Comparing the effectiveness of popular algorithms for detecting network intrusions

Matthew Nettleton 16633508

School of Computer Science

University of Lincoln

2020

Contents:

Abstract

Detecting network intrusions can be a difficult process involving monitoring and analysing network activity to detect security threats or concerns. There are many different methods to use when it comes to detecting network intrusions. However, three of the most popular machine learning methods used for detecting network intrusions include Decision Tree classifiers, Naïve Bayes classifiers and Random Forest classifiers. This paper seeks to compare the effectiveness of these three algorithms when it comes to detecting anomalous network intrusions and discovering which of theses popular algorithms is best or worst suited to the task.

Introduction

Software application security, such as mobile services or web-based applications, has always been at risk because of how open and expansive the internet is. As technology becomes more advanced and the world becomes more digitalised, the amount of network traffic increases, thus raising the risk of cyber-attacks. Because security threats are increasing, network intrusion detection systems have become more and more important over the years. A network intrusion detection system monitors is a system designed to monitor networks, collecting data on the network traffic, such as harmful or suspicious activities, or just environmental and contextual trends in the network traffic. Network intrusion detection systems are classified into three types: misuse detection systems, anomaly detection systems and specification-based detection systems. However, these will be discussed more thoroughly in the Literature Review segment. A network intrusion is a set of actions that attempt to compromise the proprietary business plans or loss of critical business data or disruption of services or system resources [14]. There are many different methods of detecting network intrusions. In this report, however, we will look at three of the most common machine learning algorithms that are used to detect network intrusions: Decision Tree, Naïve Bayes and Random Forest classifiers, to compare their effectiveness at detecting anomalous network intrusions within the NSL-KDD dataset. The NSL-KDD dataset is a public dataset of network data, designed to train machine learning models to detect network intrusions, however, we will take a more in-depth look at the NSL-KDD dataset within the Design, Development and Evaluation section.

Literature Review

Detecting network intrusions can be a difficult task. There are many ways to detect such intrusions, and there are many was to implement said detection methods. This problem is discussed in Sekar, Gupta, Frullo, Shanbhag, Tiwari, Yang and Zhou's research study (2002) [1], who go on to state that intrusion detection methods can be divided into three categories: misuse detection, anomaly detection and specification-based detection. This paper also explains that while misuse detection systems can detect previously known attack signatures accurately, misuse detection systems are ineffective against new and unseen attack signatures. This is

backed up by Sabhnani and Serpen in their 2003 paper [2], which argues that "it is not possible for any trainable pattern classification or machine learning algorithm to demonstrate an acceptable level of misuse detection performance on the KDD testing data subset" (Sabhnani and Serpen, 2003). While this paper makes a strong case for not using misuse detection for the NSL-KDD dataset, it does briefly explain that the dataset would be better suited for anomaly detection.

Anomaly detection systems work by creating models of normalised data and then detecting deviations from the normal model. This is a similar method to what is used in specification-based detection systems. Specification-based detection systems also detect intrusions as deviations from the normal model. The main differences are highlighted in García-Teodoro, Díaz-Verdejo, Maciá-Fernández and Vázquez's 2008 study into anomaly-based network intrusion detection [3]. They explain that specification-based detection systems provide good detection rates for manually specified parameters. This allows the developer to program the system to specifically ignore legitimate behaviour that could still be considered anomalous. On the other hand, anomaly detection systems do not allow the developer any control over what anomalous behaviours are considered legitimate and what are considered attacks. This topic is covered in Sekar, Gupta, Frullo, Shanbhag, Tiwari, Yang and Zhou's research study (2002) [1]. This paper presents the idea that while specification-based detection systems are less effective at detecting new and unseen attacks compared to anomaly detection systems, they produce fewer false positives/alarms. However, this reduction of false positives comes at a cost of development time as detailed specification can take a long time to implement. In addition, specification-based detection systems can result in increased false negatives in comparison to anomaly detection systems.

