Project 4: Supervised vs Unsupervised Anomaly Detection

Scursatone Matteo - s332153, Lorenzato Fabio - s332186, Milani Alessandro - s332136

1 Task 1: Dataset Characterization and Preprocessing

1.1 Explore the dataset

Q: What are your dataset characteristics? How many categorical and numerical attributes do you have? How are your attack labels and binary_label distributed?

The training dataset contains 18831 records, whereas the test set contains 5826 records. Each record in the training set is described by 43 features, 4 (`protocol_type`, `service`, `flag`, `label`) of which are categorical and the remaining 39 are numerical. The distribution of both the attack labels and the binary labels is highly imbalanced, as shown in Figure 1. 71% of the records in the training set are labeled as normal, while the remaining 29% are distributed among the 3 different attack types (DoS, Probe, R2L). Of those 3 classes, only a handful of records are labeled as R2L, while the other two classes are more evenly distributed.

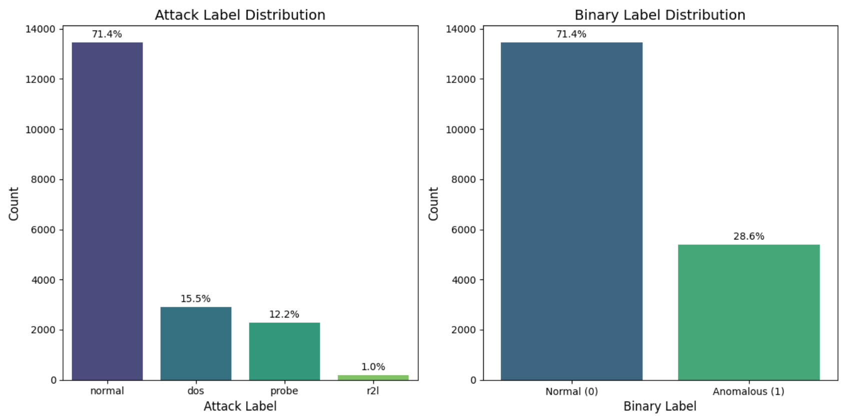


Figure 1: Distribution of the attack and binary labels in the training set.

Q: How do you preprocess categorical and numerical data?

1. We **removed** any **NaN**, **infinite** or duplicate value, as usual.
2. We dropped (one of) each features with a **correlation higher** than .97, as they are redundant. This resulted in the removal of 3 features (`num_compromised`, `rerror_rate`, `serror_rate`)
3. `urgent`, `num_outbound_cmds`, `is_host_login` were also dropped, as attributes with zero variance (single value) carry no useful information.
4. Numerical features were standardized using the **StandardScaler** from `sklearn`
5. Categorical features were One-Hot encoded due to the lack of any ordinal relationship between their values, and this required some feature engineering to avoid having too many dimensions:
    - `protocol_type` has 3 possible values, so we encoded it using 3 binary features as is.
    - `flag` has 11 possible values, but only the top 3 most frequent values contribute to more than 95% of the total number of records, so we grouped anything outside the top 3 into a single `other` category.
    - `service` has 65 possible values, which would result in a very high number of dimensions. To reduce this number, we grouped anything outside the top 10 most frequent values into a single `other` category, since each individual contribution to the toal number of records is at most 1%.

Q: Looking at the different heatmaps, do you find any main characteristics that are strongly correlated with a specific attack?

Figure 2 shows a heatmap of the aggregated statistical informations (mean, std and median) of the correlation between features and attack labels. The mean heatmap suggests that some features are strongly correlated with specific attack types. For example, dos attacks have much higher mean values in the `wrong_fragment`, `srv_serror_rate` and `dst_host_srv_serror_rate` features, suggesting a more frequent presence of fragmented packets and server errors during such attacks, and in some cases are almost exclusively associated with some attacks (e.g. `hot` with R2L attacks).

