Improving Model Performance with Robust PCA

by

Marissa Bennett

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2020

APPROVED:

Professor Randy Paffenroth, Thesis Advisor

Professor Robert Walls, Thesis Reader

Professor Craig Wills, Head of Department

Abstract

As machine learning becomes an increasingly relevant field being incorporated into everyday life, so does the need for consistently high performing models. With these high expectations, along with potentially restrictive data sets, it is crucial to be able to use techniques for machine learning that increase the likelihood of success. Robust Principal Component Analysis (RPCA) not only extracts anomalous data, but also finds correlations among the given features in a data set, in which these correlations can themselves be used as features. By taking a novel approach to utilizing the output from RPCA, we address how our method effects the performance of such models. We take into account the efficiency of our approach, and use projectors to enable our method to have a 99.79% faster run time. We apply our method primarily to cyber security data sets, though we also investigate the effects on data sets from other fields (e.g. medical).

Acknowledgements

I would like to express my gratitude to my advisor Prof. Randy Paffenroth who made sure I was always “write, write, write”-ing, for checking over my mathematical equations, and for his guidance, direction, and support during the implementation of this work. My thanks are also due to my reader Prof. Robert Walls for reviewing my written work. I would also like to acknowledge the Academic & Research Computing group at Worcester Polytechnic Institute for providing computing support that contributed to the results outlined within this report. Thanks also to my lab-mates in the ALAS/Fossil/CAKE Lab (a.k.a. the Machine Learning Machine Feeling Lab) for their advice and space to work in the lab.

Contents

1 Introduction **1**

1.1 Related Work 2

1.2 Inspirational Example 3

2 Background **6**

2.1 PCAP Files 7

2.2 PCA 9

2.3 Robust PCA 11

2.4 Projectors 13

3 Methodology **16**

3.1 Data Sets 16

3.1.1 Data Set One 17

3.1.2 Data Set Two 18

3.1.3 Data Set Three 18

3.2 Approach 19

3.2.1 Main Program 20

3.2.2 Configurations 21

3.2.3 Pre-Processing 22

4 Results **26**

4.1 Main Data Set 27

4.2 LLDOS Data Set 29

4.3 GOSSIS Data Set 30

4.4 Method Efficiency 31

4.5 Summary 32

5 Conclusion & Future Work 35

A Data Set Specifics 36

A.1 UNB Features Table 36

A.2 LLS DDOS Features Table 38

A.3 GOSSIS Features Table 39

B Package Information 47

B.1 Configuration File 47

B.2 Machine Learning Models 50

B.2.1 Machine Learning Model Codes 50

B.2.2 Decision Tree 50

B.2.3 Gradient Boosting 50

B.2.4 K-Nearest Neighbors 51

B.2.5 Logistic Regression 51

B.2.6 Naive Bayes 51

B.2.7 Pytorch Neural Network 52

B.2.8 Random Forest 52

B.2.9 Support Vector Machine 52

C Supplemental Results 53

C.1 Confusion Matrices 53

List of Figures

1.1 Apartment Example . . . 3

1.2 Computer Network Example . . . 4

2.1 Wireshark Pane Layout . . . 7

2.2 Sigma Matrix Example . . . 10

2.3 RPCA Heat-Map . . . 13

3.1 RPyCA UML Package Diagram . . . 19

3.2 Matrix Divisions . . . 20

3.3 One-Hot Encoding Example . . . 23

3.4 Normalization Example . . . 24

3.5 Randomization Example . . . 25

4.1 Confusion Matrix Example . . . 27

4.2 SVM Confusion Matrices . . . 34

B.1 Decision Tree Example . . . 51

B.2 Neural Network Example . . . 52

C.1 KNN Confusion Matrices . . . 53

C.2 Decision Tree Confusion Matrices . . . 54

List of Tables

Notation Table 6

4.1 UNB Thursday Morning Matrix Splits 28

4.2 UNB Thursday Morning Results 28

4.3 LLDOS Matrix Splits 29

4.4 LLDOS Results 30

4.5 GOSSIS Results 31

4.6 Method Efficiency 32

A.1 UNB Feature List 37

A.2 LLS DDOS Feature List 38

A.3 GOSSIS Feature List 46

B.1 Default Configuration Parameters 47

B.2 ML Model Codes 50

Chapter 1

Introduction

Cyber-attacks occur every 39 seconds [Cuk18]. They are an on-going issue that effects millions of people across all industries [Fir20], and the cost to recover from attacks can be substantial. The amount of data and information that is required to be sifted through in order to identify these cyber-attacks is too much for manual processing to be effective. This is where Machine Learning (ML) can assist. There are many ML applications that involve trying to solve the problem of intrusion detection [YJ13, BG15, THLL09]. However, when ML is applied as a solution, there can be many models to choose from, although only a few of those models might perform well on a given problem, there are limited ways to improve on them. One obvious option is to use a better data set, or one with more data. However, a better or more complete data set may not exist. Another option is to use feature engineering[1], though this can often require substantial user expertise that may not always be available.

We propose a new way of applying techniques for enhancing data sets with our

[1]A process of manipulating data to improve accuracy in a model or algorithm.

Python software package, RPyCA[2]. This package combines several different types of feature engineering and provides novel capabilities for creating new features, thus expanding the data set. We use Robust PCA (RPCA) to accomplish this, since unlike Principle Component Analysis (PCA), we're not only able to extend and utilize the capabilities of dimensionality reduction, but also detect anomalous values in every feature. After the RPCA algorithm is run once on a data set, projectors can then be used to execute this same task, but more efficiently, in place of the RPCA algorithm. By running the original data set through RPCA and taking its output, we can create new sets of data to be used in ML models.

Further, we propose that by using RPCA on a preprocessed data set, even the simplest machine learning models will be able to show an increase in their performance compared to using the same data set without using RPCA, and, that if given minimal data, a decent model can still be achieved.

1.1 Related Work

Our research was inspired by the following cited papers [PKS18, CLMW11]. Our technique appears to be a novel contribution for the application we are targeting. There has been a publication on the reverse situation, improving RPCA with neural networks [SS02], and much more on various applications using PCA as a dimension reduction technique to assist neural networks [EY07, GDAD08, ZWN+09]. The closest paper to our proposed thesis focuses on using RPCA with pulse-coupled neural network to improve the accuracy in multi-focus image fusion [ZWN+09], however its purpose is far off from ours.

[2]Link to public repository: `https://github.com/Marissa4/RPyCA`

1.2 Inspirational Example

This example describes, in a more relatable sense, how computer networks operate. In Figure 1.1, you live in the apartment building on the left, and have created the best lasagna recipe ever. In order to share this recipe with your friends, you write down the details of how to make this amazing lasagna, and enclose it in an envelope addressed to your friend to send in the mail. You repeat this process to send all of your friends a letter with this recipe. The letters are sent via the USPS[3] mail service. Many events can happen when sending a letter. As depicted on the right side of Figure 1.1, two of your friends receive your letter without any issues. However, one friend didn't care for the recipe, so they throw it away. The third friend never received your letter (denoted with the red ? in the figure). This wasn't their fault necessarily, the letter could have gotten lost in the mail system, the address could have been incorrect, etc.

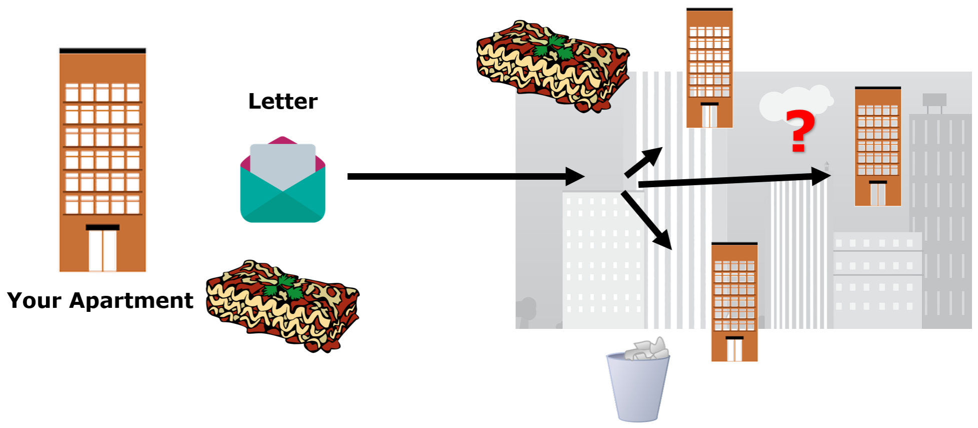


Figure 1.1: This example shows a letter being sent with a lasagna recipe to the sender's friends and the different scenarios that can happen to the letter in transit.

Now let's focus on the friend who received your letter and kept it. They decide

[3]United States Postal Service.

to make the lasagna and agree that it is the best lasagna they have ever had. They write a thank you note and put it in an envelope addressed to you, to send back to you. The same process you took to send your recipe occurs when your friend sends you something. Relating this back to a computer network, we replace our apartment example with an example of an office of computers in Figure 1.2.

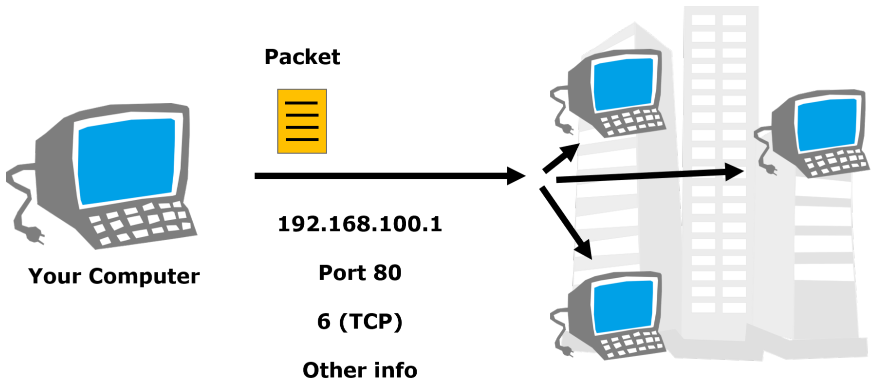


Figure 1.2: This example shows a packet with the designated IP address, port number, protocol number, and other information being sent to other computers on the same network.

If you had typed up the lasagna recipe to send to your co-workers on your office computer, a similar process would occur as in our apartment example. In this case, your envelope is what's called a packet. Packets contain information needed to send a communication from computer to computer. In order, from the list in Figure 1.2, a packet contains Internet Protocol (IP) Addresses (source and destination), port numbers (source and destination), a protocol used, and other various pieces of information, like the message content. IP Addresses can be related back to our apartment example as your and your friends' apartment building addresses. They are used to define which computer is sending a communication to which other computer(s). Port numbers can be the equivalent to your and your friends' apartment numbers. They specify more precisely than IP addresses which channel on the computer is to receive the communication. The protocol can be compared with the mail system you chose to send the letter in (USPS, but could have been FedEx, UPS, etc.). It defines how the communication will be sent and with what rules (e.g. tracking). The other information would contain the lasagna recipe itself, along with other details.

Packets do not always carry lasagna recipes in them. Malware and viruses are dispersed to other computers by the sending of these packets. This can be thought of as receiving blackmail, or something that is meant to cause harm to you and/or your computer. An example of one such attack is a SQL[4] injection. This involves an attacker finding security flaws in an application and using malicious query statements in order to retrieve information from a database they would otherwise not be allowed access to. We classify the packet(s) involved as an attack, and label them as bad packets. Unfortunately, it is not easy to avoid these as every communication from outside of your computer could contain some form or part of an attack, and there are ways to disguise them as good packets. However, we can't pay people to detect these bad packets manually; it's too hard of a problem! This is where machine learning techniques can come into play for solving this problem.

