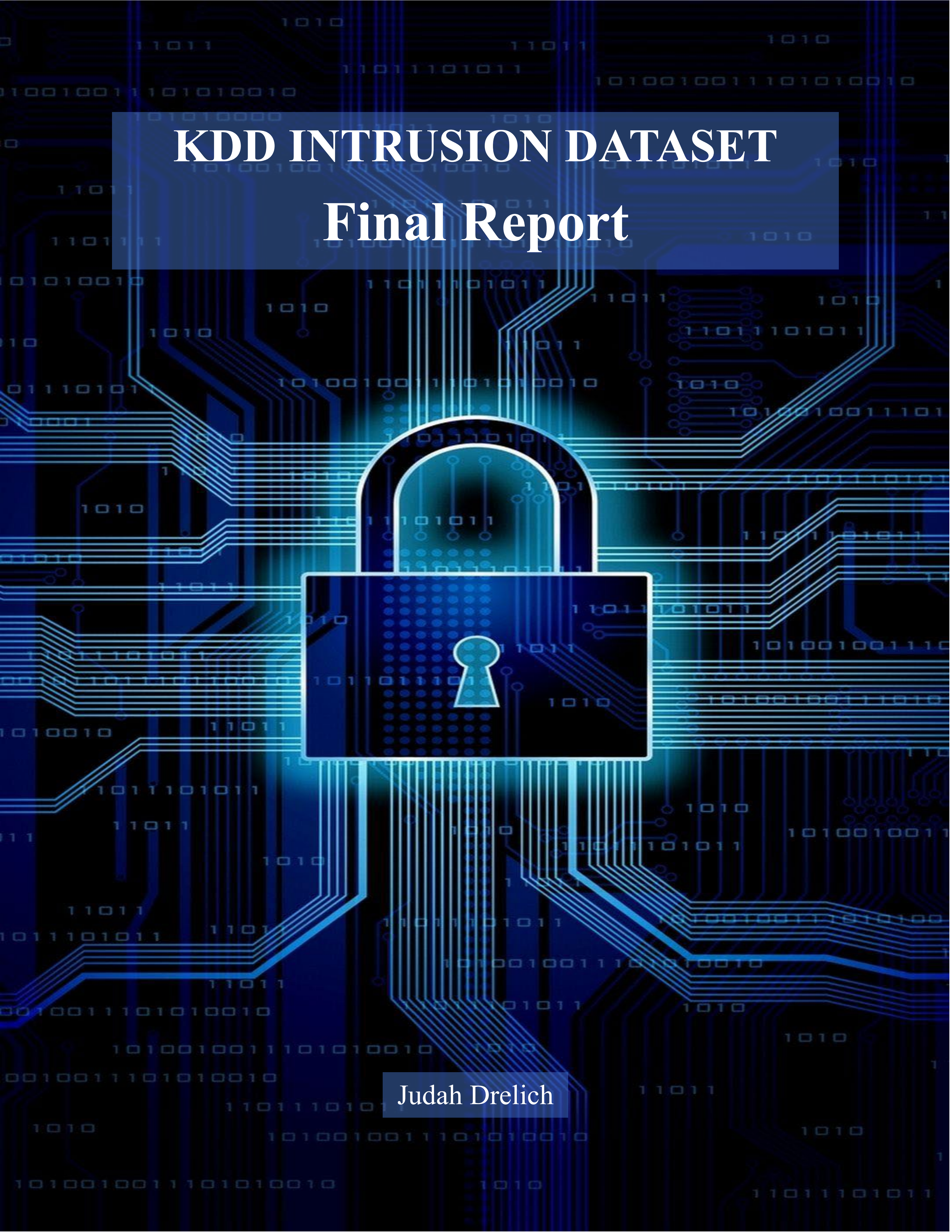
KDD INTRUSION DATASET

Final Report



Judah Drelich

Abstract

Intrusion Detection Systems (IDS) are widely used to protect private and public organizations from malicious hacks. There are two capacities that an IDS should have, the first is to find blindspots in the network that could precipitate an attack, while the second is to build a model that could predict when an intrusion is happening. In 1998, the DARPA Intrusion Detection Program created one of the few publicly available datasets that has enough information to build an IDS. The purpose of this project was to build an effective IDS from the DARPA dataset. I accomplished the first part of an IDS by clustering the observations and analyzing the likelihood that a given cluster would hold an attack. For the predictive part, I built both a Neural Network model and a Random Forest classifier that perfomed significantly better than a trivial classifier. However there were still some intrusions such as snmp attacks that the model was unable to identify. I tried to come up with new ideas for how to detect these intrusion types but my attempts were unsuccessful and so it still remains an open question.

Introduction

Cyber security is a major issue that is going to get worse. Global cyber crime costs are expected to grow 15% per year for the next 5 years reaching $10.5 trillion by 2025[1]. That would be close to a quarter of the entire world's GDP[2] and larger than the GDP of every country except the U.S. and China. From identity fraud to stealing financial assets or gathering important data, cybersecurity is a crucial industry to protect innocent people from immense harm.

[1] Cybercrime To Cost The World $10.5 Trillion Annually By 2025 (cybersecurityventures.com)

[2] Gross Domestic Product (GDP) - Worldometer (worldometers.info)

The dataset that I analyzed came from a 1998 DARPA Intrusion Detection Program managed by MIT's Lincoln Labs.[3] The data was created from TCP dump data from a simulated Air Force local area network (LAN). The data has 5 million connections, 41 features and a label for each observation that shows whether it was a safe connection or a certain type of intrusion.

Effective Intrusion Detection Systems (IDS) in cyber security are valuable from the government to the private sector. By filtering out the connections that are less worrying while paying attention to more troublesome connections, I can give organizations a better chance of protecting themselves and catching criminals.

Data Wrangling

I separated the feature variables (X) from the target variable (y). With X, I converted the categorical columns into dummy columns. This will make it possible to analyze the entire dataset instead of just the numerical columns.

To get started I used the "ProfileReport" function from the ydata-profiling package to gain an overall sense of the data. "ProfileReport" has a lot of great preliminary explorations that show the state of the dataset. When I ran the function on the original dataset some things stood out.

- No missing values
- Constant features
- Skewed features:
- Large Multicollinearity
- High numbers of zeros:

[3] KDD-CUP-99 Task Description (uci.edu)

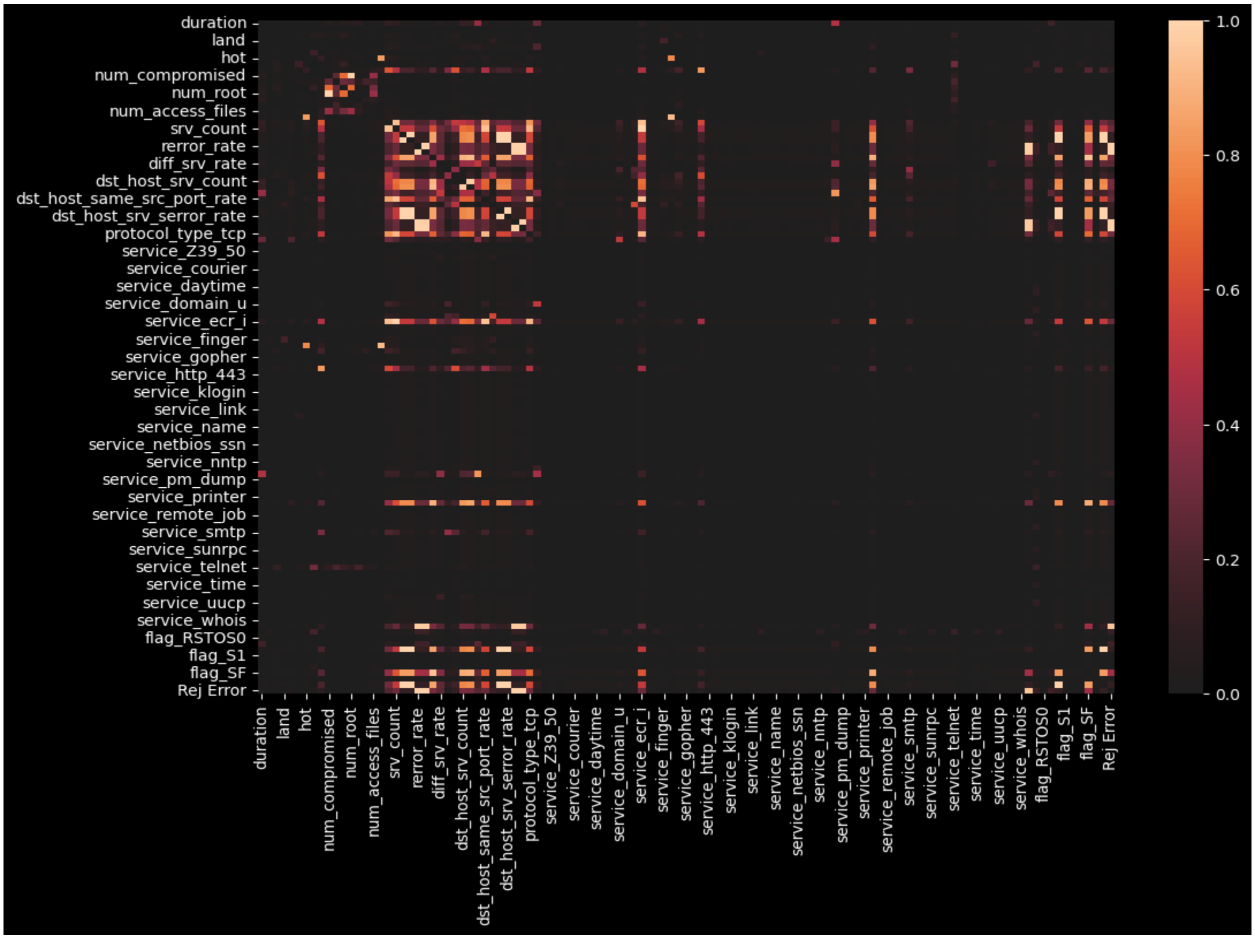
The first three items of the list are simple and easy to deal with. A dataset without missing values is great because I do not have to impute values or discard observations. Constant features are also straightforward because I can drop them entirely from the dataset since they provide no predictive ability or correlation with the target variable.

Skewed features impede a model's ability to accurately describe interactions between atypical features because the size of the highest values dwarfs every part of the calculations. To fix this, I scaled the features with a standard scaler to preserve the patterns of each feature while scaling down the largest values of the skewed features.

Multicollinearity

The most glaring problem from "ProfileReport" is the number of variables that have high correlations with each other. This is because a sizable number of the columns measure similar things. For instance, "srv_rerror_rate" and "dst_host_srv_rerror_rate" both measure the rate at which a server will deny a connection request. The former is recording only the last two seconds while the latter is using the last hundred connections. While both features have slightly different information, the information is so similar that it is impossible to leave both features in the dataset. The challenge now is to go through the dataset and try to find all instances of similar features and either combine them or discard them.

Below is the starting point of the multicollinearity for the entire dataset.

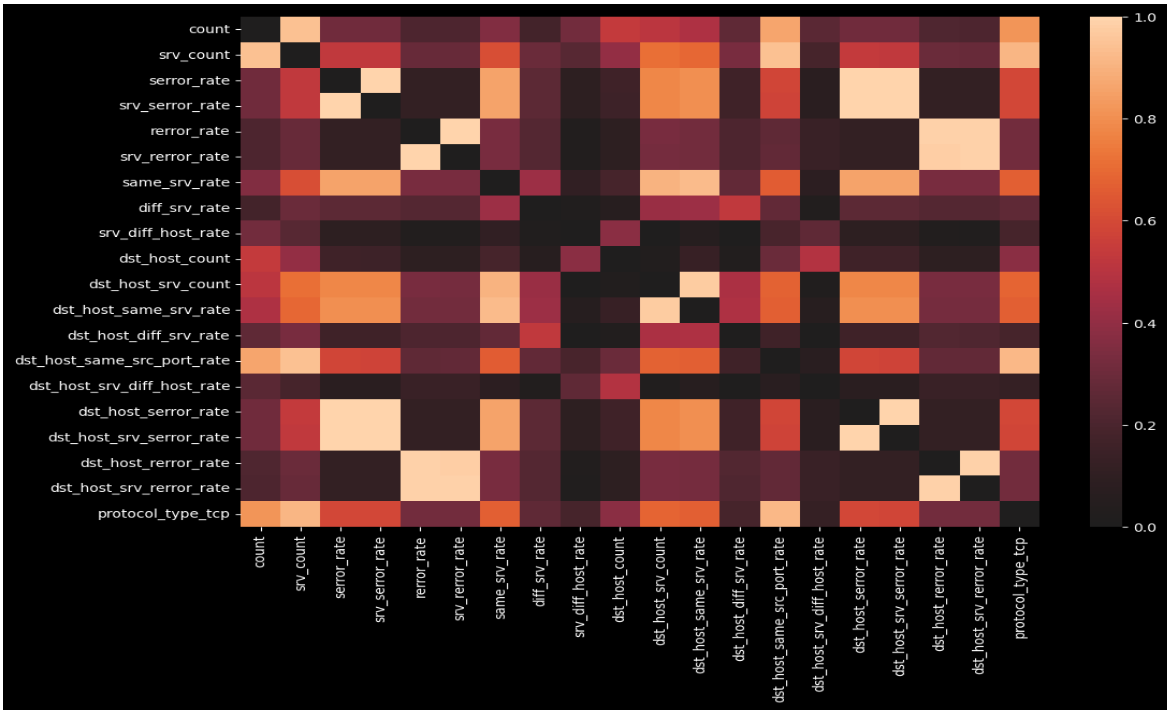


While it may look like most of the data does not have a multicollinearity problem, the dark part of the graph holds all the dummy variables from the categorical columns. Because I dropped the first value from each of the categorical columns, it is mathematically impossible for those dummy columns to be collinear.

The heatmap shows a concerning square from features 15 to 39. While there are other problem areas, if I can clean up that square, many of the other collinearities will dissipate.

Cleaning up the collinearity is not easy. While it may seem tempting to simply drop all the problematic columns, there is too much information in those columns to discard them.

