Lab 4 – Supervised vs Unsupervised Anomaly Detection

Pierfrancesco Elia - s331497, Alessandro Meneghini - s332228, Ankesh Porwal - s328746

Overview

This laboratory explores the application of anomaly detection techniques in cybersecurity, focusing on the comparison between supervised and unsupervised learning approaches. Using network intrusion detection datasets, various machine learning algorithms are evaluated in their ability to identify malicious activity. The goal is to assess detection performance, generalization capabilities, and robustness.

1 Task 1: Dataset Characterization and Preprocessing

1.1 Dataset Exploration

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 18,831 instances and 43 attributes, while the test dataset includes 5,826 instances. Among the 43 features, 4 are categorical (`protocol_type`, `service`, `flag`, `label`) and 39 are numerical.
Attack labels are grouped into four main categories: `normal`, `DoS`, `Probe`, and `R2L`. The `binary_label` feature distinguishes between normal (0) and anomalous (1) traffic.

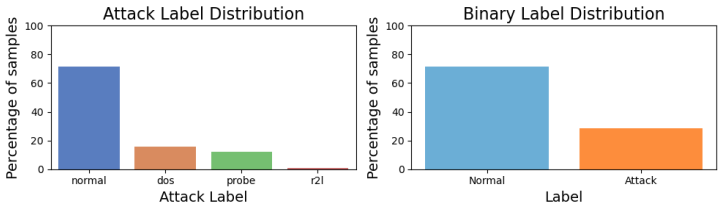

Figure 1: Distribution of attack labels and `binary_label`.

Figure 1 shows the class distribution of both multi-class attack labels and the binary anomaly label. This helps understand the balance of normal versus anomalous samples, as well as the prevalence of each attack category. As the task focuses on anomaly detection, it is favorable that the dataset contains significantly more normal data, which aligns with real-world scenarios where anomalies are rare and models are expected to generalize from abundant normal patterns.

1.2 Preprocessing

Q: How do you preprocess categorical and numerical data?
Categorical features are preprocessed as follows:

- **Service**: the 9 most frequent values are retained; all others are grouped into a 10^{th} category labeled `other`.
- **Flag**: the 4 most common values are retained; less frequent ones are grouped into a fifth category labeled `other`.
- **One-Hot Encoding** is then applied to all categorical features.

Numerical features are standardized using z-score normalization to ensure uniform scale across all attributes.

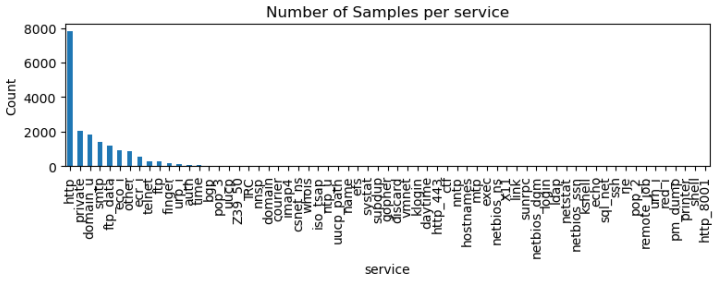

Figure 2: Distribution of service categories before encoding.

Figure 2 illustrates an example of exploratory analysis performed on the `service` feature. This step helps evaluate the feature's cardinality and distribution, which is crucial to decide how to handle high-cardinality categorical data effectively before encoding.

1.3 Domain Expert Heatmaps

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

To identify class-specific patterns, we computed the mean, standard deviation, and median for each feature grouped by attack label, and visualized them using heatmaps.

