Introduction

Digital networks are heavily integrated into our modern society and will only continue to become more prevalent over time. As a result, it is imperative for network security to be trustworthy and reliable. Anomaly detection is the identification of rare behavior and patterns in real world data. Anomaly detection is an extremely useful tool for maintaining trust and reliability in our digital networks and is widely applicable to a variety of industries. For example, anomaly detection can be used in healthcare to diagnose diseases, initiate treatments, and enhance patient health overall.

However, a more common use case for anomaly detection is within the financial industry, specifically with respect to credit card data. Not only does anomaly detection protect both the client and the bank from fraud, but it also has the ability to identify more subtle trends, like possible economic shifts. In essence, anomaly detection, whether applied to network security, finances or healthcare serves as a critical tool in identifying the abnormal and mitigating risks.

As technology advances and cyber attacks become more advanced, detecting and mitigating malicious network anomalies is critical in maintaining the integrity of the digital network systems. In this project, we will utilize a widely recognized network security data set, KDD Cup 1999, to build a model capable of detecting network attacks through various unsupervised learning techniques. The lack of constraints around unsupervised learning, compared to that of supervised learning, provides a unique opportunity to create a model that is flexible to rapidly changing cyber security threats and can identify subtle patterns in the data that are hard to identify with supervised approaches. Additionally, an important note is that unsupervised learning focuses on minimizing false positives as it continues to learn, so when an anomaly is detected it most likely is correct.

We will be evaluating the results from unsupervised learning methodologies, including ML clustering models like K-means and GMM, as well as neural networks like autoencoder, to see which will identify anomalies most accurately. In addition, we will augment the KDD Cup 1999 dataset with synthetic data to test the flexibility and explore the limitations of each model.

Problem Statement

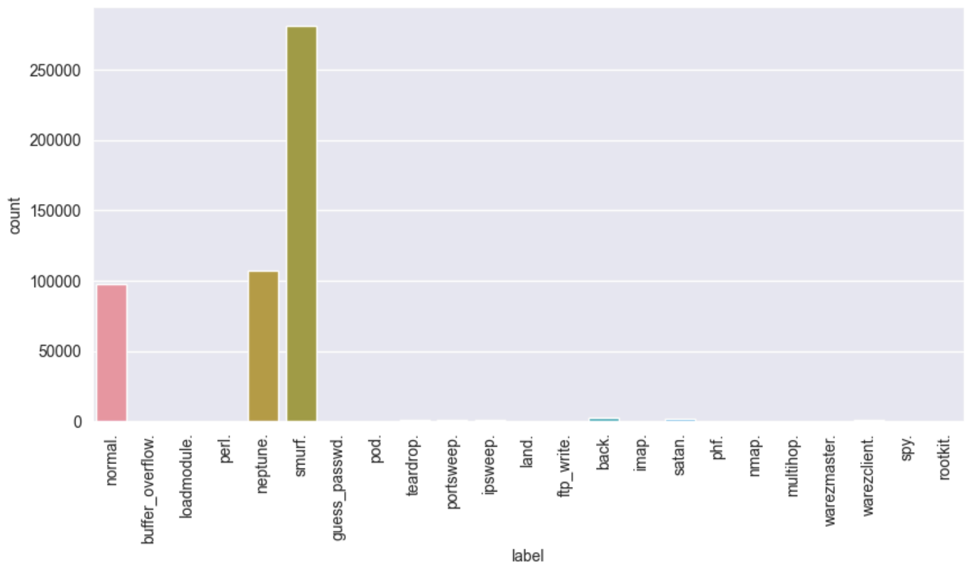
Given the KDD Cup 1999 data set, which unsupervised learning model is the most accurate at detecting labeled attacks.

Hypothesis

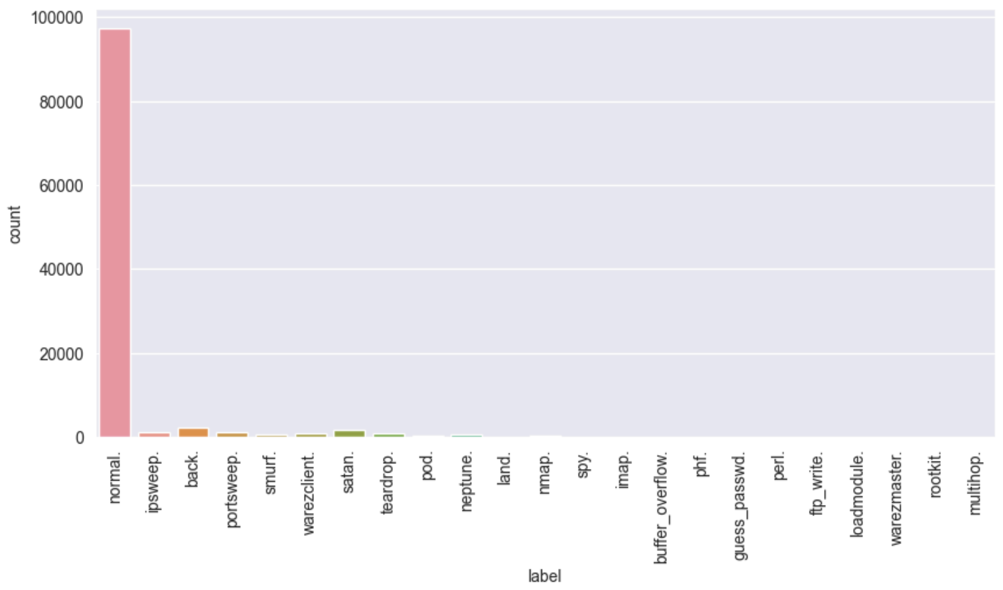
We hypothesize that the autoencoder will have the highest accuracy for anomaly identification because neural networks are more likely to be the most sensitive to the fraudulent data. We also hypothesized that amplifying attack data exposes vulnerabilities in certain anomaly detection models like autoencoders. While this model may be more powerful than conventional methods, the increased prevalence of attack data will make it harder to detect them as anomalies for a data reconstruction-based approach.