As Lazarevic, Ertoz, Kumar, Ozgur and Srivastava state in their comparative study into anomaly detection schemes [4], most approaches for detecting anomalous network intrusions build a model over the normalised data and then compare how well new data fits the model. Some of these approaches to detecting anomalies include some of the more straightforward machine learning algorithms, such as using decision trees [5] or Naïve Bayes classifiers [6]. However, other authors have used more complicated intrusion detection methods, such as using neural networks as a method of obtaining the detection model [7]. In [8], it is discussed that intrusion detection systems can be made to perform better and improve accuracy by obtaining good training parameters and choosing the right features to design an Artificial Neural Network. A downside to this method, however, is that the Artificial Neural Network requires a large amount of data to train the model. In a 2010 study comparing network intrusion detection systems [9], the authors state that the key to creating a successful and reliable system to detect network intrusions is about knowing "how to choose an effective classification approach to build accurate intrusion detection systems in terms of high detection rate while keeping a low false alarm rate" (Panda, Abraham, Patra, 2010). This paper goes on to suggest that the optimal algorithm to detect network intrusions is a Discriminative Multinomial Naïve Bayes with an N2B classifier. This paper also criticises the Naïve Bayes algorithm, claiming that "the attribute conditional assumption of Naïve Bayes rarely holds in real world applications. So, it needs to relax the assumption effectively to improve its classification performance" (Panda, Abraham, Patra, 2010), hence the addition of the N2B classifier. However, in their more recent 2012 study into using hybrid intelligent systems to detect network intrusions [10], the idea is put forward of combining Naïve Bayes with a decision tree. By using this hybrid method, they have achieved a greater accuracy than any of the previous methods for their data set.

Methodology

Project Management –

There are many different software development methodologies that have been developed and each one is best suited for different projects. Although there are many software development methodologies, there are also fundamental systems development life cycle activities which are shared between all software development methodologies [11]. These systems development life cycle activities are [12]:

1. Planning: This is the process answers why the software should be built and how the developers will build it.
2. Analysis: This stage answers what the software will do, whilst also answering who will use the software and how will they use it.
3. Design: This stage in the process answers how the software will function while also answering what programs and files will be required for the software.
4. Implementation: This final stage involves building the software. This process includes system construction, testing, and also support and improvement once the software is finished.

Because these fundamental life cycle activities are almost exactly identical to the steps outlined in the waterfall methodology for software development, waterfall was chosen as the project management methodology. This methodology was used as the basis for the original Gantt Chart, shown in figure 1, that was used to track the progress of the project and estimate how much time should be allotted for each segment of the development process.

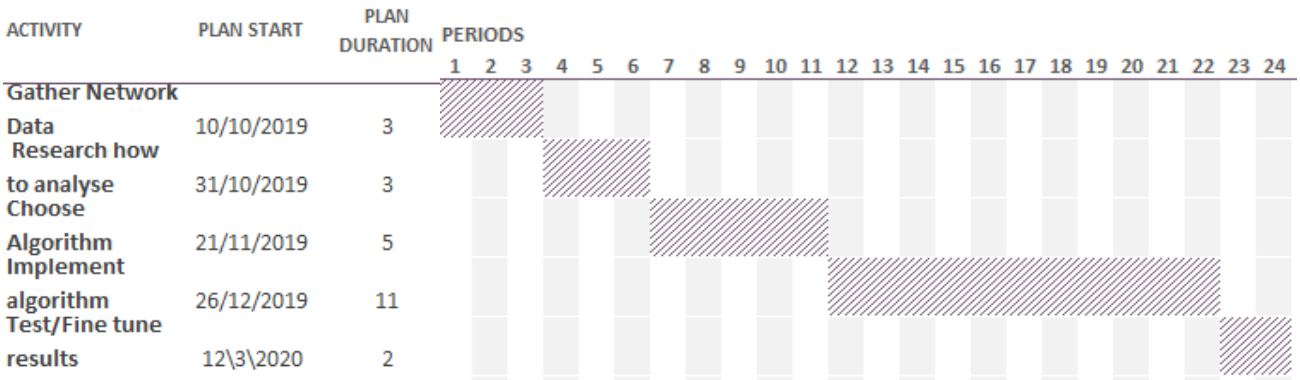

Figure 1: Original project plan

This chart, however, would be revised and updated to match the updated requirements of the development process, as will be seen in the following segment.

Software Development –

The development of the software was done in concordance with the stages that were outlined by the waterfall methodology. However, it soon became clear that the original Gantt chart shown in figure 1 did not keep up with the development lifecycle. Once it was decided that the software would compare the effectiveness of Decision Trees, Naïve Bayes and Random Forest

classifiers, the Gantt chart was updated to that shown in figure 2 to match the new requirements of the project.