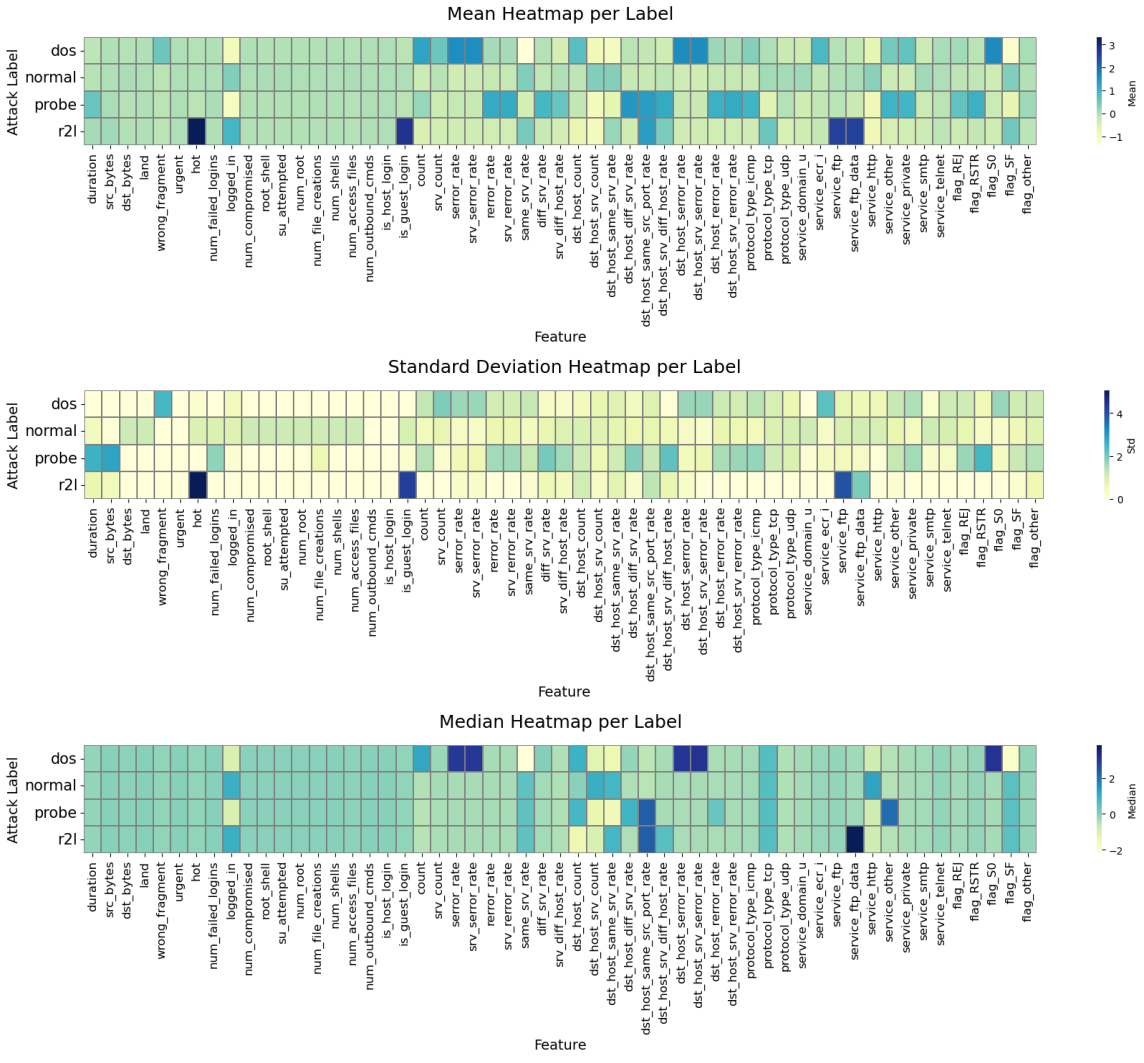

Figure 3: Heatmaps of feature statistics by attack label.

Figure 3 display statistical heatmaps computed per attack label. They provide insight into which features vary most between attacks and may serve as strong indicators for classification or anomaly detection.
Insights:

- **R2L attacks** show high values and variance in features such as `hot`, `is_guest_login`, `service_ftp`, and `service_ftp_data`, which are typically linked to unauthorized login attempts.
- **DoS attacks** are characterized by elevated values in traffic-related features, including `serror_rate`, `dst_host_serror_rate`, and `flag_S0`, reflecting their typical flooding behavior.
- **Probe attacks** present moderate increases in scanning-related features, such as `dst_host_same_src_port_rate` and `service_other`.

