Network Anomaly Detection Using Probabilistic Sketch Data Structures

EN.601.714 - Final Project

Omar Ahmed

Contents

1 Abstract

Networks have been facing increasing levels of security threats from attackers using a wide array of attack strategies [2]. These security threats can have serious implications given the crucial role that datacenters play in today's internet where they service many of the requests from users nowadays. Therefore, quickly detecting these security threats is a needed ability to ensure organizations can keep their networks up and running anomaly-free for as much as possible. Previous approaches such as sampling can be biased by frequent items in the dataset, and therefore are not an optimal solution. My project proposes and implements an anomaly detection method for security threats that is based on sketching data-structures (MinHash and HyperLogLog). The key intuition of this project is that by using sketches to capture "normal" and "anomalous" behavior, we can quickly classify network traffic based on how similar it is to our reference sketches. Our results show that using this approach allows you to achieve $>90\%$ accuracy in classifying windows of traffic as either normal or anomalies. Furthermore, our approach shows promising results in estimating the amount of anomalous behavior within a time window of traffic data.

The repository containing the source code, experiment and analysis scripts can be found at the following link: https://github.com/oma219/pacsketch.

2 Introduction

Network datacenters play a crucial role in the internet ecosystem by handling a majority of the user requests. Common user requests such as the checking of your email or streaming video content typically are processed and sent back to the user at the datacenters of these large content providers. Therefore, it is crucial these networks stay up and running for as much as possible to avoid losing money due to network outages. One potential threat to these large networks as well as smaller networks such as college or military networks are security anomalies or attacks.

There are two types of network anomalies: performance and security anomalies. In this project, we will be focusing on security anomalies which fall into three different categories that include point, collective, and contextual anomalies [3]. Security anomalies essentially consist of malicious user activity that can be focused on stealing information or possibly taking down a service. Therefore, it is clear that companies and organizations would want to ensure they can prevent or detect these security threats as much as possible.

This motivation leads us to the field of network anomaly detection where the goal is to detect network anomalies as accurately as possible to be able to react appropriately. The problem of network anomaly detection can be formalized in the following terms, $S = (M, D)$, where S is the anomaly detection engine, M is the model of normal behavior, and D is the proximity measure to distinguish normal and anomalous behavior. Previous approaches have been developed and fall into a wide array of categories such as classification, statistical, clustering, and information-theory based approaches [3]. A select number of these approaches will be discussed in the following section.

3 Related Work

This section will discuss some of the related methods that also perform network anomaly detection. My project was partially motivated by some of the shortcomings of these methods.

3.1 Classification-Based

Classification-based methods for network anomaly detection rely on storing some model of normal behavior that be used to classify the current data. This can be one of the shortcomings of these approaches given the ever-changing landscape of network traffic, so it can be hard to keep a "normal" model up-to-date [2]. Nevertheless, these types of approaches have been shown to be effective in classifying anomalous behavior such as a previous approach using Support-Vector Machines (SVM) [6]. This approach uses a Support-Vector Machine to find the hyperplane with the maximal margin to separate normal and anomalous behavior. Therefore,

