Analysis of Machine Learning Techniques for Anomaly Detection in the Internet of Things

Shane Brady, Damien Magoni, John Murphy, Haytham Assem, A. Omar Omar Portillo-Dominguez

Analysis of Machine Learning Techniques for Anomaly Detection in the Internet of Things

Shane Brady*, Damien Magoni‡, John Murphy*, Haytham Assem†, A. Omar Portillo-Dominguez*
*Lero@UCD, School of Computer Science, University College Dublin, Ireland
†Cognitive Computing Group, Innovation Exchange, IBM, Dublin, Ireland
‡LaBRI, University of Bordeaux, France
Email: shane.brady.1@ucdconnect.ie, damien.magoni@u-bordeaux.fr, j.murphy@ucd.ie,
haythama@ie.ibm.com, andres.portillodominguez@ucd.ie

***Abstract*—A major challenge faced in the Internet of Things (IoT) is discovering issues that can occur in it, such as anomalies in the network or within the IoT devices. The nature of IoT hinders the identification of issues because of the huge number of devices and amounts of data generated. The aim of this paper is to investigate machine learning for effectively identifying anomalies in an IoT environment. We evaluated several state-of-the-art techniques which can identify, in real-time, when anomalies have occurred, allowing users to make alterations to the IoT network to eliminate the anomalies. Our results offer practitioners a valuable reference about which techniques might be more appropriate for their usage scenarios.**

***Keywords*-Anomaly Detection; Internet of Things; Machine Learning; Comparative Study**

I. Introduction

The Internet of Things (IoT) is a notable technological revolution, contributing to a drastic growth in the number of Internet-connected devices worldwide. Also, how effectively these devices can communicate (over potentially heavily congested networks, or low power lossy networks) can have an important impact on the quality of the services supplied, and in their degree of success (or failure). Thus, correctly monitoring and evaluating IoT systems is critical [1].

Monitoring the performance and overall health of an IoT system offers many challenges [2]. This is a consequence of the creation and transmission of vast amounts of data, which considerably complicates the identification of anomalous behaviors. Log analytic tools can be used to detect such anomalies when they are well-defined and can be captured by strict parameters (e.g., thresholds on values), but are not suitable for detecting anomalies whose properties are less well-defined, or change over time. Machine Learning (ML) techniques have the potential to effectively detect this kind of anomalies [3] as they can learn from past behavior, making them more efficient at detecting anomalies when they occur, as they are trained to observe data which does not fit with previously observed behaviors. The contribution of this work is a comparative analysis of ML techniques to detect anomalies that could occur in an IoT environment. In this paper, methods to detect anomalies in both time series and non-time series data are evaluated with statistical metrics to properly capture their usefulness and effectiveness at detecting anomalies.

II. Related Work

ML has many uses in anomaly detection [4]. Among ML techniques, Artificial Neural Networks (ANNs) are widely use in the literature for classification tasks, specifically in the area of anomaly detection [5]. The following algorithms, often used in classification tasks and anomaly detection scenarios [6], are used in this paper: k-Nearest Neighbor (k-NN) is an intuitive and accurate classifier. It is a method to classify an object based on the majority class amongst its k-NN. There are problems of noise, time and memory with 1-NN. To avoid this, 3 or 4-NN are used for accurate performance measure. Logistic Regression (LR) is a method for statistically analyzing a data set with one (or more) independent variables that conclude an outcome (measured with a binary variable). Linear Discriminant Analysis (LDA) is a technique that attempts to find a linear combination of features that best separates two or more classes. Unlike LR, which tries to fit a line through data points to minimize the distance between the line and the points, LDA tries to maximize the distance between the data points and the line (while minimizing the distance across the points). Decision Trees are a technique for predicting the relationship between measurements of an item and its class value. A decision tree is a classifier represented as a recursive partition of an instance space. Moreover, one of the most popular algorithms to create trees is C4.5, which uses information gain as splitting criteria. Naive Bayes (NB) uses a normal distribution and rules of conditional probability to model numeric attributes. It is termed as *naive* because it relies on two simplifying assumptions: no hidden attributes influence the prediction process, and the predictive attributes are independent given the class.