[4]SQL stands for Standard Query Language and it is a coding language that is used for building and interacting with databases.

Chapter 2

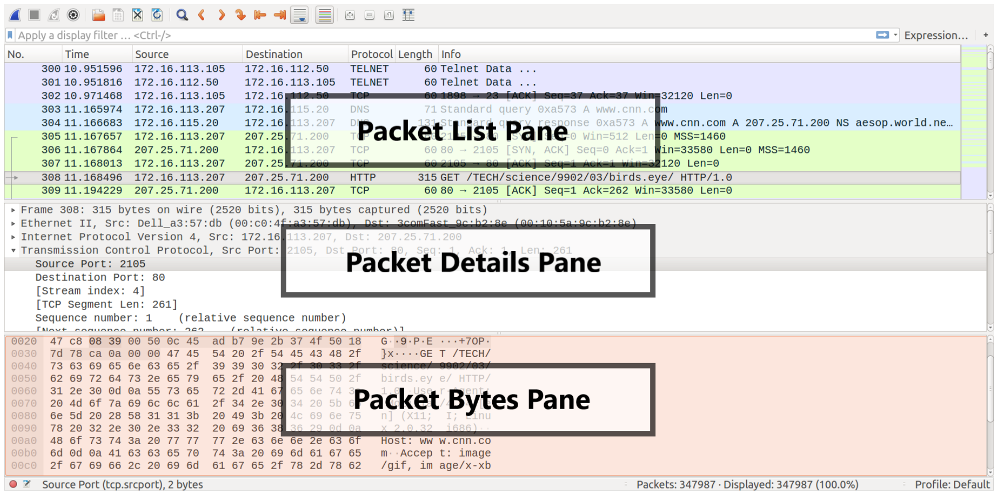
Background

We divide the background section into several subsections, each of which address an important aspect of this thesis. In particular, this thesis draws from many disciplines and we will extract from each the important concepts we need here. For example, as our focus is on detecting attacks in cyber networks we begin by defining some necessary concepts and terminology. We would like to emphasize the importance of Section 2.4 as it contains novel accomplishments of this thesis. The following table illustrates the notation used in the rest of this paper.

Notation	**Definition** *italic*
y, $\hat{y}$	Scalars are non-bold and *italic*
$\mathbf{z}$, $\mathbf{w}$	Vectors are lower-case and **bold**
$\mathbf{X}$	Matrices are UPPER-CASE and **bold**
$\mathbf{X^T}$	$\mathbf{T}$ indicates matrix transpose
$\mathbf{X_s}$	Indicates sub-matrix $\mathbf{s}$ of $\mathbf{X}$
n	Number of samples
m	Number of features
k	Number of reduced samples or features

2.1 PCAP Files

Packet CAPture (PCAP) files are a type of file that contain various information about the interactions between computers. This information can include the time and date the communication occurred, the source and destination IP addresses, the protocol used, the length of the communication, and more. The particulars are gathered from the user recording the activity on their computer to a PCAP file. This can be done through an application called Wireshark, a free and open-source network protocol analyzer [ORB06]. Figure 2.1 shows the basic panes used to view the information.



```
No.  Time       Source          Destination     Protocol Length Info
300 10.951596   172.16.113.105  172.16.112.50   TELNET     60 Telnet Data ...
301 10.951816   172.16.112.50   172.16.113.105  TELNET     60 Telnet Data ...
302 10.971468   172.16.113.105  172.16.112.50   TCP        60 1898 → 23 [ACK] Seq=37 Ack=37 Win=32120 Len=0
303 11.165974   172.16.113.207  172.16.115.20   DNS        71 Standard query 0xa573 A www.cnn.com
304 11.166683   172.16.115.20   172.16.113.207  DNS       131 Standard query response 0xa573 A www.cnn.com A 207.25.71.200 NS aesop.world.ne…
305 11.167657   172.16.113.207  207.25.71.200   TCP        60 2105 → 80 [SYN] Seq=0 Win=512 Len=0 MSS=1460
306 11.167864   207.25.71.200   172.16.113.207  TCP        60 80 → 2105 [SYN, ACK] Seq=0 Ack=1 Win=33580 Len=0 MSS=1460
307 11.168013   172.16.113.207  207.25.71.200   TCP        60 2105 → 80 [ACK] Seq=1 Ack=1 Win=32120 Len=0
308 11.168496   172.16.113.207  207.25.71.200   HTTP      315 GET /TECH/science/9902/03/birds.eye/ HTTP/1.0
309 11.194229   207.25.71.200   172.16.113.207  TCP        60 80 → 2105 [ACK] Seq=1 Ack=262 Win=33580 Len=0

▸ Frame 308: 315 bytes on wire (2520 bits), 315 bytes captured (2520 bits)
▸ Ethernet II, Src: Dell_a3:57:db (00:c0:4f:a3:57:db), Dst: 3comFast_9c:b2:8e (00:10:5a:9c:b2:8e)
▸ Internet Protocol Version 4, Src: 172.16.113.207, Dst: 207.25.71.200
▾ Transmission Control Protocol, Src Port: 2105, Dst Port: 80, Seq: 1, Ack: 1, Len: 261
    Source Port: 2105
    Destination Port: 80
    [Stream index: 4]
    [TCP Segment Len: 261]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 262    (relative sequence number)]

0020  47 c8 08 39 00 50 0c 45  ad b7 9e 2b 37 4f 50 18   G·9·P·E ···+7OP·
0030  7d 78 ca 0a 00 00 47 45  54 20 2f 54 45 43 48 2f   }x····GE T /TECH/
0040  73 63 69 65 6e 63 65 2f  39 39 30 32 2f 30 33 2f   science/ 9902/03/
0050  62 69 72 64 73 2e 65 79  65 2f 20 48 54 54 50 2f   birds.ey e/ HTTP/
0060  31 2e 30 0d 0a 55 73 65  72 2d 41 67 65 6e 74 3a   1.0··Use r-Agent:
0070  20 4d 6f 7a 69 6c 6c 61  2f 34 2e 30 34 20 5b 65    Mozilla /4.04 [e
0080  6e 5d 20 28 58 31 31 3b  20 49 3b 20 4c 69 6e 75   n] (X11;  I; Linu
0090  78 20 32 2e 30 2e 33 32  20 69 36 38 36 29 0d 0a   x 2.0.32  i686)··
00a0  48 6f 73 74 3a 20 77 77  77 2e 63 6e 6e 2e 63 6f   Host: ww w.cnn.co
00b0  6d 0d 0a 41 63 63 65 70  74 3a 20 69 6d 61 67 65   m··Accep t: image
00c0  2f 67 69 66 2c 20 69 6d  61 67 65 2f 78 2d 78 62   /gif, im age/x-xb

Source Port (tcp.srcport), 2 bytes     Packets: 347987 · Displayed: 347987 (100.0%)     Profile: Default
```

Figure 2.1: Wireshark Pane Layout. These panes can be customized to display all information about the packets captured in the file.

Within a PCAP file, one can find the details on every communication (packets) sent on the network. The Packet List Pane shows quick one-line summaries of each packet in the capture. This would include the information listed above. The Packet Details Pane provides, in a tree-like structure, a more comprehensive look at the specifics of the packet, such as which protocol was used and with what settings. The Packet Bytes Pane displays the raw captured data in both hexadecimal and text format. This information represents the payload of the packet that will be used by the end user. Examples include text-encoded payloads for HTML representations of web pages, and binary-encoded payload for video streams. These are useful as each packet can be thought of from two perspectives. First, a packet is a transport mechanism and contains information such as IP addresses and protocol type (e.g. UDP versus TCP). On the other hand, each packet is also a part of some end user process. At the simplest, a packet might be a “ping” packet which asks for a response to check for if a certain computer exists on the network. At the other extreme, a packet can be a small piece of a web page. For each type of detail, like the length of the packet, the destination and source IP addresses, etc., they become a feature for us to use in our data set.

However, some data sets provide packet flows. The main difference between packet captures and packet flow captures is that the former shows every single communication on a network, while the latter groups the sequences of communications on a network. By specification, “a flow is a sequence of packets sent from a particular source to a particular...destination that a node desires to label as a flow” [ACJR11]. This means that instead of seeing every single packet sent and received in, for example, the action of sending an email to someone, we see a single “packet” summarizing the details of that action, which we call a flow. This includes all of the basic details of a packet we’ve covered previously, like the source and destination IP addresses and port numbers, the protocol, etc., but it also provides other details not seen in single packet captures, such as how many packets were sent (Total Forward Packets), how many were received (Total Backward Packets), how long the flow lasted (Flow Duration), and other statistics on the packets within the flow.

For our work, it’s useful to turn this information into Comma Separated Values (CSV) files as the data that is generated by Wireshark needs to be shared with other parts of the processing chain. Fortunately, the raw Wireshark data can be exported into standard formats, such as CSV, for later being easily consumed by our Python package.

2.2 PCA

Principal components analysis (PCA) is a standard method for modeling correlation [BHPT06]. It's used for reducing the number of dimensions (features) in a data set. Specifically, PCA is most often implemented using the Singular Value Decomposition (SVD). In particular, consider the following equation:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}} \tag{2.1}$$

where $\mathbf{X} \in \mathbf{R}^{m \times n}$ is a matrix of the data, $\mathbf{U} \in \mathbf{R}^{m \times m}$ is a unitary matrix, $\mathbf{\Sigma} \in \mathbf{R}^{m \times n}$ is a diagonal matrix with non-negative real numbers on the diagonal, and $\mathbf{V}^{\mathbf{T}} \in \mathbf{R}^{n \times n}$ is a unitary matrix.

It is important to note that $\mathbf{\Sigma}$ in equation 2.1 provides important structural information about the data of interest, as do $\mathbf{U}$ and $\mathbf{V}$. For example, many of the items we measure are **not** independent of each other. One of the key research results of this work is to study how the dependencies between our measured quantities change depending on the presence or absence of an attack.

So, we can use the values on the diagonal in $\mathbf{\Sigma}$ to our advantage for extracting the important features. The values on the diagonal vary from a few large singular values to many smaller values close to zero. The order on the diagonal also serves a purpose, as the largest value is the top, left-most value in the matrix. This order can be seen in equation 2.2.

$$\sigma_{\mathbf{1}} \geq \sigma_{\mathbf{2}} \geq \sigma_{\mathbf{3}} \geq \ldots \geq \sigma_{\mathbf{n}} \geq \mathbf{0} \tag{2.2}$$

We are the most interested in the larger singular values. We can reduce the $\mathbf{\Sigma}$ matrix down to only the rows and columns that contain the largest sigma values on the diagonal. This produces $\hat{\mathbf{\Sigma}}$, and the process can be viewed in Figure 2.2.

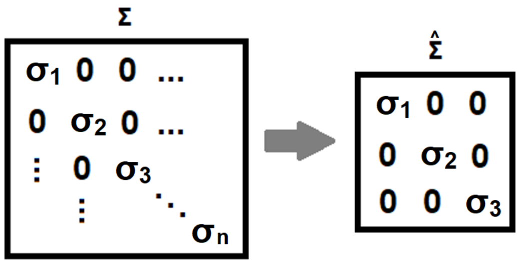


Figure 2.2: Example showing how the values on the diagonal of the $\mathbf{\Sigma}$ matrix are reduced to only the largest σ values to produce $\hat{\mathbf{\Sigma}}$, and thus $\hat{\mathbf{X}}$.