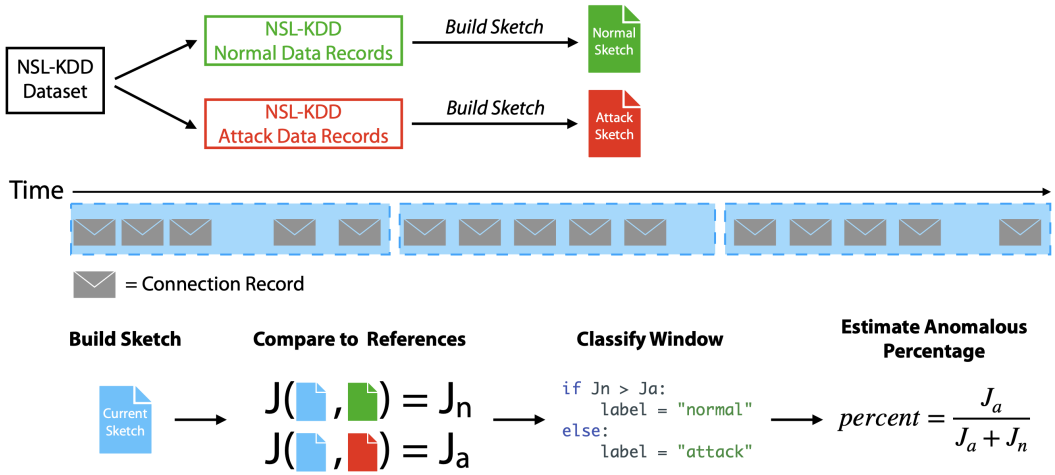


Figure 1: Shows the overall approach of Pacsketch where it starts by building the reference sketches, and then compares the test windows to the reference sketches, and estimates the percentage of anomalous content.

this allows the users to quickly distinguish the input data item based on which side it falls with respect to the hyperplane.

3.2 Clustering-Based

An additional class of solutions are clustering-based methods. These approaches allow you to use the unsupervised aspect of clustering to avoid labeling each data item but rather let the clustering method naturally try to separate the different classes. One such approach uses k-means clustering to separate out normal and anomalous data [8]. Therefore, to classify data items, each item can be plotted into the k-means space, and compared with the centroid of each cluster, and whichever cluster it is closer to becomes the classification.

3.3 Statistical-Based

Similarly to classification-based solutions, statistical-based solutions rely on some model of normal behavior, and in this case, it is a statistical distribution. Then, using test statistics to determine if the observation is similar or not to what we expected [2]. An example of a solution in this field is an approach that uses Principal Component Analysis (PCA) to reduce a high-dimensional data to small set of dimensions, which can then be converted to score based on the resulting components. This score can then based converted into a classification by using a threshold [10].

3.4 Sampling

Lastly, another key approach used for performing anomaly detection in practice is this idea of sampling where the network operators sample a small number of data items from their traffic flow to be able to determine if there are anomalies present. There has been previous work that has mentioned for example, in Cisco's NetFlow software, they use a sampling rate of 1 in every 256 items to compute traffic-related measurements [11]. This sampling approach is just a way to reduce the dataset you are looking at, and therefore, it could be combined with any of the network anomaly detection methods mentioned in the previous sections to detect anomalies.

However, sampling has an interesting theoretical comparison with sketching which will be discussed in later sections. One of the weaknesses of sampling-based approaches are that they could be influenced by frequent items in the stream. In this case, it could the presence of many identical normal records. Therefore, this weakness helps to motivate using sketching-based approaches that allow you to similarly reduce the memory needed to solve

this problem, while not suffering from frequent item bias.

4 Dataset

The dataset used in this project is called NSL-KDD [1]. This dataset was derived from the KDD-Cup99 dataset which originally consisted of millions of connection records from US Air Force LAN data. Each record consisted of 41 features where 34 were real values, and 7 were categorical variables. These features typically characterized the connection present between two machines in this network, and some of the example features are duration, bytes sent, and number of shell prompts used. The NSL-KDD improves upon its predecessor, KDD-Cup99, by removing redundant records, balancing difficulty level, and making it a more manageable size for performing analysis [1]. Each record in this dataset is labeled as either normal or it is labeled with a specific attack type. There are 24 distinct attack types included in this dataset.

4.1 Pre-processing

The raw data from the NSL-KDD could not be used directly due to the mixture of numerical and categorical features present in each record. One of the key ideas for using these sketch data-structures is that you have to input an item into them from a set of discrete, possible items. That is what allows you to find common items between two different sketches when you compute similarity measures such as jaccard. Therefore, to input these NSL-KDD items into the sketch data-structure, I had to start by discretizing all the features so there were no longer any real number features. The approach used for performing this conversion was by converting each number into a one of eight labels (1-8), and it was based on the z-score for that particular value. The exact boundaries between each label can be viewed by looking at the *analyze_dataset.py* script in the code repository. Then, after discretizing each real number feature, each record is composed into a single item by concatenating all the features and joining them by using a "_" character. This means each record can be represented as a single string that can then be input into the sketch data-structure.