A multilayer perceptron (MLP) is a neural network model mapping input data with a set of outputs [7], that can also distinguish data which is not linearly separable. A recurrent neural network (RNN) is a type of neural network where connections between units form a directed cycle, creating an internal state which allows it to exhibit dynamic behavior [8]. Long Short Term Memory (LSTM) is a form of RNN which attempt to solve the problem of long term dependencies [9]. This is achieved by having three gates used to decide if information gets passed through the network (thus protecting

the state of the cell). Finally, Gated recurrent units (GRUs) is a popular gating mechanism in recurrent neural networks [10].

III. Evaluation Metrics

This work focuses on classification tasks and standard metrics used for evaluating the performance of the ML models. Models are trained on a set of data which is the training set and predictions are made on a separate set of data containing the same features that exist in the test data. The results of this anomaly detection classification is binary (i.e., 1 for a detected anomaly, or 0 for a normal event). To perform an evaluation of these models we need to know the number of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) that occur when we make predictions on the test set [11]. A TP is a predicted anomaly which was an anomaly, a TN is a predicted normal event which was a normal event, a FP is a predicted anomaly which was actually a normal event and a FN is a predicted normal event which was actually an anomaly. Given these values, the following commonly used metrics are calculated: (1) *Precision* is used to test the ability of the model to retrieve relevant items. It is the fraction of relevant instances among the retrieved instances. (2) *Recall* is used to test the probability that the model will retrieve a relevant item. In binary classification tasks, recall is also known as sensitivity. It is not enough to use recall alone as if every instance was predicted as an anomaly then the recall of the model would be 100%. This is why it is often used in conjunction with precision. (3) *F1-score* is the harmonic mean of precision and recall, meaning it takes into account how high the two values are as well as how similar they are, making it a good indicator of the accuracy of the model. (4) *Accuracy* is a fraction of correctly classified instances over the total number of instances. In all cases, higher values are better.

IV. Comparative Analysis

The aim of these experiments is to analyze, using the standard ML metrics previously discussed, the performance of a set of state-of-the-art classification algorithms on different types of data which may be found in an IoT environment. In particular, we look at both time series and non-time series data. This is because these two complementary data types are commonly found in a typical IoT environment.

Non-Time Series: NSL-KDD Dataset. Non-time series data is data which has no discernible time-element. The ordering of the data in the dataset is not important, as it does not relate to time steps. This makes classifying non-time series data a fundamentally different problem from classifying time-series data as we are not trying to predict the next step (or several steps) in a sequence; instead, we are looking at the features of known data and given new data we try to predict what class it belongs to. The purpose of this experiment was to test various classification algorithms on a non-time series dataset. We chose 5 ML commonly-used techniques that have been found in the literature to be suitable for classifying non-time series data. They are k-Nearest Neighbors (kNN), Logistic Regression (LR), Linear Discriminant Analysis (LDA), Decision Trees (CART) and Nave Bayes (NB). We also look at using neural networks in the form of a multilayer perceptron (MLP). Also, we chose the NSL-KDD dataset [12], a revised version of the KDD 99 dataset with more variance of data and no duplicate records or missing values. This dataset comprises of rows of 41 features (independent variables) used to describe a network connection and one dependent binary variable that states whether the connection is a network attack or not. The tools we are using to perform this classification do not support categorical variables, so firstly we had to carry out one-hot encoding on each row of the dataset. This involves splitting a categorical feature (e.g. a feature whose value can be one of several values) into multiple binary features which indicate which of the values it is [13]. This task increases the number of features from 42 (including the output variable) to 124.