We use $\hat{\mathbf{\Sigma}}$ to derive equation 2.1 to produce our equation for PCA:

$$\hat{\mathbf{X}} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathbf{T}} \tag{2.3}$$

where $\hat{\mathbf{X}} \in \mathbf{R}^{m \times n}$ is a matrix of the data, a unitary matrix $\hat{\mathbf{U}} \in \mathbf{R}^{m \times k}$, a diagonal matrix with non-negative, non-zero numbers on the diagonal $\hat{\mathbf{\Sigma}} \in \mathbf{R}^{k \times k}$, and a unitary matrix $\hat{\mathbf{V}}^{\mathbf{T}} \in \mathbf{R}^{k \times n}$.

PCA chooses a low dimensional representation of the input data $\mathbf{X}$ based upon minimizing the squared distance between the original points and the low dimensional points. Since PCA uses the square of the distance, outliers in the data can skew its performance greatly. Given that our goal is to classify good packets from the bad ones, PCA lacks what we need.

2.3 Robust PCA

Robust Principle Component Analysis (RPCA) can be used in the detection of anomalous data. It is similar to PCA, as it also performs dimension reduction on a data set, though it surpasses PCA in overcoming the interference of outliers in the data.

One of the main input parameters we use to tune the RPCA algorithm is lambda (λ). It is a float value normally greater than 0 and less than 1. The value for λ will vary per data set, and ML model used. So for one model, a good λ value could be 0.07, while for another it could be 0.2. The process of choosing a λ value is not well understood for the kinds of problems that we want to solve. There is an equation 2.4 that can be used to update λ based on the size of the data set.

$$\lambda = 1/\operatorname{sqrt}(\max(m, n)) \tag{2.4}$$

However, the theoretical assumptions behind this equation are different than those in the problems we consider. A detailed analysis of this equation is beyond the scope of this thesis, and we merely observed that 2.4 is not designed for the case where one is using RPCA for feature engineering. Accordingly, we iterate over a range of possible λ values to run our package on to find acceptable λ(s). Then, we scale down that original range depending on the results to find the best λ possible. Although, either method may not produce an optimal λ. To date, there is no existing algorithm or method for computing an optimal λ for when the output of RPCA is used for feature engineering.

One of the novel aspects of this work is to study RPCA and how λ effects how the data in the $\mathbf{X}$ matrix is dispersed. This is shown in equation 2.5 from [CLMW11].

$$minimize\|\mathbf{L}\|_* + \lambda\|\mathbf{S}\|_1 \text{ s.t. } \mathbf{L} + \mathbf{S} = \mathbf{M} \tag{2.5}$$

where $\mathbf{M}$ represents our $\mathbf{X}$ matrix, $\mathbf{S} \in \mathbf{R}^{m\times n}$ is a sparse matrix (i.e. has only a few non-zero entries), and $\mathbf{L} \in \mathbf{R}^{m\times n}$. Here we can observe that λ effects the sparse matrix and helps to distribute the data between the $\mathbf{S}$ and $\mathbf{L}$ matrices based on its value. In addition, from the above equation $\mathbf{L}$ is constrained by the nuclear norm 2.6 [Fan51], while $\mathbf{S}$ by the L_1 norm in 2.7 [DZHZ06].

$$\|A\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i(A) \tag{2.6}$$

$$\|A\|_p = \left(\sum_1^m \sum_1^n \left|a_{i,j}^p\right| \right)^{1/p} \tag{2.7}$$

While beyond the scope of this thesis, we observe that the RPCA equation can be efficiently solved using an alternating direction method of multipliers (ADMM) [PKS18]. The $\mathbf{L}$ matrix is computed from the PCA equation 2.3, and can be written as:

$$\mathbf{L} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathbf{T}} \tag{2.8}$$

RPCA produces both $\hat{\mathbf{\Sigma}}$, a measure of how related measurements are, as well as $\mathbf{S}$, a measure of how some measurements are not related (the anomalous data). In Figure 2.3, we can see a point that clearly sticks out in the $\mathbf{S}$ matrix that otherwise would not seem significant in $\mathbf{X}$. The same point is circled in red for the $\mathbf{X}$ and $\mathbf{L}$ matrices in the figure.

Note that the real-world network data is quite low-dimensional, which means it's quite predictable. It is interesting to consider the decomposition in equation 2.5 in

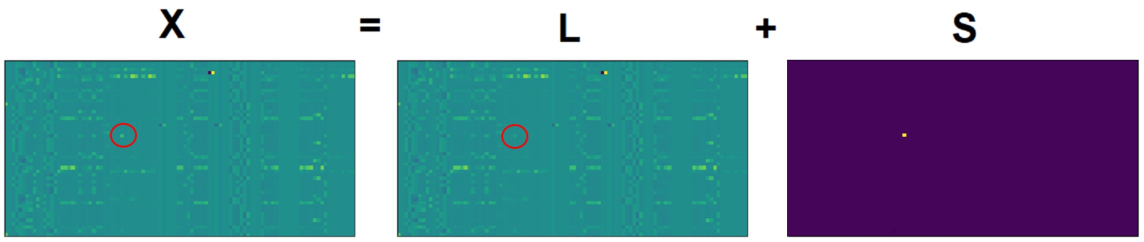


Figure 2.3: A visual representation of the RPCA equation in the form of matrix heat-maps for the same rows of packets from **X**, **L**, and **S** that highlight an anomaly.

the context of cyber-data. In particular, when thinking of the data that is generated by a computer network it is not hard to imagine that various of the measurements we make are quite correlated. Using RPCA, one can observe that the predictability changes (as shown in **S**) are important in figuring out which features most attribute to categorizing "an attack". However, once the output of RPCA becomes two dimensional, the produced **S** matrix may not contain all of the points where it thinks there are "anomalies" in the data, and this information will bleed into the **L** matrix. In order to handle this case, we will need to resort to using machine learning tools to further extract the anomalies, which can be thought of as unsupervised feature engineering.

2.4 Projectors

Inspired by RPCA, to help reduce the time complexity of our technique and increase the flexibility of how a data set could be manipulated, we use projectors. They allow us to be able to run the RPCA algorithm only once on a data set and apply the same reduction to other data (such as testing data from an ML algorithm). From the RPCA algorithm, the matrices $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$, $\hat{\mathbf{V}}^{\mathbf{T}}$, and **S** are produced (see 2.5). Only the matrix $\hat{\mathbf{V}}^{\mathbf{T}}$ is used to further calculate the projection onto other matrices. In the following we develop the details of producing our desired projected matrices.

Our package takes a data set, given as a matrix $\mathbf{X} \in \mathbf{R}^{m\times n}$, and splits it into three matrices to use in the rest of the package. The purposes of them are

- $\mathbf{X_{train}} \in \mathbf{R}^{k\times n}$ runs on the RPCA algorithm and trains the ML models,
- $\mathbf{X_{validate}} \in \mathbf{R}^{k\times n}$ tunes the hyper-parameters e.g. λ,
- and $\mathbf{X_{test}} \in \mathbf{R}^{k\times n}$ evaluates the performance of the ML models.

Since the RPCA equation (2.5), in part, uses the PCA equation (2.3), we can benefit from this relationship between them. Specifically, with the calculation for $\hat{\mathbf{X}}$ of 2.3, we know this to be equal to the $\mathbf{L}$ matrix for 2.8. If we then take equation 2.3 and dot both sides with $\hat{\mathbf{V}}$ (see 2.9), we can obtain equation 2.10:

$$\begin{aligned} \mathbf{X_{validate}} &= \hat{\mathbf{U}}_{\mathbf{validate}}\hat{\mathbf{\Sigma}}_{\mathbf{validate}}\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{validate}} \\ (\mathbf{X_{validate}})\hat{\mathbf{V}}_{\mathbf{train}} &= (\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathbf{T}})\hat{\mathbf{V}}_{\mathbf{train}} \qquad (2.9) \\ \mathbf{X_{validate}}\hat{\mathbf{V}}_{\mathbf{train}} &= \hat{\mathbf{U}}\hat{\mathbf{\Sigma}} \qquad (2.10) \end{aligned}$$

Where $\hat{\mathbf{V}}_{\mathbf{train}} \in \mathbf{R}^{n\times k}$ is the transpose of $\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}} \in \mathbf{R}^{k\times n}$, and $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathbf{T}}$ are from $\mathbf{X_{validate}}$. Note that in the equation above we assume that $\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{validate}}\hat{\mathbf{V}}_{\mathbf{train}} = \mathbf{I}$. Of course, this is, in general, not true. However, as long as it is approximately true we demonstrate in Section 4 that the following procedure is effective in improving the capabilities of ML algorithms. Taking equation 2.10, we can again dot both sides, though this time with $\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}}$ to obtain equation 2.11:

$$\mathbf{X_{validate}}\hat{\mathbf{V}}_{\mathbf{train}}\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}} \qquad (2.11)$$

However, we stop here and see something familiar! If we remember the note from earlier in this section, the right-hand side of 2.11 is equal to **L** in the RPCA equation (2.5). Presented in equation 2.12:

$$\mathbf{L_{validate}} = \mathbf{X_{validate}}\hat{\mathbf{V}}_{\mathbf{train}}\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}} \tag{2.12}$$

Where $\mathbf{L_{validate}} \in \mathbf{R}^{m \times n}$ is the projected **L** matrix of $\mathbf{X_{validate}}$. From here it's as simple as using equation 2.5, and reordering the terms to produce:

$$\mathbf{S_{validate}} = \mathbf{X_{validate}} - \mathbf{L_{validate}} \tag{2.13}$$

Where $\mathbf{S_{validate}} \in \mathbf{R}^{m \times n}$ is a sparse matrix that is the projection of the **S** matrix of $\mathbf{X_{validate}}$. These equations, 2.12 and 2.13, are also used to calculate $\mathbf{X_{test}}$ as well:

$$\mathbf{L_{test}} = \mathbf{X_{test}}\hat{\mathbf{V}}_{\mathbf{train}}\hat{\mathbf{V}}^{\mathbf{T}}_{\mathbf{train}}$$
$$\mathbf{S_{test}} = \mathbf{X_{test}} - \mathbf{L_{test}}$$

To summarize, we utilized this process in order to obtain $\mathbf{L_{validate}}$ and $\mathbf{S_{validate}}$ of $\mathbf{X_{validate}}$ instead of needing to rerun the RPCA algorithm on $\mathbf{X_{validate}}$, and again for $\mathbf{X_{test}}$. This gives us more flexibility and better time complexity in our method.

Chapter 3

Methodology

The first step in answering our research question, "does using RPCA improve the output performance of ML models?", was to create the code base for running and testing this theory. In this chapter, we detail the data sets utilized, and explain our approach used to conduct the research.

3.1 Data Sets

We utilized three data sets in our research and experiments, listed below. The primary data set used while developing our package is the first data set in the list from the University of New Brunswick. Further details on each of these data sets can be found in the following subsections, and tables of how we feature engineer the data sets are in appendix A.

1. Intrusion Detection Evaluation Dataset (CICIDS2017), from the University of New Brunswick[1]

[1]UNB link: `https://www.unb.ca/cic/datasets/ids-2017.html`

2. MIT LL 2000 DARPA Intrusion Detection Scenario Specific Dataset; LLDOS 2.0.2 - Scenario Two[2]

3. Data set by MIT's GOSSIS community initiative (From the WiDS Datathon 2020)[3]

3.1.1 Data Set One

The UNB data set was created to improve upon previous intrusion detection data sets. It provides the most up-to-date common attacks, and is very realistic to true network traffic in an every-day office setting. We chose to use this set as we wanted to apply our research to cyber security applications, and as its purpose of creation states, is very detailed and has a variety of benign and attack data. Thus, the goal of using this data set is to classify normal packets from malicious packets.