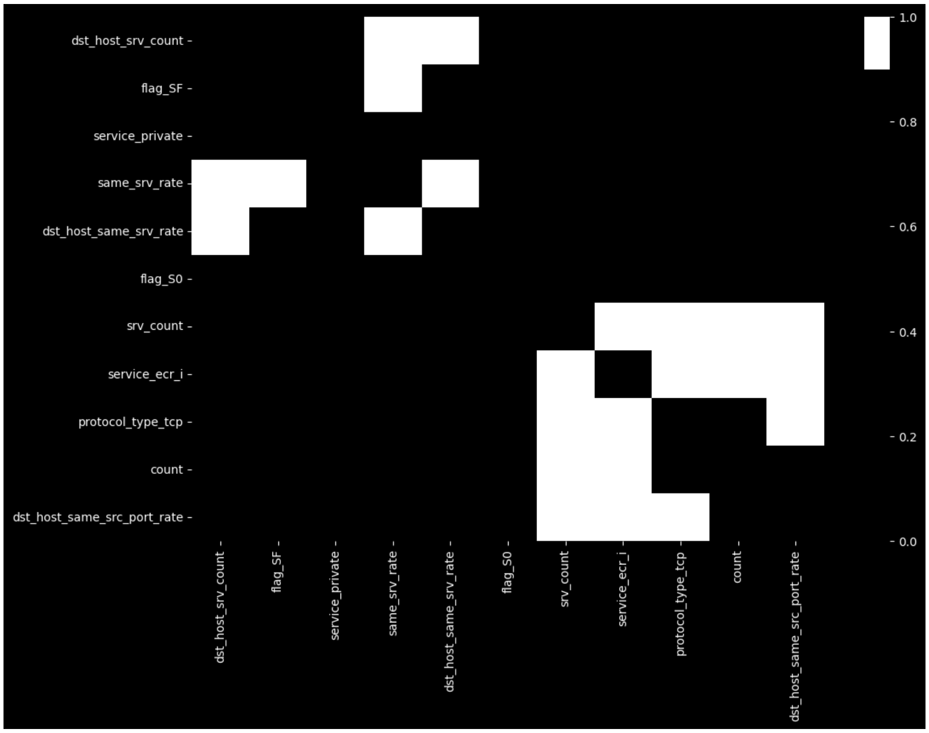
The first step was to zoom in on bright spot of the heatmap to see if there were any groupings of features that had clear real-world explanations for their high correlations. This produced the following graph:



In the heatmap above "serror" or "rerror" account for four variables each that create boxes of prominent levels of correlation. Given their clear real-world connection, I combined each grouping of the four variables into 'Syn Error' and 'Rej Error' by taking the maximum value of those four variables for each observation. That way if there is a spike in one of the four variables, then Syn and Rej Error will spike while if all four of the original variables are low, then the combined feature will be low as well.

Without an obvious next step from the heatmap above, I then created a dictionary with the features as keys and the number of correlations with other features over 0.7 as the values. The dictionary's largest value was eight collinearities followed by six from 'Syn Error' which is the synthetic feature from the "serror" variables. By evaluating 'Syn Error' last, I can try to preserve it while ending multicollinearity.

I then created a binary heatmap without ‘Syn Error’ where all the boxes of correlations that were greater than 0.9 were white and the rest of the graph is black:



From the graph above, I saw two groupings of features that need to be combined or dropped.

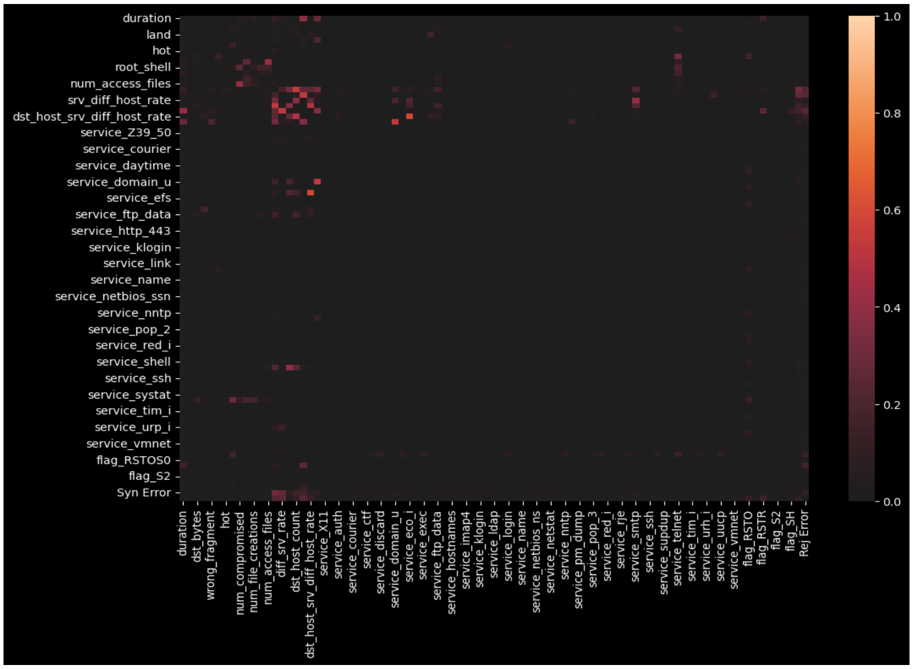
- Top left: ['dst_host_srv_count', 'same_srv_rate', 'dst_host_same_srv_rate']
- Bottom right corner: ['srv_count','service_ecr_i', 'dst_host_same_src_port_rate']

The top left group should be combined because the variables in it are measuring connections to the same server in different ways. Similar to the Syn and Rej error I took the max of 'same_srv_rate', 'dst_host_same_srv_rate'. Since there is no way to combine 'dst_host_srv_count' and it is highly collinear I will drop it. For the bottom right there is no clear thread tying the three together. I decided to keep 'protocol_type_tcp' because the other two have a massive correlations with count. If I keep 'protocol_type_tcp' then I can keep 'count'.

I continued to iterate through the dataset with the same process where I will create a dictionary to see most collinear features. I then look to see if there is any connection between the variables that would justify combining them instead of dropping them. If there is not a clear similarity, then I drop columns in order of importance. The order is defined as:

1) Combinations of Original Columns:
    a. ‘Syn Error’, ‘Rej Error’, ‘srv_rate'
2) Numerical Original Columns
    a. ‘count’, ‘src_bytes’
3) Binary Original Columns
    a. ‘su_attempted’, ‘is_guest_login’
4) Dummy Variables from the Categorical Variables
    a. 'protocol_type_tcp', ‘flag_SF’

The more information a column holds, the higher the priority is for it to stay in the dataset. When I finally dropped all the collinear columns, the heatmap of collinearity looked like this.



It is a significant improvement over the start of the process that had a massive bright square in the top left quadrant.

To make sure that I had fully ended the multicollinearity I calculated every remaining feature's Variance Inflation Factor (VIF). The formula for VIF is:

$$VIF = \frac{1}{1-R^2} \tag{1}$$

Where R^2 is the R-squared value that represents correlations.[4] When R^2 approaches zero, the VIF approaches one whereas when R^2 approaches 1, the VIF approaches infinity. The guidelines for an acceptable VIF score can range from 2.5 to 10 although anything over five is suspicious.[5] With the remaining features that I had, 87 out of the 89 had a VIF of under 2.5 and 2 had a VIF < 3.03.

Lasso Regularization

Lasso Regularization is a technique for variable selection that uses regression to evaluate the effect that features have on a target variable. The idea is to add a penalty term to the regression loss function that has the coefficient or slope of the independent variable with respect to the target variable, multiplied by a parameter α.

$$Loss(\beta) = SSD + \alpha|\beta| \tag{2}$$

[4] How to Calculate VIF in Excel - Sheetaki
[5] Variance Inflation Factors (VIFs) - Statistics By Jim

Where β is the coefficient of the feature, *SSD* is the sum of squared distances of the observations to the regression line and α is the penalty's parameter. If there are many features, then we generalize equation (2) by summing the coefficients:

$$Loss(\beta_1, \dots \beta_n) = SSD + \alpha \textstyle\sum_{i=1}^{n} |\beta_i| \qquad (3)$$

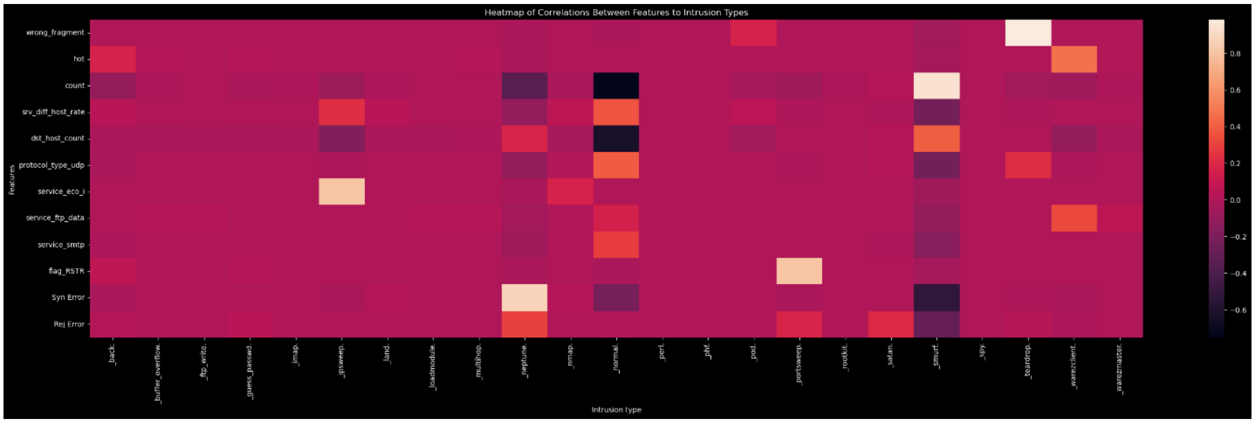
Lasso aims to reduce the loss in equation (3). If a feature is not important, then changing its slope will not move the regression line close enough to the data points to decrease the loss function with a non-zero value. If a feature is important, the regression line will move towards the data and minimize the SSD faster than it increases the penalty term. Since only features with non-zero coefficients are meaningful, I can discard all the features with a coefficient of zero.

After running the Lasso Regularization with the target variable being a binary choice between attack and no attack, twelve out of the eighty-nine features have non-zero coefficients. By selecting out these features I have created a small dataset. The problem with the small dataset is that it is possible that it is missing some of the features that would distinguish between different attack types. To be safe, I iterated through all the different intrusion types using lasso regularization to see if there were other notable features that I had missed in smaller dataset. That process yielded sixty-one features and became a bigger data frame.

Exploratory Data Analysis

Small Data

The simplest way to explore this data was to create a correlation heatmap between the twelve features and all the different intrusion types.



This graph shows that the features hold predictive power, but that I need to do a lot more analysis to find it. To do that I can create clusters in the data that will be able to show how multiple features can come together to create an attack.

One problem with the heatmap is that most of the correlations are hovering around zero. This is because the frequency of the intrusion types varies wildly. Below is a list of the most common intrusions and the percentage of observations in the dataset.

Smurf.	56.8378%
Neptune.	21.6997%
Normal.	19.6909%

The most common intrusion types make up over 98% of the dataset which is why much of the heatmap has little to no correlation.

Clustering

The diverse types of intrusions clearly fall into separate groups. On the original challenge's webpage the contest makers list four distinct groups:

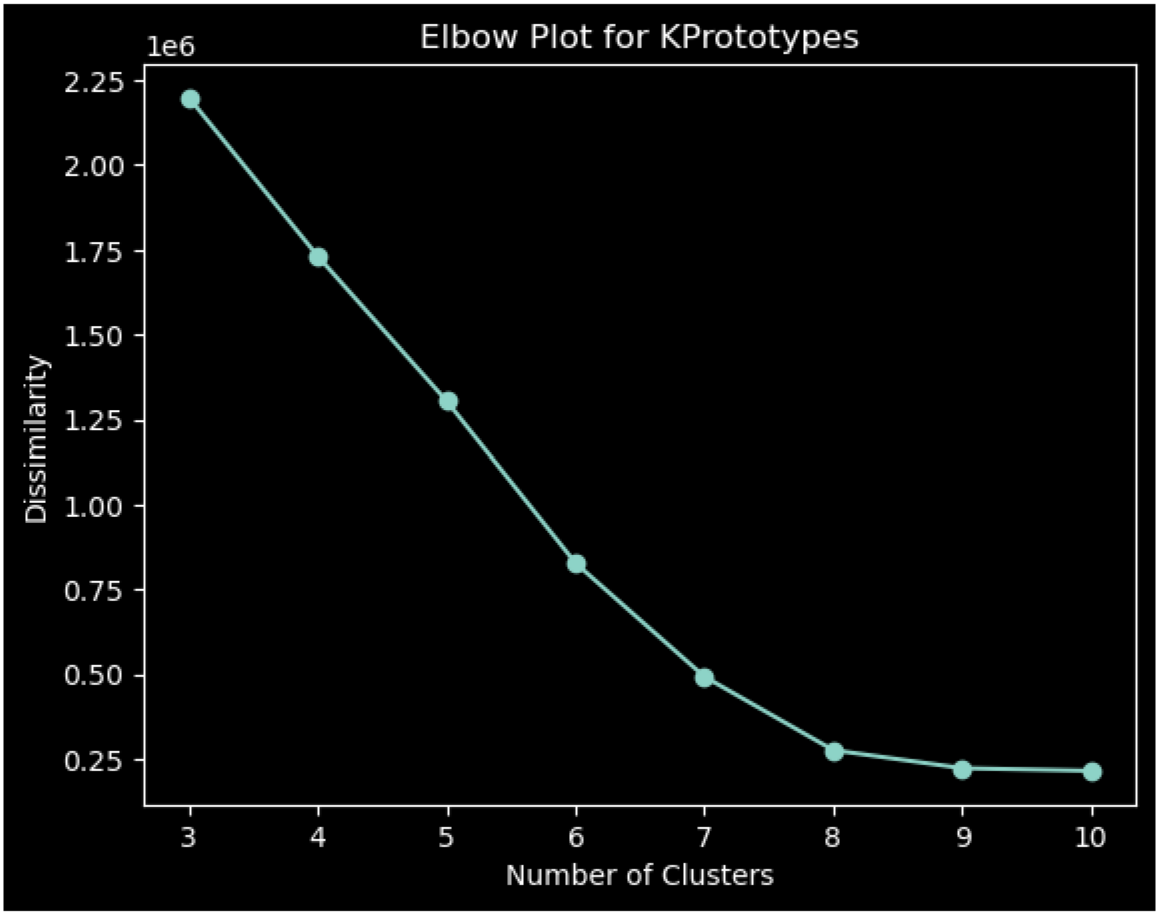
- DOS: denial-of-service, e.g., syn flood.
- R2L: unauthorized access from a remote machine, e.g., guessing password
- U2R: unauthorized access to local superuser (root) privileges, e.g., various `buffer overflow' attacks.
- probing: surveillance and other probing, e.g., port scanning.