EDA & Data Preprocessing

When examining the labels in the KDD Cup dataset, we noticed that two attack types, “neptune” and “smurf”, had very high occurrences in the dataset. The occurrences were so high, in fact, that these attack frequencies were higher than “normal” user activity in the dataset. Anomaly detection models also often rely on unbalanced data.

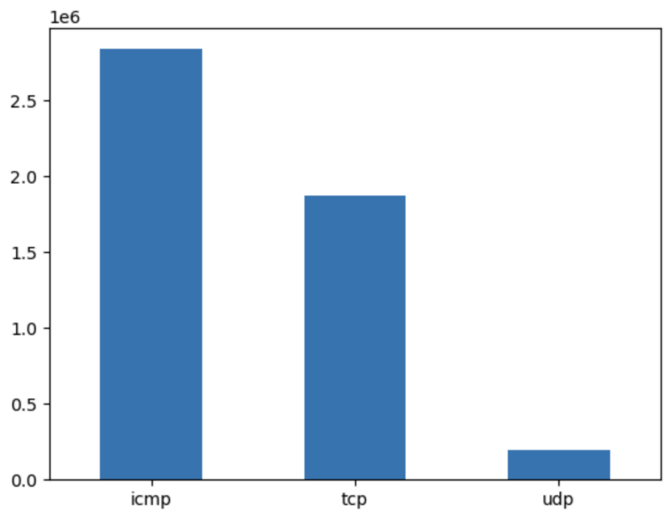


To adjust for these issues, we randomly undersampled the “smurf” and “neptune” data points to 500 each in order to create a highly imbalanced dataset required for better anomaly detection performance. The resulting dataset contained 97,275 normal observations and 9,752 attacks.

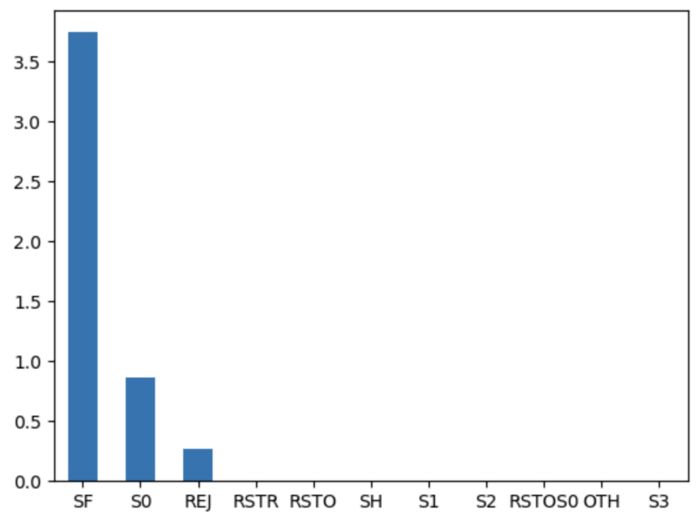


As seen below, the three categorical variables were very skewed towards one or two values. A correlation matrix ran on the numeric features to evaluate if any creatures were highly correlated. While most were uncorrelated, there were some features that appeared to have relatively high correlation. That being said, we decided the models discussed below on the entire dataset.

Protocol Type:



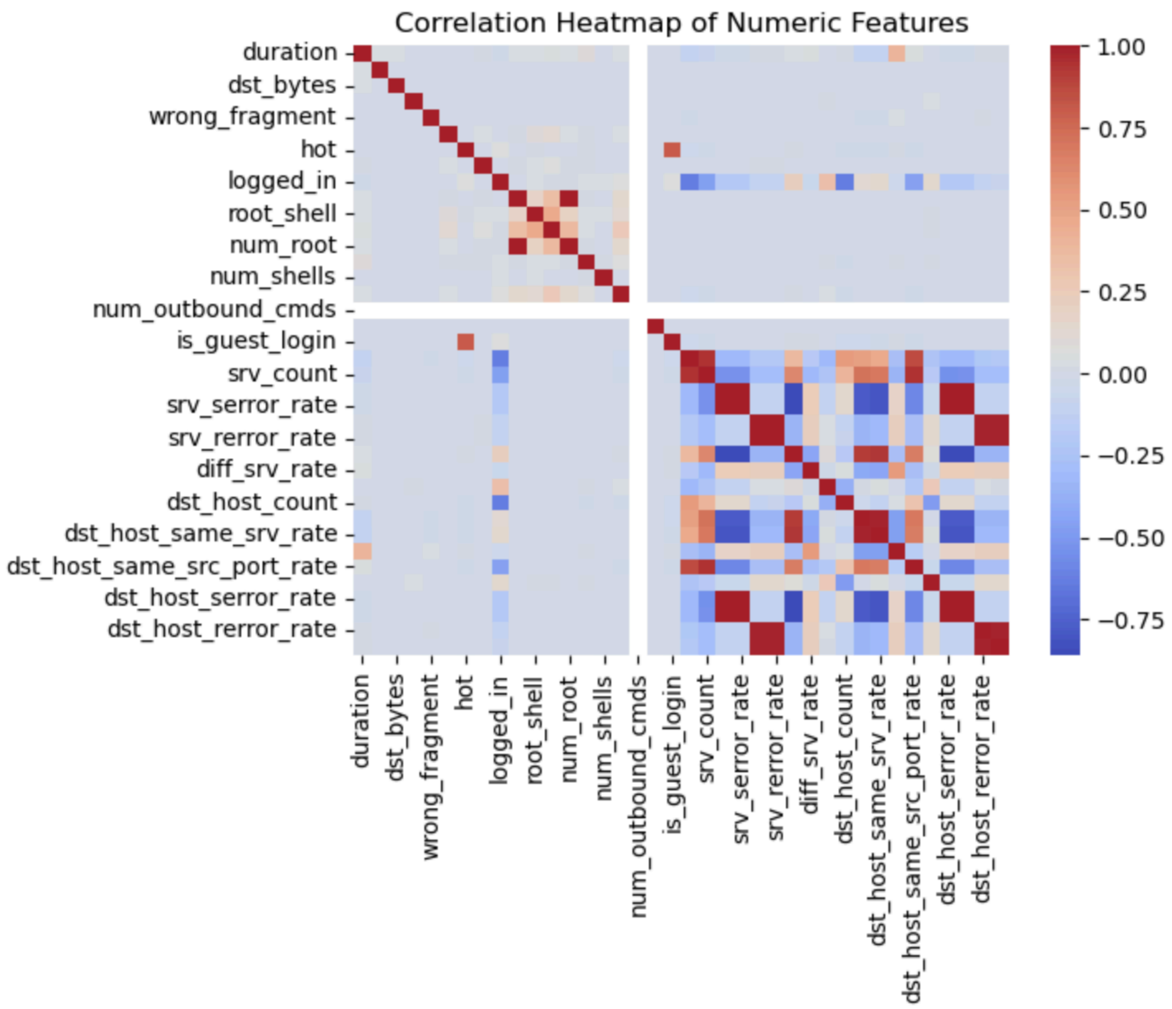
Flag:



Service (too many labels to nicely plot):

```
: ecr_i        2811660
  private      1100831
  http          623091
  smtp           96554
  other          72653
                ...
  tftp_u             3
  harvest            2
  aol                2
  http_8001          2
  http_2784          1
  Name: service, Length: 70, dtype: int64
```

Correlation Matrix of Numeric Variables:



Data processing for each of the models below included converting the categorical variables into numerical dummy variables (one-hot encoding) and combining the all network intrusion (attack) labels as one group, anomaly. The resulting labels were a binary column where normal = 0 and anomaly = 1 for each datapoint.

Methodology

First, we trained the full dataset on three unsupervised learning models: K-means, GMM and autoencoder. With this set of models, we aimed to understand which unsupervised approaches work best to identify network intrusions and attacks on the KDD Cup 1999 dataset. Once the models were trained and evaluated, we visualized the clustering models in low dimensional space and the autoencoder reconstruction scores to achieve a better understanding of how the models can separate the labeled attacks and normal network data. We trained each model twice – once on the full dataset and once on the undersampled set. All methods included two steps, training the unsupervised model and using the labeled data to evaluate the results and determine an appropriate threshold for anomaly detection.

K-means

We performed preprocessing, including undersampling, data standardization and principal component analysis. We varied the number of clusters (k) to assess its impact on the system's performance. Below are steps in our methodology:

1. Data Preprocessing:
    a. First, we undersampled the "neptune" and "smurf" attacks, as mentioned above.
    b. We scaled the data to ensure that each feature contributed proportionally to the distance calculations.
    c. Because the original dataset contained a large number of features (119), we performed PCA to reduce the dimensionality of the data to 10 components.
2. Train, Test, Split: Simulate how the model performs with new data by splitting the dataset into a training set (70%) and testing set (30%).
3. K-means Clustering: We applied the k-means algorithm with a range of cluster sizes (k=2 to k=11) to identify natural groupings in the data on the training set (**X_train** and **y_train**). The steps in the k-means clustering algorithm are
    a. Randomly select k data points from the dataset as initial cluster centroids.
    b. Assign each data point to the cluster whose centroid is closest using Euclidean distance.
    c. Recalculate the centroids of the clusters based on the mean of the data points assigned to each cluster.
    d. Repeat the assignment and update steps until convergence or a specified number of iterations is reached.
    e. Convergence is often determined by checking if the cluster assignments and centroids remain unchanged between iterations.
4. Distance Calculation: For each data point, we calculated the Euclidean distance to its assigned cluster centroid. The Euclidean distance is a common choice due to its suitability for high-dimensional data and its intuitive geometric interpretation. The distances were stored in an array, capturing the proximity of each instance to its assigned cluster centroid. This distance metric served as a basis for determining anomalies.
5. Threshold Setting: Anomaly thresholds were set for each cluster based on statistical measures, specifically the mean plus two times the standard deviation of the distances between data points and assigned centroids. This statistical measure provides a robust threshold that considers the spread of distances between data points and centroids.
6. Anomaly Identification: Instances with distances greater than this threshold were considered to be anomalies. This results in a binary classification of instances as normal or anomalous based on their proximity to the cluster centroids.

7. Model Evaluation: We evaluated the performance of the anomaly detection system using various metrics, including accuracy, precision, recall, and F1 score. Additionally, confusion matrices and classification reports were generated to provide a detailed breakdown of model performance.
8. Finding Optimal K: We plotted the evaluation metrics for a range of cluster sizes (2 to 11) and observed the change in evaluation metrics as k varied. We determined the optimal k cluster based on the training set.
9. Evaluation Performance on Test Set: We assessed the generalizability of our model by performing steps 3-7 on the test set (**X_test** and **y_test**).