5 Methods

In this section, I will detail the methods used in my network anomaly detection approach. I will start by discussing the intuition behind the sketching data-structures used in my approach, and then detail how they are used to classify windows of traffic data from the NSL-KDD.

5.1 MinHash

The MinHash data-structure was first developed by Andrei Broder as an approach to quickly compare websites to find duplicate websites for the AltaVista search engine [4]. The key intuition of MinHash is that if you have a dataset of items, and you apply a hash function to each item, and visualize them in terms of the range of potential hashes as shown in Figure 2, they will roughly be uniformly distributed. This is a reasonable assumption if you assume the hash function is uniformly placing each input item along this range. Therefore, if we assume the values are being uniformly distributed, it means we can estimate the cardinality of the set by keeping track of the minimum hash value. For the example shown in Figure 2, where the hash value range is from 0 to 1, the cardinality can be computed by the following formula:

$$E[X] = \frac{1}{min\_hash}$$

In practice, only using the smallest hash value can lead to noisy cardinality estimates. This is why typically tools in practice that use MinHash such as Mash which is widely used for comparing genomes [9] typically use the kth smallest hash. This has the effect of essentially averaging the random estimates instead of just relying on the smallest hash.

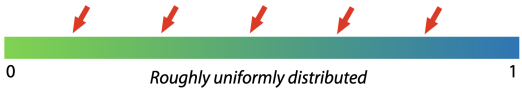


Figure 2: Diagram shows the expected behavior of hashes when looking at the total range of potential hashes.

The last key idea regarding MinHash is understanding how to compare two MinHash sketches.

This can be done by computing the jaccard similarity coefficient which is computed as shown below:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Therefore, we need to be able to generate the intersection and union of two MinHash sketches, and luckily it is rather straightforward. As previously mentioned, our MinHash sketch for a set consists of the k smallest hashes. Therefore, to compute the intersection sketch, we determine which values occur in both sketches, and to determine the union sketch, we determine how many unique values there are among the two sketches. Then, the ratio of those two values will be the jaccard between the two sets.

5.2 HyperLogLog

Another widely used cardinality estimation datastructure is the HyperLogLog which is based on the concept of probabilistic counting [7]. The key intuition behind the HyperLogLog is if you hash all the items in a set and look at the leading zero count in the hash value, and keep track of the largest value, then you can estimate the cardinality. The reason the leading zero count is relevant is because we can think of each zero as a coin flip with a 50% probability of occurring. Therefore, if we see leading zero count of x, we can say there was a $(1/2)^x$ chance of seeing this value which gives a sense of how many unique items we have seen (i.e. the cardinality).

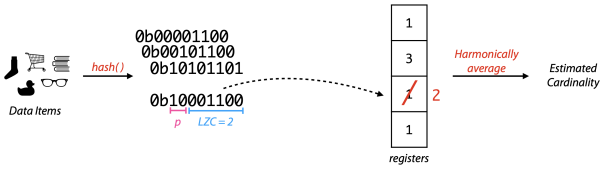


Figure 3: Diagram shows the steps taken in adding an element to a HyperLogLog sketch.

The exact workflow for how items are added to the HyperLogLog sketch can be viewed in Figure 3. Each data item is hashed and the first p bits tell us which register the value will be directed to, and then we compute the leading zero count of the rest of the hash value. If that leading zero count is larger than the current value in the register, then we will replace it. After adding all the data items, we can harmonically average all the registers to get an estimated cardinality.