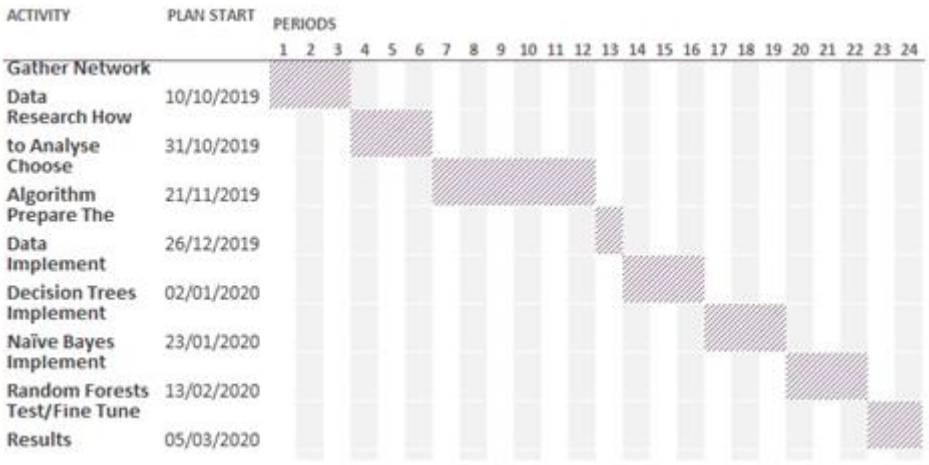

Figure 2: Revised project plan

After the software development life cycle had reached the analysis stage, it was decided that Decision Trees, Naïve Bayes, and Random Forests would be implemented for the software to use to compare the results against each other. The changes to the project plan in figure 2 were necessary to ensure that both the design and implementation stages of the development life cycle could continue smoothly without the need for additional changes to the proposed plan, as doing so would disrupt the software development life cycle and waste unnecessary time, as unforeseen requirements can take a long time to accommodate when using the waterfall methodology. The revision to the project plan shown in figure 2 was also factored into the design stage of the software development life cycle, as the project plan outlines how the software will function and what must be completed before moving onto the next step during the implementation stage of the development life cycle, such as preparing and processing the data from the NSL-KDD dataset before moving forward with implementing the algorithms that would be used to analyse the data.

Toolsets and Machine Environments –

The main tool used for the software development process was Python. Python is an ideal programming language for machine learning due to the large number of libraries available to process or analyse data, such as NumPy [13], Pandas and SciKit-Learn. Although other languages and libraries exist that are also well suited for machine learning, such as the R language [16] or the Shogun Library for C++ [17], Python was chosen in the end as it is a good, all-round language that is well suited for simple programs [18] as well as advanced and sophisticated applications.

Design, Development and Evaluation

Dataset Used –

The NSL-KDD dataset is an updated and revised version of the original KDDCUP'99 dataset. The KDDCUP'99 dataset is the most commonly used dataset for anomaly detection [19], however there were three big issues with the KDDCUP'99 dataset. Because of these issues, the NSL-KDD dataset was created. The NSL-KDD dataset was comprised of only selected values from the complete KDDCUP'99 dataset, solving the three issues that were present within the KDDCUP'99 dataset [20].

The first of these issues was that the KDDCUP'99 dataset had many redundant records that overlapped between the training and test datasets, causing machine learning classifiers to become biased towards the repeating values in the dataset. The NSL-KDD dataset solved this issue by not including any of the redundant records that previously existed in the KDDCUP'99 dataset, causing the machine learning classifiers to not become biased the more frequent records in the dataset.

The second issue the NSL-KDD dataset solved was that the number of values selected from each difficulty group within the dataset is inversely proportional to the percentage of values within the original KDDCUP'99 dataset. This causes the classification rates of different machine learning algorithms to have a greater variance, making it more efficient to have an accurate evaluation of different machine learning methods [19].

Finally, the third issue the NSL-KDD dataset fixed was reducing the number of values in both the training and test dataset to a manageable amount, removing the need to select a portion of the KDDCUP'99 dataset at random. This means that the analysis of different researchers will produce consistent and comparable results.

The NSL-KDD dataset is made up of 41 features and 5 classes. Of these 5 classes, one of these classes is normal and the other 4 are different types of attacks: R2L, U2R, DoS and Probe. The features in the NSL-KDD dataset can be seen in figure 3.

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

Figure 3: Features in the NDL-KDD dataset

Romote to Local attacks (R2L) are attacks where a user sends packets to a machine that they do not have access to over the internet, exposing vulnerabilities and exploiting privileges. User to Root attacks (U2R) occur when an attacker starts with normal user access to a system and are then able to exploit vulnerabilities to gain root access to the system. Denial of Service (DoS) attacks occur when an attacker overloads a computing or memory resource to prevent it from giving legitimate users access. Probing Attacks are when attackers attempt to learn information about a network to avoid its security protocols [19].

Algorithms Used –

Decision Tree –

Decision trees are simple classification models that are often used for the classification of data within data mining. The objective of a decision tree can be split into 4 tasks [22]:

1- Correctly classify as much of the training data as possible.
2- Look beyond the training data so that unknown data can be classified as accurately as possible.
3- Ensure that the decision tree is easy to update.
4- Be as simple as possible.