GMM

Gaussian Mixture Model (GMM) is a probabilistic model that represents the KDD Cup 1999 dataset as a mixture of multiple Gaussian distributions. Compared to other clustering models like K-means, GMM allows for a more flexible assignment by applying a probabilistic value to each cluster for each data point. GMMs ability to capture complex patterns in data made it a strong candidate for this project.

After initial data preprocessing discussed above, the feature data was further scaled to have zero mean and unit variance. Then, PCA was run to reduce the dimensionality of the dataset to 10 components. The first two principal components were evaluated to ensure they included most of the variance of the dataset and visualized. Before running GMM, we did a train test split on the data where test data made up 20% of the full dataset.

GMM was initialized with 2 components or clusters, a full covariance matrix, and random seed for reproducibility. The model was then fit to the feature training set and evaluated. The optimal number of components were assessed and evaluated further. For example, log likelihood of the model was calculated to assess how well the model fits the data. Additionally, accuracy, precision, recall and F1 scores were calculated comparing the predicted values back to the test set. This process was then repeated for undersampled data.

Autoencoder

Autoencoders perform anomaly detection by compressing the original data into a low dimensional space. An autoencoder is a combination of two neural networks, an encoder and decoder network. The encoder compresses the data to a “bottleneck” layer of neurons in

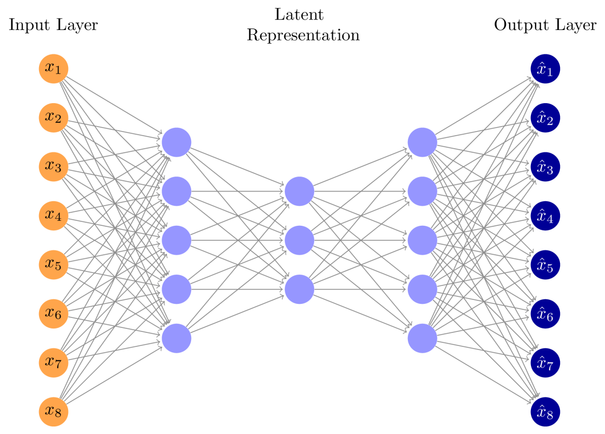
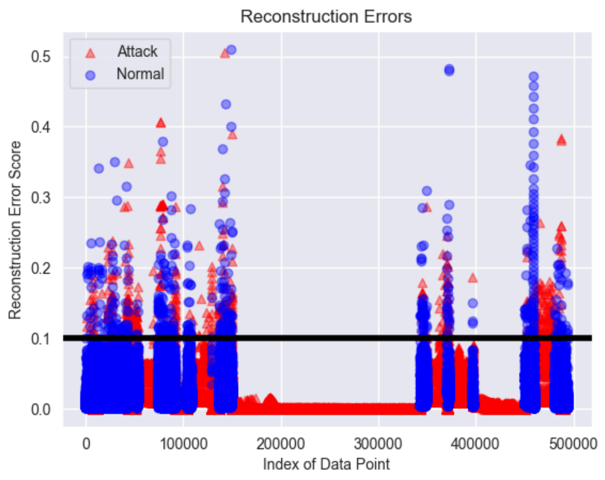
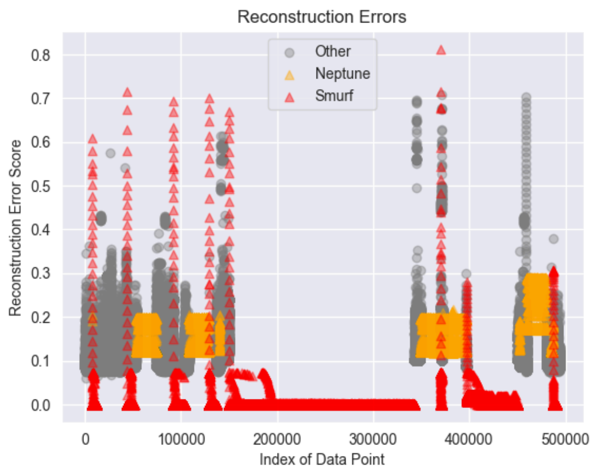


Image from: https://tikz.net/autoencoder

the latent space (similar to PCA). The decoder network then learns how to reconstruct the compressed data back to the original dimensionality. Autoencoders use lossy compression, meaning that the reconstructed data will not be the same as the input dataset because we lose some information during data compression. Anomalies are detected by calculating the reconstruction error of each data point. This metric can be calculated as the Euclidean distance between the input data point and the reconstructed data point but can also be evaluated by the sum of the incorrectly reconstructed features for each data point. More anomalous data will have higher reconstruction errors because the model will only learn the patterns exhibited by normal data. In order to flag data points as anomalies, we used a threshold to determine the appropriate decision boundary to classify points as anomalies or normal data.





Performance from original dataset before undersampling “smurf” and “neptune”

The labeled data provided a good benchmark to evaluate how the unsupervised models performed on known network attacks and if they can identify some types of attacks better than others. We hypothesized that this will create issues for autoencoders that are trained to recognize anomalous data. The model would certainly learn patterns associated with these attacks and be able to reconstruct the data points leading to poor model performance.

When we examine the data labels and plot reconstruction errors, we want to see normal data (blue) at the bottom of the graph and attack data (red) at the top. These charts show that the first model did not only learn the patterns for normal activity. As you can see, the autoencoder learned to reconstruct smurf and neptune very well (left plot) and would not classify these attacks as anomalies because they are over represented in the dataset. In general, there is not any separation between attacks and normal data in the y-axis for reconstruction error. Another observation from the base model showed that network attacks and normal data were grouped together by index throughout the dataset. This could create issues when training the model because the autoencoder uses batches of data to train the model and could cause overfitting to anomalous data. To account for these issues, we undersampled as mentioned in the preprocessing steps above and shuffled the dataset to ensure that labels would not be grouped together.