Lastly, in order to compute the jaccard between two HyperLogLog sketches, instead of computing the intersection sketch as I did with MinHash, I instead used the inclusion-exclusion principle as shown below:

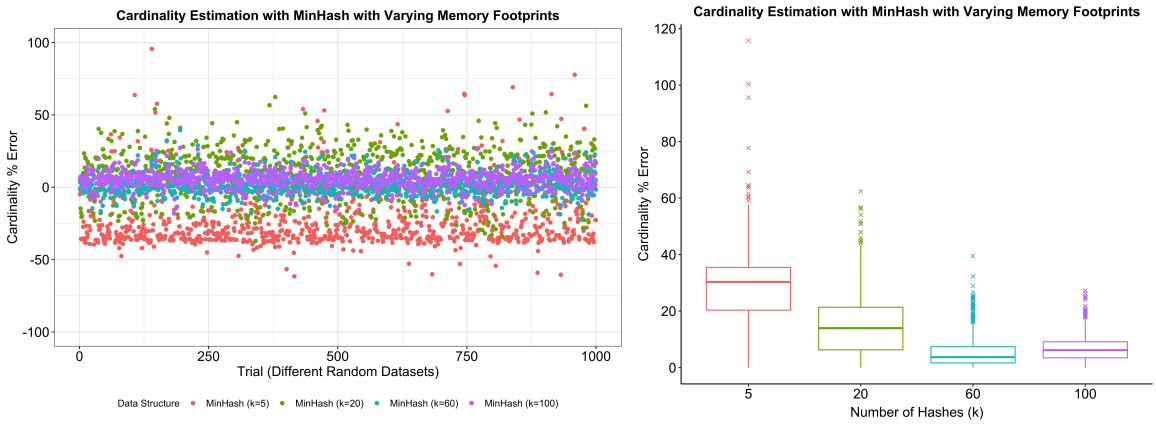
$$J(A,B) = \frac{|A| + |B| - |A \cup B|}{|A \cup B|}$$

The reason for doing this is because performing the union of two HyperLogLog sketches is rather trivial by just taking the maximum value at each register between the two sketches. The intersection on the other hand is non-trivial, and it was developed much later by Otmar Ertl [5]. However, by using the inclusion-exclusion principle, we can compute the jaccard between two HyperLogLog sketches.

5.3 Pacsketch

In this section, I will detail my network anomaly detection method which is called Pacsketch. The goal of Pacsketch is to be able to detect security anomalies in US Air Force LAN traffic data, which is the type of data in the NSL-KDD. There are two main metrics that Pacsketch can be evaluated on which include the binary classification ability for each time window, and the estimated percentage of anomalous activity in each window.

The overall workflow for Pacsketch can be viewed in Figure 1. The method starts by building a model of what normal and anomalous behavior looks like by splitting the records into those groups, and building sketches over them. Then, Pacsketch can start to classify real test data by first splitting the data into windows of records. This would be analogous to a situation in real-life where maybe the network operator gets data in batches, and wants to compute whether there was any anomalous activity in that time window. To continue, Pacsketch will build a sketch over each time window, and compare it to the reference sketches for normal and anomalous behavior and get two jaccard values. Then, Pacsketch can simply binary classify the window as



(a) cardinality percent error

(b) absolute cardinality percent error

Figure 4: Shows the cardinality for the MinHash data-structure as we vary the parameter k which stands for number of hashes kept. As we store more hashes, our cardinality estimates improve.

either ”normal” or ”anomaly” based on which jaccard is the largest. An **important** note to make is that this binary classification label just tells the user that a majority of the traffic in this window seems to be either normal or anomalous. The second value that Pacsketch computes the estimated percentage of the data in this current window that is anomalous, and it does this by taking the ratio of the jaccard with respect to the attack sketch to the sum of the two jaccards. This measure gives the user some estimate of just how much anomalous activity is present in the window since the binary label only tells you what the majority of the traffic is.

6 Evaluation

In this section, I will detail the evaluation I cared out on my method called Pacsketch. At a high level, the approach I took was to test out my data-structures, and then I went to dataset analysis, and then testing out my Pacsketch method on test data.

