized library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.
- **MLPClassifier**: The Multi-Layer Perceptron classifier, also known as neuronal network classifier implements the widespread technique of deep learning. In scikit-learn the model can only be modified in basic ways compared to specific deep learning libraries like Keras[17].
- **SVC**: This implementation of a support vector machine is based on libsvm. Its training time scales at least quadratically with the number of samples. For this reason the training set was reduced for this model [16].

E. Training

Due to the high number of libraries and datasets used, hyperparameter tuning is not in the scope of this work. During training the models were used with default parameters provided by the library. Some common-sense parameter-tuning was done to some models especially to the *HistGradientBoostingClassifier* as it seemed the most promising. Exact configurations can be found on GitHub [18]. Each model was evaluated using the metrics Precision, Recall, ROC and F1-score with normal being the positive class. Additionally, the time to train the models, as well as the duration of running the test set were measured. Size of train and test set can be taken from tables II to IV. Training the MLP and SVC was done using a smaller set of size 100000 to keep the training time in a reasonable period.

TABLE IV
DISTRIBUTION OF CLASSES FOR KDD CUP 99

Dataset	training	training	testing	testing
Anomal	527372	94.7%	46340	58.8%
Normal	55170	5.3%	32495	41.2%
Total	582542	100%	78835	100%

TABLE V
PERFORMANCE OF CLASSIFIERS ON KDD CUP 99

Classifier	Precision	Recall	ROC	F1
HistGradientBoostingClassifier	0.843	0.964	0.853	0.899
BaggingClassifier	0.850	0.980	0.867	0.910
RandomForestClassifier	0.844	0.974	0.860	0.905
XGBClassifier	0.843	0.983	0.860	0.907
MLPClassifier	0.864	0.984	0.882	0.920
SVC	0.853	0.980	0.870	0.912

TABLE VI
PERFORMANCE OF CLASSIFIERS ON NSL-KDD

Classifier	Precision	Recall	ROC	F1
HistGradientBoostingClassifier	0.737	0.757	0.775	0.746
BaggingClassifier	0.672	0.771	0.742	0.718
RandomForestClassifier	0.680	0.760	0.743	0.718
XGBClassifier	0.733	0.806	0.791	0.768
MLPClassifier	0.720	0.720	0.766	0.739
SVC	0.248	0.003	0.497	0.006

V. RESULTS

During the experiments six different measurements were taken for six classifiers on three datasets resulting in 108 values. When interpreting the results the different sizes and compositions of the datasets (see tables II to IV) need to be taken into account. While ROC and F1-Scores can be compared independently, Precision and Recall need to be looked at in combination.

A. Performance

The performance of the models can be taken from tables V to VII. When comparing the results across all three datasets several interesting findings can be made.

Generally, one can see the best results were achieved on the Kyoto dataset using the *HistGradientBoostingClassifier* with

TABLE VII
PERFORMANCE OF CLASSIFIERS ON KYOTO2006+

Classifier	Precision	Recall	ROC	F1
HistGradientBoostingClassifier	0.978	0.966	0.982	0.972
BaggingClassifier	0.834	0.621	0.804	0.712
RandomForestClassifier	0.892	0.541	0.767	0.673
XGBClassifier	0.808	0.505	0.746	0.062
MLPClassifier	0.684	0.369	0.676	0.480
SVC	0.018	0.018	0.457	0.018

almost perfect F1 and ROC-scores. The second-best scores were attained by all classifiers on the KDD CUP 99 data with ROC-scores around 0.86 and F1-scores around 0.91. Classifiers on the NSL-KDD data performed mostly with ROC and F1-scores around 0.75. While all classifiers performed relatively similar on the KDD CUP 99 data, results on NSL-KDD and especially Kyoto2006+ vary in between classifiers. This was especially true for the *SVC* as well as the *MLP-Classifier*. It is quite surprising as the data is relatively similar for all three datasets. Another interesting discovery is the ratio of Precision to Recall. On the KDD CUP 99 data Recall is always higher than Precision. On NSL-KDD the classifiers performed quite similar on both scores while on the Kyoto2006+ dataset, the models have higher Precision than Recall. This means classifiers on the KDD CUP 99 dataset made more false positive errors while classifiers on the Kyoto data had the tendency to falsely classify data as negative.