From the data set, we run tests using the "Thursday Morning" CSV file, and utilize most of the features provided. After some preliminary research and unusually accurate results, we realized using the source IP address as a feature was causing bias in our method, and thus do not include it as a feature. This is due to there only being a few attacker IP addresses, compared to the many victim IP addresses. We do use the destination IP address, though we spilt it into each of the address' bytes (4), then one-hot encode those bytes. This is to reduce the number of overall features. We also capped the source and destination port numbers to 1024 if they were at or above this number to again help reduce the overall number of features, and also since the ports 0 to 1023 are privileged ports[4]. See Table A.1 for more details on every feature in the data set, and how we handle them.

[2]LLDOS link:`https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_2.0.2.html`

[3]WiDS link: `https://www.kaggle.com/c/widsdatathon2020/overview`

[4]Meaning there are certain protocols reserved for these ports.

3.1.2 Data Set Two

This data set is an enhanced version of an earlier data set produced by Joshua W. Haines for DARPA[5]. It is comprised of five phases leading up to a DDoS attack[6] by the adversary. We chose to use this set in order to compare with the work done from the paper [PKS18]. Again, the goal of using this data set is to classify normal packets from malicious packets.

It includes PCAP files that we converted into CSV files. To keep consistency for comparison with the work done from [PKS18], we only use the first 3 phases of the data and try to match as much as possible the same feature engineering choices. We did only use the first byte of the source and destination IP addresses for features to reduce the overall number of features. And we again capped the source and destination port numbers to 1024 if they were at or above this number to help reduce the overall number of features. Other than these changes, the rest of the feature engineering choices stayed the same. See Table A.2 for more details on every feature, and how we handle them.

3.1.3 Data Set Three

This data set emerged from the Kaggle competition for the 2020 Women in Data Science (WiDS) Datathon, though was originally started with the GOSSIS[7] Project [Fab17]. We participated in a team for the competition, and used our package to assist our team members. This also gave us another variety of data sets to try. The goal of

[5]The first being MIT LL 2000 DARPA Intrusion Detection Scenario Specific Dataset; LLDOS 1.0 - Scenario One.
Link to site: `https://archive.ll.mit.edu/ideval/data/2000/LLS_DDOS_1.0.html`

[6]Distributed Denial of Service. The attack floods the victim's computer system with traffic to compromise the system.

[7]Global Open Source Severity of Illness Score

this competition was to create a model that uses data from the first 24 hours of intensive care to predict each patient's mortality rate.

There were two main CSV files provided for the data set. One file had labelled data, and the other was unlabelled. For our contributions with our package to the competition, we performed basic feature engineering to all features. See Table A.3 for more details on every feature, and how we handle them.

3.2 Approach

Our package consists of a few distinct parts during execution. The first being the use of a configuration file, for setting the parameters, operations, and data set of the run. Second, the creation of the data matrix **X**, including the normalization and randomization of the data. Third, the use of RPCA and projections on the matrix **X**. And finally, the running and evaluation of machine learning models. This process can also be viewed in the UML[8] diagram below:

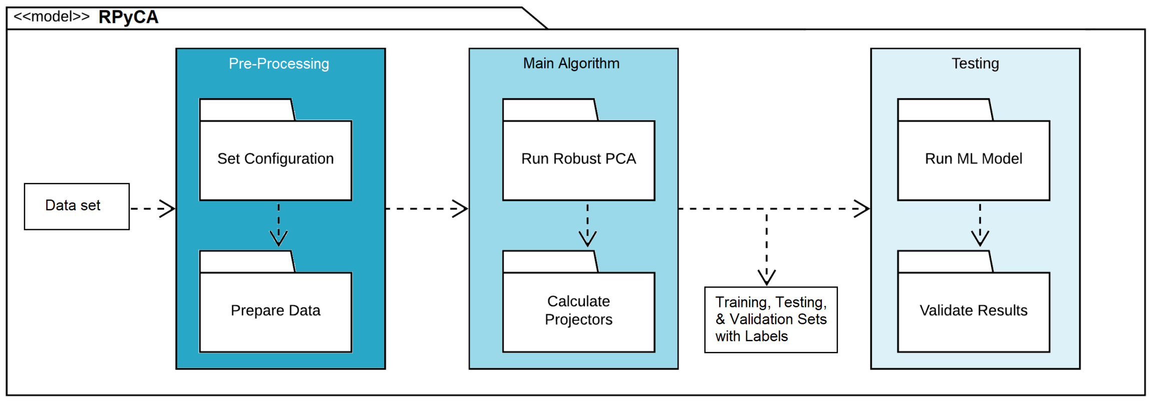


Figure 3.1: RPyCA Python Software Package structure.

[8]Unified Modeling Language.

3.2.1 Main Program

For each split matrix from $\mathbf{X}$, we utilize their corresponding output from RPCA and the projectors to produce two other matrices to use as different feature engineering techniques we test. These are, as depicted in Figure 3.2, the *concatenation* of the $\mathbf{L}$ and $\mathbf{S}$ matrices, $\mathbf{LS}$, and the *concatenation* of the $\mathbf{X}$, $\mathbf{L}$, and $\mathbf{S}$ matrices, $\mathbf{XLS}$. For example, if we take $\mathbf{X_{train}}$, its matrices would only use $\mathbf{X_{train}}$'s corresponding $\mathbf{L}$ and $\mathbf{S}$ matrices from the RPCA algorithm.

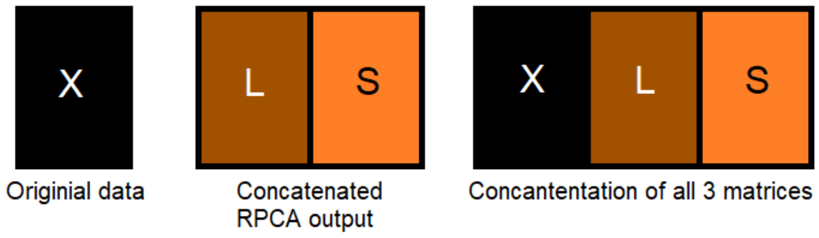


Figure 3.2: Feature engineered matrices we test on ML models.

The purpose of using these matrices is to test our hypothesis that RPCA generated features assist in ML tasks. For the $\mathbf{LS}$ matrix, this can be thought of as an expanded version of the $\mathbf{X}$ matrix (see equation 2.5). For $\mathbf{XLS}$, this is taking the original data, $\mathbf{X}$, and appending extra information to each row (in our case, the rows are packets). These are the matrices we run through the ML models, in addition to each $\mathbf{X}$ matrix. When we describe our process in the following, it also includes testing with the related $\mathbf{LS}$ and $\mathbf{XLS}$ matrices.

Our process of running our package is described in the following and can be referenced in the "Main Algorithm" and "Testing" blocks in Figure 3.1. We start by executing the RPCA algorithm on $\mathbf{X_{train}}$. Then we project, as detailed in section 2.4, onto the $\mathbf{X_{validate}}$ matrix, with the results from the RPCA algorithm. Then we assess how $\mathbf{X_{train}}$ and $\mathbf{X_{validate}}$ perform on the ML model. Only if the results from the ML model are good, do we move on to evaluate with the final matrix, $\mathbf{X_{test}}$. If a result is not produced by any of our matrices, we repeat the above process with a new lambda value. Else, we again project onto $\mathbf{X_{test}}$, then use $\mathbf{X_{train}}$ and $\mathbf{X_{test}}$ on the same model with the chosen hyper-parameters. This is where we retrieve our final results.

If we want to reproduce an exact experiment or run, there is a logger included in the package that creates log files for the user that saves the necessary information in order to achieve this. Of course, if the package is to be used for commercial purposes, the logger should be handled appropriately for security.

3.2.2 Configurations

Configurations of the package lie in the config.ini file in the project structure. This file allows the user to define the necessary parameters for using the package. It also allows for the ease of reuse on a variety of data sets. Users can save custom configurations in the file by typing them into the file. Another option is users can start off with the default configuration.

The default run is recommended for first-time runs of a data set, as it will perform parameter tuning and testing for the best results and create a new configuration with these parameters. This is, of course, a much longer run time of the package, but it is a necessary step for determining the most optimal parameters and achieving the best output from the models. Though this tuning and setting of the parameters can be done manually, we cannot guarantee the benefits of using this package for this case. It is also recommended to perform this initial run in a place where it will only run once or run it once separately from the program being used to benefit from this package. More details on the configuration file can be read in the appendix B.1.

3.2.3 Pre-Processing

After the configuration is set, the code moves on to extracting the data from the specified database file, then creates and normalizes the $\mathbf{X}$ matrix.[9] We first specify which columns from $\mathbf{X}$ to perform one-hot encoding on, then normalize each column, and finally we randomize the rows to produce the three final matrices of $\mathbf{X}$..

One-Hot Encoding

One pre-processing step we use is one-hot encoding. This technique is not needed for every feature in our data, but it is useful for enabling those features it is performed on to produce more meaningful information. The process involves taking a column of data and transforming it into multiple columns with binary values. This means that if we have a feature that is categorical, we can convert the data in that column to be more useful to a model. This is necessary, as when a model receives a value from a categorical feature, like "Breeds of Dogs", the category of the actual strings "German Shepard" and "Golden Bois" do not provide semantic value to the model. However, if instead we make each breed of dog a feature, then assign a value of 1 if that row has that breed, 0 otherwise, and provide those features, we are able to convey much more information to the model.

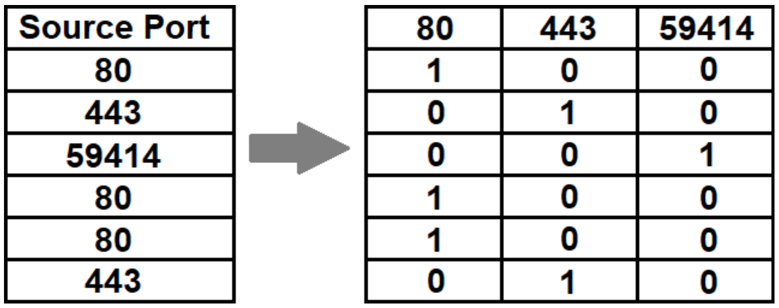
Categorical features do not have to be strings or words as data, they can be numbers as well. One instance is when we handle port numbers.[10] For example, port 21 is for the File Transfer Protocol (FTP), while port 22 is for Secure Shell (SSH). These ports vary greatly in function, even though these numbers are close

[9]While creating the $\mathbf{X}$ matrix, an accompanying $\mathbf{y}$ vector where $\mathbf{y} \in \mathbf{R}^{m \times 1}$ is also created for the purpose of the corresponding labels. See the note in the appendix for Table A.1 for more information.

[10]Port numbers are logical access channels for communication between two devices. In simpler terms, they act as passages to allow communications to flow between two computers.

numerically, they actually could not be any more different.

As shown in figure 3.3, equivalent port numbers are grouped together to create their own column to be used in matrix **X**. These expanded columns replace where the one original column was in the matrix. Continuing with this example, if we only had those three port numbers appear in the Source Port column in **X**, then those three columns would replace the one for Source Port in **X**.