These groupings make sense. For example, portsweep and ipsweep are clearly in the probing group as they are types of sweeps that search for vulnerabilities.

But it is foolish to group intrusions of the same kind together. A hacker could use ipsweep as a first step in a ping of death (pod) attack, while rarely ever using an ipsweep and portsweep in the same attack together. Without clear information on how to group the different intrusions, we need to cluster the data to supply a more rigorous analysis.

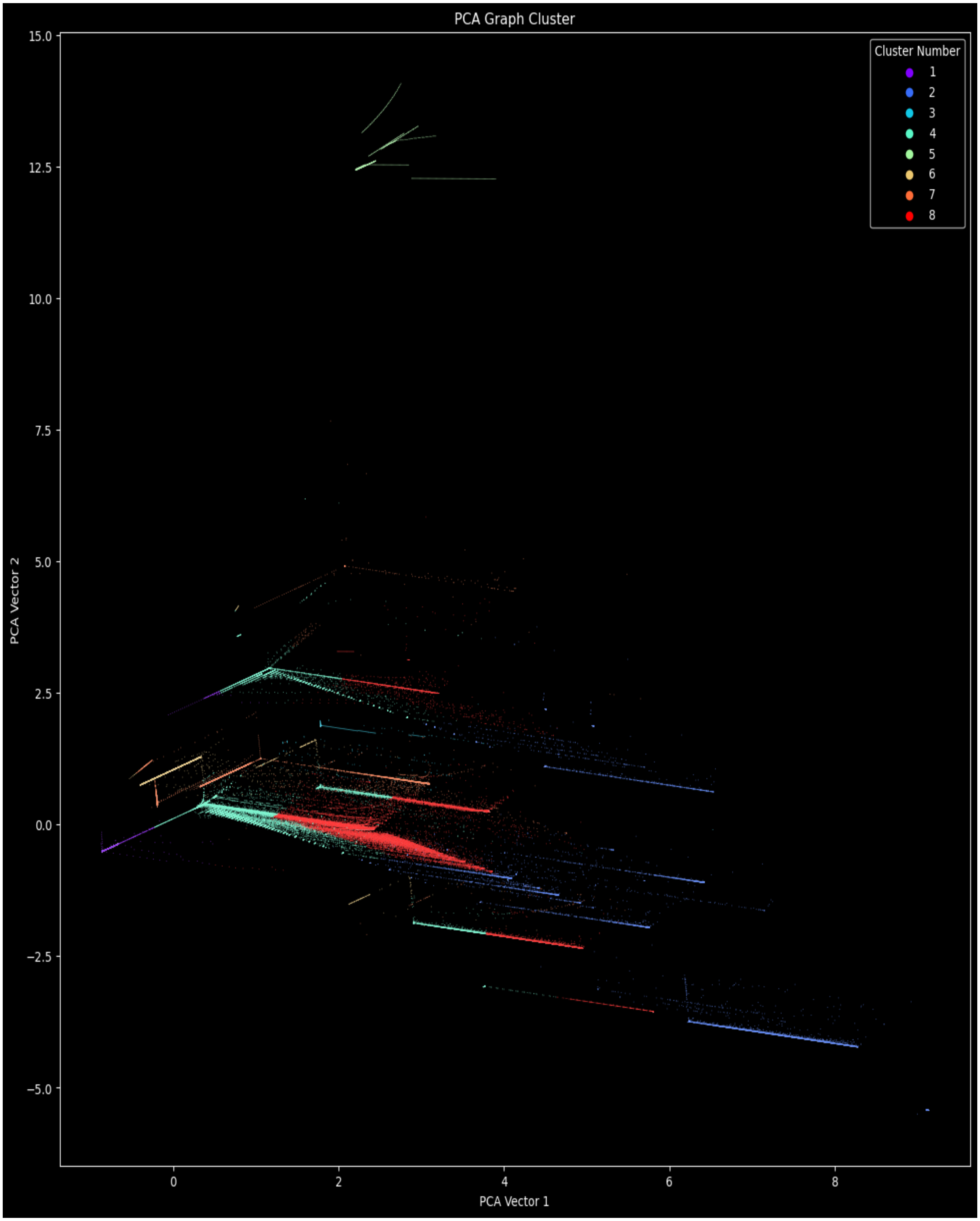
K-Prototype is a clustering algorithm that is like K-means, but it takes both numerical and categorical data. While none of the remaining columns were technically categorical, the dummy columns still have the same properties as categorical data. I thought that the best algorithm would be one that supports the dummy columns as well as the numerical columns.

To find the best number of clusters, I graphed the loss function relative to each value of K.

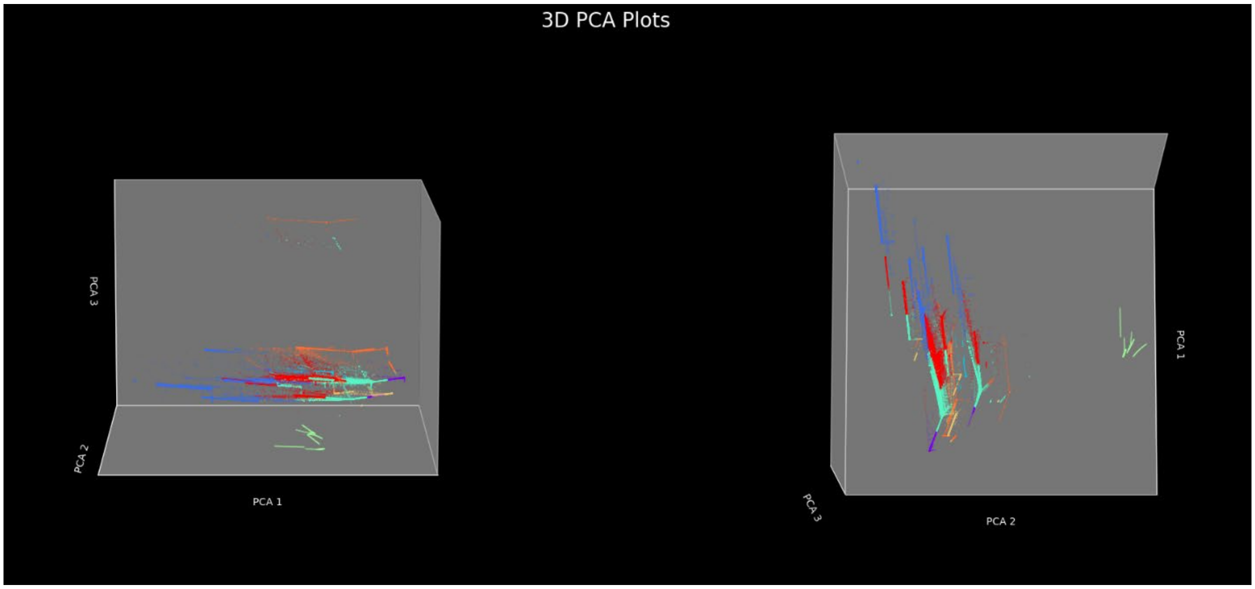


From this elbow plot the best value is K = 8.

The first way to try to explore the clusters is finding a way to look at them. If they look like they form a clear structure, then it is another piece of evidence for their existence. Because the clusters themselves are in twelve dimensions, we need a way to reduce their dimensionality to a number that we can see. Principle Component Analysis (PCA) is a method to reduce dimensionality by taking the features with the highest variances and combining them into a smaller number of dimensions. We can take the twelve features and combine them into two components and then graph those components to see the structure of the data.



This graph shows that there is a clear structure in the data, but it also gives the impression that it would be in three dimensions. Below is a three-dimensional version of the graph above:



The 3-D version gives a much better view of the clusters than the 2-D version, but it is not a fully correct representation of the clusters. The K-Prototype algorithm has different weightings for

each of the datapoints and has zero information loss. PCA may look nice, but it is not a complete picture. This means that I need a more rigorous way of confirming the existence of the clusters.

Cluster Analysis

The most straightforward way to evaluate the structure is a silhouette score which measures the distance or dissimilarity from other cluster's centroids (center of the cluster) compared to the dissimilarity to its own centroid. The closer a point is to its own centroid, while being far away from other centroids, the clearer it is that the structure is clustered. Unfortunately, when I ran the computation in R, the dataset was so large that the dissimilarity matrix (store of all the distances) returned a 900+ GB vector. That is far outside the ability of my computer or any normal commercial computer to manage. Even if I were to use just a tenth of the dataset for a representative sample, it would still generate a 9+ GB vector. As such I must use different algorithms that are not as direct but can still lend some insight into validity of the clustering structure.

The algorithms that I chose are the:

Adjusted Rand Index (ARI)

$$ARI(true, new) \ = \ ADJ(\frac{MatchingPairs}{TotalPairs}) \qquad (4)$$

- Where true is the "true value of the clusters,
- new is the estimated value of the clusters.
- A pair is each combination of cluster labels from the true, and new datasets,
- ADJ is the adjustment, it corrects for the number of pairs that will match out of pure chance.

Normalized Mutual Index (NMI)

$$NMI = \frac{2 * I(true, new)}{H(true) + H(new)} \qquad (5)$$

- true: ibid
- new: ibid
- $I(true, new)$ is a function that measures the commonality of the two cluster sets.
- $H(true)$ and $H(new)$) are the entropy of the cluster labels. More simply it measures the distribution. of clusters over the data. The more evenly distributed the clusters, the higher the entropy.

Fowlkes-Mallow Index (FMI)

$$FMI = \frac{TP}{\sqrt{(TP+FP)(TP+FN)}} \tag{6}$$

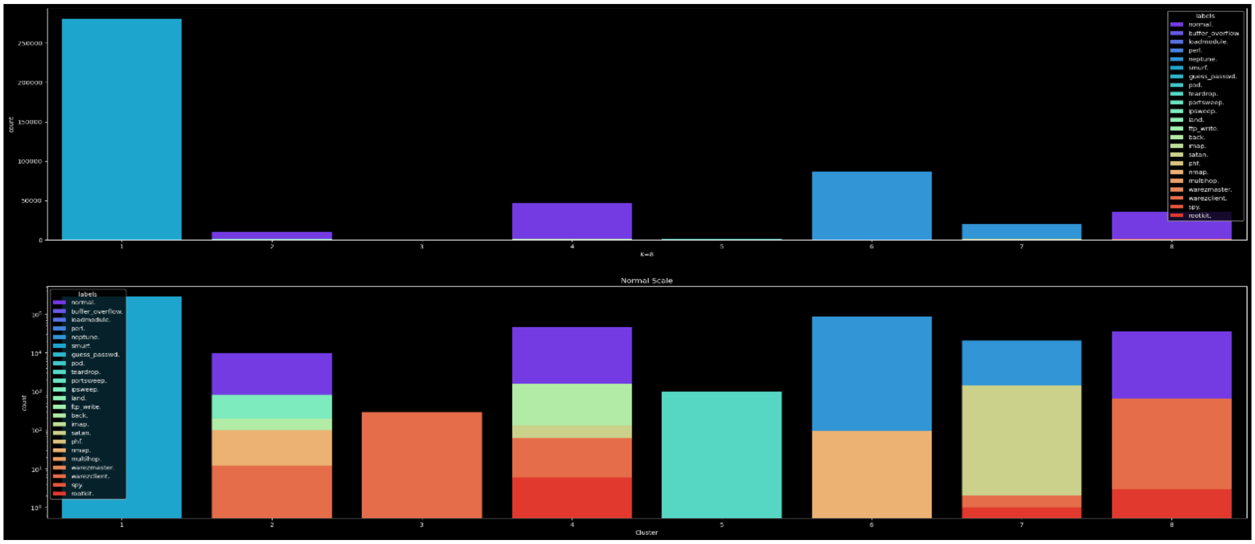
- TP is the number of pairs that have the same cluster label as the first set of labels,
- FP is the number of pairs that have the same cluster label as the second set of labels.

The ARI measures the similarity between two sets of labels on a scale of -1 to 1 where negative values show clear difference and one is a perfect match. NMI and FMI are from 0 to 1 and measure similarity. To assess the structure of this dataset, I can take a representative sample, run the clustering algorithm on the sample and then compare the sample's clusters to the full dataset's labels. If there are no clusters and the data is random, then I will get two vastly different sets of labels that will have low similarity scores. I do not have a clear threshold to decide if there are clusters in the data but the closer the scores are to one, the higher the likelihood that the data has a clustered structure.

After running the three tests I got these three results.

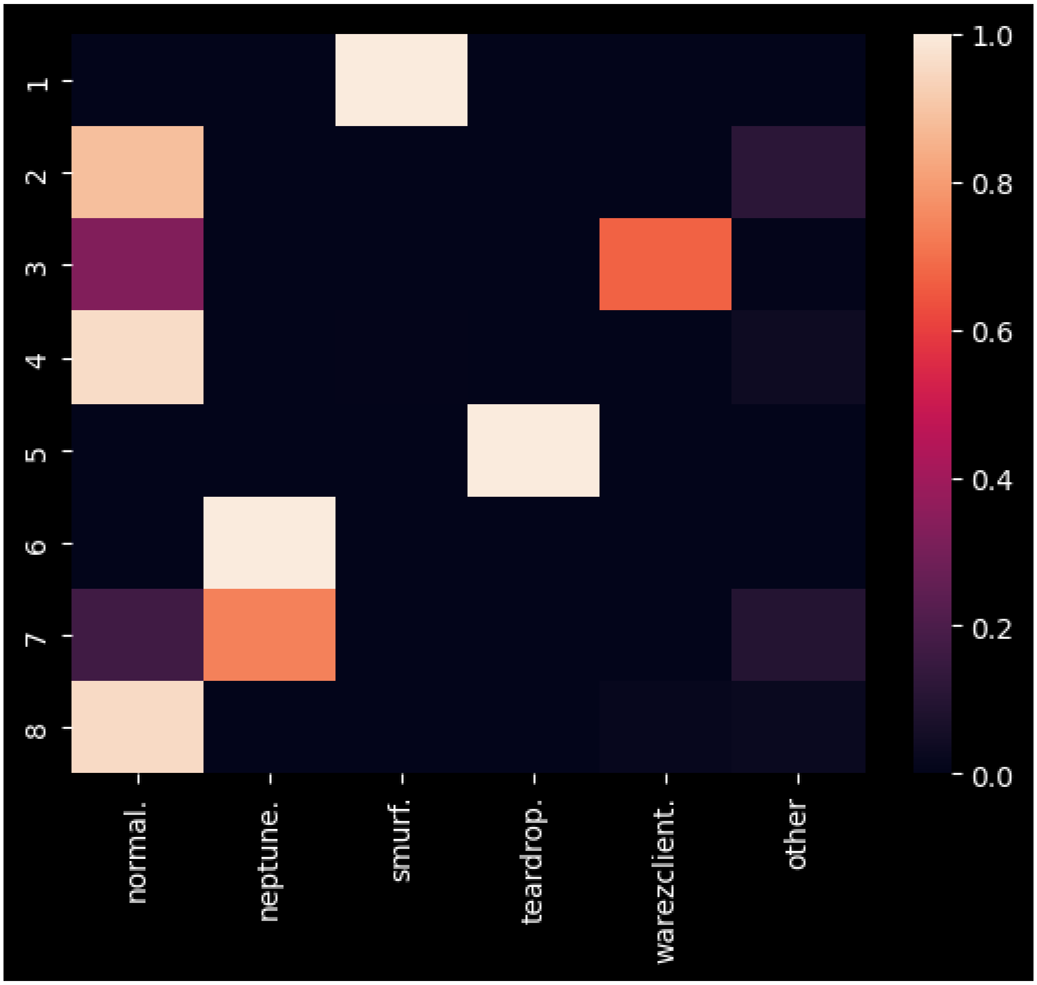
Adjusted Rand Index: 0.99997   Normalized Mutual Index: 0.9997   Fowlkes Mallow Index: 0.99998

Which strongly confirms the existence of clusters. With the clusters confirmed, I plotted a



histogram of the different clusters colored by intrusion type. The top graph has a normal y-axis while the bottom graph's y-axis is a logarithmic scale. These two histograms show the power of the clusters. In the first histogram, we can see that clusters 2, 4 and 8 are broadly safe connections. The second graph shows that the non-common intrusions will be harder to predict.