6.1 Cardinality Estimation

The first evaluation I conducted on my data-structures was to test their ability to estimate the cardinality of a multi-set. For generating random datasets with known cardinality, I generated random FASTA files which is the file format for DNA sequences. I used this format since it easy to generate and easy to compute the known cardinality of substrings which are called kmers since they have a length of k letters. The evaluation I performed consisted of simulating 1,000 random datasets with known cardinality and feeding them into each data-structure, and estimating its cardinality. Figure 4 shows the percent error in both (a) a scatter plot and (b) box plot form, and we can see as store more hashes, our estimates are more accurate which is consistent with what we would expect. The results for the HyperLogLog are similar but the parameter that we are varying is b which represents the number of registers in our sketch.

6.2 Jaccard Estimation

The next evaluation I conducted was testing the ability of the MinHash and HyperLogLog data-structures to estimate the jaccard between two multi-sets. Similar to the previous evaluation, I generated random pairs of FASTA files, and computed their exact jaccard similarity. Then, I would feed those into each of the sketching data-structures, and estimate their jaccard.

As we can see in Figure 5, the jaccard estimates improve as we store more hash values, and at k = 100, most of jaccard estimates are within 0.1 of the true jaccard. Once again, this plot was similar for the HyperLogLog as you increase the value of b or the number of registers. The only difference worth mentioning is that typically with the HyperLogLog there are more outliers in terms of the jaccard dis-

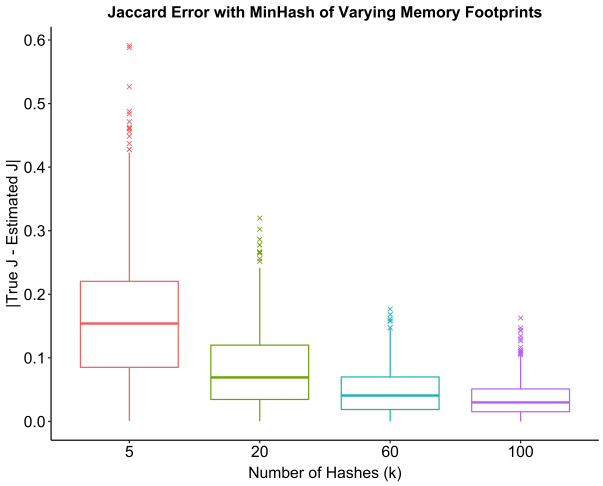


Figure 5: Shows the jaccard estimation error for MinHash as we vary the number of hashes we retain. Similar to cardinality, we can see the error decreases as we store more hashes.

tribution. This could be explained by the small cardinality correction discussed in the HyperLogLog paper [7]. Since the estimator needs to apply corrections when the cardinality is small, it could be that these outliers are due to instances when the cardinality is small and needs a correction that possibly does not do a good job.

6.3 NSL-KDD Feature Analysis

At this point in the evaluation, we have evaluated both of our data-structures and they appear to be working as expected in terms of cardinality and jaccard estimates. Now, I transitioned to working with the NSL-KDD dataset.

Prior to feeding the dataset into the MinHash and HyperLogLog data-structures, I performed some feature analysis prior to ensure that visually we can see a difference in the feature distributions between normal and attack records. In order to do this, I extracted all the normal and attack records from the training dataset, and plotted the feature distributions in each of those groups.

As you can see in Figure 6, there is a clear distinction between the duration feature in normal and attack records. It is important to mention that they y-axis is log-transformed so makes it a little clearer that the two distributions are very different. This similar pattern of differing distributions between normal and attack records was also present in various of the other real numerical features.

6.4 Jaccard Distributions for Test Windows

I started to test out the workflow shown in Figure 1 where I build the two reference sketches, and then on test windows I compare them to each reference sketch. In this experiment, the parameter that I held consistent was the percentage of the records in the window that were security anomalies, that way I would know the exact percentage and binary label for that window (normal or anomaly).