Source Port
80
443
59414
80
80
443

80	443	59414
1	0	0
0	1	0
0	0	1
1	0	0
1	0	0
0	1	0

Figure 3.3: The source port feature goes through one-hot encoding to produce the three new features on the right-hand side of the figure.

We can also choose how to manipulate the groupings of the data in column by grouping columns before one-hot encoding them. One example of a grouping could be by grouping all protected ports, port numbers above 59414. Overall, by one-hot encoding the data we can bring meaning to the categorical values of the column.

Normalization

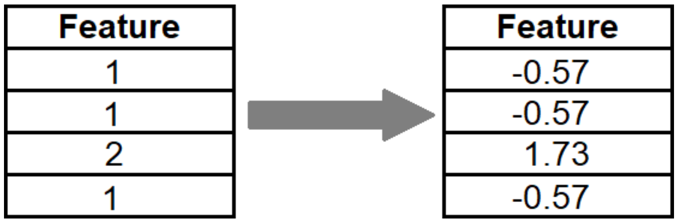
Once all the columns from **X** have been pre-processed, we then normalize **X** using Z-transform (a.k.a. z-score). We use the equation 3.1 to compute the score across each column individually.

$$\mathbf{z} = \frac{\mathbf{x_i} - \mu_\mathbf{i}}{\sigma_\mathbf{i}} \tag{3.1}$$

Where $\mathbf{z} \in \mathbf{R}^{m\times 1}$ is the resulting normalized column vector of the data, $\mathbf{x_i} \in$

$\mathbf{R}^{m\times 1}$ is the i^{th} column of matrix $\mathbf{X}$ that is being normalized, $\mu_{\mathbf{i}} \in \mathbf{R}$ is the mean of the i^{th} column of matrix $\mathbf{X}$, and $\sigma_{\mathbf{i}} \in \mathbf{R}$ is the standard deviation of the i^{th} column of matrix $\mathbf{X}$.

It is necessary to do this as it prevents our columns of data from having large skews between the values in the column. For example, as shown in figure 3.4, a point that has a value of 1 compared with another point in the same column of 2 will distribute the two points from the mean. This type of normalization shifts and scales the data over the original distribution centered around μ.



Feature
1
1
2
1

Feature
-0.57
-0.57
1.73
-0.57

Figure 3.4: A feature in the data set is normalized based on the data in the column by using the Z-transform equation 3.1

Randomization

After processing on $\mathbf{X}$ is complete, we use a random seed to randomly assign chosen rows (data points) from $\mathbf{X}$ to create the new matrices referred to in Section 2.4. A visual representation of how the randomization happens can be seen in Figure 3.5. We perform this step to help ensure that the order of the data provides no statistical relevance in how our model performs. That is to say, we do not rely on the order of the data as a “feature”, and our technique performs independently of it.

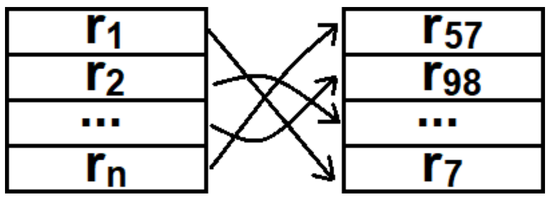


Figure 3.5: Rows on the left-hand side of the figure get reordered on the right-hand side after randomization happens. This process is based on a randomly generated seed.

Note that the actual size of these resulting matrices vary on the size of **X**, and what parameters are used in the configuration file. A random seed is generated to section which data points are included in each matrix. Once this is finished, the matrices are returned and are ready to be used in the RPCA algorithm.

Chapter 4

Results

In this section we evaluate the effectiveness of our technique across various machine learning models and the data sets. General descriptions for each model can be found in the appendix B.2. For the majority of our data sets, we use F1-score to evaluate the performance output of the machine learning models[1]. We use the equation 4.1 to calculate the score of the model on each $\mathbf{X}$ matrix as well as on each feature engineered matrix [VR79].

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{4.1}$$

This score can be derived from the confusion matrix. This is a table that can be used to describe the performance of a model through showing the True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) as shown in Figure 4.1. To achieve the best performance for a data set that has binary classes, there should only be values in the TP and TN squares.

We chose this metric as it takes into consideration both precision and recall, as

[1]AUC (Area Under the Curve) score was used for the WiDS competition.

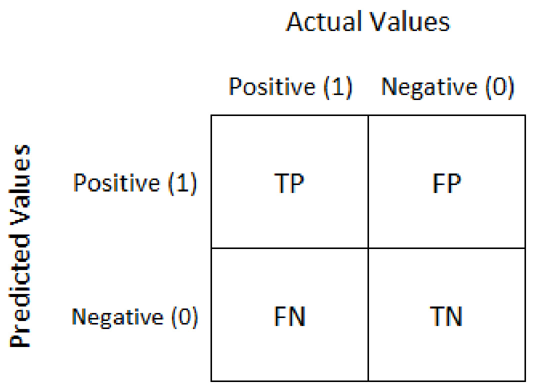


Figure 4.1: This figure from [Nar19] shows the different squares in a confusion matrix. To achieve the best performance for a data set that has binary classes, there should only be values in the TP and TN squares.

shown by the equation. This is useful as we perform binary classification[2] on the data sets, and need to consider whether the model chose packets that are relevant (precision), and that all relevant packets are chosen (recall).

4.1 Main Data Set

As stated previously in Section 3.1, we utilize the Thursday Morning CSV file to execute our testing on. We chose this file due to the nature of the cyber-attacks it covers which include different varieties of Web Attacks[3]. The goal of these type of attacks is to try to gain information users have out on the internet, like credit card information or personally identifiable information (PII).

No matter the original size of the file, we restrict the size of our resulting matrices to ratios of 0.2 of the total data set to the training set, and the remaining portion of the data set to 0.4 for the validation set, and the remaining to the testing set.

[2]Classifying a normal packet as 0, and an attack as 1.
[3]Specifically Brute Force, XSS, and SQL Injections.

These ratios are set in the configuration file. In addition to these ratios, we also set in the configuration file a sample size of a quarter (0.25) of the entire file size to cut down on total run time, and to show that little data is needed in order for our method to perform well.

For the data used from this file, we roughly split the matrices as shown in Table 4.1 below. These numbers are estimates, as with every run, a random number will skew the exact count of normal and attack packets per matrix. To regulate the distribution, we set those ratios for the training and validation data matrices. The total sample size for the Thursday Morning file ends up being 42,592 packets.

Matrix	**Normal**	**Attack**	**Total**
Train	8407	112	8519
Validate	20168	275	20443
Test	13453	177	13630

Table 4.1: These are estimates of the number of normal vs. attack packets used in producing these results. The total is 42592 rows of data (a quarter of the actual file).

Below in Table 4.2 we show the results from the file for each ML model.

Model	Lambda	Average F1 Score			# of times outperformed X	
		X	LS	XLS	(LS)	(XLS)
dtree	0.0005	**0.99572**	0.48474	0.48026	0	0
gb	0.0002	0.99414	**0.99442**	0.99384	1	1
knn	0.004	0.86315	**0.95181**	0.92323	10	10
logreg	0.03	0.98912	0.55940	**0.99324**	1	7
nb	0.1	**1.0**	0.99863	**1.0**	0	0
pynn	0.0001	0.51398	0.49328	**0.53868**	2	7
rf*	0.07	0.0	0.04841	**0.60229**	1	2
svm	0.14	0.98544	**0.98627**	0.98571	2	1

Table 4.2: Results from running our method on the Thursday Morning file from the UNB data set. Averages are from 10 randomly seeded runs. Shown in bold are the max scores for each model.
* *These models did not produce results for all 10 runs.*

Seven out of the nine models we used benefited from using either the **LS** or

XLS matrices. Some models did benefit more than others. In the case of Random Forest (rf), while the **X** matrix was not able to detect the attacks, **XLS** did, and with a decent accuracy of about 60%. For this data set, the K-Nearest Neighbors (knn) model benefited the most from the other matrices, especially **LS**, as they outperformed **X** for all ten runs. Further, it improved the accuracy of the model by 8.8%.

4.2 LLDOS Data Set

For this data set, we use as much of the data as possible while only including the first three phases of the attack. For the data used from this file, we roughly split the matrices as shown in Table 4.3 below. These numbers are estimates, as with every run, a random number will skew the exact count of normal and attack packets per matrix. To regulate the distribution, we set the ratios for the training and validation data matrices as 0.3 and 0.6 in the configuration file. The total sample size for the file ends up being 90,399 packets.

Matrix	**Normal**	**Attack**	**Total**
Train	27095	25	27120
Validate	25293	18	25311
Test	37929	39	37968

Table 4.3: These are estimates of the number of normal vs. attack packets used in producing these results. The total is 90399 rows of data.

Below in Table 4.4 we show the results from the file for each ML model.

While these results are not as substantial as the last data set, they do show some interesting outcomes. Again, we see that Random Forest was not able to produce any score for **X**, yet we were able to get scores from **LS** and **XLS**. We also observe that for K-Nearst Neighbors it did not matter which matrix it was given. For some other models, it is odd how **X** performs so well while the other matrices do not.

Model	Lambda	Average F1 Score			# of times outperformed X	
		X	LS	XLS	(LS)	(XLS)
dtree	0.22	**0.99192**	0.04409	0.03063	0	0
gb	0.2	**0.99086**	0.12855	0.16048	0	0
knn	0.005	**0.87681**	**0.87681**	**0.87681**	0	0
logreg	0.17	**0.94884**	0.91374	0.91374	1	1
nb	0.13	**0.08061**	0.06324	0.0613	2	2
rf	0.14	0.0	0.08564	**0.09348**	10	9
svm	0.17	**0.88972**	0.84607	0.85091	0	0
pynn	0.19	**0.81826**	0.15587	0.15484	0	0

Table 4.4: Results from running our method on the LLDOS data set. Averages are from 10 randomly seeded runs. Shown in bold are the max scores for each model.

This could be due to having less features used. The values of lambda are also much higher than the last model, and it is very possible there are better lambdas for these.

4.3 GOSSIS Data Set

Since this data set differs greatly from the others, results were gathered in a slightly different manner. As stated in Section 3.1, the goal was to predict the patient's mortality rate. This was predicted as a percentage, and is thus not a binary classification. This percentage was evaluated using the AUC score metric, and not the F1 score. As a team, we utilized LightGBM, a variant of Gradient Boosting, for our model [KMF+17]. The total data set included 91,713 encounters (with different patients).

The website the competition was hosted on produces the results for testing how our model performed. These scores in Table 4.5 are produced from the website. During the competition, the "Public" column was used to rank teams, and evaluated team's result on only half of the solutions file. The "Private" column was only revealed to teams after the competition ended, and evaluated with the other half of

Run	X		LS		XLS	
	Private	Public	Private	Public	Private	Public
1	0.87868	0.86403	**0.87928**	**0.87103**	0.87874	0.86547
2	0.87545	0.86225	**0.87689**	**0.8666**	0.8751	0.86358
3	0.87616	0.86075	**0.87769**	**0.86395**	0.87723	0.85904
4	0.87571	0.86197	**0.87703**	**0.86591**	0.87637	0.85955
5	**0.87624**	**0.86147**	0.87573	0.86017	0.87595	0.85878
6	0.87565	**0.86193**	**0.87628**	0.86085	0.87614	0.85932
7	0.8767	0.86104	**0.87785**	**0.86398**	0.87717	0.85975
8	0.87669	0.86191	**0.87712**	**0.86286**	0.87689	0.86045
9	0.87662	0.86601	**0.8792**	**0.87016**	0.87809	0.86359
10	0.87593	0.85836	**0.87801**	**0.86417**	0.87736	0.85985
Average	0.876383	0.861972	**0.877508**	**0.864968**	0.876904	0.860938

Table 4.5: Results from running our method on the GOSSIS data set with Light GBM. Public scores were evaluated on half of the solutions file, while private scores evaluated the other half. Each run was randomly seeded and scores were calculated with AUC. Shown in bold are the max scores for each run. Overall, **LS** had the highest average.

the solutions[4].