Next, we can make a heatmap with the odds that an attack will happen in each cluster.



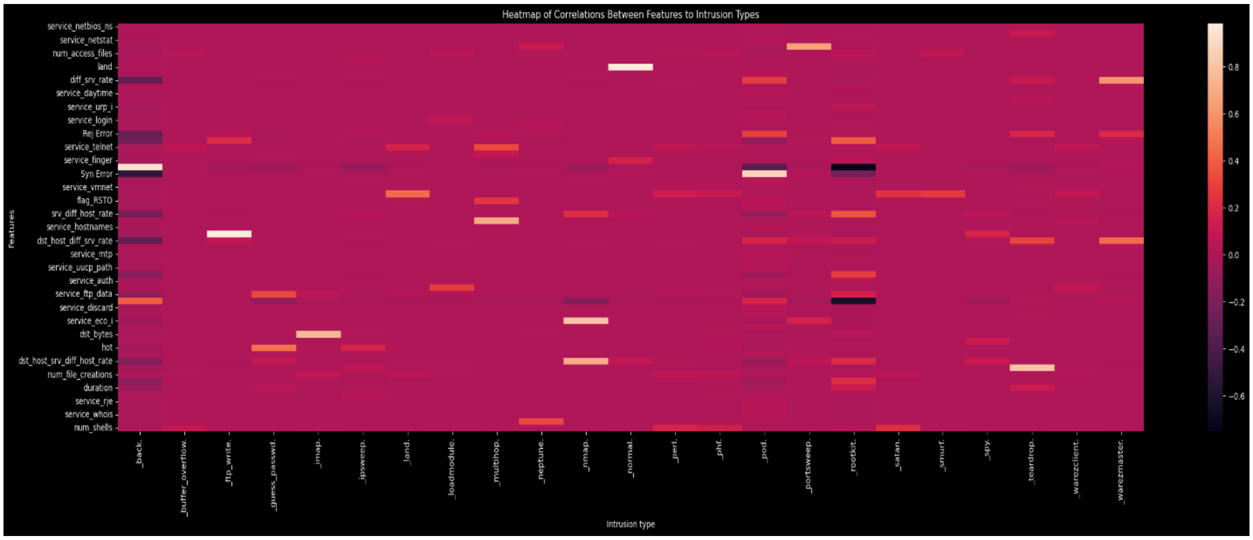
When I use the data that created the above heatmap which is in the EDA notebook, I can come to the conclusions for each cluster that:

- Cluster 1: High count, slightly elevated dst_host_count --> 99.9658% chance of being a smurf attack, .0342% are normal.
- Cluster 2: High srv_diff_host_rate, low dst_host_count, high service_eco_i, or high service_smtp --> 88.3967% of being normal, 11.4258% of rare attack type, tiny chance of neptune or warezclient.
- Cluster 3: High hot --> 66.9767% Warezclient, 32.5581% normal
- Cluster 4: Nothing major --> 95.7344% normal, 3.5693% rare attack types, slight instances of everything but Neptune
- Cluster 5: High wrong_fragment, high protocol_type_udp --> 100% teardrop.
- Cluster 6: High Syn_Error --> 99.748% Neptune, slight chance of rare type or normal
- Cluster 7: High flag_RSTR, High Rej_Error --> 73.710% Neptune, 9.4552% rare type, slight chance of all the types, 16.8060% normal
- Cluster 8: Low dst_host_count, high service_ftp_data --> %95.4880 normal, everything else is under 2.6%, barely any for teardrop or smurf.

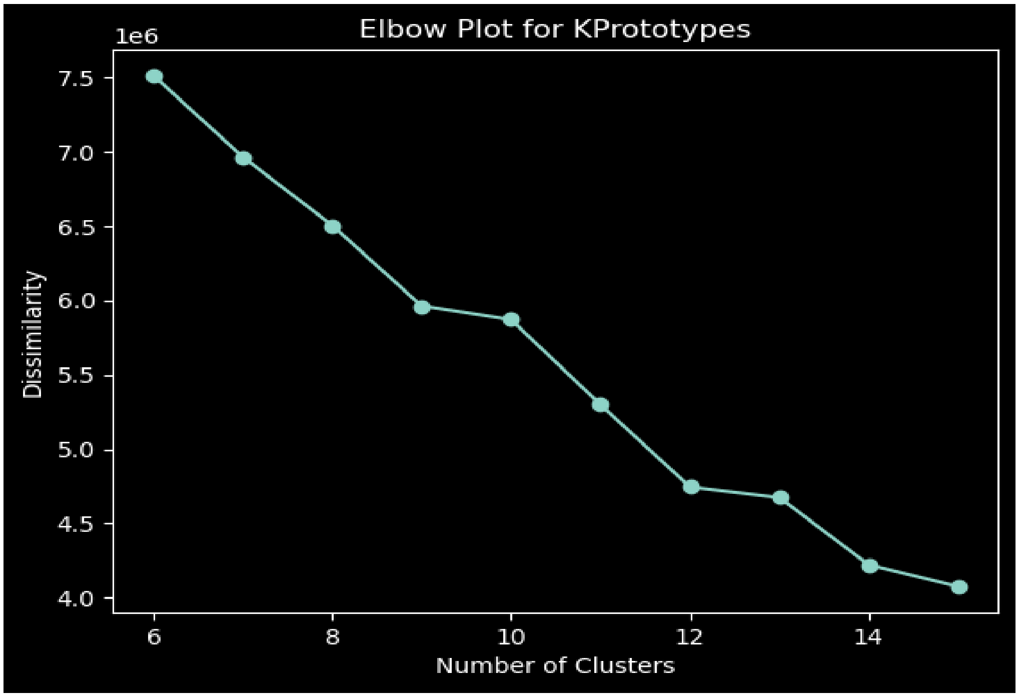
With all the insights that the clusters provide, I evaluated their importance using Lasso regularization from the data wrangling section. The clusters had a coefficient of zero, so they are not going to aid in the modelling process.

Big Data

The feature selection process until now used a lasso regression that regularized between normal and non-normal labels in the data. It had no data that solely distinguished between the different attack types. This next section tries the clustering algorithm with the extra data. First, I created a correlation heatmap to see if there are any obvious correlations that were not in the first dataset.

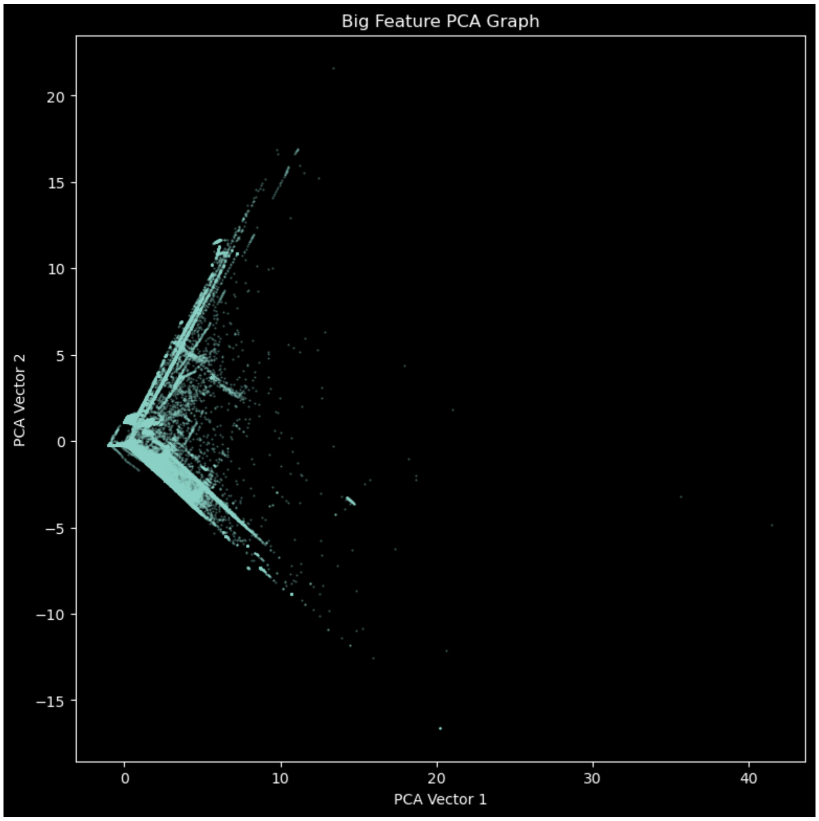


There appears to be slightly more information here than the first time that I ran the correlation heatmap. We can try to do the same process with this dataset as with the 12-feature dataset by creating an elbow plot of the different numbers of clusters.

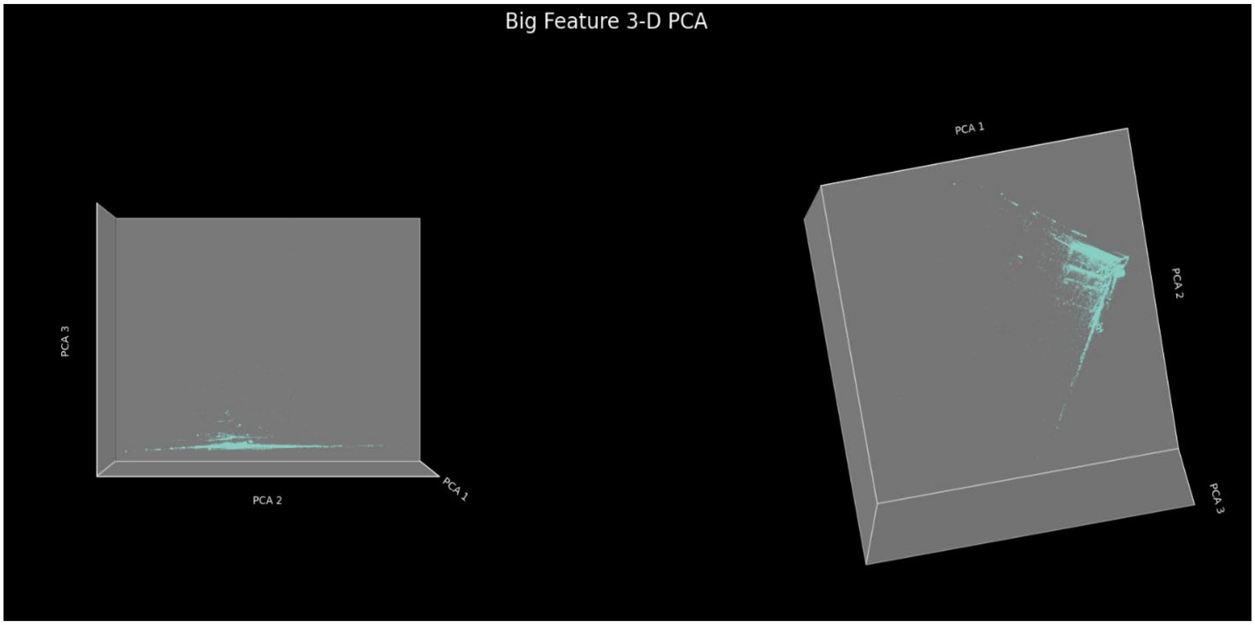


This elbow plot does not have the smooth downward descent that the previous elbow plot had. This is from the substantial number of dummy variables that function as categorical variables. As K-Prototype has more clusters added, some of the observations that shared dummy values with their centroid, will have to switch to centroids with different dummy values. The large cost of unmatched categorical data in K-Prototypes creates random bumps and changes in slope from one value of K to the next. It is possible that the correct number of clusters is fourteen, but it is also possible that the loss function keeps decreasing with higher values of K. The amount of time to process each cluster increased severely as K increased. It would therefore be impractical to find out the best number of clusters in a reasonable amount of time.

If there is some clustered structure in this data, then I can use PCA to discover it. Below is the PCA graph of the big dataset.



This graph does not look as though there is a clustered structure in this dataset. I can also a 3-D version to see if there is another angle in which the clusters are hiding.



From these graphs, there is not a clear cluster structure in this larger dataset. It is possible that for a high value of K there are truly obvious clusters, but I do not have the computing power nor the time to find them.