These feature patterns help distinguish attack types and guide both supervised and unsupervised anomaly detection strategies.

2 Task 2: Shallow Anomaly Detection

We evaluate several One-Class Support Vector Machine (OC-SVM) models to detect anomalies under different training conditions.

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`? Which is the impact on the training performance?
We train OC-SVM using only normal samples, setting `nu` to 0.01, which reflects a small expected outlier fraction. This configuration yields an F1-macro score of 0.9368, outperforming the default `nu` = 0.5 (F1-macro = 0.6354). A small `nu` allows the model to tolerate minimal noise and define a compact boundary around normal behavior.
Figure 4 shows the confusion matrices comparing performance with the tuned and default `nu` values.

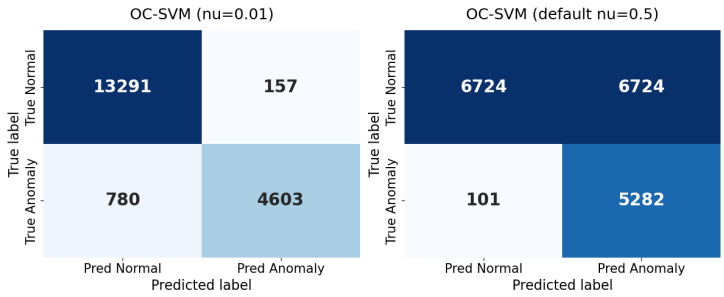

Figure 4: Confusion matrices for OC-SVM with `nu`=0.01 vs default on normal-only training data.

2.2 One-Class SVM with All data

We then train OC-SVM on the full dataset, including anomalies. The estimated anomaly ratio (`nu`) is 0.2859, resulting in an F1-macro score of 0.7376.
Q: Which model performs better? Why do you think it is the case?
The model trained only on normal data performs better. Including anomalies in the training set distorts the notion of normality and weakens the model's boundary. Since OC-SVM assumes mostly clean data, using purely normal samples enables more accurate anomaly detection.

2.3 Impact of Adding Anomalies During Training

We train OC-SVM models with varying proportions of anomalies in the training data: `[0, 0.1, 0.2, 0.5, 1]`.
Q: Plot the f1-macro score for each scenario. How is the increasing ratio of anomalies impacting the results?

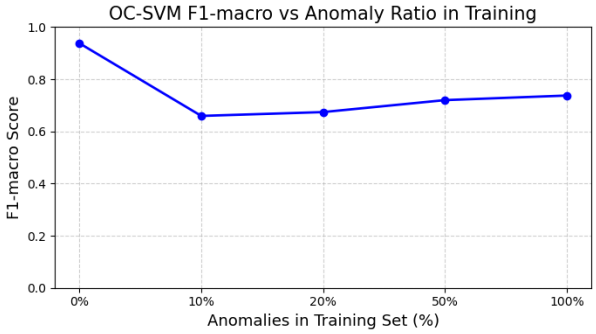

Figure 5: F1-macro scores with increasing anomaly contamination during training.

Figure 5 shows how the F1-macro score changes as more anomalies are included during training. The model performs best when trained only on clean (normal) data. Interestingly, the lowest F1 score occurs when 10% of anomalies are added, and then slightly improves with more contamination. However, the variation is not significant. These results suggest that OC-SVM is most effective when trained on purely normal traffic, as the presence of anomalies in training may hinder the model's ability to define a precise decision boundary.