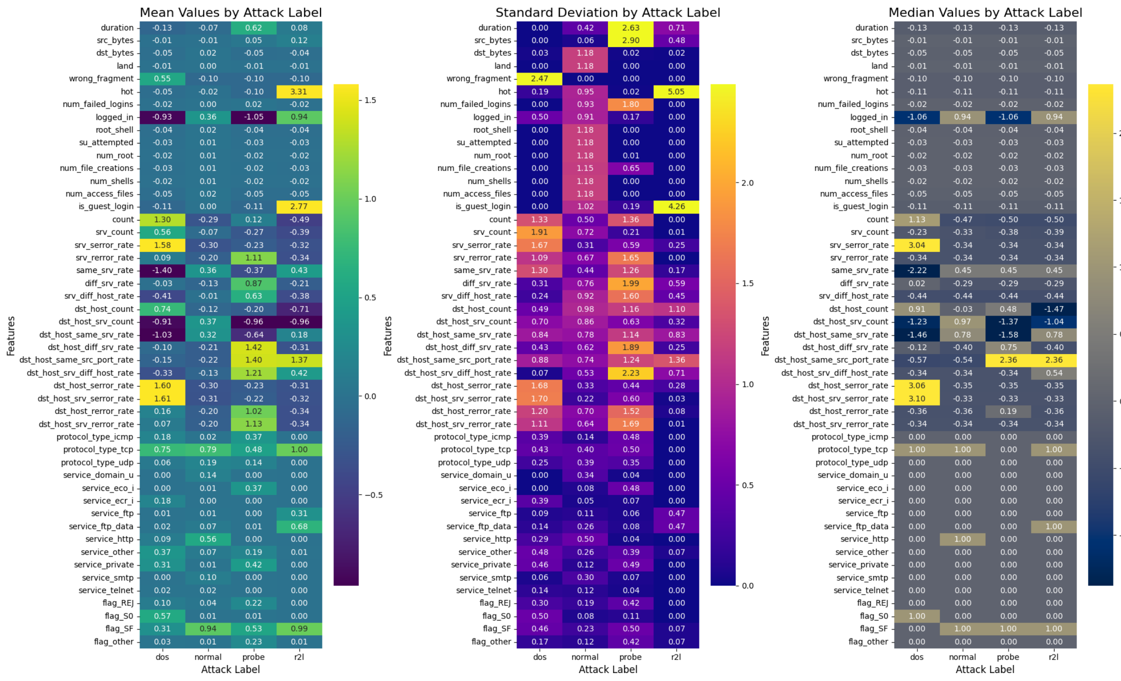


Figure 2: Heatmap of the correlation between numerical features and attack labels.

We were able to make some observation about which features are more relevant for each attack type:

- **DoS** attacks are associated with a higher number of fragmented packets (`wrong_fragment`), a higher rate of server errors (`srv_serror_rate`, `dst_host_srv_serror_rate`), a lower rate of successful connections (`dst_host_srv_rerror_rate`) and an higher count of connections to the same service (`count`).
- **Probe** attacks are associated with a higher number of connections from different hosts(`dst_host_diff_srv_rate`), much different connection durations (`duration`) and much larger packet sizes (`src_bytes`.
- **R2L** attacks are associated with a higher number of hot indicators (`hot`) as well as guest logins (`is_guest_login`).
- **Normal** traffic is associated with a higher number of successful connections (`dst_host_srv_rerror_rate`) and almost exclusive presence of shell and root related features (`root_shell`, `num_root`, `num_file_creations`,. . .).

Those patters can be used to help distinguishing between the different attack types during the classification phase.

2 Task 2: Shallow Anomaly Detection - Supervised vs Unsupervised

2.1 One-Class SVM with Normal data only

Q: Considering that you are currently training only on normal data, which is a good estimate for the parameter nu?

A good estimate for the parameter nu when training on normal data only can be a **small value (like 0.05 or 0.1)**. This is because nu represents the upper bound on the fraction of training errors and a lower value is more appropriate when the training set is composed solely of normal instances.

Q: Which is the impact on the training performance? Try both your estimate and the default value of nu.

We have decided to test 2 possible value on nu: 0.01 and 0.1 plus the standard value of the OneClassSVM class: 0.5. The results, for the F1-macro score, are shown in the table 1:

nu	F1-macro Score
0.01	0.9064
0.1	0.8778
0.5	0.6378

Table 1: F1-macro scores for different nu values

These results show how a **lower nu value leads to better performance** (our best performing model is with nu=0.01 $\rightarrow$ F1-score = 0.9064) in terms of F1-macro score when training on only normal data, where an higher nu value (like the default 0.5) results in a significant drop in performance because it leads to overflagging normal data.

