




Network Anomaly Detection Using Machine Learning Techniques [†]

Julio J. Estévez-Pereira ^{1,*}, Diego Fernández ^{1,2} and Francisco J. Novoa ^{1,2}

¹ Department of Computer Science and Information Technologies, Faculty of Computer Science, Universidade da Coruña, Campus de Elviña, s/n, 15071 A Coruña, Spain; diego.fernandez@udc.es (D.F.); francisco.javier.novoa@udc.es (F.J.N.)

² Research Center of Information and Communication Technologies (CITIC), Campus de Elviña, s/n, 15071 A Coruña, Spain

* Correspondence: julio.jairo.estevez.pereira@udc.es

[†] Presented at the 3rd XoveTIC Conference, A Coruña, Spain, 8–9 October 2020.

Published: 19 August 2020



Abstract: While traditional network security methods have been proven useful until now, the flexibility of machine learning techniques makes them a solid candidate in the current scene of our networks. In this paper, we assess how well the latter are capable of detecting security threats in a corporative network. To that end, we configure and compare several models to find the one which fits better with our needs. Furthermore, we distribute the computational load and storage so we can handle extensive volumes of data. The algorithms that we use to create our models, Random Forest, Naive Bayes, and Deep Neural Networks (DNN), are both divergent and tested in other papers in order to make our comparison richer. For the distribution phase, we operate with Apache Structured Streaming, PySpark, and MLlib. As for the results, it is relevant to mention that our dataset has been found to be effectively modelable with just a reduced number of features. Finally, given the outcomes obtained, we find this line of research encouraging and, therefore, this approach worth pursuing.

Keywords: machine leaning; IDS; network security; distributed computing; network flow

1. Introduction

A network anomaly can be defined as a variation of the regular behavior of the network. That includes both unfortunate unintended events, and deliberate attacks planned to compromise the network's availability. In both cases, it is essential to be able to detect those quickly so we can react in time [1].

In the past few years, machine learning has been slowly taking its place as an alternative to policies-based traditional intrusion detection systems, as it presents quite a few advantages. Machine learning techniques allow the development of non-parametric algorithms, adaptative to our network and its modifications, and portable across applications [1].

Although there is still a gap between the deployment of machine learning based intrusion detection systems and their predecessors due to different challenges [2], it is a reality that traditional detection methods fall behind when it comes to handle large-scale volumes of data, as their analysis processes are complex and time-consuming. Nevertheless, machine learning and big data tools and techniques can help us overcome these shortcomings [3].

In this paper, we go through different machine learning and big data alternatives, as we model the Intrusion Detection Evaluation Dataset (UNB ISCX IDS 2012) and deploy it so it supports extensive data processing. The solution we will be deploying will be eventually based on the best model and implemented on Apache Structured Streaming, using PySpark and MLlib. The storage will be also distributed with Hadoop. Furthermore, for performance monitoring, we will use Zabbix and JMX.

2. Results

2.1. Machine Learning Process

The phases that we went through in this project are those defined by CRISP-DM methodology: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. In this subsection, we will be focusing on the modeling and evaluating phase, as our priority is to show the outcomes acquired. In the next subsection, we will address deployment related aspects.

2.1.1. Naive Bayes

For this project, we used a two-level Naïve Bayes model. First, we separated our binary features from our continuous ones. For the first group, we used Bernoulli Naïve Bayes, and for the latter Gaussian Naïve Bayes. After both were trained, we ensembled the probabilities given by each model for training a last Gaussian Naïve Bayes model. With this approach, we reached an 86.9% accuracy score in the ensembled model. As a relevant remark, we were reaching about a 96.9% accuracy in the ensembled model before taking out a couple of the correlated redundant features. These features happened to be quite descriptive, so, by letting them in, we were imposing a “positive bias” in the model, as the information encoded in those features was taken into account twice.

2.1.2. Random Forest

Due to the flexibility of Random Forest in terms of feature input, we could easily assess how well it behaves with different feature set approaches. We discovered that we could use a reduced set of original features and still get a 99.7% of accuracy on our dataset. After trying a wider range of features and applying hyperparametrization on the number of trees and the number of maximum features per tree split, we managed to raise that number to 99.8% with the configuration of 2000 trees and 15 maximum features per tree split.

2.1.3. Deep Neural Networks (DNN)

The DNN based model, as expected by the properties of the algorithm, was the one that entailed the largest amount of configuration of the three. Feature wise, we scaled the continuous variables, as DNN is quite sensitive to their values. For the initial architecture of the network, we started from the final disposition of layers and neurons described in Gabriel Fernandez’s project [4]. Even so, after performing hyperparametrization, our architecture changed, ending with two hidden layers of 160 neurons each. The score was archived after all of the configurations were 99.6%.