I initially started with test windows where 100% of the records were anomalous since this should be an easy scenario for Pacsketch. It turned out to be so, since the distribution of jaccards with respect the attack sketch was pushed farther to the right while the distribution of jaccards with respect to the normal sketch was essentially centered right around 0.

Next, I tested out a more difficult scenario where the percentage of attack records in the windows was

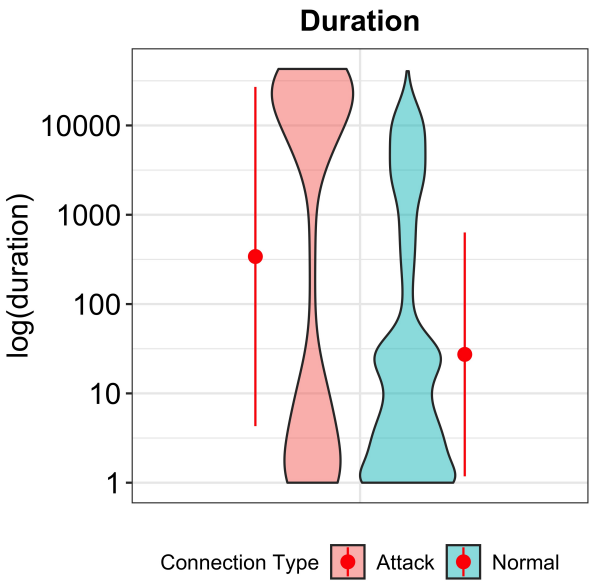


Figure 6: Shows the distribution of duration of the connection for normal and attack records in the NSL-KDD training set. Note the y-axis is log-scale, and the red-dot represents the mean, and each segment is one standard deviation.

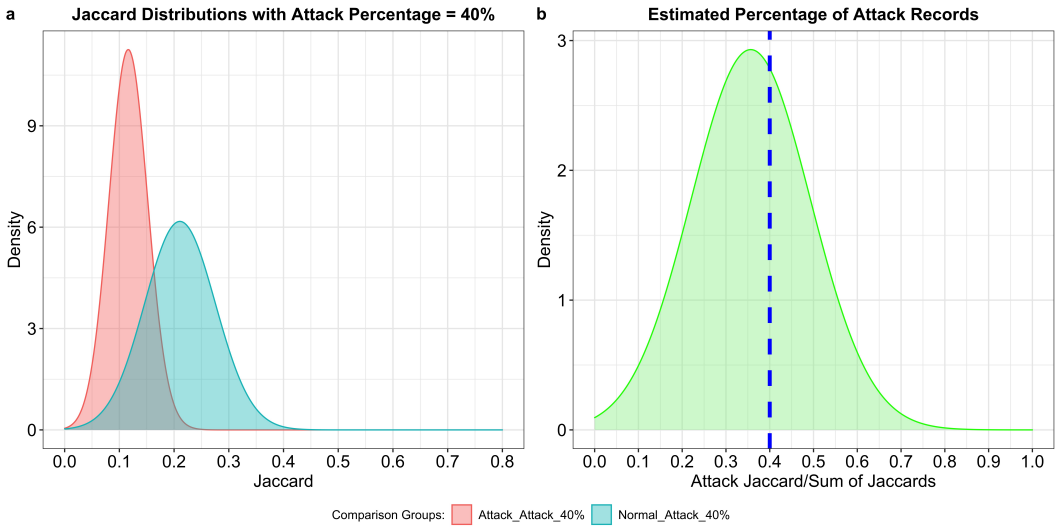


Figure 7: Shows (a) distribution of jaccards with respect to each of the reference sketches, and (b) estimated percentage of the attack records based on the ratio of the jaccards. The blue vertical line represents the true percentage which was 40%.

set at 40%. Therefore, Pacsketch should classify it as a "normal" window since a majority of it is normal records, and estimate the anomalous percentage as around 0.40.