After getting the data into a form that we can process, the next step is to divide it into train and test data. The NSL-KDD dataset provides pre-defined train and test datasets. The training set consists of 125K records and the test set consists of 22K records. As we want to see how the algorithms we have chosen scale in terms of performance (both in classification and execution time), we also split the training set into multiple training sets of different sizes. We finish with 5 different training sets: The original training set with 100% of the values, as well as one with 80%, one with 60%, one with 40% and one with 20% of the original values of the training set. Finally, we also want to see how our algorithms perform when we do not use all of the features of the dataset, as well as seeing how necessary all of the features of the dataset are for effective classification. To do this, we use an extra trees classifier to gather the importance of each feature in the dataset. Based on the obtained importances, we can eliminate those features which are deemed by the classifier to be effectively useless for classification. This brings us from 124 features , after one-hot encoding, to just 49. This should give improved execution time for our algorithms with minimal cost to classification accuracy.

First, we will look at the 5 classification algorithms mentioned above. Figure 1 shows the classification accuracy for the 5 algorithms, using 100% of the training set for training

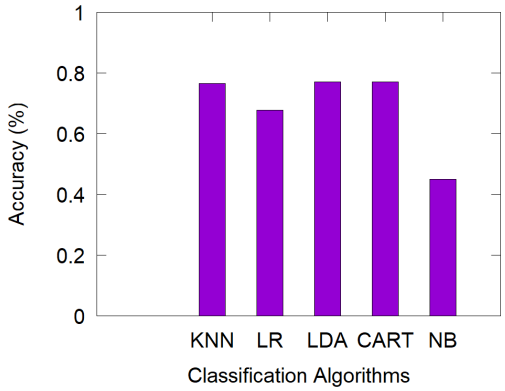


Fig. 1. Classification accuracy

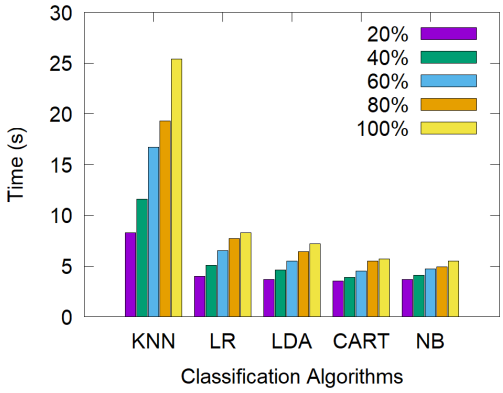


Fig. 2. Algorithm scaling using various percentages of the training set

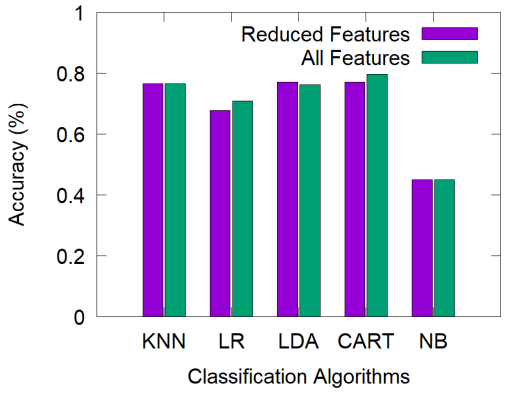


Fig. 4. Classification accuracy with and without feature reduction

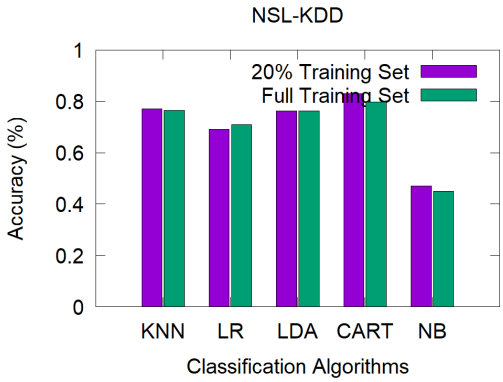


Fig. 3. 20% training set vs 100% training set

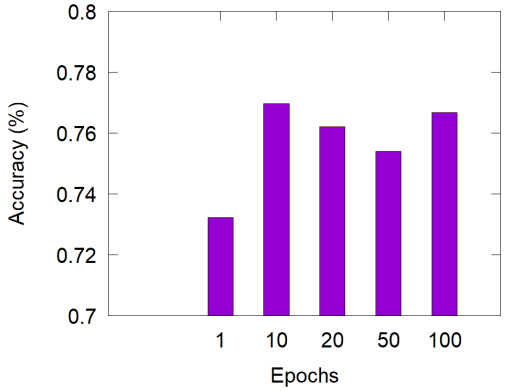


Fig. 5. MLP classification accuracy

and classifying on the test set, with all features included. Apart from the NB algorithm, the other four algorithms perform well, all hitting at least 70% accuracy. The decision trees show an impressive 80% classification accuracy, with little difference between the KNN and LDA algorithms. Moreover, the low accuracy exhibited by NB was the result of over-fitting the model with multiple dependent variables. Next, we look at what the trade-off is in execution time to get these scores in Figure 2. We observe that, while there is little difference in accuracy between KNN and LDA, it takes considerably longer to train KNN. We also see that the decision tree algorithm (CART), which consistently outperform the other algorithms in terms of classification accuracy, also scales very well in terms of time taken to train.