Classifiers trained on the Kyoto2006+ data had the biggest differences in performance while also achieving the best overall score using the *HistGradientBoostingClassifier*. The results are better than those produced by [15]. As the *HistGradientBoostingClassifier* seemed promising since the beginning of the experiments, hyperparameter optimization was done to it resulting in these good results. On this dataset bagging (*BaggingClassifier, RandomForestClassifier*) as well as boosting techniques (*HistGradientBoostingClassifier, XGBClassifier*) seem applicable. Also the *MPLClassifier* has further potential if tuned correctly.

As all algorithms performed similar on KDD CUP 99 no further comparisons within the dataset can be made. Compared to the results from [1] with an F1-Score of 0.95 the outcome of the experiments is very promising as none of the models were tuned while [1] implemented a GAN especially for identifying anomalies in the KDD CUP 99 dataset.

On the NSL-KDD data boosting-techniques performed little bit better than bagging. Also compared to the original KDD data the *MPLClassifier* hat a bigger loss in performance compared to the other models. The *SCV* didn't work at all similar to the Kyoto2006+ data.

B. Computational effort

Since the size of the computed data varies from dataset to dataset, comparing training and runtime only makes sense within one dataset. Some differences in time originate from the complexity of the algorithm other parts from how well a library is multithreaded. Higher training times increase time or the hardware cost of developing a model while runtime is important for running a model in real time. All measurements can be found in tables VIII to X. The most interesting findings are discussed in the following section.

The biggest outlier in training time is the *MPLClassifier* which is no surprise as Neuronal Networks are known to be expensive to train. Also the *SVC* has a long training time considering both the *SVC* and *MPLClassifier* were trained on a reduced training set.

TABLE VIII
COMPUTATIONAL EFFORT OF CLASSIFIERS ON KDD CUP 99

Classifier	trainingtime	runtime
HistGradientBoostingClassifier	17658. ms	134.8ms
BaggingClassifier	68185.1 ms	1123.6 ms
RandomForestClassifier	8523.5 ms	190.0 ms
XGBClassifier	21138.7 ms	27.8 ms
MLPClassifier	212428.2 ms	333.5 ms
SVC	44595.8 ms	5331.9 ms

TABLE IX
COMPUTATIONAL EFFORT OF CLASSIFIERS ON KYOTO2006+

Classifier	trainingtime	runtime
HistGradientBoostingClassifier	69880.6 ms	551.5ms
BaggingClassifier	227561.1 ms	3671.3 ms
RandomForestClassifier	75139.2 ms	734.2 ms
XGBClassifier	323574.4 ms	91.3 ms
MLPClassifier	1705627.8 ms	1115.9 ms
SVC	47161.1 ms	15868.7 ms

Training time doesn't correlate to runtime which can be clearly seen for the *MPLClassifier*. Considering their runtime, the boosting algorithms are clearly performing best followed by the *RandomForestClassifier*. The *SVC* and *BaggingClassifier* have long runtimes which makes them unsuitable for use cases with high throughput which is often needed for A-NIDS.

C. Combination of models

After comparing the individual classifiers, ensemble models were build using the best performing classifier on each dataset. This was done using the scikit-learn *VotingClassifier*. On the KDD CUP 99 dataset the *BaggingClassifier* , *MLPClassifier* and *SVC* were combined resulting in a F1-Score of 0.919. Using NSL-KDD Data an ensemble was build out of the *HistGradientBoostingClassifier*, *MLPClassifier* and *XGBClassifier* which scored with 0.747. Finally, for the Kyoto2006+ dataset only the *BaggingClassifier* and *HistGradientBoostingClassifier* were combined because other classifiers had performances which didn't seem worth combing. This combination lead to a slight increase in precision to 0.990 while Recall dropped to 0.612 resulting in ROC and F1 scores of 0.806 and 0.756.