Pre-processing

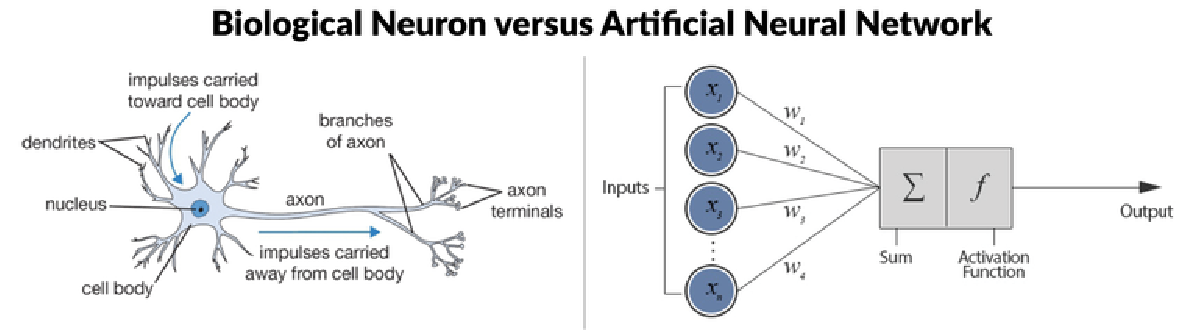
The dataset that I have been working with so far was only 10% of the entire dataset. I created a processing function that prepared the entire dataset the exact same way as I had left the 10% version after the data wrangling. I also grouped the new intrusion types so that the model could potentially give more information than just if an attack were happening for the new intrusion types.

Modelling

With the data wrangled, explored, and processed I can now move on to the modelling stage. The two models that I have decided to use are neural networks and random forests.

Neural Networks

The term neural networks is inspired by the structure and function of biological neurons. The diagram below shows both a diagram of a neuron, and a node of a neural network.



This Photo by Unknown Author is licensed under CC BY-SA

In both, a node receives and processes many inputs, then produces an output. When many nodes are connected, they form a network that can emulate non-linear complex systems. The formula for a node in a neural network is:

$$Output = f(\textstyle\sum_{1}^{n} w_i x_i) + b \qquad (7)$$

Where f is the activation function of the node, w_i is the weight of a given input into the node, x_i is the value of a connected node, n is all the inputs into the node, and b is the bias of the node.

The output of one node can become one of the inputs of another node or it can be the final node in the network that bears its prediction.

When a neural network is training, the network updates its model weights to minimize the difference between its predictions and the target values in the training set. This is achieved by using the derivative of the function representation of the entire model. The network moves in the direction with the lowest derivative value to find the minima until there are no more directions with negative derivative values. When the model finishes, it can be used to predict a test set that it has not been trained on but has the same type of data as the model to make predictions.

For these datasets, a neural network is an excellent choice because intrusions are a complex, non-linear system. The training datasets that I prepared have 12 and 61 features and five million observations. So, the model should have no problem with being fully able to train itself.

The smaller neural net that I created has an input layer of twelve nodes to match the size of the smaller dataset, 3 hidden layers of 12 nodes, a hidden layer of 8 nodes, and a final output layer of a single node. The activation function that I used for every layer except the last was Relu. which stands for rectified linear unit. The Relu function is:

$$Relu(x) = \begin{cases} x \geq 0, & x \\ x < 0, & 0 \end{cases} \tag{8}$$

Where the idea is to discard the negative nodes that are not helpful while allowing the useful nodes to equally contribute to the model.

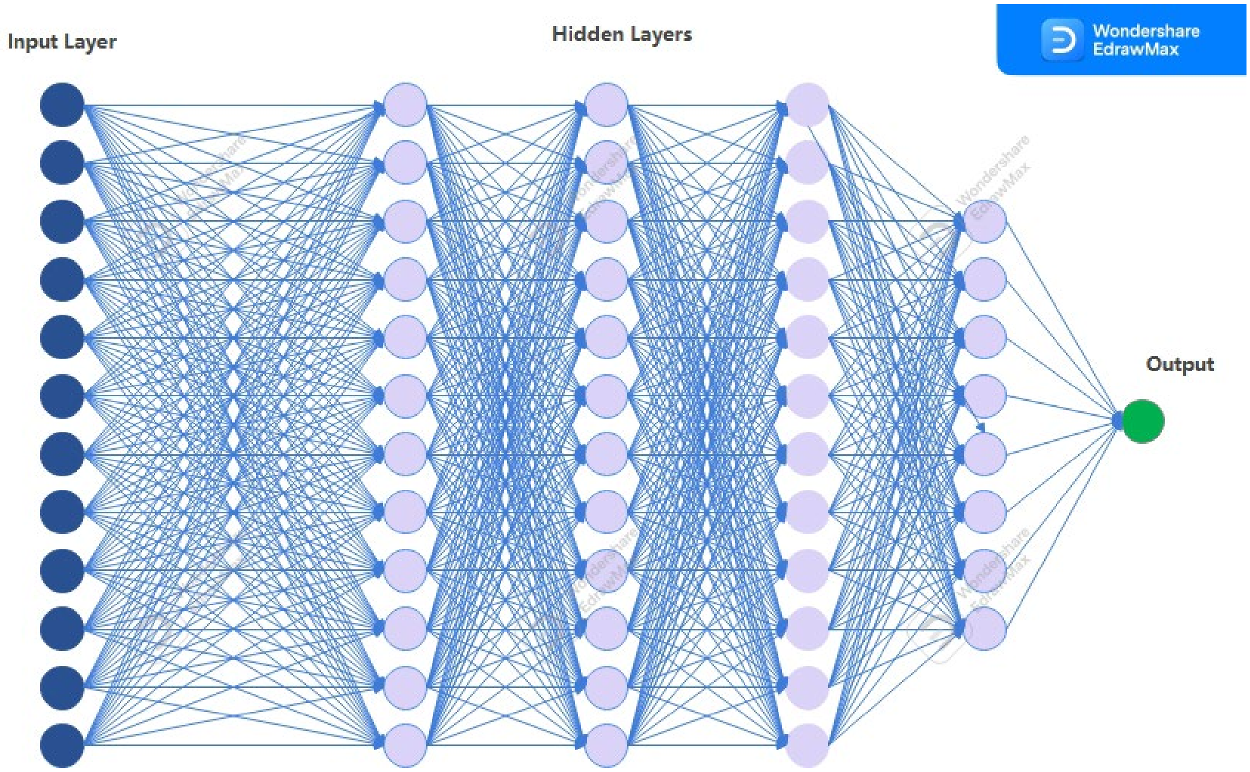
The final layer uses a function called sigmoid which converts the outputs of the earlier nodes into probabilities. The formula for sigmoid is:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{9}$$

This formula only outputs values between zero and one. As x goes to infinity, the exponential term disappears, and the fraction simplifies to one. If x goes to negative infinity, the denominator also goes to infinity and the fraction tends toward zero.

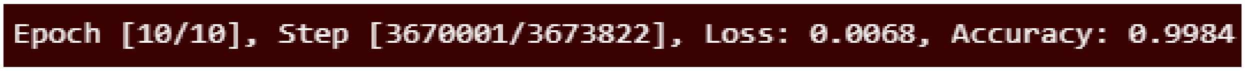
Small Data

Below is a graphical representation of the first neural network that I had, all the nodes use the Relu activation function except for the last node which uses the Sigmoid function to create a probability between zero and one.



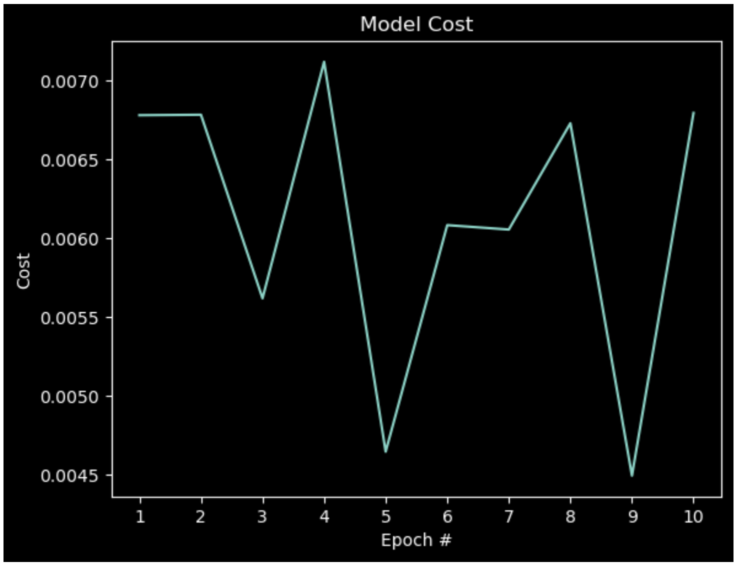
There are many options for hyper-parameters in a neural network, the type of model, size and shape of the model, the batch size, number of epochs, and the loss function can all play crucial roles in how the model fares. Batch size refers to how many observations the model handles before it updates its weights. Higher batch sizes allow the model to work much faster, but it also does not allow the model to learn as well. An epoch refers to how many times the model has gone through the dataset. The assumption is that as the model goes through the epochs it will have its cost function decrease asymptotically until finally the model has been fully trained.

The first model that I am going to run will use only the training data. The official test set has new intrusion types so by using the training data as both the training and the test set, I can assess if a neural network is the right model for this dataset. After training the model with a batch size of five thousand gave this output:



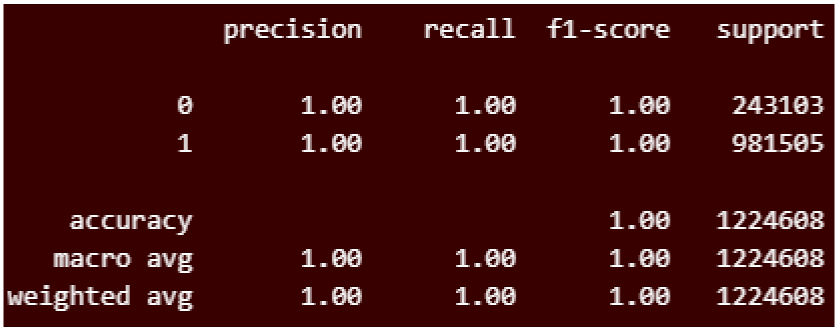
```
Epoch [10/10], Step [3670001/3673822], Loss: 0.0068, Accuracy: 0.9984
```

The loss and accuracy were exceptionally low. I also graphed the loss with respect to the epochs.



The model did all its learning in the first epoch because of the small batch size. The rest of the graph contains random fluctuations as the model cannot continue to improve in a meaningful way.

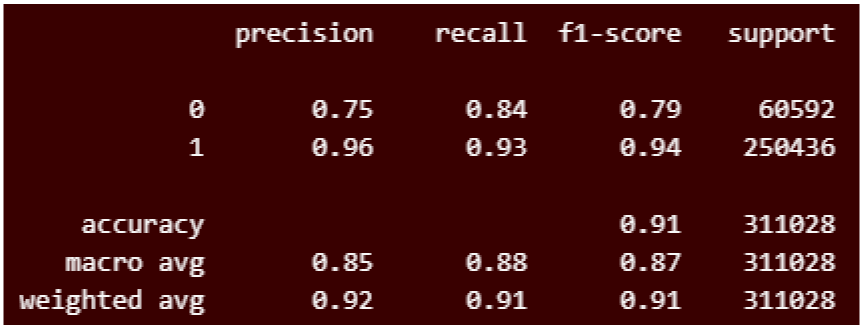
With this result I can assess the model on the test set portion of the training data. When I ran the test, I got a stunning result.



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    243103
           1       1.00      1.00      1.00    981505

    accuracy                           1.00   1224608
   macro avg       1.00      1.00      1.00   1224608
weighted avg       1.00      1.00      1.00   1224608
```

Classification report rounds its scores, so while this was not a perfect result, it was remarkably close. Now that I know that this model is remarkably effective for this dataset, I will use the entire dataset to train for the official test set.

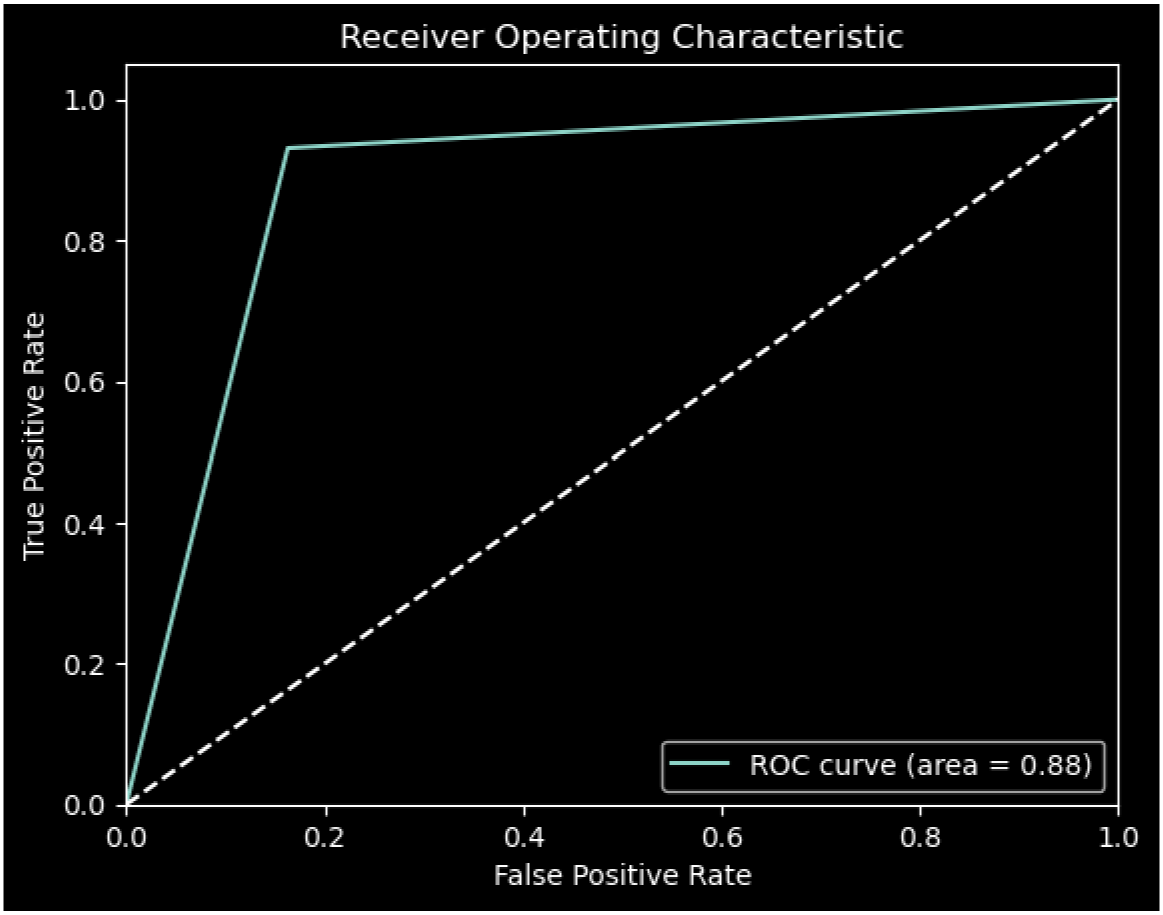
When running the model on the official test set, I got this result.