Decision trees are made up of nodes and branches, where each non-terminal node is a decision point on the data set. Depending on the outcome of the decision at the node, a specific branch is chosen to follow. This process starts at the root node, where a branch is selected to move to the next node. This process repeats itself for each layer in the decision tree until a terminal node, also known as a leaf, is reached. Once this point has been reached, a decision is made, and the process begins again on the next value in the dataset. One of the advantages of implementing a decision tree for detecting network intrusions is that the person developing the decision tree that will be used does not need any knowledge about network or network intrusions to implement an accurate decision tree. Decision trees are well suited for dealing with and categorising large datasets. This makes decision trees ideal for detecting network intrusions as the datasets that are used to train these models, such as the NSL-KDD dataset, are usually very large. One of the disadvantages to using decision trees to detect network intrusions, however, is that the output of the decision tree must be categorical data, such as normal or anomaly.

Naïve Bayes –

Bayesian models are machine learning models that establish probabilistic relationships between class labels and values of interest, represented by feature vectors, where the feature vector can be written as $X = (x_1, \dots, x_n)$ Naïve Bayes classifiers are probabilistic algorithms that use the Bayes' Theorem to classify new data instances. These classifiers can be be easily simplified by assuming that vectors are independent given class [23], therefore $P(X|C) = \prod^{n}_{i=1} P(X|C)$, where X is a feature vector and C is a class. Because these models simplify the computation when dealing with data, Naïve Bayes models show a high speed and accuracy when it comes to categorising large datasets, such as the NSL-KDD dataset. One of the biggest disadvantages for the Naïve Bayes model, however, is that it is based on a conditional independence assumption [24]. This assumption does not always translate well to real life situations and applications, and this assumption can cause the Naïve Bayes model to lose accuracy if the assumption does not match with the data being analysed. Despite this issue, Bayesian models can still give fairly accurate results as the model focuses on the classes, not on the probabilities.

Random Forests –

A random forest classifier is a machine learning model that is made up of many decision trees. Instead of averaging the prediction of the decision trees, the random forest classifier instead takes random values in the dataset when it builds the decision trees. By training each decision tree on different values at random, the forest will have a lower variance without an increased

bias. In addition, only a small number of features in the data are considered when it comes to splitting each node in each decision tree in the random forest. The random forest classifier creates hundreds or sometimes thousands of decision trees, where each decision tree is trained on randomly selected data values and each node in each decision tree is split considering a limited number of features. The final prediction of the random forest classifier is calculated by averaging the predictions of every single decision tree within the random forest classifier. This method should produce more accurate results than a single decision tree. A downside to using a random forest classifier is that because lots of decision trees are created, this model takes more computational power and time to run.

Building and Coding –

For the coding of the software, after the Scikit-Learn, NumPy and Pandas libraries were imported, the first stage is the pre-processing of the NSL-KDD dataset. During this stage, the dataset is first loaded into a Pandas data frame. However, because the 'protocol_type', 'service' and 'flag' columns are stings, they must be converted into integers before the data is inputted into any of the machine learning models otherwise they will not function. This was done by creating dictionary files for the three columns and correlating each unique string variable in the columns into integers (*Advances in Information and Communication: Proceedings of the 2020 Future of Information and Communication Conference (FICC), Volume 2.* (2020) Springer International Publishing, 726) [25]. Once the strings have been replaced with their corresponding integer values, the dataset is then split into two datasets. The first dataset is created from the first 40 columns from the original dataset that is now only numeric values, while the second dataset contains just the final 'class' column, which contains only two strings: 'normal' and 'anomaly'. This dataset will be used at the end once the machine learning models have categorised the data to compare the results and calculate the accuracy. After the 'class' column has been separated from the rest of the dataset, the numeric dataset is then split into 80% training data and 20% test data. This stage marks the end of the data pre-processing.
The next stage in the software is the creation of the machine learning models that will be used to analyse the data. The decision tree classifier is created first, where the maximum layers in the decision tree is set to 10. Afterwards, the Naïve Bayes classifier is created whilst making sure that it is created with no prior assumptions, reducing bias in the model. Finally, the random forest classifier is created with the number of decision trees it will create set to 500, whilst the maximum number of features it will consider for splitting each node is set to the square root of how many features exist within the dataset. This is the standard when fitting this model for the categorisation of data. Once the three prediction models have been created, the models are then trained using the training dataset before they are then given the testing dataset to make their predictions as to what entries in the dataset are anomalous results and what are normal.