VI. DISCUSSION

The previous section details the results of the experiments. This section will discuss those findings with the properties of each dataset in mind.

TABLE X
COMPUTATIONAL EFFORT OF CLASSIFIERS ON NSL-KDD

Classifier	trainingtime	runtime
HistGradientBoostingClassifier	3480.8 ms	28.9ms
BaggingClassifier	16544.2 ms	858.2 ms
RandomForestClassifier	2843.7 ms	225.5 ms
XGBClassifier	2940.3 ms	11.8 ms
MLPClassifier	146024.6 ms	107.5 ms
SVC	9034.1 ms	1338.2 ms

The most obvious conclusion one can draw from the results is that classifier performance is highly influenced by the choice of the dataset as results have a higher correlation by dataset than by the algorithm used. The same findings were made by [4]. This leaves the question how classifiers trained on these datasets would perform in real world applications. There are several opinions [4, 15] doubting the applicability of the KDD CUP 99 data to real world applications.

Second one can see boosting algorithms performing constantly good, having the best score in two of the three datasets. [4] suggests the reason for this could lie in the ability of ensemble methods to build classifiers for each subclass of malicious data (see I) allowing it to become an expert for all the classes.

Comparing the computational effort, boosting algorithms are also the clear winner as they the need the shortest amount of time both in training the model as well as predicting new data. This is especially important in the case of real time application of A-NIDS.

While combing models did not result in an increase in overall performance, for the Kyoto2006+ dataset a slight increase in precision was achieved. This means the *BaggingClassifier* could identify some samples the *HistGradientBoostingClassifier* couldn't. [2] has found ensemble models to perform well on imbalanced datasets which is often the case for A-NIDS. For ensemble models to perform better than their base estimators they need to possess pairwise diversity which didn't seem to be the case for models tested in this work. Ensemble models seem promising especially if their base estimators are tuned to work well on specific parts of the data.

The results show how scikit-learn is able to provide almost state of the art results in the case of the *HistGradientBoostingClassifier* on the Kyoto data even slightly improved results over other scientific approaches[15]. Combined with the significantly lower development effort needed to develop a scikit-learn model compared to developing a model from scratch this promotes the use of predefined models for the use in A-NIDS. For projects with a lot of resources, scikit-learn can establish a baseline from which more specialised models can be developed if the scikit-learn models leave room for improvement. For ventures with limited resources scikit-learn offers the capabilities to develop a unique model for a special use case in a reasonable timeframe.

VII. Conclusion

This work evaluates six different machine learning algorithms on the three well known datasets KDD CUP 99, NSL-KDD and Kyoto2006+ comparing their computational effort as well a classification performance.

The results show how the performance is highly influenced by the choice of dataset. Even though KDD CUP 99 has been used by more than 50% of the researchers in the network security area [11] it has many flaws including being outdated as well as having a skewed class distribution. For future research the Kyoto2006+ suggest by [15] or UNSW-NB15 suggest by [4] should be used as they provide newer data with more balanced classes. Generally, all ensemble methods performed well due to their ability to build classifiers for each subclass of malicious data (see I) allowing them to become an expert for all the classes. The best results were achieved by boosting techniques, especially the *HistGradientBoostingClassifier*. On the Kyoto2006+ data the *HistGradientBoostingClassifier* had a ROC-score of 0.982 while the best model in [15] work had a score of 0.9741. For the KDD CUP 99 and NSL-KDD similar scores to [4] were achived. Also, regarding their computational effort boosting techniques performed best out of all the evaluated techniques. This shows how scikit-learn models are very useful due to their ability to provide good results with lower development effort than hand crafted models. They can be used to develop a baseline in projects with a lot of resources or develop a unique model for a special use case in smaller projects with limited staff.