```
              precision    recall  f1-score   support

           0       0.75      0.84      0.79     60592
           1       0.96      0.93      0.94    250436

    accuracy                           0.91    311028
   macro avg       0.85      0.88      0.87    311028
weighted avg       0.92      0.91      0.91    311028
```

This appears promising, The model is not only better than a coinflip, but also significantly better than a trivial model that would predict the most frequent class every time. Only 19.5% of the test set is safe. If I had predicted an intrusion every time, the classification report would have had a precision of eighty for intrusion, 100 for recall intrusion and zeroes on everything for no intrusion.

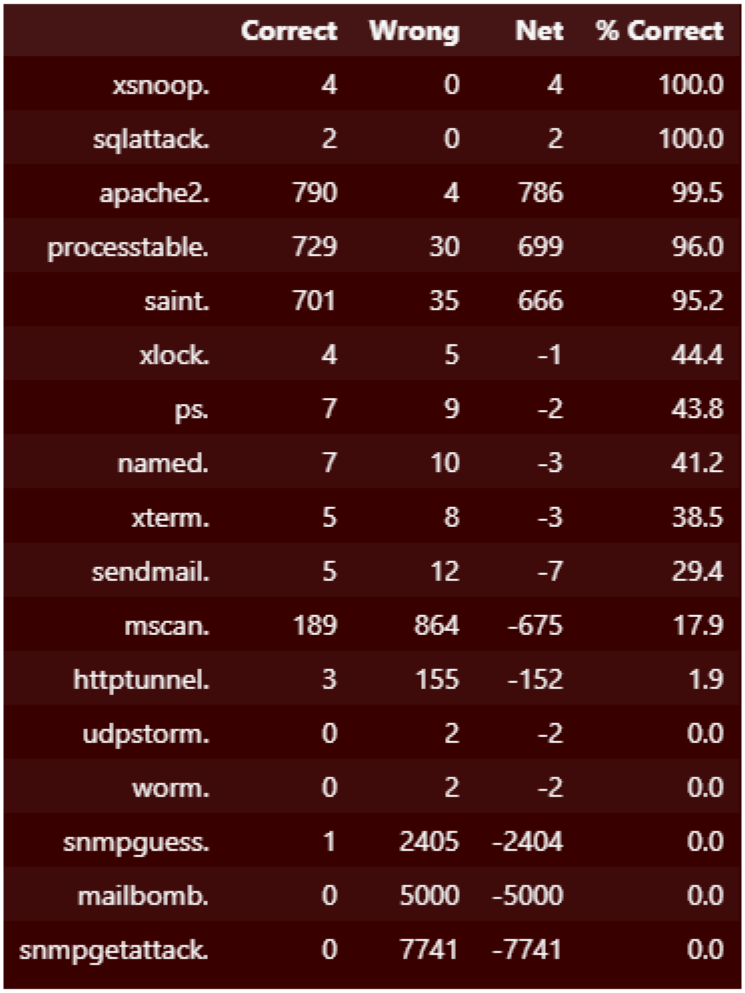
To get a better understanding of how well the model performed I can use the Receiver under the Curve metric which measures the True Positive Rate (TPR) against the False Positive Rate (FPR). A ROC area of greater than 0.8 is good and an ROC curve of 0.9 or greater is excellent.



The ROC curve of .88 is good but not great.

There is a deeper issue with this result which is that when I compare it to the cross-validation dataset there is a clear drop off. Out of all the observations that had new attack types, the model correctly classified only 14.24%.

To further investigate, I looked at which new attack types the model was best at predicting and which attack types the model struggled with.



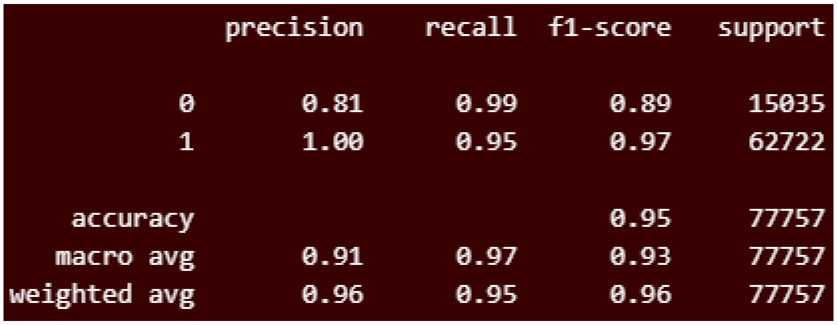
	Correct	Wrong	Net	% Correct
xsnoop.	4	0	4	100.0
sqlattack.	2	0	2	100.0
apache2.	790	4	786	99.5
processtable.	729	30	699	96.0
saint.	701	35	666	95.2
xlock.	4	5	-1	44.4
ps.	7	9	-2	43.8
named.	7	10	-3	41.2
xterm.	5	8	-3	38.5
sendmail.	5	12	-7	29.4
mscan.	189	864	-675	17.9
httptunnel.	3	155	-152	1.9
udpstorm.	0	2	-2	0.0
worm.	0	2	-2	0.0
snmpguess.	1	2405	-2404	0.0
mailbomb.	0	5000	-5000	0.0
snmpgetattack.	0	7741	-7741	0.0

The varying efficiency of the model comes from the wide range in the level of similarity between the new attacks and the training attacks that the model trained on. For snmpgetattack, mailbomb and snmpguess there were not a lot of intrusions that had the same signature as them. Since the safe classification is likely to not have any specific markers, anything that did not match an intrusion would be considered safe.

There is a fundamental problem in this challenge which is that it is impossible to train for what has never happened. These models are great at predicting the future based on the patterns of the past but when they are unable to see any future patterns in the data, they cannot predict the next attack. This is why cybersecurity specialists need to consistently update the training sets with new types of attacks. Without that data, companies and organizations are blind to the new ways that hackers can attack them.

To assess if the Neural network is still workable, I can split the official test set into training and testing data and see how the model performs on the subset of test data after having trained on the testing data. If this were a company or the government, eventually future observations would replace the test data and turn it into training data.

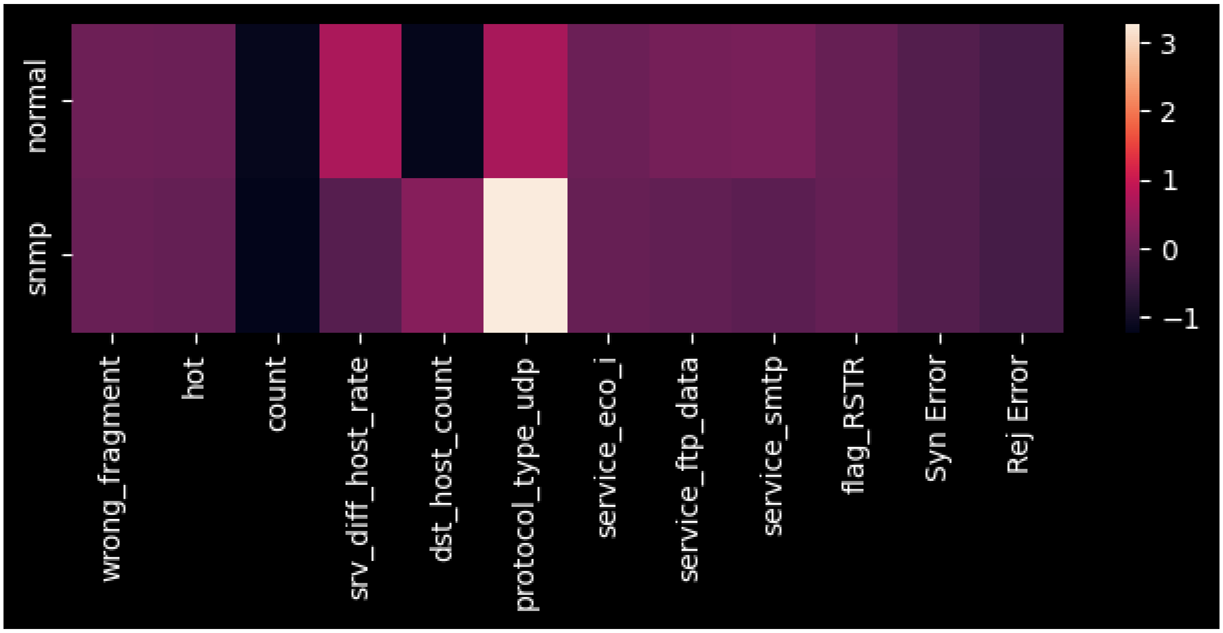
When I run the model, I get this result.



```
              precision    recall  f1-score   support

           0       0.81      0.99      0.89     15035
           1       1.00      0.95      0.97     62722

    accuracy                           0.95     77757
   macro avg       0.91      0.97      0.93     77757
weighted avg       0.96      0.95      0.96     77757
```

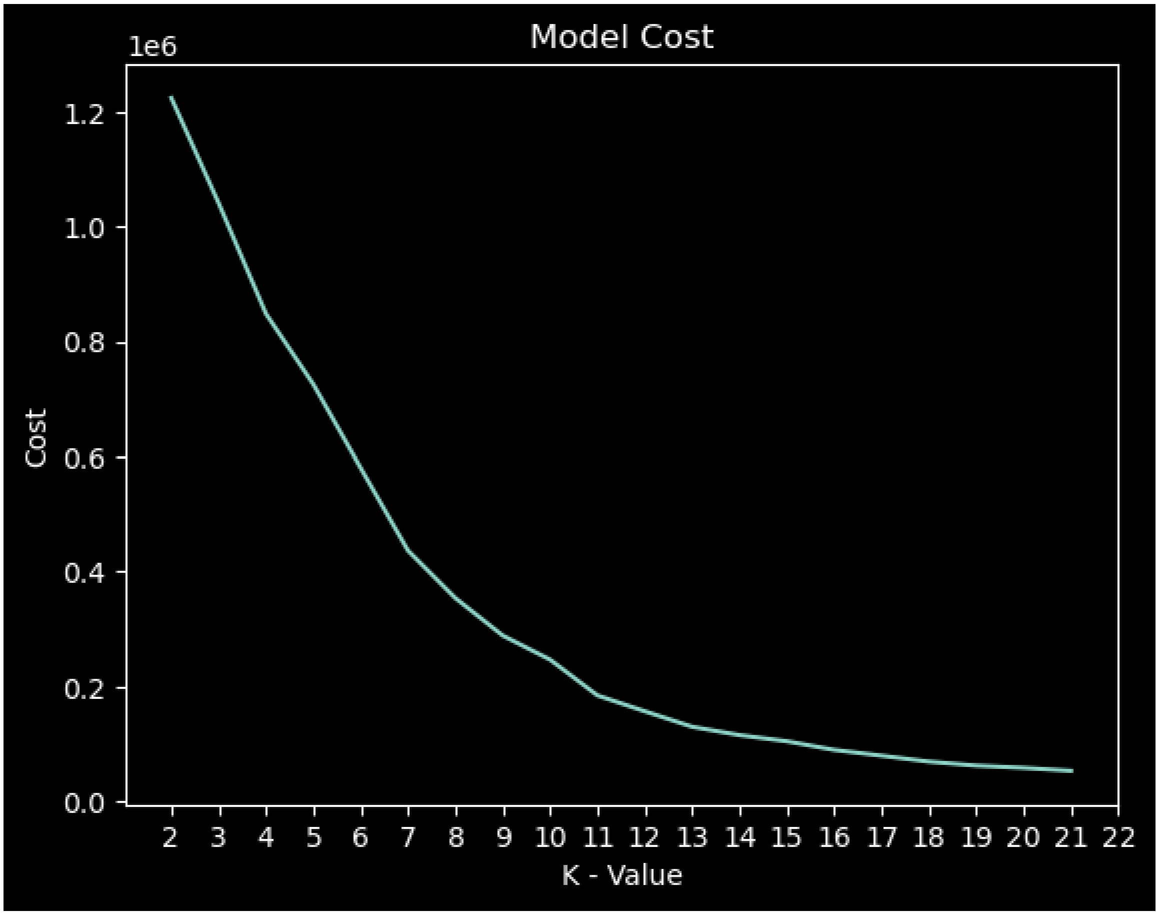
The accuracy is not near the training set cross validation accuracy which is disappointing. It appears that all the clear patterns that were in the training set are not fully in the test set. When I did lasso regularization, I did it only with the training set which did not have any of the new attacks.

The model treats snmp intrusions from this dataset exactly the same as no intrusion. In my cluster analysis I explored how a combination of different traits can lead to some stark conclusions about the type of intrusion an observation has. I can look at all means of the features of the observations that were snmp intrusions and compare them to the means of features in safe observations.



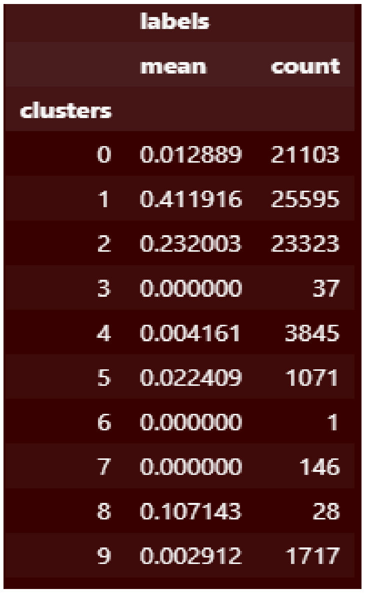
The heatmap looks like there should be enough for snmp to classified correctly by the model as protocol_type_udp which would appear to be significantly different in SNMP. What happened was that the model discarded protocol_type_udp during training as it did not have a significant impact on the training intrusions. If I create a sequential second layer that re-examines all the observations that the model predicted were safe, I can hopefully catch some obscure patterns in the rarer intrusion types that could significantly improve the model.

This heatmap makes it look very possible to create a classifier that could differentiate between snmp and normal. One method that comes to mind is K-Means which will differentiate between snmp and normal given the difference they both display with their means. Below I tried to find a value of K that was suitable for all the observations that were either normal or snmp.



While there is no clear value of K the clustering is effective as the line is decreasing significantly.

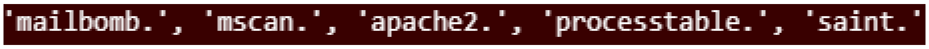
Clustering can supply some insight into what makes a safe observation. If there are other methods that can supply insights into places where the neural net failed, then I would significantly improve the IDS.



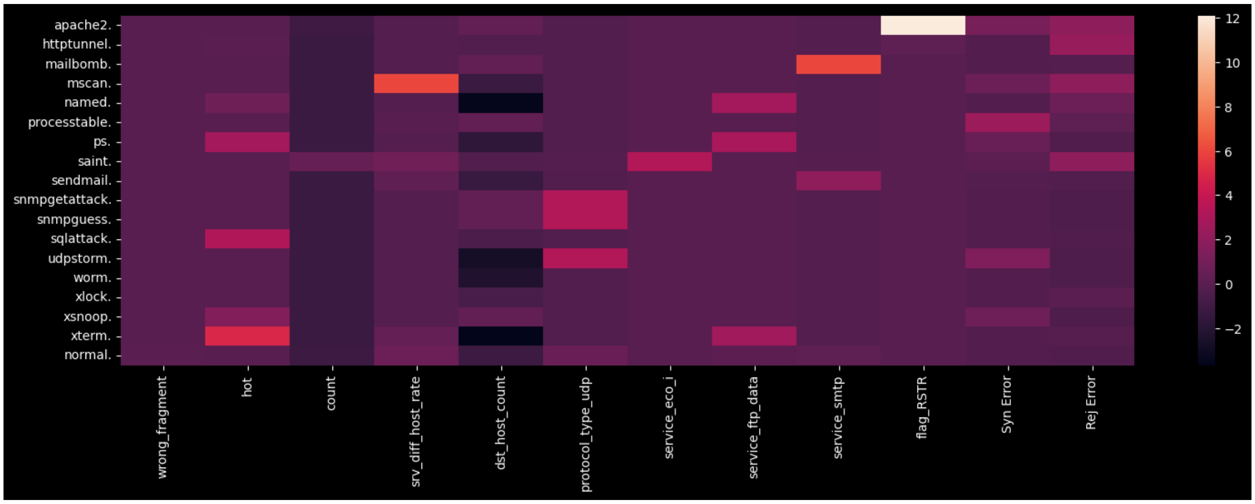
	labels	
	mean	count
clusters		
0	0.012889	21103
1	0.411916	25595
2	0.232003	23323
3	0.000000	37
4	0.004161	3845
5	0.022409	1071
6	0.000000	1
7	0.000000	146
8	0.107143	28
9	0.002912	1717

While the only clusters that do not have intrusions are tiny, many of the clusters that do have intrusions have very few. Clusters 1, 2, and 8 are the only ones that have more than 2.5% intrusions. By separating out these clusters and analyzing them the model could be a lot better if there was new data to come.

Below I created a heatmap that compares all the means of the features of each new intrusion type to the safe observations to see how similar they are. If it is the case that there is no clear difference, then the model will not be able to distinguish between safe and attack.



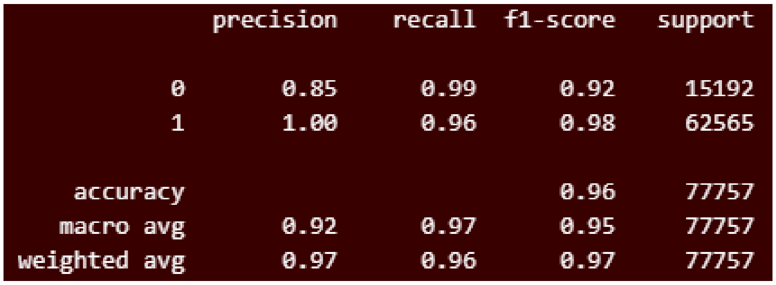
```
'mailbomb.', 'mscan.', 'apache2.', 'processtable.', 'saint.'
```



The attack type with the clearest hot spot (apache2) is one of the easiest new attack types for the model to classify. I printed above the heatmap the 5 new attacks that the model is the best at classifying. A lot of them have clear differences with the last row of the heat map in some features. When I look at snap attacks do have differences between them and the safe observation but means could be close enough that the model has no way to effectively differentiate between the two.

I believe that the reason the model is struggling with these new intrusion types is because when I did lasso regularization, I perfected only the intrusion types in the training set and not any in the test set. To build a better model that could be useful for the next round of data. I need to keep perfecting it with all the relevant features.

When I run the model with the new features, I get the following report.