As we can see panel (a) of Figure 7, the distribution of jaccards with respect to the normal sketch is slightly farther to the right than the one with respect to the attack sketch. This makes sense since each test window consists of 60% normal records, and 40% attack records. In panel (b) of the same figure, we can see that Pacsketch's estimates for the anomalous percentages are close to the true value of 40% since the green distribution is nearly centered at 0.40.

Additionally, I tested Pacsketch on binary classifying random windows of records from test data, which included attacks not present in the training set to make it more realistic. The goal was to binary classify each window as either containing a majority of normal or anomalous records. Pacsketch achieved an accuracy of 93.3% and a recall of 92.6%.

6.5 Comparison to Sampling Approach

In this section, I will detail an experiment I performed to compare a sampling-based approach to our Pacsketch approach which is sketch-based. We previously mentioned that sampling-based approaches suffer from frequent item bias.

Pacsketch's two main forms of output are a binary label for each window of data as either normal or anomaly, and an estimate for the percent of anomalous records in a window. Therefore, I decided to compare Pacsketch and the sampling-based approach on the latter of those two metrics. Furthermore, since I did not have a specific sampling-

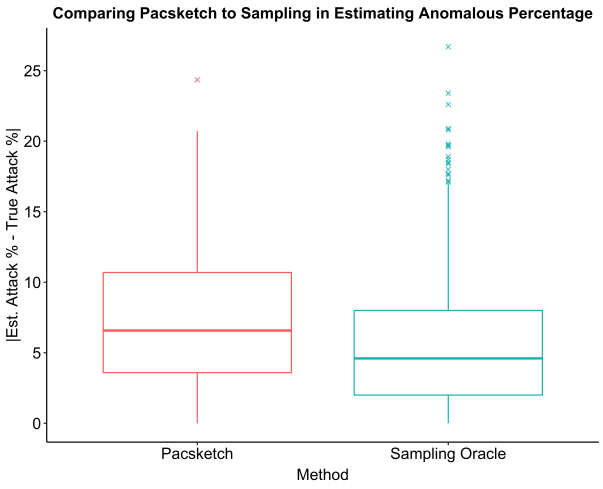


Figure 8: Shows the distribution of anomalous record percentages as determined by the Pacsketch and a Sampling Oracle method.

based method to use, I just assumed that the sampling method had an Oracle who could classify each record with 100% accuracy whether it was normal or not. This is a *very important* point to understand when it comes to this evaluation.

The results are summarized in Figure 8, and we can see that the sampling oracle method does have a slightly lower absolute error. This could be due to the fact that we are just assuming it is an Oracle, and not actually feeding the sampled records into a downstream classifier. It could also be due to the fact that NSL-KDD dataset contains very little duplicate records so it could be lessening the issue with frequent item bias. Regardless, Pacsketch is still quite close to the sampling-based method in terms of its estimates for the anomalous percentage in windows of data.

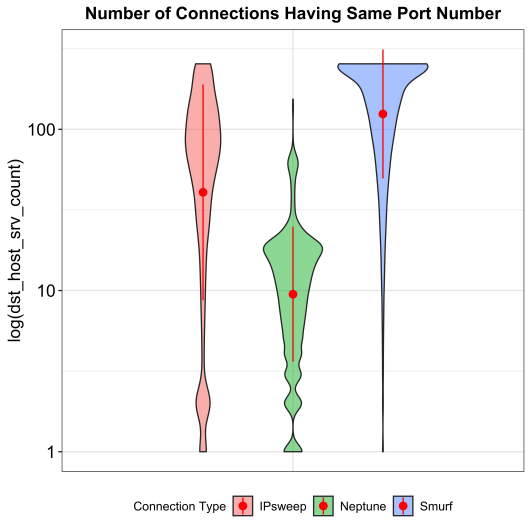


Figure 9: Compares the feature of the number of connections with same port number across three different attack types.