In addition, we adjusted the model parameters by increasing the dropout rate of each layer in the encoder and decoder models from 0.0 to 0.8. Dropout specifies the probability of switching off a neuron in each layer. In theory, this would prevent overfitting and create a more generalized model because the autoencoder is forced to learn more patterns rather than overfitting to a single dominant path in the neural nets.

The final model contained 5 encoder layers to compress the data linearly from 115 features to 5 features at the bottleneck. It then reconstructed the data points with a symmetric 5 layer decoder network.

<u>Autoencoder Architecture (Encoder & Decoder layers)</u>

Encoder Training:

```
 encoder((encoder_L1): Linear(in_features=115, out_features=50, bias=True)

  (encoder_R1): LeakyReLU(negative_slope=0.4, inplace=True)

  (encoder_L2): Linear(in_features=50, out_features=25, bias=True)

  (encoder_R2): LeakyReLU(negative_slope=0.4, inplace=True)

  (encoder_L3): Linear(in_features=25, out_features=10, bias=True)

  (encoder_R3): LeakyReLU(negative_slope=0.4, inplace=True)

  (encoder_L4): Linear(in_features=10, out_features=5, bias=True)

  (encoder_R4): LeakyReLU(negative_slope=0.4, inplace=True)

  (dropout): Dropout(p=0.8, inplace=True))
```

Decoder Training:

```
decoder((decoder_L4): Linear(in_features=5, out_features=10, bias=True)
 (decoder_R4): LeakyReLU(negative_slope=0.4, inplace=True)
 (decoder_L1): Linear(in_features=10, out_features=25, bias=True)
 (decoder_R1): LeakyReLU(negative_slope=0.4, inplace=True)
 (decoder_L2): Linear(in_features=25, out_features=50, bias=True)
 (decoder_R2): LeakyReLU(negative_slope=0.4, inplace=True)
 (decoder_L3): Linear(in_features=50, out_features=115, bias=True)
 (decoder_R3): LeakyReLU(negative_slope=0.4, inplace=True)
 (dropout): Dropout(p=0.8, inplace=True))
```

Evaluation

Due to the highly imbalanced nature of this dataset and given the large number of "normal" instances, we paid careful attention to metrics that account for imbalances, such as precision and recall. The imbalanced dataset will impact the interpretation of certain metrics like accuracy because accuracy does not appropriately account for the class imbalance. In particular, a model could theoretically achieve high accuracy by simply predicting the majority class (normal) all the time while neglecting the minority class (anomalies/attacks).

Precision, recall, and F1 score are metrics that specifically address the model's performance on the minority class, providing insights into its ability to correctly identify anomalies. To calculate these metrics, we need to understand the components of the calculations, which is also used to interpret the confusion matrix and classification report:

- True Positive (TP): Anomalies correctly identified as anomalies.
- True Negative (TN): Normal instances correctly identified as normal.
- False Positive (FP): Normal instances incorrectly identified as anomalies (Type I error).
- False Negative (FN): Anomalies incorrectly identified as normal (Type II error).

In the following section, we discuss the definitions and tradeoffs between precision, recall, and F1 scores as evaluation metrics for k-means, GMM, and autoencoders. Precision focuses on the accuracy of positive predictions, recall emphasizes the ability to capture all positives, and F1 score provides a balance between precision and recall. In the context of network intrusion detection,

- Precision: Out of all the instances that were predicted/identified as anomalies, how many were actually anomalies.

- Recall: Out of all the true anomalies, how many did we correctly predict/identify as anomalies.
- F1: Harmonic mean between precision and recall
- Accuracy: Out of all the predictions, how many were correctly predicted/identified.

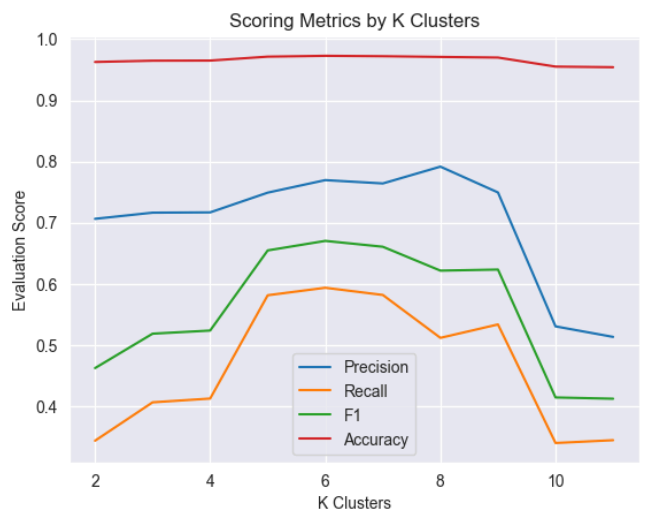
In the context of network intrusion detection, a false negative means that the intrusion detection system fails to detect an actual intrusion or attack. An intrusion that is missed can lead to severe security breaches, unauthorized access, and data theft. The attacker can exploit vulnerabilities and compromise systems that safeguard sensitive information, intellectual property, and user privacy. False negatives can have costly consequences, including financial losses, reputation damage, legal repercussions, and disruption of critical services. Minimizing false negatives is a “better safe than sorry” approach where an organization emphasizes a proactive approach to threat mitigation and early detection of intrusions even if they must tolerate more false positives. In this case, the organization would focus on precision scores.

On the other hand, specific priorities may vary based on an organization’s risk tolerance, nature of the network, and criticality of the systems being protected. There are circumstances or models where it is more appropriate to emphasize minimizing false positives, as discussed in autoencoder evaluation methods below.