```
              precision    recall  f1-score   support

           0       0.85      0.99      0.92     15192
           1       1.00      0.96      0.98     62565

    accuracy                           0.96     77757
   macro avg       0.92      0.97      0.95     77757
weighted avg       0.97      0.96      0.97     77757
```

There is improvement with these new features, but it is slight. While this improvement could be incredibly helpful depending on the size of the dataset, it is frustrating to not be able to replicate comparable results to the near perfect scores on the test set of the training part.

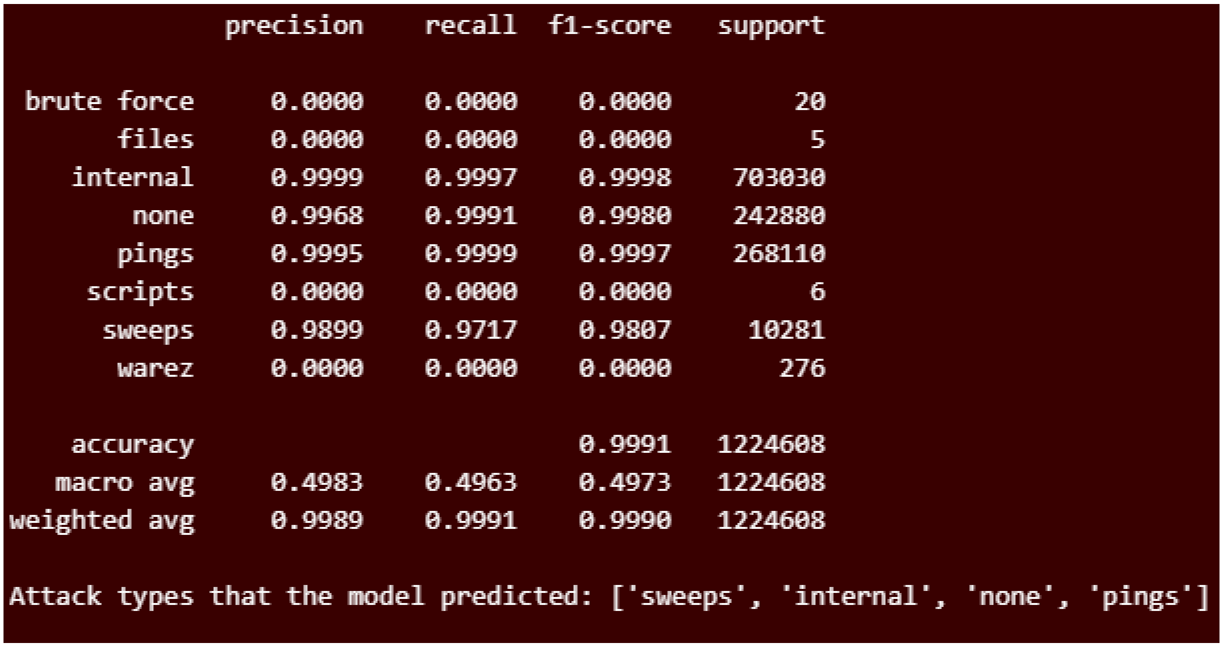
It is always important to keep perspective. If the model was able to predict certain types of attacks well and struggled with others, then that is a lot better than if it had the same overall accuracy but had uniform success with each attack type. With the knowledge that the model struggled with snmp attacks it is possible to install more security around snmp while continuing to use the model for the rest of the intrusions.

Another thing to note is the near perfect precision the model has when it predicts that there is an attack. Having at least one classification that a model can give that a person can put confidence in is valuable.

Big Data

As helpful as it is to know whether there is an attack or not, it would also be helpful to know what kind of attack is happening. To do this I will use the big dataset that is made up of every feature that influences the type of attack in an observation.

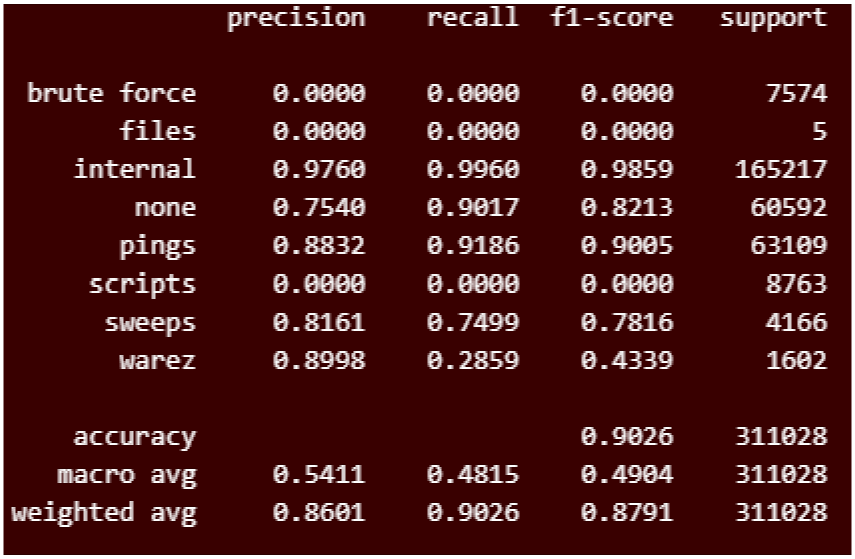
When I run the model on the cross-validation set like how I did for the smaller dataset I get this result



```
              precision    recall  f1-score   support

 brute force     0.0000    0.0000    0.0000        20
       files     0.0000    0.0000    0.0000         5
    internal     0.9999    0.9997    0.9998    703030
        none     0.9968    0.9991    0.9980    242880
       pings     0.9995    0.9999    0.9997    268110
     scripts     0.0000    0.0000    0.0000         6
      sweeps     0.9899    0.9717    0.9807     10281
       warez     0.0000    0.0000    0.0000       276

    accuracy                         0.9991   1224608
   macro avg     0.4983    0.4963    0.4973   1224608
weighted avg     0.9989    0.9991    0.9990   1224608

Attack types that the model predicted: ['sweeps', 'internal', 'none', 'pings']
```

The bottom line shows that the model did not even predict ‘brute force,’ ‘files,’ ‘scripts,’ or ‘warez.’ It is hard to know if this is because there are so few observations that fit these groups or if these intrusion types are hard to single out. The test set has a different distribution for the groupings of the attack types. It is possible that the model will have an easier time classifying the larger classes.

When I run the large model on the official test set, I get this.



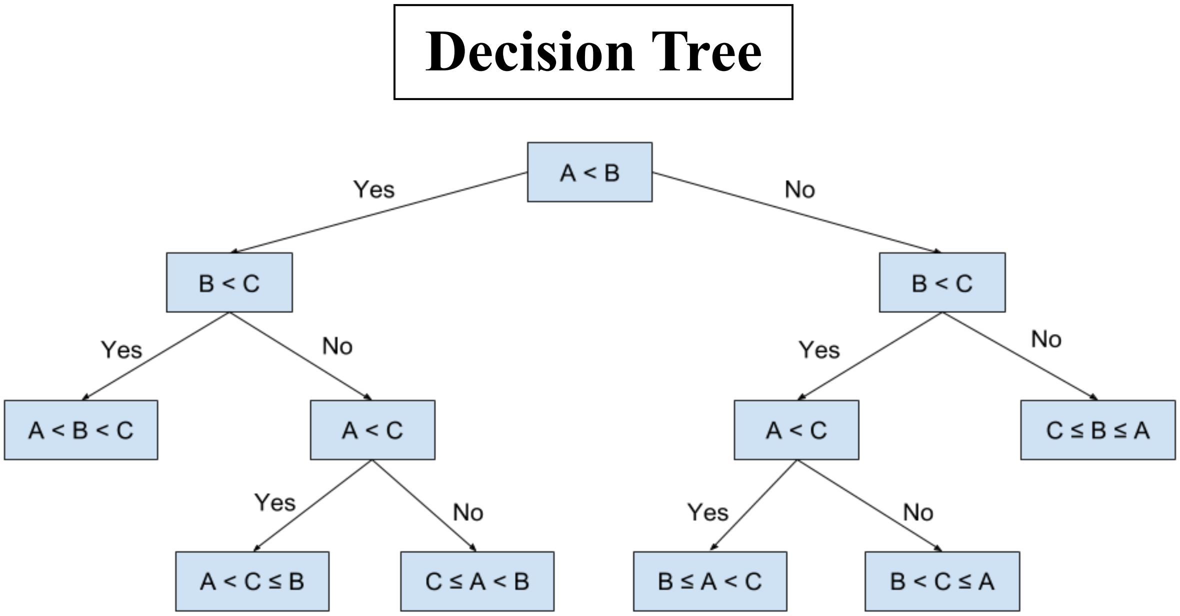
```
              precision    recall  f1-score   support

 brute force     0.0000    0.0000    0.0000      7574
       files     0.0000    0.0000    0.0000         5
    internal     0.9760    0.9960    0.9859    165217
        none     0.7540    0.9017    0.8213     60592
       pings     0.8832    0.9186    0.9005     63109
     scripts     0.0000    0.0000    0.0000      8763
      sweeps     0.8161    0.7499    0.7816      4166
       warez     0.8998    0.2859    0.4339      1602