Overall, **LS** had the highest average for this data set. We can see that given other random seeds, like in runs 5 and 6, **X** sometimes did outperform **LS** and **XLS**. However, over these ten runs, **LS** did have more consistently high scores.

4.4 Method Efficiency

The comparison of the efficiency of using projectors instead of the RPCA algorithm can be seen in table 4.6. These results were gathered by running on the UNB data set. We chose to set the λ value for these runs to 0.0431, as it is the average of the λ values used for each model in Table 4.2. In order to produce these time results, we again ran our method ten times, using a different random seed each time. On

[4]The solutions file is not directly available. In order to score the results, a file with the predictions must be ran through the competition website.

Run	RPCA (seconds)	Projector (seconds)
1	602.40	1.29
2	606.93	1.32
3	602.02	1.22
4	564.97	1.22
5	576.50	1.21
6	587.40	1.24
7	601.05	1.23
8	625.76	1.30
9	613.63	1.22
10	595.21	1.19
Average	597.58	1.24

Table 4.6: Comparison of running the RPCA algorithm vs. the projection equations on the Thursday Morning file from the UNB data set. These were produced from 10 randomly seeded runs with a λ value of 0.0431.

average, the RPCA algorithm completed in 597.58 seconds, or roughly 10 minutes. Contrarily, the projection equations completed in 1.24 seconds. That is a 99.79% decrease in time! Based on the results gathered, it is quite obvious that projectors are faster than the RPCA algorithm[5].

4.5 Summary

These results help to further support our hypothesis. Although they are quite minimal given the potential our technique can cover, they provide a nice base line for future testing. We've shown that for some models where **X** does not score, our matrices can, and that even by restricting our data sets to smaller sizes, we can have the models continue to perform at a reasonable caliber. Additionally, by using projectors, our method provides minimal time in order for use.

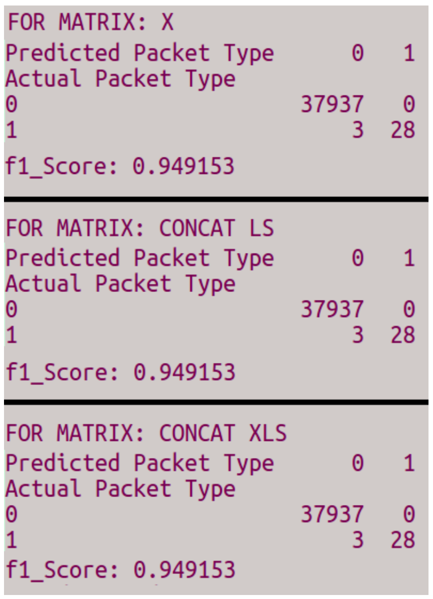
During the operation of our method, the RPCA algorithm can be run just once,

[5]The RPCA algorithm must be ran at least once in order to use projectors. After that initial run, there is no need to run the algorithm again unless the data used to run the algorithm changes.

and its output can be reused for later calculations, as with the projectors, which are very efficient. This allows for our method to be conducted in a streaming fashion (i.e., some small block of data arrives and is immediately processed to see if there is an attack). Finally, the RPCA algorithm can be run “offline”, perhaps just once a day, to supply new projections for any changes in the data.

Note that for the column in the first two data set results tables, “number of times outperformed **X**”, only applies to F1 Scores that were larger than what **X** scored during each run. This means that if **LS** or **XLS** met the same score as **X** it would not be counted here. The averages columns in each table are meant to act as a measure of consistency, as if for every run the model’s perform equally, their average F1 scores would be the same. In some tables, we do not see this, although the other matrices may have outperformed **X** a few times, or vice versa, as there may have been runs where they performed less well, or not at all. Of course, all of these results depend on the random numbers used.

Even when **LS** and/or **XLS** do not outperform **X**, our method can still benefit the user. In the case for when **LS** and or **XLS** match the performance of **X**, this information can be used to pin point the exact number of features needed to achieve peak performance. For example, with the Support Vector Machine (svm) of the LLDOS data set, at a lambda value of 0.17, the confusion matrices and calculated f1 scores were identical. This is shown in Figure 4.2.



```
FOR MATRIX: X
Predicted Packet Type      0   1
Actual Packet Type
0                      37937   0
1                          3  28
f1_Score: 0.949153
```

```
FOR MATRIX: CONCAT LS
Predicted Packet Type      0   1
Actual Packet Type
0                      37937   0
1                          3  28
f1_Score: 0.949153
```

```
FOR MATRIX: CONCAT XLS
Predicted Packet Type      0   1
Actual Packet Type
0                      37937   0
1                          3  28
f1_Score: 0.949153
```

Figure 4.2: This is the output from a run of the results of the SVM model from the LLDOS data set with all matrices producing the same F1 score.

All of the matrices had the same performance metric, however, during the projection, only 84 out of 143 features were used. This is beneficial for users to know as reducing the number of features can improve run time and the accuracy of the model. More confusion matrices can be viewed in the appendix C.1.

Chapter 5

Conclusion & Future Work

Our hypothesis is that with the enhanced data provided by RPCA, machine learning models will perform better than if given the data as-is. As machine learning models become more prevalent in day-to-day life, there will be a greater need to improve these models quickly, reliably, and possibly with limited data available. Our method provides relief to this area, and the ability to be utilized among virtually any model and data set.

There are many suggestions for improvements that could be made on top of our work. Applying the method to other machine learning models, other data sets, or adding or removing pre-processing techniques are just some. One of the more zealous advancements is to create an equation or algorithm for finding an optimal lambda for RPCA. Another possible area would be to try the method with image data sets. With our technique being very malleable there are many options for future work.

Appendix A

Data Set Specifics

*Note: The Label feature in all tables is not used in the context of creating the **X** matrix of data being performed on. It is used for creating the vector **y** for evaluation.

A.1 UNB Features Table

Order	**Feature**	**Pre-process**
1	Flow ID	Not Used
2	Source IP Address	Not Used
3 & 5	Source/Destination Port	One-Hot
4	Destination IP Address	One-Hot
6	Protocol	One-Hot
7	Timestamp	Not Used
8	Flow Duration	Z-transform
9 & 10	Total Fwd/Bwd Packets	Z-transform
11 & 12	Total Length Fwd/Bwd Packets	Z-transform
13-20	Fwd/Bwd Packet Length Max/Min/Mean/Std	Z-transform

21	Flow Bytes/s	Z-transform
22	Flow Packets/s	Z-transform
23-36	Flow/Fwd/Bwd IAT Mean/Std/Max/Min & (Fwd/Bwd Total)	Z-transform
37-40	Fwd/Bwd PSH/URG Flags	One-Hot
41 & 42	Fwd/Bwd Header Length	Z-transform
43 & 44	Fwd/Bwd Packets/s	Z-transform
45-49	Packet Length Min/Max/Mean/Std/Variance	Z-transform
50-57	FIN/SYN/RST/PSH/ACK/URG/CWE/ECE Flag Count	One-Hot
58	Down/Up Ratio	Z-transform
59	Average Packet Size	Z-transform
60 & 61	Avg Fwd/Bwd Segment Size	Z-transform
62	Fwd Header Length	Z-transform
63-68	Fwd/Bwd Avg (Bytes/Bulk) / (Packets/Bulk) / (Bulk Rate)	Z-transform
69-72	Subflow Fwd/Bwd Packets / Bytes	Z-transform
73 & 74	Init_Win_bytes_Forward/Backward	Z-transform
75	act_data_pkt_fwd	Z-transform
76	min_seg_size_forward	Z-transform
77-80	Active Mean/Std/Max/Min	Z-transform
81-84	Idle Mean/Std/Max/Min	Z-transform
85	Label	Not Used*

Table A.1: List of features and how they were encoded.

A.2 LLS DDOS Features Table

*Note: The Label feature is not used when creating the **X** matrix, but only for the **y** labels vector.

Encoding	**Feature**	**Pre-process**
0 or 1	Source IP Address	One-hot
0 or 1	Destination IP Address	One-Hot
0 or 1	Source Port	One-Hot
0 or 1	Ports below 1024	One-Hot
0 or 1	Ports above 1024	One-Hot
0 or 1	Missing source port number	One-Hot
0 or 1	Missing destination port number	One-Hot
0 or 1	Protocol	One-Hot
Numerical	Number of bytes in the packet	Z-transform
0 or 1	Label	Not Used*

Table A.2: Selected features from the PCAP files from the data set to keep consistent with feature list in [PKS18].

A.3 GOSSIS Features Table

*Note: The hospital_death feature is the label feature in this data set, and is not used when creating the **X** matrix, but only for the **y** labels vector.

Order	Feature Name	Pre-process
1	encounter_id	Not Used
2	patient_id	Not Used
3	hospital_id	Not Used
4	hospital_death	Not Used*
5	age	Z-transform
6	bmi	Z-transform
7	elective_surgery	Z-transform
8	ethnicity	One-Hot
9	gender	One-Hot
10	height	Z-transform
11	hospital_admit_source	One-Hot
12	icu_admit_source	One-Hot
13	icu_id	Not Used
14	icu_stay_type	One-Hot
15	icu_type	One-Hot
16	pre_icu_los_days	Z-transform
17	readmission_status	Not Used
18	weight	Z-transform
19	albumin_apache	Z-transform
20	apache_2_diagnosis	Z-transform

21	apache_3j_diagnosis	Z-transform
22	apache_post_operative	Z-transform
23	arf_apache	Z-transform
24	bilirubin_apache	Z-transform
25	bun_apache	Z-transform
26	creatinine_apache	Z-transform
27	fio2_apache	Z-transform
28	gcs_eyes_apache	Z-transform
29	gcs_motor_apache	Z-transform
30	gcs_unable_apache	Z-transform
31	gcs_verbal_apache	Z-transform
32	glucose_apache	Z-transform
33	heart_rate_apache	Z-transform
34	hematocrit_apache	Z-transform
35	intubated_apache	Z-transform
36	map_apache	Z-transform
37	paco2_apache	Z-transform
38	paco2_for_ph_apache	Z-transform
39	pao2_apache	Z-transform
40	ph_apache	Z-transform
41	resprate_apache	Z-transform
42	sodium_apache	Z-transform
43	temp_apache	Z-transform
44	urineoutput_apache	Z-transform
45	ventilated_apache	Z-transform

46	wbc_apache	Z-transform
47	d1_diasbp_invasive_max	Z-transform
48	d1_diasbp_invasive_min	Z-transform
49	d1_diasbp_max	Z-transform
50	d1_diasbp_min	Z-transform
51	d1_diasbp_noninvasive_max	Z-transform
52	d1_diasbp_noninvasive_min	Z-transform
53	d1_heartrate_max	Z-transform
54	d1_heartrate_min	Z-transform
55	d1_mbp_invasive_max	Z-transform
56	d1_mbp_invasive_min	Z-transform
57	d1_mbp_max	Z-transform
58	d1_mbp_min	Z-transform
59	d1_mbp_noninvasive_max	Z-transform
60	d1_mbp_noninvasive_min	Z-transform
61	d1_resprate_max	Z-transform
62	d1_resprate_min	Z-transform
63	d1_spo2_max	Z-transform
64	d1_spo2_min	Z-transform
65	d1_sysbp_invasive_max	Z-transform
66	d1_sysbp_invasive_min	Z-transform
67	d1_sysbp_max	Z-transform
68	d1_sysbp_min	Z-transform
69	d1_sysbp_noninvasive_max	Z-transform
70	d1_sysbp_noninvasive_min	Z-transform