K-means

For k-means, we iteratively evaluated the performance of the model for **k** number of clusters 2-11. We calculated the precision, recall, and F1 scores to evaluate the performance of the k-means clustering algorithm and stored the scores in the table below. The plot of the evaluation metrics by number of clusters for the training set is also included below.

K Clusters	**Accuracy**	**Precision**	**Recall**	**F1 Score**
2	96.26%	70.63%	34.36%	46.23%
3	96.47%	71.63%	40.64%	51.85%
4	96.49%	71.68%	41.25%	52.37%
5	97.13%	74.90%	58.14%	65.46%
6	97.27%	76.95%	59.36%	67.02%
7	97.20%	76.41%	58.18%	66.06%
8	97.08%	79.16%	51.17%	62.16%
9	96.98%	74.93%	53.36%	62.33%
10	95.50%	53.05%	33.99%	41.43%
11	95.40%	51.33%	34.44%	41.22%



Based on the above plot, we conclude that the optimal number of clusters might be in the range of 5 to 7, where performance metrics exhibit a balance between precision, recall, and F1 score. Regarding accuracy, the results are expected to be very high because of the imbalanced nature of our dataset, with 95% of the data as normal records, so we've focused on precision, recall, and F1 score, rather than accuracy.

After narrowing down to precision and recall, we believe that when in the context of k-means in building a network intrusion detection system, it's more important to minimize false negatives, as discussed above. In other words, we aim to minimize falsely labeling a normal record as an attack. Therefore, we have focused more on recall, which is a ratio of the number of correctly predicted anomalies out of all the actual anomalies. In other words, we're optimizing for a model that's good at finding all the positives/anomalies to ensure system security. Since recall is maximized at k=7 while also showing a reasonable balance between precision, recall, and F1 score, we will use seven clusters in the final model for testing.

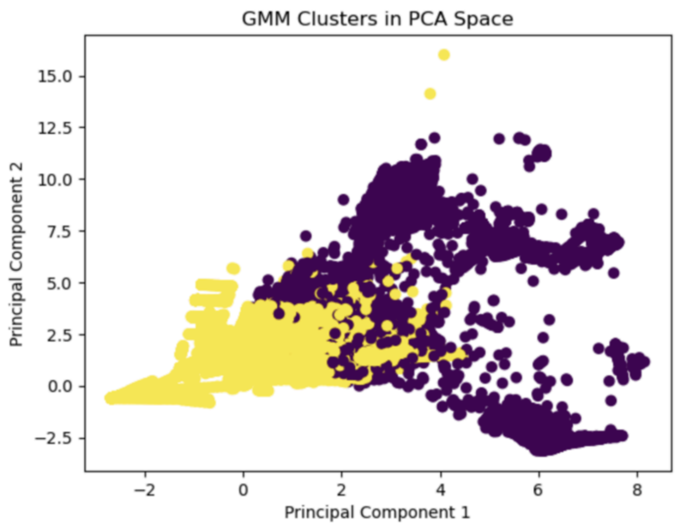
After identifying the optimal k cluster to perform k-means clustering, we evaluate the model and hyperparameters chosen on the new, unseen data to assess the generalizability of our model. We performed steps 3-7 on the test set (**X_test** and **y_test**) i.e. calculated distances for the test set, applied the thresholds, and compared the predictions to the true labels. Below are the results.

Metric	**Train Scores**	**Test Scores**
Accuracy	97.27%	96.41%
Precision	76.95%	78.40%
Recall	59.36%	58.20%
F1 Score	66.02%	66.96%

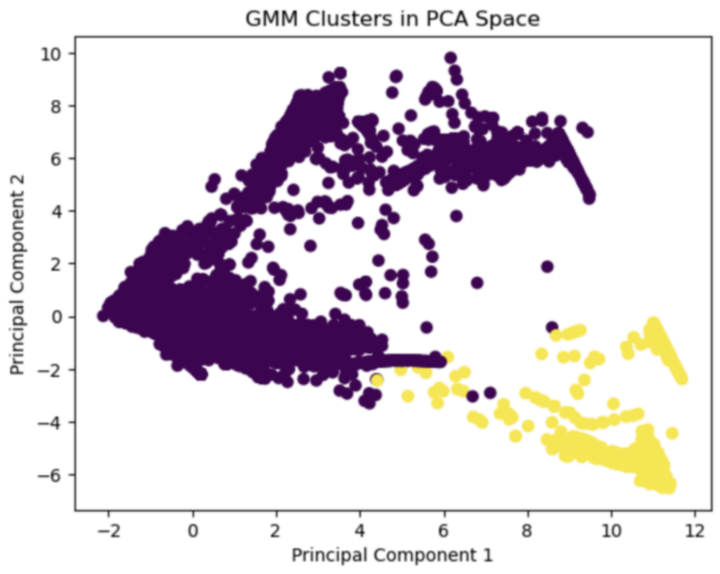
Since accuracy, precision, recall and F1 scores are very similar between the training and test set, the results suggest that the k-means model is performing consistently and has learned patterns in the training data that generalize well to unseen data and it is not overfitting to the training data. This is a sign that the model is stable and is not overly sensitive to variations in the data and the model.

GMM

Initially, looking at the PCA charts, we can see a clearer distinction between the normal and anomalous data points between the full dataset model and the undersampled model. The yellow data points represent the anomalies and the purple represent the normal points. Furthermore, looking at the initial log-likelihood of the models (directly under the PCA visuals), it is clear that the undersampled model has a larger log likelihood value indicating that it represents the data better (-44 is a better score compared to -13).



```
Log-Likelihood: -13.158496584993363
```



```
Log−Likelihood: −44.2833345052347
```

Various numbers of components were evaluated when running the GMM model and based on the evaluation metrics, it was determined that 2 components were the best. The additional other component over 2 only decreased the evaluation metrics as seen in the example below.