2.2. Distribution

After all the algorithms were evaluated, we decided to prioritize the distribution of the one whose results were better, which is Random Forest. The next step consisted of reimplementing the model using PySpark and MLlib, which is the Spark’s machine learning library. Then, we defined the right transformations to deliver the input dataset in the right format for Random Forest to process in a pipeline, and finally we trained the model. After that, we developed a PySpark script to submit to Spark in which we defined aspects such as the input and output streams of data, and the query we wanted to make over the streaming data.

We assessed the efficiency of the distribution strategy with one and four workers, comparing the metrics “inputRate” and “processingRate”. We found out that the performances were very similar. In fact, the rates became slightly worse when using four workers. We think that this phenomenon is caused by the overhead introduced by Spark, as we use it both for the ingestion and processing of data.

3. Conclusions and Discussion

From a general point of view, we define the outcomes obtained as positive. They certainly align with both our hypothesis and other authors' ideas [1,3] about how machine learning in the network security domain can make a difference.

About the models, and concretely respecting Naïve Bayes, its simplicity and strong assumptions about feature independency [5] probably take a part in why it falls behind in terms of capability of prediction with respect to our other choices. On the other side, Random Forest overtook every other alternative in almost every aspect. With a default configuration and using only a few of the original features, it was already the best scoring algorithm. Concerning DNN, its outcomes came close to the ones of Random Forest. In addition, this is the algorithm that benefits the most from hyperparametrization.

In reference to the distribution outcomes, the results display an issue that needs to be addressed to successfully take advantage of one of the benefits that we have been claimed machine learning can offer, handling the processing of extensive volumes of data.

In future lines of research, we would perform similar experiments to other datasets as the results could be influenced to some extent by the peculiarities of the data. As for the models, we probably could somewhat improve the results for DNN if we explored more sophisticated feature engineering techniques, such as Feature Embedding. Finally, in view of the outcomes of the distribution phase, we would delegate the data ingestion to a separated solution, such as Kafka.

4. Materials and Methods

The dataset used for this project was the UNB ISCX IDS 2012, and the main tools that allowed us to work through all the phases of the machine learning process but the deployment were: Scikit-Learn v0.23.1, Tensorflow v2.1.0, Pandas v1.0.5, and Numpy v1.18.5. These supplied us with all the core functionality we needed for handling our dataset, modeling, and evaluating. However, we also made use of some complementary libraries. Concretely, for visualization, we employed Matplotlib v3.2.2 and Seaborn v0.10.1, and for the transformation of IP related features, Netaddr v0.8.0.

Our distributing architecture was based on Spark v3.0.0 and Hadoop v2.7.0. For developing applications on this architecture, we employed PySpark and MLlib v3.0.0. Finally, we delegated the monitoring and stream metrics visualization to Zabbix v5.0.0 and JMX.

Author Contributions: Conceptualization, D.F. and F.J.N.; methodology, J.J.E.-P., D.F., and F.J.N.; software, J.J.E.-P., D.F., and F.J.N.; validation, D.F. and F.J.N.; formal analysis, J.J.E.-P.; research, J.J.E.-P.; resources, J.J.E.-P.; data curation, J.J.E.-P.; writing—original draft preparation, J.J.E.-P.; writing—review and editing, J.J.E.-P., D.F. and F.J.N.; visualization, J.J.E.-P.; supervision, D.F. and F.J.N. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ahmed, T.; Oreshkin, B.; Coates, M. Machine Learning Approaches to Network Anomaly Detection. In Proceedings of the Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07), Cambridge, MA, USA, 10 April 2007.
2. Sommer, R.; Paxson, V. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In Proceedings of the 2010 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 16–19 May 2010; Volume 1.
3. Amrollahi, M.; Hadayeghparast, S.; Karimipour, H.; Derakhshan, F.; Srivastava, G. Enhancing Network Security Via Machine Learning: Opportunities and Challenges. In *Handbook Of Big Data Privacy*; Springer International Publishing: Cham, Switzerland, 2020; p. 169.

4. Fernandez, G. Deep Learning Approaches for Network Intrusion Detection. Ph.D. Thesis, The University of Texas at San Antonio, San Antonio, TX, USA, 2019; pp. 19, 20, 50, 51.
5. Panda, M.; Ranjan Patra, M. Network Intrusion Detection Using Naïve Bayes. *IJCSNS Int. J. Comput. Sci. Netw. Secur.* **2007**, *7*, 258–263.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).