We also find that using smaller amount of values for training the dataset has little effect on the accuracy, which is believed to be due to the large volume of data contained in the initial dataset. Figure 3 shows the accuracy scores for the 5 algorithms with the 20% training set versus the full training set. From Figure 4, we also do not see a huge difference between the classification accuracy when using reduced features and all features. This would appear to be due to the number of features left, after feature reduction, still being quite large (49). Next, we look at the neural network approach, using MLP to train a more complicated network with our various training sets, then feeding data from our test set to the trained model for classification. For this experiment, we run the training process for different numbers of epochs (1, 10, 20, 50, 100) and gather the same metrics mentioned above. Figure 5 shows the accuracy when using 100% of the training set with all features. What we see from that figure is that the classification accuracy is greatly increased when we train for larger numbers of epoch (i.e., $\geq$ 10). This was due to the high complexity of the KDD dataset, which required more epochs to get a satisfactory accuracy. From the results for 10, 20, 50 and 100 epochs we also see variance but no real improvement in the accuracy. This is likely due to the network finding a local minimum when reducing the error rate during training and deviating either away (or back) towards that minimum during further epochs. To verify this, we can look at results for the other training sets, for instance Figure 6 which depicts the results of the 80% training set. From this figure, we can see a more predictable trend for the 1, 10, 20 and 50 epochs with the classification accuracy increasing by a reasonable margin for each value of epochs. However, for the 100 epochs the accuracy drops a bit, again likely due to finding a local minimum for the error rate during training, this time after 50 epochs. When looking at the 10 sets of results

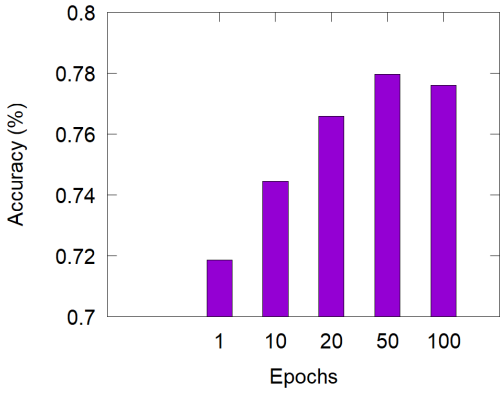


Fig. 6. MLP classification accuracy (80% train set)

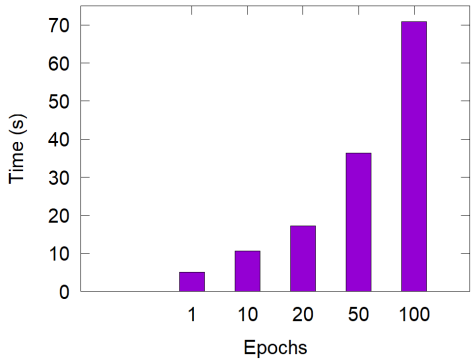


Fig. 7. MLP execution time over range of epochs

(for the 5 different number of epochs and with and without feature reduction), we find that 8 (out of 10) times 50 epochs gives a higher classification accuracy than 100 epochs.