    accuracy                         0.9026    311028
   macro avg     0.5411    0.4815    0.4904    311028
weighted avg     0.8601    0.9026    0.8791    311028
```

The larger model suffers from the same inability to predict new and rarer intrusions as the smaller model did. It is interesting that as warez increased, the model was able to start picking it up. One of the hardest parts of creating a model for this dataset is that some of the intrusion types have under ten occurrences out of either five million or 300,000 observations.

Overall, the large model did better than the smaller model. The precision and recall scores for no intrusion increased while the model had success in predicting the attack type.

Random Forests

Random forests are a collection of decision trees that predict a target variable based on the features of the dataset. Below is an example of a decision tree.



This Photo by Unknown Author is licensed under CC BY

When decision trees are applied to datasets, they have specific ways of creating decisions. If the data is binary, then the decision tree will create two branches for each of the binary values. If the data is categorical then the decision tree can choose any number of the categories to become another branch. If the data is numerical, then the decision tree creates a cutoff value where all observations with values below the cutoff go to one branch, and the rest go to another branch.

The decision tree then partitions the least homogeneous feature of the dataset as measured by the Gini index which is:

$$Gini = 1 - \sum_{i=1}^{n} {p_i}^2 \tag{10}$$

Where p_i is the probability of each of the classes and n is all the classes. The decision tree will continue to iterate through this process until it reaches 100% purity and has nothing left to partition. When all the observations in each group are the same, then y-class that is the most popular for that group becomes the prediction of the model.

This can lead to overfitting as the decision tree has learned all the patterns of the training data that may or may not be helpful. To fix this problem we can create a forest of decision trees with bootstrapped datasets. Bootstrapping is when the data is randomly sampled with replacement to create another dataset that has the same parameters as the original dataset but does not carry the same noise. When all the trees in the forest have finished their processes, they vote on the correct prediction. The prediction that the greatest number of trees support is the prediction of the forest.

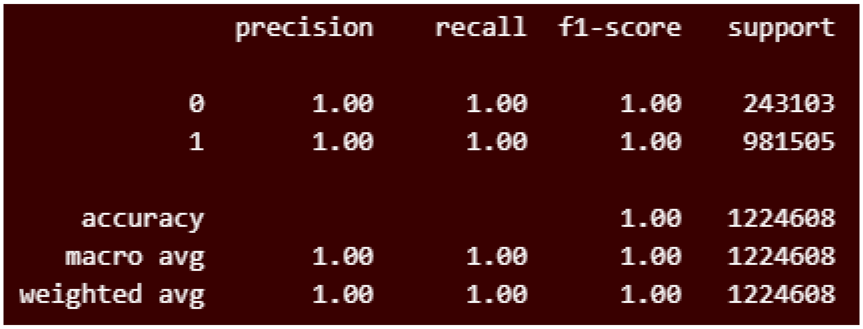
There are many different hyper-parameters for a random forest model. Some examples are:

- Number of decision trees

- Homogeneity metric
- Maximum depth of an individual tree

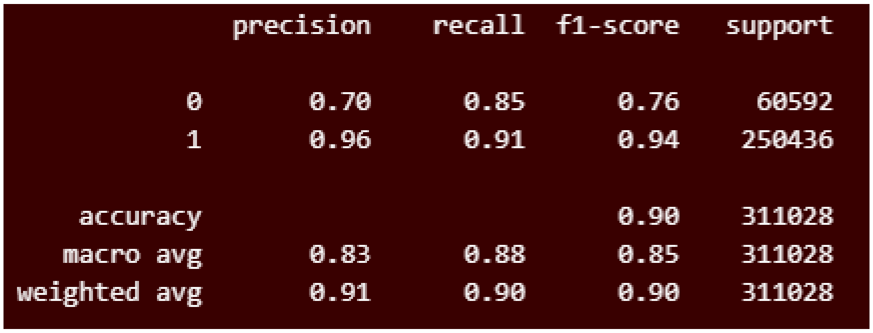
The only hyper parameter that I had time to test was the number of decision trees because the calculations took significantly more time as the number estimators increased. It is possible that there are some hyper-parameters that would have produced significantly better results, but I did not have the time to find them.

I treated the random forest the exact same as the neural network. The first test was to use the cross-validation test set to see if it could match the neural network.



```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    243103
           1       1.00      1.00      1.00    981505

    accuracy                           1.00   1224608
   macro avg       1.00      1.00      1.00   1224608
weighted avg       1.00      1.00      1.00   1224608
```

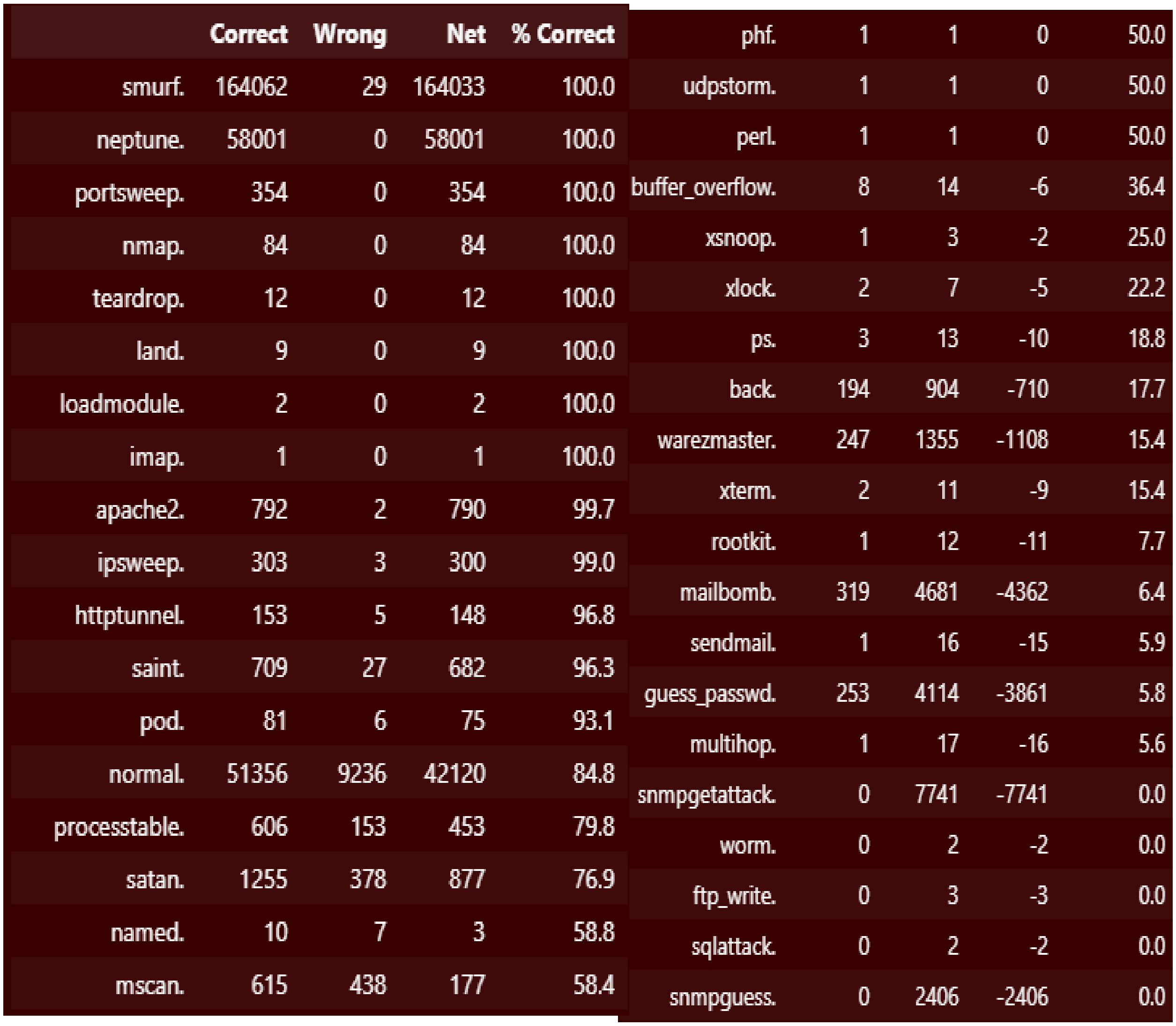
It is interesting that the random forest was able to get the same result as the neural network. It shows that there were clearly decipherable patterns in the training data that these models were able to notice. Below is the official test set.



```
              precision    recall  f1-score   support

           0       0.70      0.85      0.76     60592
           1       0.96      0.91      0.94    250436

    accuracy                           0.90    311028
   macro avg       0.83      0.88      0.85    311028
weighted avg       0.91      0.90      0.90    311028
```

Both models struggled when moving to the official test set. This is slightly worse than the neural network, but it is close. It could be that if the hyper parameters were correctly tuned, then the random forest's predictions would be more correct.

Below I did a deep dive into how the model performed with different intrusion types.

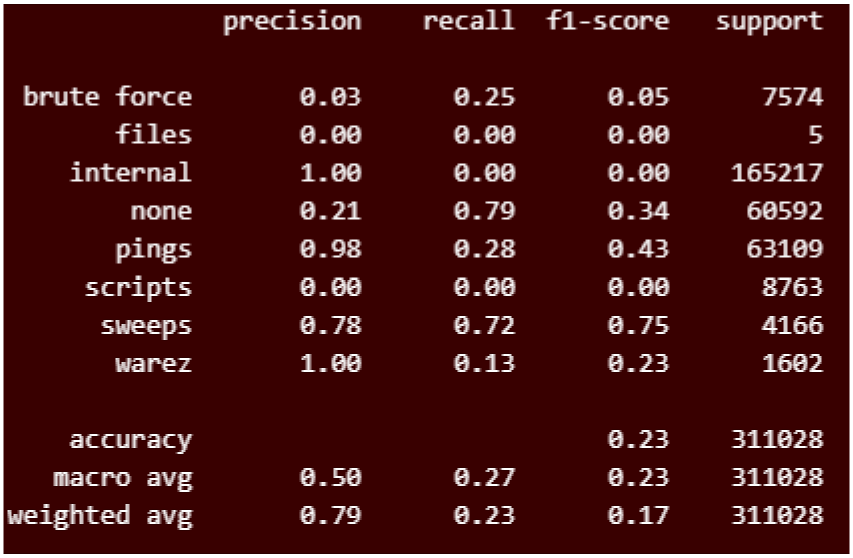


	Correct	Wrong	Net	% Correct
smurf.	164062	29	164033	100.0
neptune.	58001	0	58001	100.0
portsweep.	354	0	354	100.0
nmap.	84	0	84	100.0
teardrop.	12	0	12	100.0
land.	9	0	9	100.0
loadmodule.	2	0	2	100.0
imap.	1	0	1	100.0
apache2.	792	2	790	99.7
ipsweep.	303	3	300	99.0
httptunnel.	153	5	148	96.8
saint.	709	27	682	96.3
pod.	81	6	75	93.1
normal.	51356	9236	42120	84.8
processtable.	606	153	453	79.8
satan.	1255	378	877	76.9
named.	10	7	3	58.8
mscan.	615	438	177	58.4
phf.	1	1	0	50.0
udpstorm.	1	1	0	50.0
perl.	1	1	0	50.0
buffer_overflow.	8	14	-6	36.4
xsnoop.	1	3	-2	25.0
xlock.	2	7	-5	22.2
ps.	3	13	-10	18.8
back.	194	904	-710	17.7
warezmaster.	247	1355	-1108	15.4
xterm.	2	11	-9	15.4
rootkit.	1	12	-11	7.7
mailbomb.	319	4681	-4362	6.4
sendmail.	1	16	-15	5.9
guess_passwd.	253	4114	-3861	5.8
multihop.	1	17	-16	5.6
snmpgetattack.	0	7741	-7741	0.0
worm.	0	2	-2	0.0
ftp_write.	0	3	-3	0.0
sqlattack.	0	2	-2	0.0
snmpguess.	0	2406	-2406	0.0

The decision tree struggled in the exact same way with none of the snmp attacks correctly predicted. This shows that there is nothing in the data that would allow a model to predict snmp attacks. If a business used these models, they would still have to take additional security measures to protect against snmp attacks.

Big Data

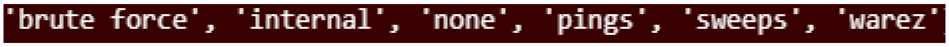
The final calculation that I did was to use the big dataset to predict the group that the intrusions belonged to. These are the results:



```
              precision    recall  f1-score   support

 brute force       0.03      0.25      0.05      7574
       files       0.00      0.00      0.00         5
    internal       1.00      0.00      0.00    165217
        none       0.21      0.79      0.34     60592
       pings       0.98      0.28      0.43     63109
     scripts       0.00      0.00      0.00      8763
      sweeps       0.78      0.72      0.75      4166
       warez       1.00      0.13      0.23      1602

    accuracy                           0.23    311028
   macro avg       0.50      0.27      0.23    311028
weighted avg       0.79      0.23      0.17    311028
```

This model was significantly worse when trying to predict the groups. The most glaring weakness of this model is the precision score for none. The group that by far matters the most is the one that finds out whether there is an intrusion happening or not.

I also returned all the groups that the model had predicted.



```
'brute force', 'internal', 'none', 'pings', 'sweeps', 'warez'
```

It is interesting that the random forest was able to predict more of the groups than the neural network. One possibility for future work is to combine the two models where the first level is the neural network and then the second level is the random forest. If the random forest classifies an intrusion as a warez attack or sweep, then that observation will be considered warez or a sweep.

Conclusion

When I started this project, I thought that I was going to be able to download the dataset, make a couple of graphs, run a couple of models, and get a decent classifier. I was wrong. This dataset carried tremendous richness that needed a full investigation to reveal its properties.

The most important insights from the project are:

- Clustering Analysis
- The Categorical Neural Network

The clustering insights exposed how the features could interact to create an intrusion type. While I did not fully articulate in the report how all the features impacted intrusions, by looking at how the features make up the clusters and then applying that to how clusters create intrusions, it is possible to calculate the most worrying values of features.

The best model was the Categorical Neural Network. Its ability to not only give the best binary prediction but also the best categorical prediction made it the best of any of the models. Its success rate was significantly higher than that of a trivial model and it was able to predict multiple different classes of intrusion types.

Final Thoughts

Towards the end of the project, I found a stack overflow answer about this dataset that said "As a side note, the [Intrusion Detection Systems] IDS research community vehemently discourages the use the DARPA dataset (and the derived KDD Cup dataset) despite it's appealing availability."[6] This comment stung but there should be some more context added to it. In a field that is rapidly evolving, a dataset that is 24 years old will not be able to supply much insight into current problems.

There are more problems with the data set than its age. A 2008 study[7] into the usefulness of the dataset found that:

- The background traffic generator is not publicly available so there is no way to evaluate the background traffic in the dataset.
- Some events like packet storms and strange packets are not in the dataset but would raise the false positive rate.
- There are irregularities in the data that make it possible for trivial detectors to have an effect.

These concerns were enough for one person to say "benchmarking, testing and evaluating with the DARPA dataset is useless unless serious breakthroughs are made in machine learning."

With all the problems that this dataset has "the non-availability of any other dataset that includes the complete network traffic"[7] guarantees that this dataset should be considered for analysis. Other researchers claim that "if an advanced IDS could not perform well on the DARPA dataset, it could also not perform acceptably on realistic data."

This dataset is the perfect training dataset. The complexity of attacks coupled with the randomness of normal behavior is far beyond the scope of this dataset. But it gives me a chance

[6] java - What is an syn error in TCP Protocol? - Stack Overflow

[7] darpa.pdf (iisc.ac.in)

to hone my skills and techniques on a large amount of data in an important field while getting clear and interesting results.