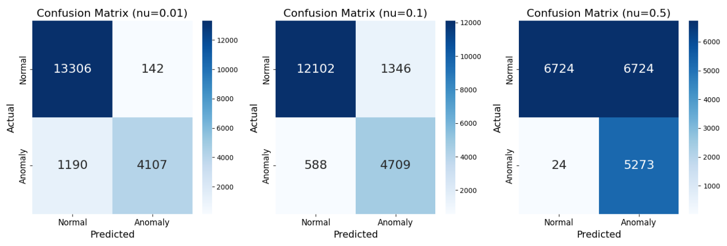


Figure 3: Confusion matrix for the three One-Class SVM models (nu=0.01, nu=0.1, nu=0.5) evaluated on the test set.

2.2 One-Class SVM with All data

Q: Which model performs better? Why do you think it is the case?

The model trained with all data (normal + anomalies) has a F1-macro score of 0.7315, which is significantly lower than the best model trained **only on normal data**. The causes for this drop in performance is the presence of the anomalies in the training data because the One-Class SVM is designed to be trained exclusively on normal data sample, and an high presence of anomalies can lead to a **misrepresentation of the normal data distribution**.

2.3 One-Class SVM with normal traffic and some anomalies

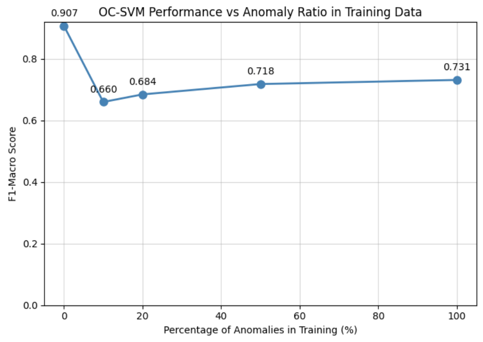


Figure 4: F1-macro scores for different anomaly rates in the training data.

Q: How is the increasing ratio of anomalies impacting the results?

The impact of increasing the anomaly ratio in the training data on the One-Class SVM model is not strictly monotonic. As shown in Figure 4 and the results, introducing a **small proportion of anomalies (10%) causes a sharp drop in F1-macro** score (from about 0.91 to 0.66). However, as the anomaly ratio increases further, the F1-macro score slightly recovers, reaching 0.73 when training only with all the anomalies. This confirms that the model is most effective when trained exclusively on normal data, while mixing in a small fraction of anomalies is particularly detrimental.

2.4 One-Class SVM model robustness

Model	F1-macro Score
One-Class SVM (nu=0.01, normal data only)	0.7540
One-Class SVM (nu=0.1, normal data only)	0.7450
One-Class SVM (nu=0.5, normal data only)	0.4691
One-Class SVM (all data)	0.6502
One-Class SVM (normal + 10% anomalies)	0.5228

Table 2: F1-macro scores of One-Class SVM with test set

Q: Is the best-performing model in the training set also the best here?

From the results in Table 2, we can see that **the best-performing model on the training set** (One-Class SVM with nu=0.01 trained only on normal data) **is also the best-performing model on the test set**, achieving an F1-macro score of 0.7540. This indicates that the model generalizes well to unseen data when trained exclusively on normal instances. If we want a more precise representation of the results, we can refer to the confusion matrices in Figure 5.

Q: Can it still spot the anomalies? Does it confuse normal data with anomalies?

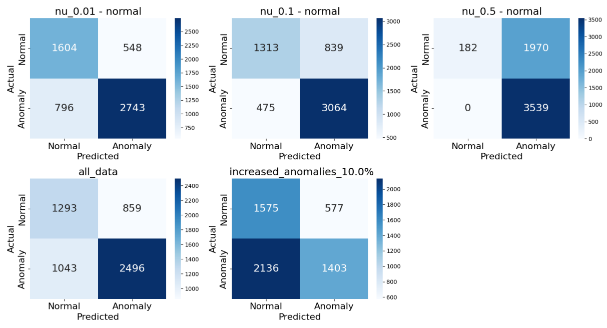


Figure 5: Confusion matrices for One-Class SVM with test set

The confusion matrices in Figure 5 provide further insight into the model's performance. Some key observations include:

- The model trained with **nu=0.5 on normal data** gets all the anomalies but also **misclassifies most all normal instances** as anomalies.
- The model trained with **the 10% anomalies** included in the training data shows the **worst performance in detecting anomalies**, by classifying over 2000 of them as normal.
- The best performing models remain the One-Class SVM with nu=0.01 and nu=0.1 trained only on normal data. The first presents less normal sample misclassifies as anomalies and the second one get correct more anomalies sample (but with also an higher false positive)

3 Task 3: Deep Anomaly Detection and Data Representation

In this task, we explore the use of a deep learning model, specifically an Autoencoder, for both anomaly detection and data representation. We will compare its performance against the shallow methods from Task 2 and also in conjunction with them.

3.1 Training and Validating Autoencoder with Normal data only

We began by defining and training an Autoencoder exclusively on normal traffic data. The architecture was designed with a symmetric encoder-decoder structure, creating a bottleneck layer to learn a compressed representation of the data.

- **Architecture:** The Autoencoder consists of an encoder with layers of size (input_dim $\rightarrow$ 128 $\rightarrow$ 64 $\rightarrow$ 8 (bottleneck_dim)) and a symmetric decoder. Batch normalization, ReLU activations, and Dropout are used for regularization.

- **Validation:** The normal data was split into training (80%) and validation (20%) sets. We experimented with four different learning rates (0.005, 0.001, 0.0005, 0.0001) to find the optimal one. The model with a learning rate of **0.0005 achieved the lowest validation loss of 0.038939** at epoch 86, and was selected as our best model.

3.2 Estimate the Reconstruction Error Threshold

Anomalies are detected by identifying data points that the Autoencoder, trained on normal data, cannot reconstruct accurately. We established a threshold for this reconstruction error using the validation set.

- **Method:** We calculated the per-sample reconstruction error on the validation data. The threshold was set at the 95th percentile of these errors as it's a common value to get a good trade-off between false positives and false negatives.

- **Threshold Value:** The obtained reconstruction **error threshold value is 0.0479**. Any data point with a reconstruction error above this value is classified as an anomaly.

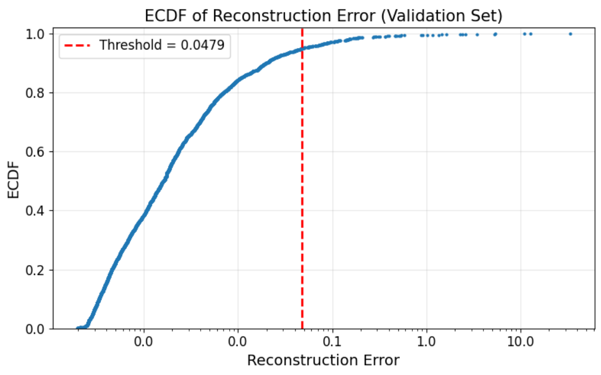


Figure 6: ECDF of Reconstruction Errors

3.3 Anomaly Detection with reconstruction error

We then evaluated the model's ability to distinguish anomalies by computing reconstruction errors on the entire training set and the test set. The Empirical Cumulative Distribution Function (ECDF) plots 7 illustrate the distribution of these errors.

Q: Plot and report the ECDF of the reconstruction errors for each point

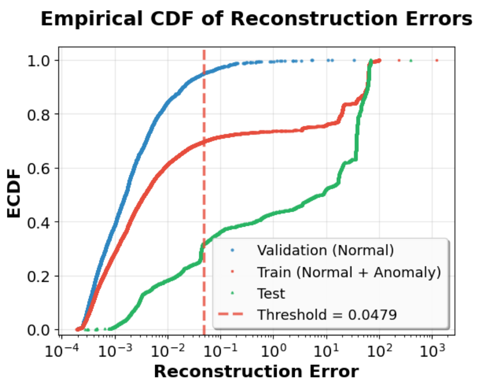


Figure 7: ECDF of Reconstruction Errors - Validation for reference + Full Training and Test sets

Q: Why the reconstruction errors are higher on the full training set than on the validation one? And why the reconstruction errors in the test set are even higher?

As anticipated, reconstruction errors are higher for both the full training set (which includes anomalies) and the test set compared to the validation set, which contains only normal data. This is expected, as the **autoencoder was trained exclusively on normal patterns** and therefore reconstructs them more accurately. The presence of anomalies unseen during training leads to increased reconstruction errors. The higher reconstruction errors observed in the test set suggest that it may include **patterns that deviate more significantly from the normal** training data than those present in the anomalous portion of the training set.