71	d1_temp_max	Z-transform
72	d1_temp_min	Z-transform
73	h1_diasbp_invasive_max	Z-transform
74	h1_diasbp_invasive_min	Z-transform
75	h1_diasbp_max	Z-transform
76	h1_diasbp_min	Z-transform
77	h1_diasbp_noninvasive_max	Z-transform
78	h1_diasbp_noninvasive_min	Z-transform
79	h1_heartrate_max	Z-transform
80	h1_heartrate_min	Z-transform
81	h1_mbp_invasive_max	Z-transform
82	h1_mbp_invasive_min	Z-transform
83	h1_mbp_max	Z-transform
84	h1_mbp_min	Z-transform
85	h1_mbp_noninvasive_max	Z-transform
86	h1_mbp_noninvasive_min	Z-transform
87	h1_resprate_max	Z-transform
88	h1_resprate_min	Z-transform
89	h1_spo2_max	Z-transform
90	h1_spo2_min	Z-transform
91	h1_sysbp_invasive_max	Z-transform
92	h1_sysbp_invasive_min	Z-transform
93	h1_sysbp_max	Z-transform
94	h1_sysbp_min	Z-transform
95	h1_sysbp_noninvasive_max	Z-transform

96	h1_sysbp_noninvasive_min	Z-transform
97	h1_temp_max	Z-transform
98	h1_temp_min	Z-transform
99	d1_albumin_max	Z-transform
100	d1_albumin_min	Z-transform
101	d1_bilirubin_max	Z-transform
102	d1_bilirubin_min	Z-transform
103	d1_bun_max	Z-transform
104	d1_bun_min	Z-transform
105	d1_calcium_max	Z-transform
106	d1_calcium_min	Z-transform
107	d1_creatinine_max	Z-transform
108	d1_creatinine_min	Z-transform
109	d1_glucose_max	Z-transform
110	d1_glucose_min	Z-transform
111	d1_hco3_max	Z-transform
112	d1_hco3_min	Z-transform
113	d1_hemaglobin_max	Z-transform
114	d1_hemaglobin_min	Z-transform
115	d1_hematocrit_max	Z-transform
116	d1_hematocrit_min	Z-transform
117	d1_inr_max	Z-transform
118	d1_inr_min	Z-transform
119	d1_lactate_max	Z-transform
120	d1_lactate_min	Z-transform

121	d1_platelets_max	Z-transform
122	d1_platelets_min	Z-transform
123	d1_potassium_max	Z-transform
124	d1_potassium_min	Z-transform
125	d1_sodium_max	Z-transform
126	d1_sodium_min	Z-transform
127	d1_wbc_max	Z-transform
128	d1_wbc_min	Z-transform
129	h1_albumin_max	Z-transform
130	h1_albumin_min	Z-transform
131	h1_bilirubin_max	Z-transform
132	h1_bilirubin_min	Z-transform
133	h1_bun_max	Z-transform
134	h1_bun_min	Z-transform
135	h1_calcium_max	Z-transform
136	h1_calcium_min	Z-transform
137	h1_creatinine_max	Z-transform
138	h1_creatinine_min	Z-transform
139	h1_glucose_max	Z-transform
140	h1_glucose_min	Z-transform
141	h1_hco3_max	Z-transform
142	h1_hco3_min	Z-transform
143	h1_hemaglobin_max	Z-transform
144	h1_hemaglobin_min	Z-transform
145	h1_hematocrit_max	Z-transform

146	h1_hematocrit_min	Z-transform
147	h1_inr_max	Z-transform
148	h1_inr_min	Z-transform
149	h1_lactate_max	Z-transform
150	h1_lactate_min	Z-transform
151	h1_platelets_max	Z-transform
152	h1_platelets_min	Z-transform
153	h1_potassium_max	Z-transform
154	h1_potassium_min	Z-transform
155	h1_sodium_max	Z-transform
156	h1_sodium_min	Z-transform
157	h1_wbc_max	Z-transform
158	h1_wbc_min	Z-transform
159	d1_arterial_pco2_max	Z-transform
160	d1_arterial_pco2_min	Z-transform
161	d1_arterial_ph_max	Z-transform
162	d1_arterial_ph_min	Z-transform
163	d1_arterial_po2_max	Z-transform
164	d1_arterial_po2_min	Z-transform
165	d1_pao2fio2ratio_max	Z-transform
166	d1_pao2fio2ratio_min	Z-transform
167	h1_arterial_pco2_max	Z-transform
168	h1_arterial_pco2_min	Z-transform
169	h1_arterial_ph_max	Z-transform
170	h1_arterial_ph_min	Z-transform

171	h1_arterial_po2_max	Z-transform
172	h1_arterial_po2_min	Z-transform
173	h1_pao2fio2ratio_max	Z-transform
174	h1_pao2fio2ratio_min	Z-transform
175	apache_4a_hospital_death_prob	Z-transform
176	apache_4a_icu_death_prob	Z-transform
177	aids	Z-transform
178	cirrhosis	Z-transform
179	diabetes_mellitus	Z-transform
180	hepatic_failure	Z-transform
181	immunosuppression	Z-transform
182	leukemia	Z-transform
183	lymphoma	Z-transform
184	solid_tumor_with_metastasis	Z-transform
185	apache_3j_bodysystem	One-Hot
186	apache_2_bodysystem	One-Hot

Table A.3: Complete list of features from the WiDS competition data set and how they were encoded.

Appendix B

Package Information

B.1 Configuration File

Table B.1 shows the parameters, in order, and their default values.

Parameter Name	**Value**	**Restraints**
lambdastartvalue	0.001	0 < n < 1
lambdaendvalue	0.1	0 < n < 1
lambdaincrvalue	0.001	0 < n < 1
csvfile	datasets/defaultDataset.csv	String; must end in .csv
labels	Label	String; name of label column
oneHot	None	String; must use , as delimiter
skip	None	String; must use , as delimiter
randomseed	0	n = -1 or any real number
samplesize	0	0 <= n < 1
ratiotraindata	1/3	0 < n and n + ratiovaliddata != 1
ratiovaliddata	1/3	0 < n and n + ratiotraindata != 1
mode	0	n = 0, 1, 2
models	all	String; See codes in appendix B.2
logfile	defaultRun	String

Table B.1: List of the DEFAULT configuration parameters where n is a possible value.

The parameters **lambdastartvalue**, **lambdaendvalue**, and **lambdaincrvalue** in the default configuration are all used to find an optimal lambda for the data set. Lambda is used to adjust the output of the RPCA algorithm As their names suggest, **lambdastartvalue** is the start value for lambda, **lambdaendvalue** is the end value, and **lambdaincrvalue** is the amount to increment the lambda value by on each iteration of the code.

The parameter **csvfile** defines the file path to and the file name of the csv file that is the data set to be used. This should be changed by the user on the first run.

The **onehot** parameter is a list of column names that need one hot encoding. Similarly, **skip** takes in the same type of input, and skips using the indexes in the data.

The **ratiotraindata** and **ratiovaliddata** define how large the training and testing portions of the data set should be, respectively. For example, with a data set of 100 rows, and the **ratiotraindata** and **ratiovaliddata** each set to 1/3, then the resulting training set size will be 33 rows, testing set size will be 33 rows, and the validation set size will be 34 rows.

The **mode** parameter can be set to either 0, 1, or 2. **mode** controls the operation for finding/using lambda. When equal to the default, 0, all lambda parameters are used to search the default range (0.001 to 0.1 incremented by 0.001) to find an optimal lambda. Setting the **mode** value to 1 is for when an optimal lambda has previously been found. It will only use **lambdastartvalue**. This mode will run the program once, which is ideal for development with the package and regular use. Lastly, setting it to 2 is used for plotting graphs of a distribution on a single lambda. Again, this only uses **lambdastartvalue**.

The **models** parameter defines which machine learning models to run. As the default, the package will run all models. For descriptions and a list of these models and their corresponding codes for use, see appendix B.2.

Finally, the **logfile** parameter is used to set the name of the .log file to be saved in the Logs folder under the project structure. If a file with this name does not exist in the Logs folder, a new one will automatically be made on runtime. If the file already exists, new log information will be appended to the file.

B.2 Machine Learning Models

B.2.1 Machine Learning Model Codes

Model	**Code**	**Parameters**
Decision Tree	dtree	random_state=0
Gradient Boosting	gb	
K-Nearest Neighbors	knn	
Logistic Regression	logreg	random_state=0, solver=‘lbfgs’, multi_class=‘multinomial’
(Gaussian) Naive Bayes	nb	
(PyTorch) Neural Network	pynn	
Random Forest	rf	max_depth=2, random_state=0
Support Vector Machine	svm	gamma=“scale”
Run all models	all	
Run custom model	(No code, leave empty)	

Table B.2: List of codes for running the corresponding machine learning models. If the Parameters column is empty for a model, we use the default parameters.

B.2.2 Decision Tree

This is one of the most simplistic machine learning models. It creates and uses a decision tree structure to perform its predictions, for example, like the one depicted in Figure B.1. The branches constitute the features in a given row of data while the leaves are the class labels with 0 being a normal packet, and 1 being an attack [SH77].

B.2.3 Gradient Boosting

This model typically combines multiple varieties of weak prediction models, like decision trees. It builds these models in stages, and allows for optimization of various differentiable loss functions [Fri01].

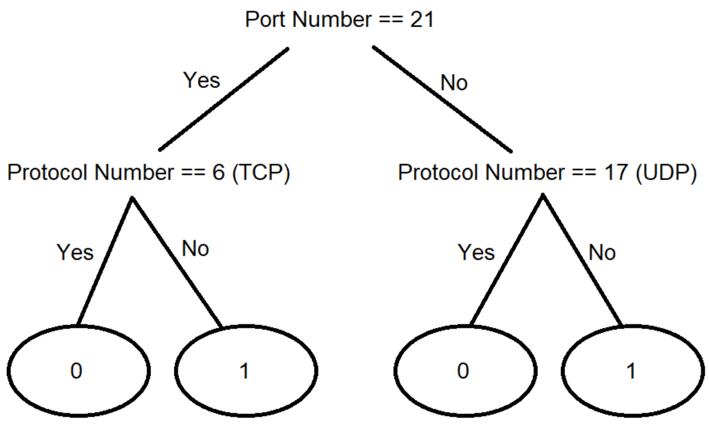


Figure B.1: In this figurative example, if a row of data contained a port number equal to 21, and a protocol number not equal to 6, it would be classified as an attack.

B.2.4 K-Nearest Neighbors

This model performs classification by grouping the k closest training examples in the feature space [Alt92]. For a row of data (a packet), based on the features in that row, it will be classified based on what other already placed packets are closest to it, and their classification. If a packet is nearest to a group of normal packets, it will be classified as normal.

B.2.5 Logistic Regression

This model uses a logistic function to classify data. Although this model is not traditionally meant for classification problems, it is able to achieve ways of classifying by using a cutoff value. This will assign the data points that lie above this value to one class, and the points below to another [KDG$^+$02].

B.2.6 Naive Bayes

This model uses a probabilistic classifier based on applying Bayes' Theorem [Joy03]. It uses naive independence assumptions between features in a row of data [R$^+$01].