VIII. Future work

Due to timely constrains models were not fine tuned during this work. However, hyperparameters are important for machine learning algorithms since they directly influence the behaviours of training algorithms and have a significant effect on the performance of machine learning models [19]. Future work should take the findings from this paper and train boosting algorithms on the UNSW-NB15 dataset suggested by [4]. This way the capabilities of the scikit-learn library could be fully explored.

Future work should also be done to determine applicability of classifiers trained on scientific datasets like KDD CUP 99 on real world applications. This could be done by training multiple classifiers on different datasets and running them in a real environment. This way, their performance could be compared in a more realistic way.

References

[1] Cuong Phuc Ngo et al. "Fence GAN: Towards Better Anomaly Detection". In: (2019). URL: https://arxiv.org/pdf/1904.01209.pdf.

[2] Xing Fan, Chung-Horng Lung, and Samuel A. Ajila. "An Adaptive Diversity-Based Ensemble Method for Binary Classification". In: *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. 2017, pp. 822–827. DOI: 10.1109/COMPSAC.2017.49.

[3] Marzia Zaman and Chung-Horng Lung. "Evaluation of machine learning techniques for network intrusion detection". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, pp. 1–5. DOI: 10.1109/NOMS.2018.8406212.

[4] Abhishek Divekar et al. "Benchmarking datasets for Anomaly-based Network Intrusion Detection: KDD CUP 99 alternatives". In: *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*. 2018, pp. 1–8. DOI: 10.1109/CCCS.2018.8586840.

[5] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[6] soumya7. *Bagging vs Boosting in Machine Learning*. 2021. URL: https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/.

[7] Kohavi Ron and Foster Provost. *UCI Glossary of Terms*. 1998. URL: http : / / robotics . stanford . edu / ~ronnyk / glossary.html.

[8] Tom Fawcett. "Introduction to ROC analysis". In: *Pattern Recognition Letters* 27 (June 2006), pp. 861–874. DOI: 10.1016/j.patrec.2005.10.010.

[9] S.J. Stolfo et al. "Cost-based modeling for fraud and intrusion detection: results from the JAM project". In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. 2000, 130–144 vol.2. DOI: 10.1109/DISCEX.2000.821515.

[10] KDD CUP. *KDD CUP 1999 Data*. 1999. URL: http : / / kdd . ics . uci . edu / databases / kddcup99/ (visited on 01/12/2021).

[11] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. "A Survey of Network Anomaly Detection Techniques". In: *J. Netw. Comput. Appl.* 60.C (Jan. 2016), pp. 19–31. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2015.11.016. URL: https://doi.org/10.1016/j.jnca.2015.11.016.

[12] John McHugh. "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory". In: *ACM Trans. Inf. Syst. Secur.* 3.4 (Nov. 2000), pp. 262–294. ISSN: 1094-9224. DOI: 10.1145/382912.382923. URL: https://doi.org/10.1145/382912.382923.

[13] Mahbod Tavallaee et al. "A detailed analysis of the KDD CUP 99 data set". In: *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*. 2009, pp. 1–6. DOI: 10.1109/CISDA.2009.5356528.

[14] Jungsuk Song et al. "Statistical analysis of honeypot data and building of Kyoto 2006+ dataset for NIDS evaluation". In: Apr. 2011, pp. 29–36. DOI: 10.1145/1978672.1978676.

[15] Marzia Zaman and Chung-Horng Lung. "Evaluation of machine learning techniques for network intrusion detection". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*. 2018, pp. 1–5. DOI: 10.1109/NOMS.2018.8406212.

[16] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[17] François Chollet et al. *Keras*. https://keras.io. 2015.

[18] Leon Lukas. *Project Github*. https://github.com/l0renor/Evaluation-of-the-scikit-learn-library-for-anomaly-detection-in-network-data. 2021.

[19] Jia Wu et al. "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb". In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40. ISSN: 1674-862X. DOI: https://doi.org/10.11989/JEST.1674-862X.80904120. URL: https://www.sciencedirect.com/science/article/pii/S1674862X19300047.