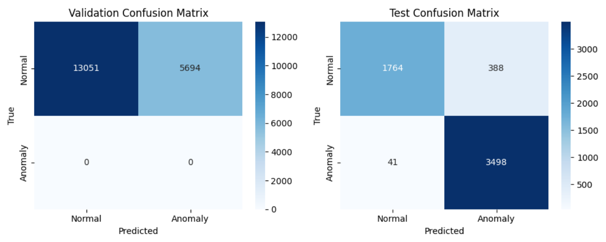


Figure 8: Confusion matrix for Autoencoder evaluated on (normal) training data and the test set

3.4 Auto-Encoder's bottleneck and OC-SVM

We leveraged the trained encoder to transform the normal portion of the original data into its lower-dimensional representation from the bottleneck layer and then use them to train a One-Class SVM. This train lead to the result showed in fig 9 and to an F1 score of 0.8697 for the training set and 0.7369 for the test set.

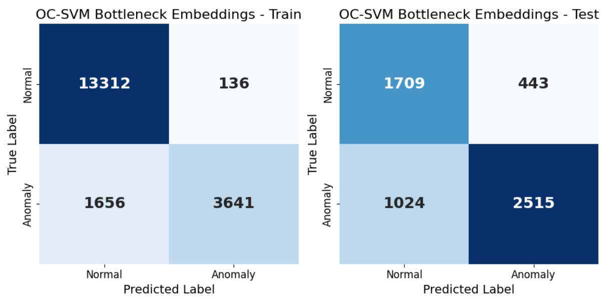


Figure 9: OC-SVM trained with Autoencoder embeddings

Q: Compare results with the best original OC-SVM. Describe the performance and where the model performs better or worst w.r.t. the original OC-SVM

Model	**Train F1-macro**	**Test F1-macro**
OC-SVM on raw features	0.9064	0.7540
OC-SVM on AE bottleneck embeddings	0.8697	0.7369

Table 3: Performance comparison between OC-SVM on raw features and OC-SVM on Autoencoder bottleneck embeddings.

IF we recover the results from the best original OC-SVM (nu=0.01, trained only on normal data) we can see that the F1-macro score for the training set is 0.9064 and for the test set is 0.7540. These result show how the **OC-SVM trained with Autoencoder embeddings performs slightly worse on both the training set and test set compared to the original OC-SVM**. This results can be explained by the fact that the Autoencoder embeddings provide a more compact and informative representation of the data, that lead to loss of some information. Also, the minor difference between the train and test results of the OC-SVM trained on embeddings suggest that it could generalize better to unseen data, compared to the original OC-SVM, with a more in depth optimization.

3.5 PCA and OC-SVM

As an alternative method for data representation, we applied Principal Component Analysis (PCA) to the normal data. To decide the number of components to retain, we analyzed the explained variance ratio (that can be seen of fig 10) and chose to **keep 20 components**, which capture a significant portion of the variance in the data.

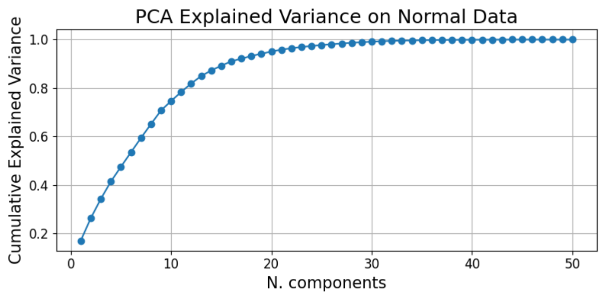


Figure 10: Explained variance ratio for PCA components

Q: compare results with original OC-SVM and the OC-SVM trained using the Encoder embeddings. Describe the performance of the PCA-model w.r.t. the previous OC-SVMs

After the application of PCA, we trained a One-Class SVM on the reduced dataset. The results are summarized in Table 4.

Model	**Train F1-macro**	**Test F1-macro**
OC-SVM on raw features	0.9064	0.7540
OC-SVM on AE bottleneck embeddings	0.8697	0.7369
OC-SVM on PCA components	0.7322	0.6585

Table 4: Performance comparison between OC-SVM on raw features, OC-SVM on Autoencoder bottleneck embeddings, and OC-SVM on PCA components.