We also find that the neural networks approach is computationally expensive, as shown by Figure 7, which depicts the results for training with 100% of the data with no feature reduction. To perform a single epoch of training takes about 5 seconds, but this time increases greatly for more epochs, taking roughly 70 seconds for 100 epochs. The highest accuracy we see from the MLP is 79%, which means it is consistently worse than the decision tree CART algorithm which regularly managed 80% (or higher) accuracy, and which takes a fraction of the time to train (approximately 10 times quicker).

Time-Series: Yahoo Anomaly Detection Dataset. Time-series data consists of data points which are indexed in order of time, meaning every piece of data has an associated time-step. When performing classification on time-series data, we first look at a sequence of data points up to a given time and then try to predict the next value (or sequence of values) for a time period. This dataset, provided by Yahoo for academic purposes [14], was created with the aim of benchmarking anomaly-detection algorithms. The dataset consists of real and synthetic time-series data, where each data point is tagged as either an anomaly or normal data. The real data consists of metrics of various Yahoo services, though it is not defined what these services are, as the data is user sensitive. Five files were chosen from the dataset (4x real, 1x synthetic) to give a spread of anomaly occurrence and distribution in the data. Four time-series algorithms are used to test for anomalies. These algorithms are Long Short Term Memory (LSTM), Gated Reccurent Unit (GRU), Recurrent Neural Network (SimpleRNN) and an Autoregressive Integrated Moving Average (ARIMA) model. The data is then split into train/test splits at a ratio of 0.5, 0.6, 0.7, 0.8 and 0.9. For ARIMA, we add every observation from the test data to our history of seen observations after we predict the next value and then retrain the model. This is why in the results the time increases for a shorter train set, as the model is retrained as many times as there are values in the test set. Each of the neural network algorithms (LSTM, GRU and SimpleRNN) are trained for 100 epochs before making predictions, and all use the same seeded random values throughout the tests. The data is normalized to small values (between 0 and 10) and an error threshold of 1 (i.e., 10% of the range) is used to predict anomalies (as this threshold is typically used to delimit the behaviour of a steady-state process [15]). For instance, if the predicted value is 0.5 and the actual value is -0.3, it is not considered an anomaly. Alternatively, if the prediction is 0.4 and the actual value is 2.2, it is considered an anomaly. Also, we compare our predicted binary classification results with the actual anomaly values to calculate the evaluation metrics discussed in Section III.

To illustrate the obtained results, we present the results of File 17, as it had the most anomalies spread out over time, and it is also representative of the overall results obtained (as the other files exhibited similar trends). Figure 8 shows the results of the TP/TN/FP/FN values for this file for the different algorithms and train/test splits. It can be noticed that, in general, all the executed algorithms consistently performed well, as they produced considerably more TP and TN values than FP and FN values. To complement this analysis, we present Figure 9 which shows how the train/test splits affect the test metrics by looking at the values, as a percentage of the total number of records in the set. There, it can be seen that the algorithms generally follow a similar pattern, mostly finding true negatives, which is explained by the fact that most of the data points are not anomalies. Meanwhile, Figure 10 shows the standard classification metrics for File 17. In general terms, the bigger the training set, the better the precision, with LSTM and GRU being the most accurate algorithms. Nevertheless, the accuracy does not improve with the training set size and is mostly similar for all the algorithms. Finally, ARIMA is a bit better at recall compared to the others.