6.6 Differences Between Attack Types

Thus far, we have focused on distinguishing between normal and attack records. However, the NSL-KDD dataset does provide specific attack labels. Therefore, it is theoretical that we could build sketches for specific attack types, and then try to classify the anomaly as a specific type of attack. I chose to focus on three of the most common attack types (IPsweep, Neptune, Smurf) in the NSL-KDD dataset, and see if I can visually see a difference in their feature distributions.

As shown in Figure 9, there does appear to be a difference between these specific attack types. This indicates that it could be possible to use Pacsketch to classify not only that a window is mostly anomalies but also which type of anomalies are present in that window of records.

7 Limitations

There are a handful of limitations with Pacsketch that I discovered as I was working through the project. Firstly, since my approach relies on building a sketch, it does not allow me to go to finer granularity such as classifying individual data items. Pacsketch needs to build a sketch first, and then compare it to the reference sketches. Secondly, during the development, I focused entirely on the NSL-KDD dataset which does have its limitations since it is simulated data, and is quite old since it is from 1999. Ideally, Pacsketch would be tested on various anomaly datasets to show it is a robust approach.

8 Conclusions

This project consisted of developing a method that can detect security anomalies in Air Force LAN traffic data that relies on using sketching data-structures. The key intuition of Pacsketch is that if we have a sketch for normal and anomalous data, then we can quickly classify windows of data based on which reference sketch it is most similar to. Our experiments show that the jaccard values with respect to the reference sketches typically allow us to classify the window correctly, as well as estimating the anomalous percentage quite close to the true value. Furthermore, our feature analysis indicates we could possibly take Pacsketch a step further to classifying anomalies as specific types of anomalies since there seems to be a difference in the feature distribution across different attack types.

9 Future Work

In terms of future directions, one natural step would be to extend Pacsketch to classify the type of

anomaly present in a window of data. This could be accomplished by building reference sketches for specific attack types instead of just one large sketch containing all attacks. Another direction would be to extend Pacsketch to additional network anomaly datasets since it could help to make Pacsketch more robust and generalizable. Lastly, another direction would be to perform some feature optimization in terms of how we convert the real number features to discrete features. Currently, each real number feature is converted to a discrete label which could be one of eight labels, however it is possible that eight is too many or too less. Therefore, it might be worth varying that parameter and seeing how it affects the downstream jaccard values.

References

[1] NSL-KDD dataset, Available at https://www.unb.ca/cic/datasets/nsl.html.

[2] AHMED, M., MAHMOOD, A. N., AND HU, J. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications 60* (2016), 19–31.

[3] BHUYAN, M. H., BHATTACHARYYA, D. K., AND KALITA, J. K. Network anomaly detection: methods, systems and tools. *Ieee communications surveys & tutorials 16*, 1 (2013), 303–336.

[4] BRODER, A. Z. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)* (1997), IEEE, pp. 21–29.

[5] ERTL, O. New cardinality estimation algorithms for hyperloglog sketches. *arXiv preprint arXiv:1702.01284* (2017).

[6] ESKIN, E., ARNOLD, A., PRERAU, M., PORTNOY, L., AND STOLFO, S. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*. Springer, 2002, pp. 77–101.

[7] FLAJOLET, P., FUSY, É., GANDOUET, O., AND MEUNIER, F. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science* (2007), Discrete Mathematics and Theoretical Computer Science, pp. 137–156.

[8] MÜNZ, G., LI, S., AND CARLE, G. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet* (2007), pp. 13–14.

[9] ONDOV, B. D., TREANGEN, T. J., MELSTED, P., MALLONEE, A. B., BERGMAN, N. H., KOREN, S., AND PHILLIPPY, A. M. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology 17*, 1 (2016), 1–14.

[10] SHYU, M.-L., CHEN, S.-C., SARINNAPAKORN, K., AND CHANG, L. A novel anomaly detection scheme based on principal component classifier. Tech. rep., MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[11] TUNE, P., AND VEITCH, D. Sampling vs sketching: An information theoretic comparison. In *2011 Proceedings IEEE INFOCOM* (2011), IEEE, pp. 2105–2113.