These results indicate that the OC-SVM model trained on PCA components performs worse than both the original OC-SVM (that so still the best performing one) and the one trained on Autoencoder embeddings. This suggests that while **PCA is effective for dimensionality reduction**, it may **not capture the same level of discriminative features** as the original feature space or the Autoencoder embeddings.

4 Task 4: Unsupervised Anomaly Detection and Interpretation

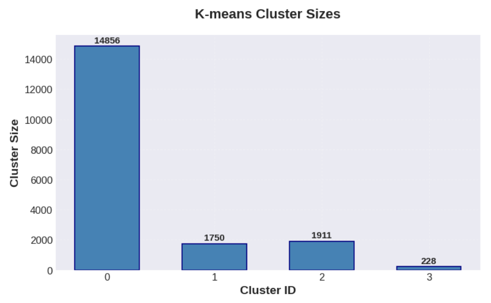
In this final task, we simulate a scenario where we have no prior labels and must use unsupervised clustering to identify anomalous behavior.

4.1 K-means with little domain knowledge

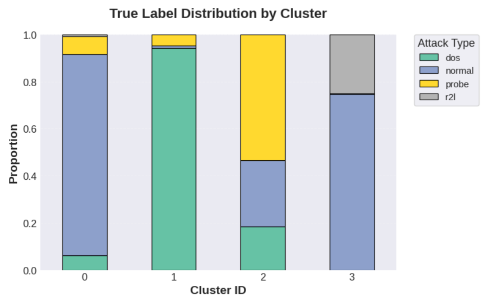
As domain experts in cybersecurity, we identified that the dataset contains three primary attack types in addition to normal traffic. Therefore, we set the number of clusters for K-means to 4 (one for each attack type and one for normal traffic).

4.2 K-means cluster interpretation

Q: How big are the clusters? How are the attack labels distributed across the clusters? Are the clusters pure?



(a) Cluster sizes and attack label distribution



(b) Cluster purity (fraction of majority label)

The clusters formed by K-means show **varying sizes and compositions**. As shown in Figure 11a, **cluster with ID 0 is significantly larger** than the others, indicating a dominant pattern in the data. The attack label distribution within each cluster reveals that **some clusters are relatively pure**, while others are more mixed. Figure 11b illustrates the purity of each cluster, with cluster with ID 1 achieving over 90% purity of dos traffic, while ID 2 and 3 have a more mixed distribution of data types. This suggests that while K-means can identify distinct patterns, it may struggle to separate certain attack types effectively.

Q: How high is the silhouette per cluster? Is there any clusters with a lower silhouette value? If it is the case, what attack labels are present in these clusters?

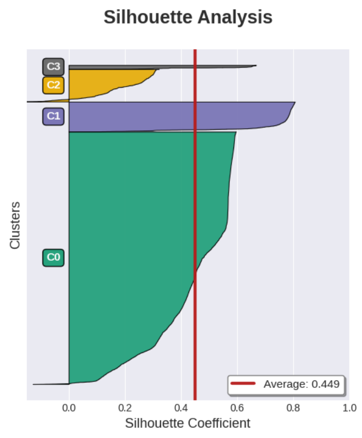


Figure 12: Silhouette analysis visualization

Metric	Value
Average Silhouette	0.4491
Cluster Details	
Cluster 0	14856 points, avg: 0.4483
Cluster 1	1750 points, avg: 0.7304
Cluster 2	1911 points, avg: 0.1897
Cluster 3	228 points, avg: 0.5124

Table 5: K-means clustering metrics

The silhouette scores per cluster reveal interesting patterns. Cluster **1 achieves the highest silhouette score of 0.7304**, indicating well-separated and cohesive points. In contrast, **Cluster 2 has the lowest silhouette score of 0.1897**, suggesting that points in this cluster may be poorly separated from neighboring clusters or internally inconsistent. This lower silhouette value in Cluster 2 often corresponds to mixed attack types of dos (18%), probe (53%) and some normal traffic (28%) that indicate a potential overlap or some similarity between the patterns.