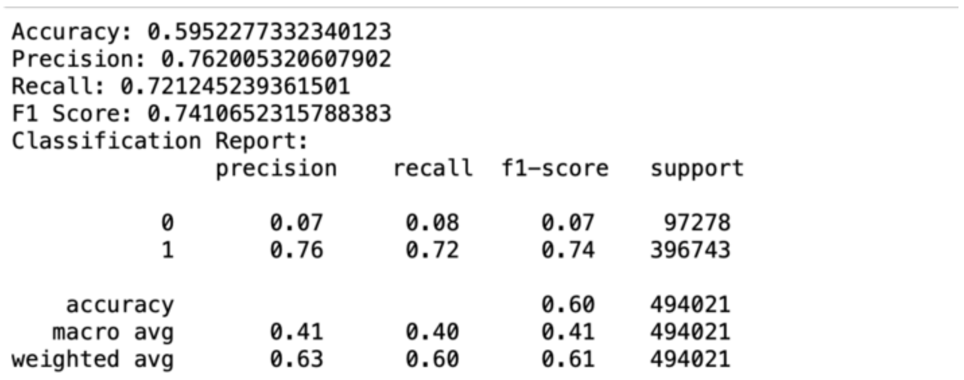
```
Accuracy: 0.5673625210264341
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00     97278
           1       0.97      0.71      0.82    396743
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0

    accuracy                           0.57    494021
   macro avg       0.24      0.18      0.20    494021
weighted avg       0.78      0.57      0.66    494021
```

Comparing the two final models, the full model and the undersampled model, we see that while the full model's accuracy and precision were low, their recall and F1 score were high. However we can see from graphing the GMM model above in PCA space that the undersampled model has a better representation of the data and precision is almost perfect.

Full Model:

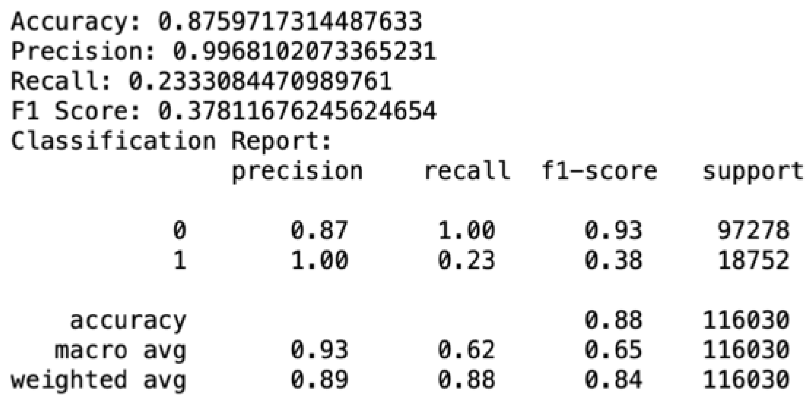


```
Accuracy: 0.5952277332340123
Precision: 0.762005320607902
Recall: 0.721245239361501
F1 Score: 0.7410652315788383
Classification Report:
              precision    recall  f1-score   support

           0       0.07      0.08      0.07     97278
           1       0.76      0.72      0.74    396743

    accuracy                           0.60    494021
   macro avg       0.41      0.40      0.41    494021
weighted avg       0.63      0.60      0.61    494021
```

Undersampled Model:



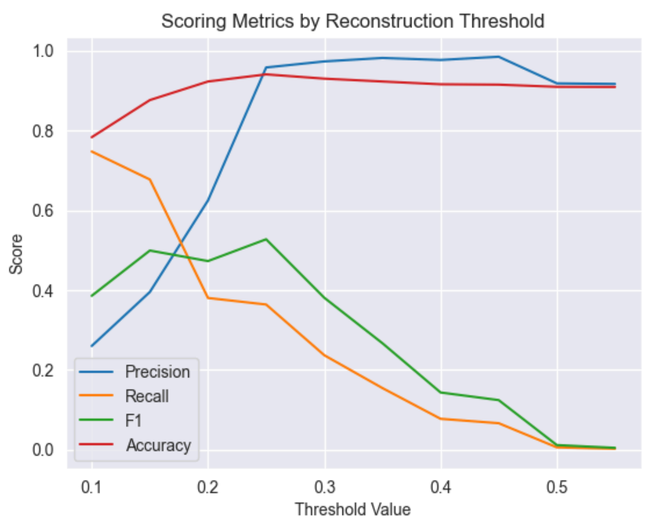
```
Accuracy: 0.8759717314487633
Precision: 0.9968102073365231
Recall: 0.2333084470989761
F1 Score: 0.37811676245624654
Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93     97278
           1       1.00      0.23      0.38     18752

    accuracy                           0.88    116030
   macro avg       0.93      0.62      0.65    116030
weighted avg       0.89      0.88      0.84    116030
```

Autoencoder

For autoencoders, we set a predefined threshold for reconstruction error, such that any data point exceeding this threshold will be labeled as an anomaly/attack. Using the reconstruction error of each datapoint, we check if each datapoint exceeds the threshold and calculate the performance metrics for the overall dataset. Although the autoencoder model is unsupervised, we used the labeled data to evaluate which threshold would be best for flagging anomalies in the sampled training data.