Accuracy –

To measure the accuracy of the predictions outputted by the machine learning models, a confusion matrix was used. A confusion matrix shows the number of correct and incorrect predictions made by the models. The confusion matrix shows the ways in which the models are making mistakes in their predictions and the types of errors they are making. Figure 4 shows how the confusion matrix displays the results.

	Predicted Normal	Predicted Anomaly
Actual Normal	True Positive (TP)	False Negative (FN)
Actual Anomaly	False Positive (FP)	True Negative (TN)

Figure 4: Confusion matrix result interpretations

True positive results are results that were correctly predicted as normal.
True negative results are results that were correctly predicted as anomalies.
False positive results are normal results that were incorrectly predicted as anomalies.
False negative results are anomalous results that were incorrectly identified as normal.

Figure 5 shows the confusion matrix for the decision tree, Naïve Bayes and random forest classifiers.

```
array([[11742,    30],
       [   37, 13386]], dtype=int64)

array([[  218, 11554],
       [  212, 13211]], dtype=int64)

array([[11751,    21],
       [    3, 13420]], dtype=int64)
```

Figure 5: Confusion matrices for the decision tree, Naïve Bayes and random forest classifiers

These results show that the decision tree classifier correctly predicted 11742 true positives and 13386 true negatives, with only 37 false positives and 30 false negatives. The Naïve Bayes classifier correctly predicted only 218 true positives and 13211 true negatives, with 212 false positives and 11554 false negatives. Finally, the random forest classifier was the most accurate with 11751 true positives identified and 13420 true negatives identified, with only 21 false negatives and 3 false positives. To calculate to accuracy of the of each machine learning model with the confusion matrix, the following algorithm must be used [26]:

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

By inputting the confusion matrix values from the decision tree, Naïve Bayes and random forest classifiers into this algorithm, the accuracy for each model can be calculated. Figure 6 shows the accuracy values for the decision tree, Naïve Bayes and random forest classifiers.

```
(99.7340742210756, 53.30025798769598, 99.9047430045644)
```

Figure 6: % accuracy for the decision tree, Naïve Bayes and random forest classifiers

Conclusion

Plotting these accuracy values onto a graph gives us the following, shown in figure 7.