Q: Use the t-SNE algorithm to obtain a 2D visualization of your points

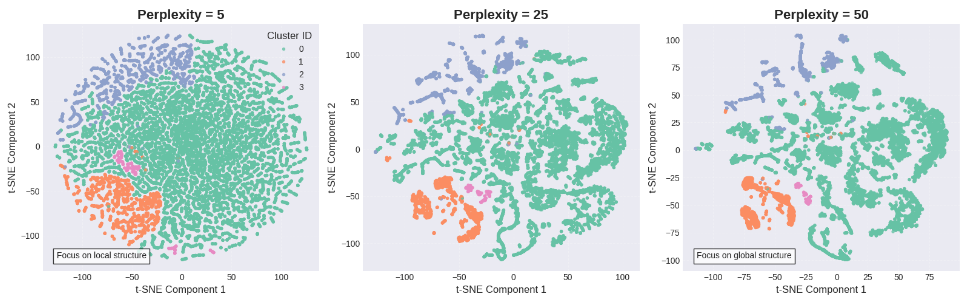


Figure 13: t-SNE comparison of cluster IDs

As shown in Figure 13, we try the 3 value: 5, 25, 50 for the perplexity. From all of the we can recognize the different clusters, but we can say that the best one is the one with **perplexity=25**, as it shows a better separation between the clusters.

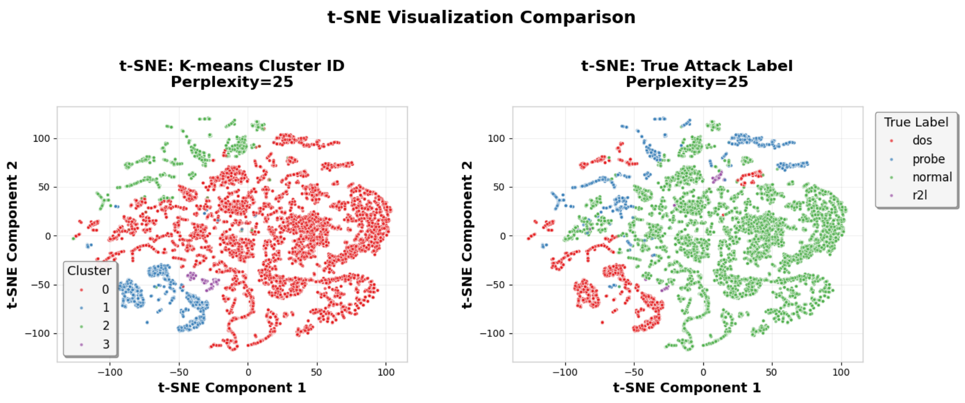


Figure 14: t-SNE comparison of attack labels

Use the t-SNE with the best perplexity and plot all the training data with the attack label. Can you find a difference between the two visualizations? What are the misinterpreted points? in the figure 14, we see the same scatter plot but colorized according to cluster ID in the first and to the attack labels in the second. With this direct comparison we can see how the clusters' representation is not perfect. Even if at first glance the clusters seem to align with the data labels we with a more detailed analysis we can see how **k-means was able to only split the biggest "groups" of attacks** (like we can notice by the split between cluster 0 (mainly normal traffic) and 1 (mainly DDoS attacks)) **and lost definition in more complex areas** (like all the top of the plot, where we can see a mix of all the attack types that in k-menas where split between cluster 0 and 2).

4.3 DB-Scan anomalies are anomalies?

Q: Create the clustering results using the entire training set using the parameters min_points and ϵ. Does the DB-Scan noise cluster consist only of anomalous points?

We decide to apply DB-Scan with the following parameters: **min_points = 5** (default value of the library) **and ϵ = 6** (the epsilon was chosen based on the k-NN distance after have tested the value 2,4,6,8,12). The results show that the **noise cluster** (labeled as -1) contains a total of 48 points, some of which are anomalous (about 6% of probe traffic) but **the majority is normal traffic** (about 94%). This indicates that the DB-Scan algorithm may not isolate anomalies effectively in this case, as the noise cluster is not exclusively composed of anomalous points.

Q: Next, consider the 10 largest clusters by size. How are the labels distributed across these clusters, i.e. how are they composed of a single label?