2.3.1 One-Class SVM model robustness

To assess robustness, we evaluate three models on the **test set**: Model 1 is trained on normal data only (`nu` = 0.05), Model 2 is trained on all data (`nu` $\approx$ 0.286), and Model 3 is trained on normal data with 10% anomalies (`nu` $\approx$ 0.038).
Q: Is the best-performing model in the training set also the best here? Compare the results of the best model in the test set with its results in the training set. Can it still spot the anomalies? Does it confuse normal data with anomalies?
We observe F1-macro scores of **0.7888 for Model 1**, 0.6560 for Model 2, and 0.5073 for Model 3 on the test set.
Compared to its performance on the training set (F1-macro = 0.9368), Model 1 experiences a moderate drop, which is expected when generalizing to unseen data. However, it still manages to correctly detect the majority of anomalies, showing good anomaly detection capability. Some false positives remain, indicating occasional confusion between normal and anomalous samples.

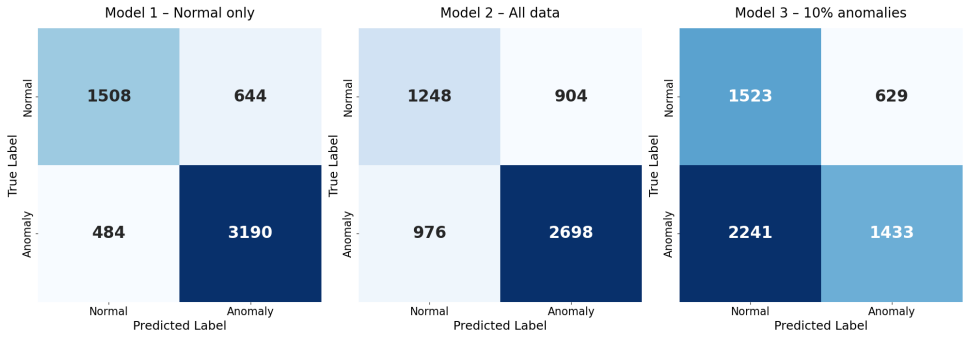

Figure 6: Confusion matrices of all OC-SVM models evaluated on the test set.

Figure 6 shows the confusion matrices for all three models evaluated on the test set.
Confusion Matrix Insights:

- **Model 1 (Normal only)**: Delivers the best balance. It detects most anomalies (3190/3674) and correctly classifies many normal samples (1508/2152), with moderate false positives and negatives.
- **Model 2 (All data)**: Performs worse, missing more anomalies (976) and misclassifying more normal points (904).
- **Model 3 (10% anomalies)**: Shows the weakest results, with 2241 false negatives, suggesting a highly diluted boundary due to insufficiently representative anomalies.

The best-performing model in training (Model 1) also generalizes best on the test set. Training OC-SVM exclusively on clean normal data ensures superior anomaly detection, both in training and real-world scenarios.

3 Task 3: Deep Anomaly Detection and Data Representation

In this task, we extend the anomaly detection pipeline by introducing Deep Learning techniques, specifically using an Autoencoder. We train and validate the Autoencoder on normal data, use reconstruction error and thresholding to detect anomalies, extract compressed representations from the bottleneck for OC-SVM training, and compare this approach with a PCA-based OC-SVM model.

3.1 Training and Validating Autoencoder with Normal data only

We train an Autoencoder on normal traffic. The model consists of an encoder that compresses the input into a lower-dimensional space (bottleneck) and a decoder that reconstructs the input from this compressed representation.
We split the normal data into training and validation sets and monitor the reconstruction loss on the validation set to evaluate performance. We apply an early stopping mechanism to halt training when the validation loss does not improve for a fixed number of epochs, thus avoiding overfitting.
Using a grid search on learning rate values (`[0.001, 0.0005, 0.0001, 0.00005]`), the best model is found at `LR = 0.0001`, with a validation loss of 0.0314.
Figure 7 shows the training and validation losses over epochs and illustrates the effect of early stopping.