Time-Series: IoT Testbed Dataset. As a second time-series dataset, we used our IoT testbed data generated by the process discussed in [16]. The testbed outputs room temperature values and those that are too low/high are flagged as anomalies. We used LSTM, GRU, SimpleRNN and ARIMA: for each algorithm, the test/train split was tried with 0.5/0.6/0.7/0.8/0.9 ratio as for the Yahoo dataset. To exemplify the obtained results, Figure 11 shows the precision, recall, F1-score and accuracy for one file from the testbed results. We can see

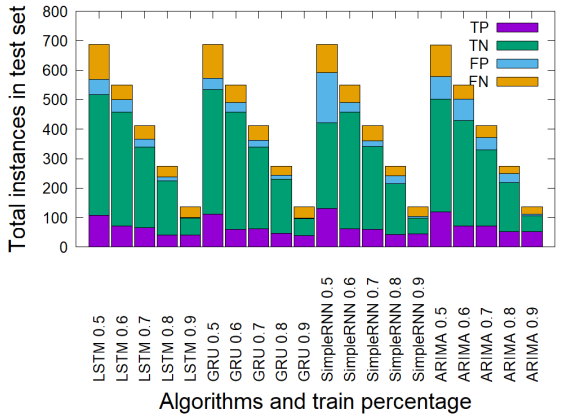
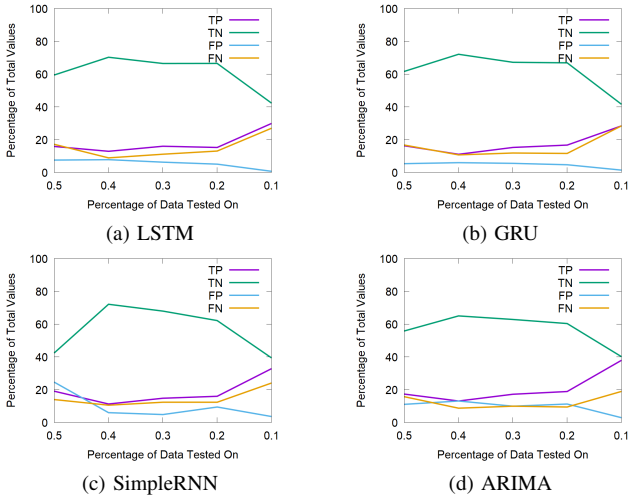


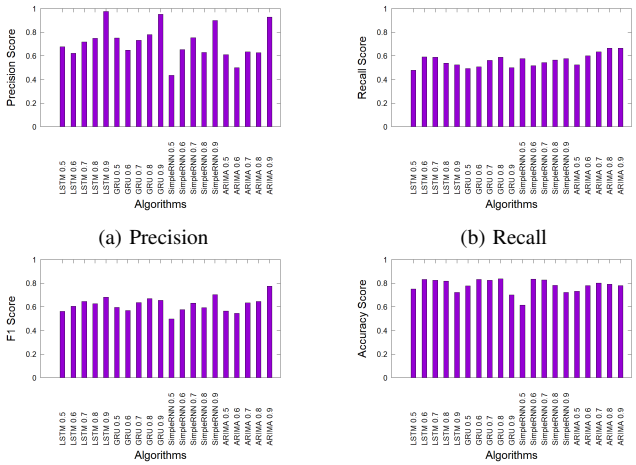
Fig. 8. Positivity values for File 17 for each algorithm and split



(a) LSTM (b) GRU

(c) SimpleRNN (d) ARIMA

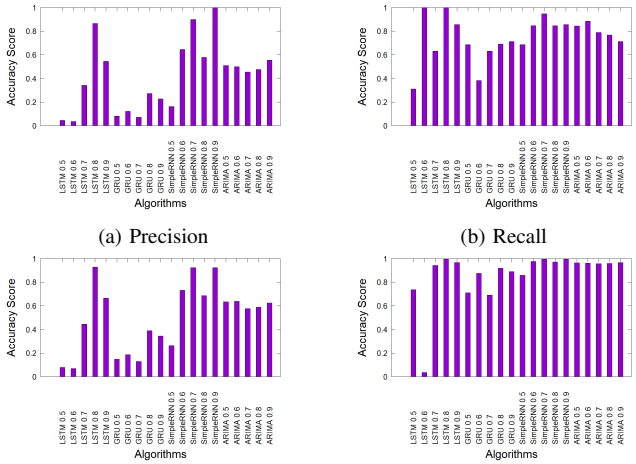
Fig. 9. Positivity values as a % of records for each algorithm and split