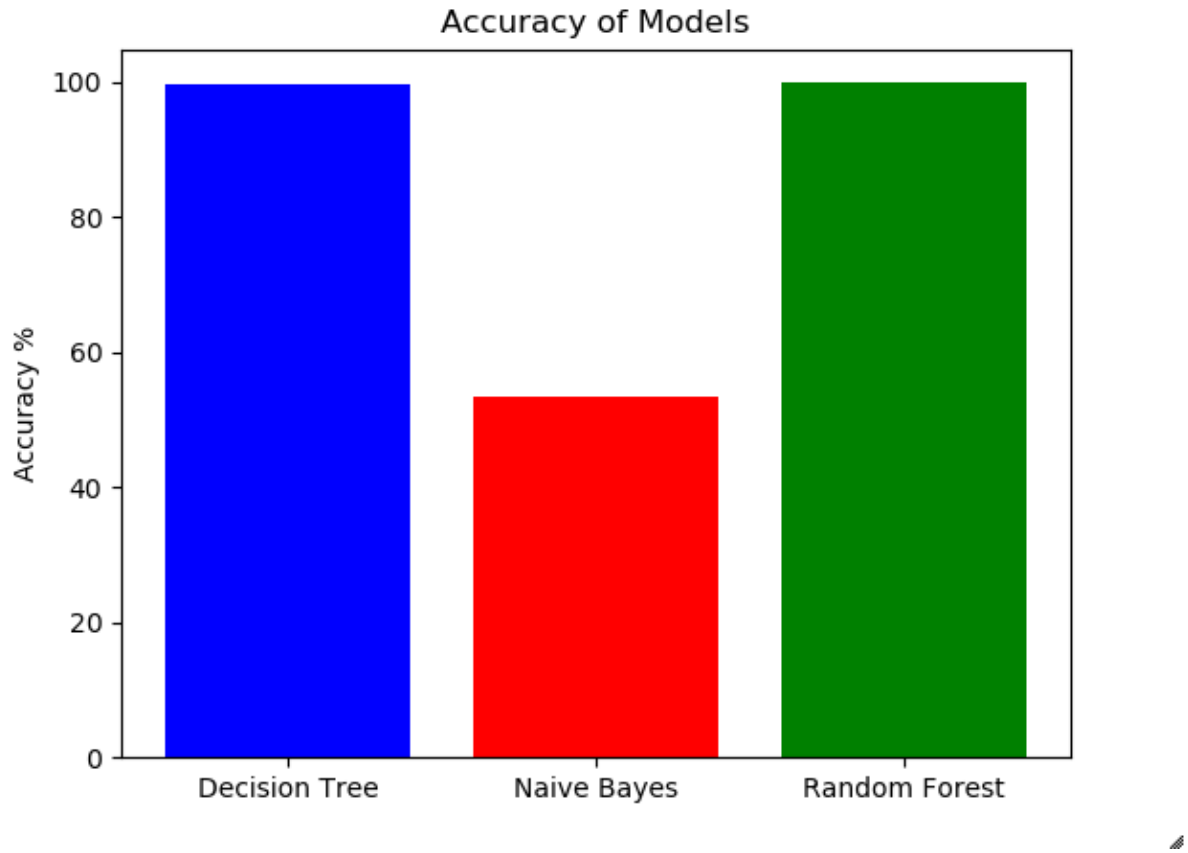

Figure 7: Accuracy of the models

These results show that when it comes to analysing the anomalous data from network intrusions, or at least analysing the network intrusion data from the NSL-KDD dataset, the decision tree and random forest models display an almost perfect accuracy, with the random forest classifier with the most accurate result of a 99.9% intrusion detection accuracy and the decision tree classifier with a 99.7% intrusion detection accuracy. The most likely reason as to why the random forest is slightly more accurate than the decision tree is most likely due to the fact that the random forest model was made up of 500 decision trees with splits at random nodes, decreasing the variance without increasing bias and therefore giving slightly more accurate results. However, the random forest model takes far longer to compute than the other two models and it requires far more computation. For very large datasets, this model may not be ideal due to the amount of time it takes to process the data in comparison to the decision tree model, in return for only a 0.2% increase in accuracy. The Naïve Bayes model, on the other hand, performed very poorly for detecting anomalous results, with only 53.3% accuracy. This poor result may be due to the fact the assumptions that the Naïve Bayes algorithm is based on does not translate well to real life scenarios, as was mentioned previously in the Design, Development and Evaluation section.

Overall, judging by the results shown with the analysis of the NSL-KDD dataset, the random forest classifier is the most accurate and therefore most suitable, followed closely by the decision tree classifier. The Naïve Bayes classifier proved to be the least suitable algorithm for detecting anomalous network intrusions within the NSL-KDD dataset.

Reflection

If I were to repeat this project again, I would have liked to implement the Naïve Bayes algorithm in combination with another algorithm, such the combination of Naïve Bayes with a decision tree mention in [10]. Doing so would have made it a fairer comparison as the Naïve Bayes algorithm on its own can struggle to product reliable results if the assumptions do not match with the dataset, as was likely the case in this comparison. Had I been more accurate with the timeframe, especially given the extra time available due to the COVID-19 outbreak, I would have also liked to have implemented a fourth method of detecting anomalous results, perhaps by using a neural network or a k-means clustering approach outlined in [15], as the artefact is not as advanced as I would like. That being said, I believe I worked well throughout this project and followed the project plan fairly closely.

References

[1] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, S. Zhou. (2002) Specification-based anomaly detection: a new approach for detecting network intrusions. In: *Proceedings of the 9th ACM conference on Computer and communication security.* Washington, DC, USA: ACM, 265-274. Available from https://dl.acm.org/doi/pdf/10.1145/586110.586146

[2] Sabhnani. M, Serpen. G. *Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set*. University of Toledo. Available from https://pdfs.semanticscholar.org/24ef/d7b974b5b8fd79ede075de13bc98b2182dc2.pdf [accessed 15th April 2020]

[3] García-Teodoro. P, Díaz-Verdejo. J, Maciá-Fernández. G, Vázquez. E (2009) Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers and Security*, 28(7-8) 335-354. Available from http://dtstc.ugr.es/~jedv/descargas/2009_CoSe09-Anomaly-based-network-intrusion-detection-Techniques,-systems-and-challenges.pdf [accessed 15th April 2020]

[4] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, J. Srivastava. (2003) *A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection.* SDM. Available from https://pdfs.semanticscholar.org/c2b8/974e83e18ee816d7610b919fb7851146ea25.pdf?_ga=2.241379146.934170256.1575536532-735326547.1575536532 [accessed 15th April 2020]

[5] Lee. W, Stolfo. S. J (1998) Data Mining Approaches for Intrusion Detection, In: *Proceedings of the 1998 USENIX Security Symposium*, San Antonio, Texas, 26-29 January. n.p: USENIX Association. Available from https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/lee/lee.pdf [accessed 15th April 2020]