Threshold with highest F1 Score: 0.25 with Score - 0.53

Threshold with highest Precision Score: 0.45 with Score - 0.98

Threshold with highest Recall Score: 0.1 with Score - 0.75

Threshold with highest Accuracy Score: 0.25 with Score - 0.94

For the best model, we decided that a reconstruction threshold of 0.25 was appropriate because it had the highest F1 score while maintaining a very high Precision score of 0.95. With the autoencoder, it appears to be very difficult to separate attacks from normal data when the size of the attack data points is large. Thus, it is more important to consider minimizing false positives. In a real world scenario, where all data labels are unknown, any data flagged as anomalous by this model would need to be manually reviewed or you would respond with an automated action (ex. immediately locking a user out of the system). In both cases, you would prefer to have high precision so that you don't waste time reviewing false positives or you don't accidentally take action against a normal user. With the size of the attack data, if recall were maximized, the threshold would classify most of the dataset as anomalous.

Results and Discussion

K-means

Classification Report	Precision	Recall	F1-Score
Normal	0.97	0.99	0.98
Anomaly	0.78	0.58	0.67

- Precision: Out of all the instances that were predicted/identified as anomalies, 78% were actually anomalies.

- Recall: Out of all the true anomalies, we correctly predicted/identified 58% as anomalies.
- F1: Harmonic mean between precision and recall is 67%.
- Accuracy: Out of all the predictions, 97% were correctly predicted/identified.

GMM

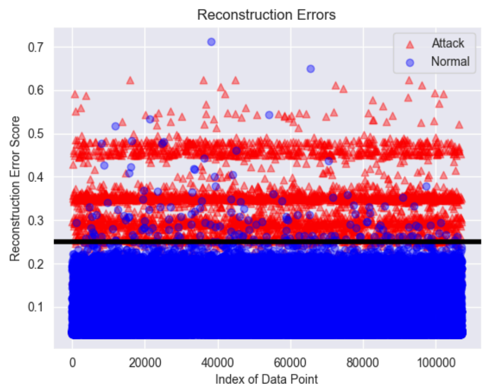
Undersampled data worked best with an overall accuracy of 88%. The model performed the best with identifying normal behavior with a 100% recall. Despite the low recall (23%) for anomaly, the model overall had a high rate of precision at 100%. The strong performance for the normal classification is important in anomaly detection since ideally you do not want to falsely flag an anomaly.

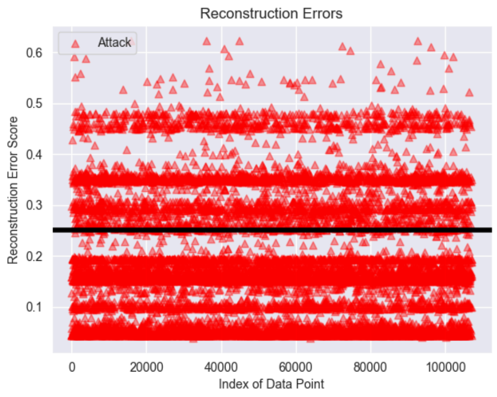
Classification Report	**Precision**	**Recall**	**F1-Score**
Normal	0.87	1.00	0.93
Anomaly	1.00	0.23	0.38

While the undersampled model's results are a good start, there is room for improvement. In the future, we could explore more data transformations to help improve recall.

Autoencoder

In the charts below, the black line denotes the decision boundary where any data points above the threshold would be classified as anomalies.





While only 37% of attacks were flagged as anomalies (recall), the model did a good job of separating some of the attack data from the normal user activity. The 95% precision score means that the data points that are flagged have a very high probability of being a network attack. With more tuning and down sampling the different attack types this model may achieve improved performance.

Classification Report	**Precision**	**Recall**	**F1-Score**
Threshold=0.25	0.95	0.37	0.53

Conclusion

Overall, the results do not necessarily meet all of our hypotheses. We expected the autoencoder model to significantly outperform K-means and GMM. However, our findings suggest that autoencoders may not be the best models for this particular dataset, This is partially because it appears that autoencoders can be very sensitive to groups of anomalous data. Even in a highly imbalanced dataset, autoencoders appear to learn patterns associated with small groups of anomalies and pass them as normal data. Because of this, recall does not appear to be the best metric to evaluate performance with this model. Instead, this type of model seems to be best suited for situations where precision is favored over recall.

Clustering models, on the other hand, are more robust at separating groups of anomalies in the data and offer more promising results when trying to catch all attacks in this dataset. In most network intrusion detection use cases, it would be more appropriate to use a clustering model with higher recall, such as the k-means model, to optimize finding all the positives/anomalies and ensure system security. Autoencoders would likely have high recall only if attacks were unique and very infrequent. We would choose an Autoencoder if we prioritized precision while still finding a good balance between precision, recall, and F1.

References

1. *https://tikz.net/autoencoder*
2. https://colab.research.google.com/github/GitiHubi/deepAI/blob/master/GTC_2018_CoLab.ipynb#scrollTo=weTTwKn6rdGb
3. https://arxiv.org/abs/1709.05254
4. https://towardsdatascience.com/anomaly-detection-how-to-tell-good-performance-from-bad-b57116d71a10

5. https://itsudit.medium.com/navigating-the-precision-recall-tradeoff-understanding-f1-score-544d481f1393

6. https://stackoverflow.com/questions/33382619/plot-a-horizontal-line-on-a-given-plot

7. https://hersanyagci.medium.com/random-resampling-methods-for-imbalanced-data-with-imblearn-1fbba4a0e6d3

8. https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html

9. https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html#imblearn.under_sampling.RandomUnderSampler.fit_resample

10. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

11. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

12. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/

13. https://medium.com/analytics-vidhya/building-an-intrusion-detection-model-using-kdd-cup99-dataset-fb4cba4189ed