B.2.7 Pytorch Neural Network

Pytorch is a framework that can be used to build neural networks [Ket17]. A neural network uses what are called neurons to create a network to classify data. An example of a simple network includes an input layer, a hidden layer, and an output layer, like in Figure B.2. The connections between the neurons are weighted, and an activation function controls the output.

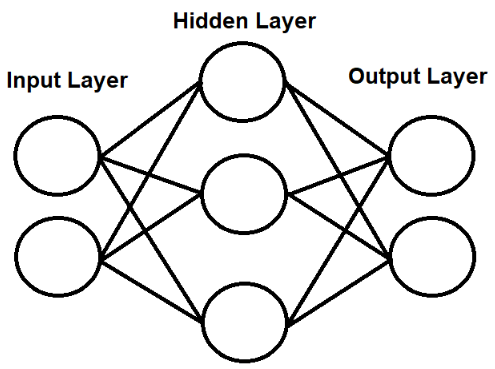


Figure B.2: An example of a what a simple neural network could look like. The lines between the neurons (circles) include weights.

B.2.8 Random Forest

This model constructs and employs multiple decision trees. When classifying a row of data, the model outputs the mode of the classes [LW$^+$02].

B.2.9 Support Vector Machine

This model maps the samples from the training data in space with as wide of a gap as possible for dividing the two classes (normal packets and attack packets). New samples (packets) get mapped in the same space based on their features, and are assigned a class based on which side of the gap they fall on [SV99].

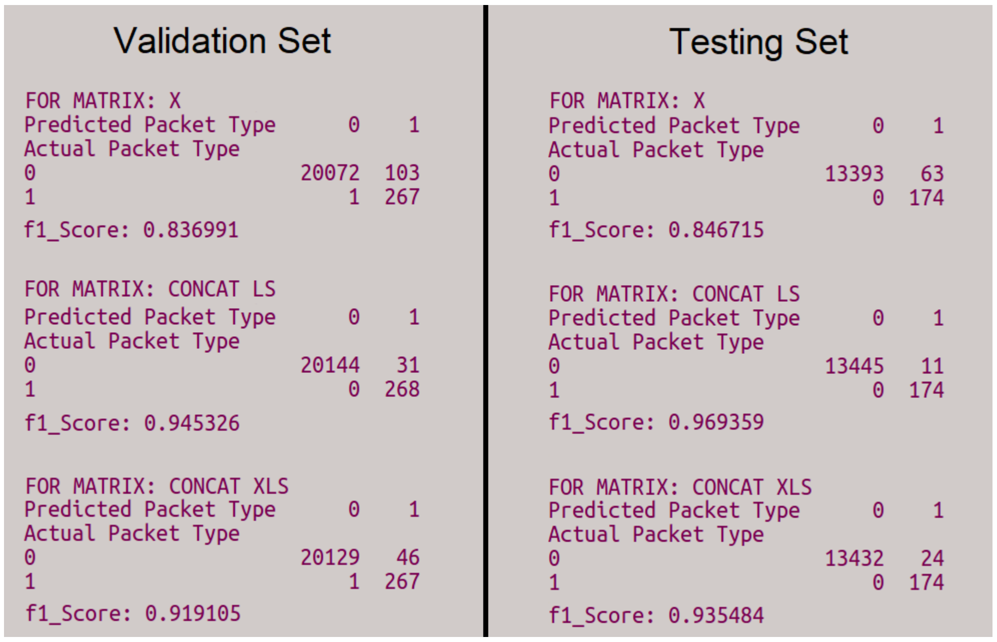
Appendix C

Supplemental Results

C.1 Confusion Matrices

Here, we show two runs of confusion matrices from the results of the UNB data set.

In Figure C.1, we show the results of a run from the K-Nearest Neighbors model.



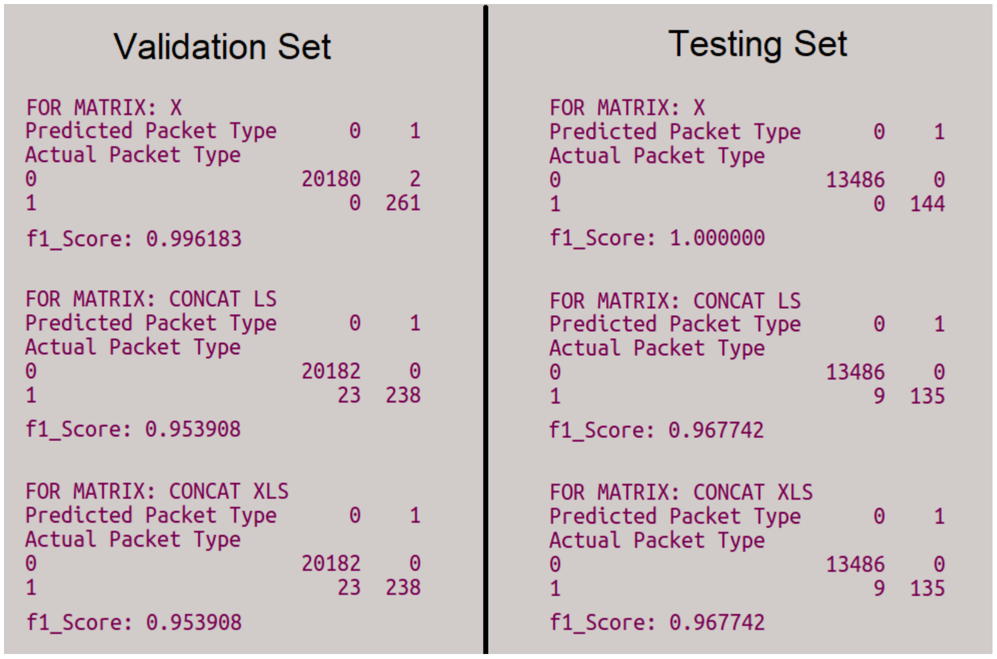
```
FOR MATRIX: X
Predicted Packet Type      0    1
Actual Packet Type
0                      20072  103
1                          1  267
f1_Score: 0.836991

FOR MATRIX: CONCAT LS
Predicted Packet Type      0    1
Actual Packet Type
0                      20144   31
1                          0  268
f1_Score: 0.945326

FOR MATRIX: CONCAT XLS
Predicted Packet Type      0    1
Actual Packet Type
0                      20129   46
1                          1  267
f1_Score: 0.919105
```

```
FOR MATRIX: X
Predicted Packet Type      0    1
Actual Packet Type
0                      13393   63
1                          0  174
f1_Score: 0.846715

FOR MATRIX: CONCAT LS
Predicted Packet Type      0    1
Actual Packet Type
0                      13445   11
1                          0  174
f1_Score: 0.969359

FOR MATRIX: CONCAT XLS
Predicted Packet Type      0    1
Actual Packet Type
0                      13432   24
1                          0  174
f1_Score: 0.935484
```

Figure C.1: This is the output from a run of the results of the KNN model from the UNB data set.

What is most interesting about these results, is that the F1 scores for all matrices, **X**, **LS**, and **XLS**, all improved by a small amount in the testing set. Then, looking at the confusion matrices themselves, we see that for the testing set (and mostly for the validation set as well), all were able to correctly classify the malicious packets, except there were varying degrees of how many benign packets were classified as malicious.

In Figure C.2, we show the results of a run from the Decision Tree model. If we remember back to Table 4.2, the dtree model performed the best when the **X** matrix was used. Here in the figure however, we show that there were some runs where the feature engineered matrices perform almost as well as **X**. And, even still, all matrices improved their F1 scores in the testing set.



```
FOR MATRIX: X
Predicted Packet Type      0    1
Actual Packet Type
0                      20180    2
1                          0  261
f1_Score: 0.996183

FOR MATRIX: CONCAT LS
Predicted Packet Type      0    1
Actual Packet Type
0                      20182    0
1                         23  238
f1_Score: 0.953908

FOR MATRIX: CONCAT XLS
Predicted Packet Type      0    1
Actual Packet Type
0                      20182    0
1                         23  238
f1_Score: 0.953908
```

```
FOR MATRIX: X
Predicted Packet Type      0    1
Actual Packet Type
0                      13486    0
1                          0  144
f1_Score: 1.000000

FOR MATRIX: CONCAT LS
Predicted Packet Type      0    1
Actual Packet Type
0                      13486    0
1                          9  135
f1_Score: 0.967742

FOR MATRIX: CONCAT XLS
Predicted Packet Type      0    1
Actual Packet Type
0                      13486    0
1                          9  135
f1_Score: 0.967742
```

Figure C.2: This is the output from a run of the results of the Decision Tree model from the UNB data set.

Bibliography

[ACJR11] Shane Amante, B Carpenter, Sheng Jiang, and J Rajahalme. Ipv6 flow label specification. *Network Working Group Request for Comments*, 6437, 2011.

[Alt92] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[BG15] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[BHPT06] Eric Bair, Trevor Hastie, Debashis Paul, and Robert Tibshirani. Prediction by supervised principal components. *Journal of the American Statistical Association*, 101(473):119–137, 2006.

[CLMW11] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[Cuk18] Michel Cukier. Study: Hackers attack every 39 seconds. *URL https://eng. umd. edu/news/story/study-hackers-attack-every-39-seconds*, 2018.

[DZHZ06] Chris Ding, Ding Zhou, Xiaofeng He, and Hongyuan Zha. R 1-pca: rotational invariant l 1-norm principal component analysis for robust subspace factorization. In *Proceedings of the 23rd international conference on Machine learning*, pages 281–288, 2006.

[EY07] Burcu Erkmen and Tülay Yildirim. *Improving classification performance of sonar targets by applying general regression neural network with PCA*. Elsevier BV, Jul 2007.

[Fab17] Maria I Fabre. *International comparison of critically ill patients*. PhD thesis, Massachusetts Institute of Technology, 2017.

[Fan51] Ky Fan. Maximum properties and inequalities for the eigenvalues of completely continuous operators. *Proceedings of the National Academy of Sciences of the United States of America*, 37(11):760, 1951.

[Fir20] Inc FireEye. M-trends 2020, 2020.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[GDAD08] Samanwoy Ghosh-Dastidar, Hojjat Adeli, and Nahid Dadmehr. Principal component analysis-enhanced cosine radial basis function neural network for robust epilepsy and seizure detection, Jan 2008.

[Joy03] James Joyce. Bayes' theorem. 2003.

[KDG+02] David G Kleinbaum, K Dietz, M Gail, Mitchel Klein, and Mitchell Klein. *Logistic regression*. Springer, 2002.

[Ket17] Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.

[LW+02] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[Nar19] Sarang Narkhede. Understanding confusion matrix, Aug 2019.

[ORB06] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[PKS18] Randy Paffenroth, Kathleen Kay, and Les Servi. Robust pca for anomaly detection in cyber networks, Jan 2018.

[R+01] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[SH77] Philip H Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977.

[SS02] Wang Song and Xia Shaowei. Robust pca based on neural networks, Aug 2002.

[SV99] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[THLL09] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion detection by machine learning: A review. *expert systems with applications*, 36(10):11994–12000, 2009.

[VR79] Cornelis Joost Van Rijsbergen. Information retrieval. 1979.

[YJ13] SLP Yasakethu and J Jiang. Intrusion detection via machine learning for scada system protection. In *1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013) 1*, pages 101–105, 2013.

[ZWN+09] Yudong Zhang, Lenan Wu, Nabil Neggaz, Shuihua Wang, and Geng Wei. Remote-sensing image classification based on an improved probabilistic neural network, Sep 2009.