[6] Sharma. N, Mukherjee. S (2012) Layered approach for intrusion detection using naïve Bayes classifier. In*: ICACCI '12: Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, Chennai, India, August. New York, USA: ACM, 639-644. Available from https://dl.acm.org/doi/pdf/10.1145/2345396.2345500 [Accessed 15th April 2020]

[7] Ghosh. A, Schwartzbard. A, (1999) A Study in Using Neural Networks for Anomaly and Misuse Detection. In: *Proceedings of the 8th USENIX Security Symposium*, Washington, DC, USA, 23-26 August. Berkeley, CA, USA, USENIX Association. Available from https://www.usenix.org/legacy/events/sec99/full_papers/ghosh/ghosh_html/ [Accessed 15th April 2020]

[8] Abdulla. S, Al-Dabagh. N, Zakaira. O (2010) *Identify Features and Parameters to Devise an Accurate Intrusion Detection System Using Artificial Neural Network*.: WASET. Available from https://www.academia.edu/2612588/Identify_Features_and_Parameters_to_Devise_an_Accurate_Intrusion_Detection_System_Using_Artificial_Neural_Network [Accessed 15th April 2020]

[9] M. Panda, A, Abraham, M. Patra. (2010) *Discriminative multinomial Naïve Bayes for network intrusion detection.* Atlanta, GA, USA: IEEE. Available from http://isda05.softcomputing.net/ias10_panda.pdf [accessed 15th April 2020]

[10] M. Panda, A, Abraham, M. Patra. (2012) *Hybrid intelligent systems for detecting network intrusions.* John Wiley & Sons, Ltd. Available from https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.592 [accessed 15th April 2020]

- [11] Ayman Al Ahmar. M (2010) Rule Based Expert System for Selecting Software Development Methodology. *Journal of Theoretical and Applied Information Technology*, 19(2) 143-148. Available from http://jatit.org/volumes/research-papers/Vol19No2/10Vol19No2.pdf [Accessed 15th April 2020]

[12] Dennis. A, Wixom. B. H, Roth. R (2018) *Systems Analysis and Design,* 4th Edition, John Wiley & Sons, Inc.

[13] Van Der Walt. S, Colbert. SC, Varoquaux. G (2011) The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2) 22-30. Available from https://arxiv.org/pdf/1102.1523.pdf%C3%AB%C2%A5%C2%BC [Accessed 15 April 2020]

[14] Adebayo. A, Shi. Z, Shi. Z, Olumide. A (2006) Network Anomalous Intrusion Detection Using Fuzzy-Bayes. *Intelligent Information Processing*, 228(3) 525-530. Available from https://link.springer.com/content/pdf/10.1007/978-0-387-44641-7_56.pdf [Accessed 1th April 2020]

[15] Muniyandi. A, Rajeswari. R, Rajeram. R (2012) Network Anomaly Detection by Cascading K-Means Clustering and C4.5 Decision Tree algorithm. In: *International Conference on Communication Technology and System Design 2011*, Coimbatore, India, 9-11 December. Science Direct, 174-182. Available from https://pdf.sciencedirectassets.com/278653/1-s2.0-S1877705812X00092/1-s2.0-S1877705812008594/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjECgaCXVzLWVhc3QtMSJHMEUCIQDz4OB0hJvek4sezvdOtaCHCmNQNTHK1BkpTGpJVhs7AwIgd4B16JtGZ%2FccEjBeUeLQyL4DwR5sTVxdaVCz7uW5tEAqtAMIURADGgwwNTkwMDM1NDY4NjUiDGRXCk7%2Fw88XSfC81iqRA6tEAPjN7KwTEPbsdwhLRI7JuOqaYMXODEmoJmDMk8dtZQ62tJcT2AhpdpDMlBwqc%2FDxrqFRVnW704duiKn%2F%2FNyKU25wv3tVUfFIwMfWi%2FhgpM1JZZz7gc3AwSYOP8YgwxXnnWE%2FDUabEfZJZ56hPTJL4Em9gdRoP2z25s9JB80MI9rlGgGPRuV7Im7vpA%2FYHSjFkKnDcrLuz5jY%2FddHSKLn6D542%2Baa7sOuxdON5lQRO1F07b9eQ91bIORzIoeR3EPAQhnkiyTqpUZjaMk7QPuyqBv7yxpjsXlE0Wgs41d7fZehdqiqOcWMBTUPsOSIW3cB%2BI%2Fzby1%2F60VIyxC11YR64t7nGen7pc2GtfgwH6bvNGKN4IcLwGxljmimDyZO%2BY7bq4LUE9fvqR8lRBNlOWA%2Ff6y2ARFva%2Bf5k0e2A1AD5VKHzGBeV2mnz%2By2sZlkKQHgPTUzLPMgnlpUIFhK4PwId3K5bdNevmDpna9mlt2m14l2RsUYet6CjbUO3ZWK7X0fmAdJyef%2BKV4doHjDL5pAMLWmg%2FUFOusB%2Bmtp2EtHt%2BRP9jH2Voq0D%2BOzOfceta510VphV1uaRSESU0AmXjtF1rX9oY9v9FS43xr9p7Hjb6exiA90%2Bvg8yA3GtzzDVfJi0NNAJH05U%2B%2BcrwlbweY0%2Bb3yO0fwfbYHfHpRS8y5tcyX88ehG0oy%2FhWKdSzZt9oL045dhsQvitEVpxAHuvqxB6x7K3NKg4QURbB5xMjcFrtFKgL%2BKZTsNDYFMB1tfBxHIUeJ9Hp5rbglJLRSM4WK%2FuOc12lovSUgA%2FfsZQxWFA8pWC0Z%2FotYDZ2xD6scbimy331MpAEhJL5FP46ziRo4TCThhg%3D%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20200422T234944Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTYZVKVKQPM%2F20200422%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=db35ce883729d99d5ec2943b61b7dc937fc6f608eb52ff55c03c23bbbbc549d8&hash=4e2c0b78d5a06b75f33380367c4b2e5329af26796469411f4d7d273b3108592c&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S1877705812008594&tid=spdf-3709ecb4-0706-410d-b01d-