(a) Precision (b) Recall

(c) F1 (d) Accuracy

Fig. 10. Metric scores for Yahoo data



(a) Precision (b) Recall

(c) F1 (d) Accuracy

Fig. 11. Metric scores for the IoT testbed

that the neural network algorithms do not perform very well in regards to precision and recall when the train/test split is skewed towards the training. The recall at times is very good but can be inconsistent (partly because there are not that many anomalies in the data, so when one is missed it has a substantial effect on the scores). The accuracy is generally high across the board. Also, ARIMA is clearly the most consistent performer as its results do not vary greatly depending on the split (consequence of retraining the model at every step).

Figure 12 shows the TP/TN/FP/FN values of these models to provide a more granual insight of the results. The algorithms should maximize the number of TP/TN values, while minimizing the number of FP and FN values. This graph clearly depicts that ARIMA is the best performing model on average. An interesting result is that the SimpleRNN model, which was the weakest of the neural network algorithms in the Yahoo tests, is now the best in these tests. We believe this is due to the random nature of the temperature values. This indicates that there is no pattern, suggesting that the memory capabilities of GRU and LSTM actually hinder their performance, as they look further back to infer patterns which do not really exist. Again, we look at these metrics as a percentage of the number of samples tested from the test data to assess how the prediction accuracy deviates with regards to the different splits. A good algorithm should have a high number of TP and TN values and a low number of FP and FN values, but it is less useful if this only occurs with certain splits. We can see from Figure 13 that ARIMA consistently performs well across all the tested scenarios.

V. Conclusions and Future Work

IoT presents many challenges, including maintaining vast and complex networks. Having many small devices sending traffic at frequent intervals makes it difficult to identify when and where problems occur in a network, compared to more traditional networks. Thus, techniques need to be evaluated to help identify when and where these issues occur. To address this issue, this paper provides a comprehensive analysis of ML techniques for anomaly detection. To ensure that the analysis was comprehensive, both non-time series and time series data were examined, as they both have their own challenges and

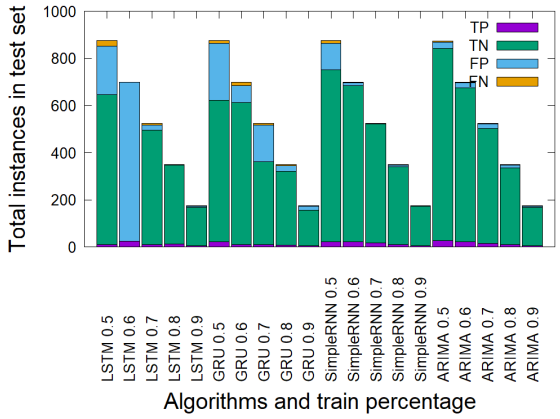
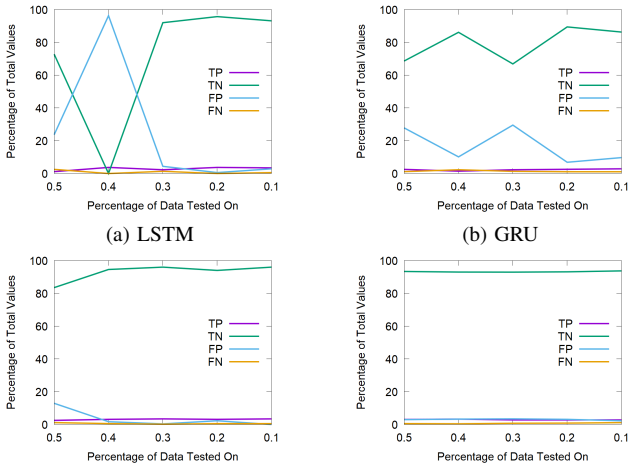