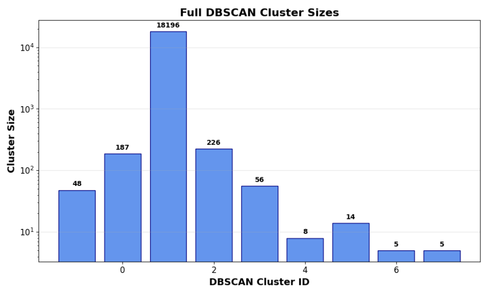


Figure 15: DB-Scan cluster sizes

From figure 15, we can see that we get a total of 8 clusters (including the noise cluster -1). After removing it and analyzing the composition of the other clusters, we get the results in table 6.

Label	**Cluster 1**	**Cluster 2**	**Cluster 0**	**Cluster 3**	**Cluster 5**	**Cluster 4**	**Cluster 6**	**Cluster 7**
Size	18196	226	187	56	14	8	5	5
normal	0.722	0.748	0.0	1.0	1.0	1.0	1.0	1.0
dos	0.149	0.0	1.0	0.0	0.0	0.0	0.0	0.0
probe	0.122	0.0	0.0	0.0	0.0	0.0	0.0	0.0
r2l	0.007	0.252	0.0	0.0	0.0	0.0	0.0	0.0

Table 6: Distribution of labels across the top DB-Scan clusters

These data show how the largest cluster **(Cluster 1) is predominantly composed of normal traffic** (72.2%), with a significant portion of dos attacks (14.9%) and probe attacks (12.2%). The second-largest cluster (Cluster 2) has a high purity of normal traffic (74.8%) but also contains a notable fraction of r2l attacks (25.2%). Other clusters, such as Cluster 3, 5, 4, 6, and 7, are entirely composed of normal traffic and **only the cluster 0 is composed entirely by dos attacks**. Overall, the clusters show varying degrees of label purity, with some clusters being more mixed than others.
We can say that the DB-Scan algorithm was able to identify some distinct patterns in the data for well-separated labels, but the presence of mixed clusters indicates that it may not be fully effective in separating all attack types that present similar characteristics.

Q: Use the t-SNE for visualization

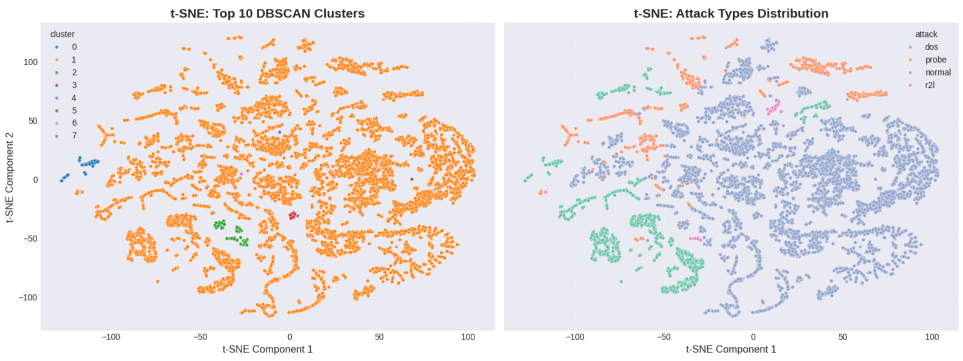


Figure 16: t-SNE visualization of DB-Scan clusters

In figure 16 we can see the use of t-SNE to compare the DB-scan clusters with the original labels. In this case it's easy to understand that the DB-Scan clusters do not align well with the original labels, as almost all the data points are grouped into a single large cluster (Cluster 0) with a few smaller clusters scattered inside in section of plot where some data point are almost overlapping.
In short, we can say that the **DB-Scan algorithm struggled to effectively separate the different types of network traffic**, leading to a clustering that does not reflect the underlying label distribution.

Q: Why do you think that DB-Scan cannot separate the normal anomalous points well?

DB-Scan cannot separate the normal and anomalous points well because its effectiveness is predicated on the assumption that clusters are dense regions separated by areas of lower density. In this dataset, **the different classes of network traffic (normal, dos, probe, r2l) do not exhibit significantly different density profiles**. Instead, they form a large, contiguous mass in the feature space without clear low-density boundaries between them. As a result, DB-Scan interprets this the majority of the entire mass as a single large cluster (Cluster 0) and only some mass with overlapping points get recognized as separated small clusters.