33f5cbe15269&sid=d3cf3cf3121761404a8b8a68266ed2b76627gxrqb&type=client [Accessed 15th April 2020]

[16] Bunn. A, Korpela. M (2019) *An Introduction to dplR*. n.p: R Foundation. Available from https://cran.r-project.org/web/packages/dplR/vignettes/intro-dplR.pdf [Accessed 15th April 2020]

[17] Sonnenburg. S, Rätsch. G, Henschel. S, Widmer. C, Behr. J, Zien. A, Bona. F, Binder. A, Gehl. C, Franc. V (2010) The SHOGUN Machine Learning Toolbox. *Journal of Machine Learning Research,* 11 1799-1802. Available from http://www.jmlr.org/papers/volume11/sonnenburg10a/sonnenburg10a.pdf [accessed 15 April 2020]

[18] Perez. F, Granger. B. (2007) IPython: a system for interactive scientific computing. *Computing in Science & Engineering*, 9(3) 21-29.

[19] Chae. H, Jo. B, Choi. S, Park. T (2013) Feature Selection for Intrusion Detection using NSL-KDD. In: *6th World Congress, Applied Computing Conference 2013*, Nanjing, China, 17-19 November. Available from http://www.wseas.us/e-library/conferences/2013/Nanjing/ACCIS/ACCIS-30.pdf [Accessed 15th April 2020]

[20] Tavallaee. M, Bagheri. E, Lu. W, Ghorbani. A (2009) A detailed analysis of the KDD CUP 99 data set. In: *2009 IEEE Symposium on Computational Intelligence for Security and Defence Applications*, Ottawa, USA, 8-10 July. n.p: IEEE. Available from https://www.researchgate.net/profile/Wei_Lu23/publication/48446353_A_detailed_analysis_of_the_KDD_CUP_99_data_set/links/09e4151490ae305c07000000.pdf [Accessed 15th April 2020]

[21] Visa. S, Ramsay. B, Ralescu. A, Knaap. E (2011) Confusion Matrix-based Feature Selection. In: *Proceedings of the Twenty-second Midwest Artificial Intelligence and Cognitive Science Conference*, Cincinnati, USA, 16-17 April. 120-127. Available from https://www.researchgate.net/profile/Atsushi_Inoue/publication/220833227_Page_Ranking_Refinement_Using_Fuzzy_Sets_and_Logic/links/54b743480cf24eb34f6e9e80.pdf#page=126 [Accessed 15th April 2020]

[22] Safavian. S, Landgrebe. D (1990) *A Survey of Decision Tree Classifier Methodology*. Indiana: NASA. Available from https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19910014493.pdf [Accessed 15th April 2020]

[23] Rish. I (2001) An empirical study of the naïve Bayes classifier. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Seattle, USA, 4 August. New York: IBM, 41-46. Available from https://www.cc.gatech.edu/~isbell/reading/papers/Rish.pdf [Accessed 15th April 2020]

[24] Zhang. H (2004) *The Optimality of Naïve Bayes*. University of New Brunswick. Available from https://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf [Accessed 15th April 2020]

[25] *Advances in Information and Communication: Proceedings of the 2020 Future of Information and Communication Conference (FICC), Volume 2.* (2020) Springer International Publishing