Fig. 12. Positivity values of testbed data across all splits



(a) LSTM (b) GRU

(c) SimpleRNN (d) ARIMA

Fig. 13. Positivity values as a % of records for each test split

methods for anomaly detection. This analysis was carried out with three data sets, one for non-time series and two for time series, with different variations of splits for training and test data, as well as identifying anomalies in the data before and after feature reduction. For the non-time series data, it was observed that the Linear Discriminant Analysis and Decision Tree algorithms provided the most consistent results, achieving roughly 80% accuracy (across all tested scenarios). Using neural networks for the non-time series data does not appear to be a viable option, as the experiments showed that, even with many epochs, neural networks do not outperform the other classification algorithms, despite taking longer to run. For the time series data, it was observed that when analyzing data with underlying trends (such as in the Yahoo data set), the neural networks with memory gates (LSTM and GRU) outperformed the others. However, with data without underlying trends (such as the IoT testbed data) the opposite was true, as the algorithms tried to infer patterns that did not exist. Also, it was observed that ARIMA generally performs well across both types of data and appears to be the most suitable for real-time prediction of anomalies, as the network can be retrained after every observation and make a new prediction even when observations are being made every second.

Finally, the algorithms used were chosen from those that are most frequently cited in the literature and which were deemed suitable for anomaly detection. Future work will focus on extending the set of evaluated algorithms to generalize the obtained knowledge of when each algorithm is best suited for. Likewise, further work will focus on diversifying more the parameters' configurations of the evaluated algorithms, with the aim of generating usability guidelines for practitioners.

ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094. Supported, in part, by Agence Nationale de la Recherche grant ANR-10-IDEX-03-02.

REFERENCES

[1] A. O. Portillo-Dominguez and V. Ayala-Rivera, "A requirements-based approach for the evaluation of emulated IoT systems," in *International Requirements Engineering Conference Workshops*, 2018.

[2] A. Shinde, "Challenges in performance monitoring of hyper connected IoT systems," in *International Conference on Internet of Things and Applications*, 2016, pp. 174–178.

[3] V. G. Aquize, E. Emery, and F. B. de Lima Neto, "Self-organizing maps for anomaly detection in fuel consumption," in *Latin American Conference on Computational Intelligence*, 2017.

[4] A. Gulenko, M. Wallschlager, F. Schmidt, O. Kao, and F. Liu, "Evaluating machine learning algorithms for anomaly detection in clouds," *IEEE International Conference on Big Data*, pp. 2716–2721, 2016.

[5] S. T. F. Al-Janabi and H. A. Saeed, "A Neural Network Based Anomaly Intrusion Detection System," *Developments in E-systems Engineering*, 2011.

[6] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[7] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.

[8] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model." in *Interspeech*, vol. 2, 2010, p. 3.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[11] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[12] "NSL-KDD dataset," Last accessed: 2018-05-11. [Online]. Available: http://www.unb.ca/cic/datasets/nsl.html

[13] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *International Conference on Machine Learning*, 2011.

[14] "Yahoo dataset," Last accessed: 2018-05-11. [Online]. Available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s

[15] A. O. Portillo-Dominguez, P. Perry, D. Magoni, and J. Murphy, "PHOEBE: an automation framework for the effective usage of diagnosis tools in the performance testing of clustered systems," *Softw. Pract. Exp.*, 2017.

[16] S. Brady, A. Hava, P. Perry, J. Murphy, D. Magoni, and A. O. Portillo-Dominguez, "Towards an emulated IoT test environment for anomaly detection using NEMU," in *Global Internet of Things Summit*, 2017.