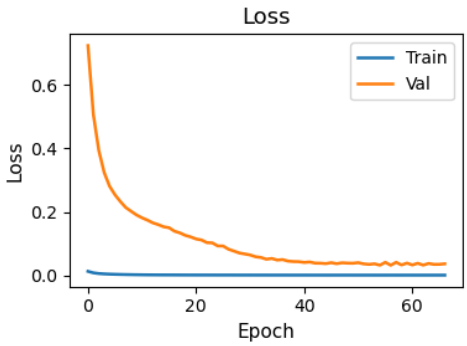

Figure 7: Training and validation loss curves during Autoencoder training.

3.2 Estimate the Reconstruction Error Threshold

Q: How do you choose the threshold? What is its value?
We compute the reconstruction error on the validation set and select the 95th percentile as the anomaly detection threshold. This approach provides a trade-off between sensitivity and false positives, assuming some variability even within normal data. Using this criterion, we obtain a **threshold value of 0.0625**.
Figure 8 shows the ECDF of reconstruction errors on the validation set. The Autoencoder is trained only on normal samples.

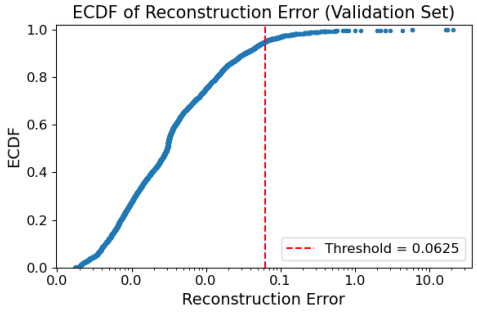

Figure 8: ECDF of reconstruction errors on the validation set.

3.3 Anomaly Detection with reconstruction error

We apply the trained Autoencoder to compute reconstruction errors on:

- The validation set (normal data only).
- The full training set (normal + anomalies).
- The test set.

Using the previously defined threshold, we flag as anomalous any sample whose reconstruction error exceeds it.
Figure 9 presents the ECDF curves of reconstruction errors across the three datasets. The red dashed line shows the anomaly detection threshold.

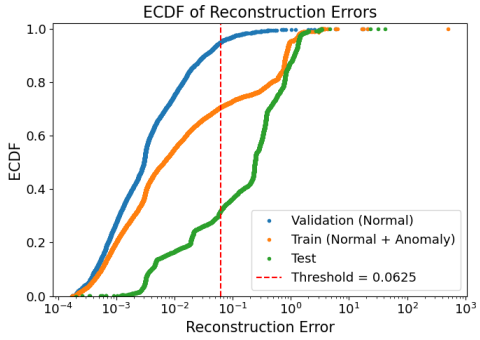

Figure 9: ECDF of reconstruction errors on validation, full training, and test sets.

The validation set shows the lowest errors, confirming effective reconstruction of normal samples. The full training set (including anomalies) has higher errors, while the test set is further shifted, reflecting unseen attack types and validating the thresholding approach.
Q: Why are errors higher on full training and test sets than on validation?
Errors are higher because these sets contain anomalies not seen during training. The Autoencoder reconstructs normal patterns well but fails on unseen, anomalous behavior.
Figure 10 shows confusion matrices over validation, training, and test sets using the reconstruction error threshold.

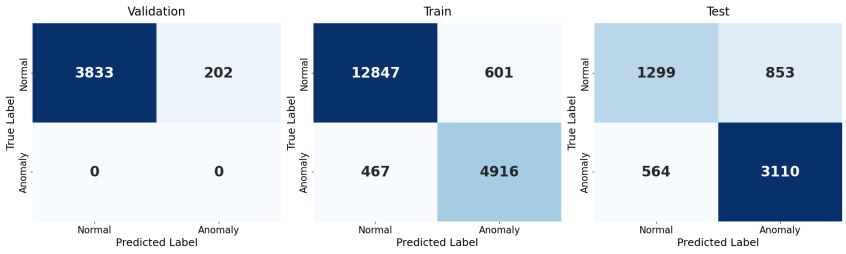

Figure 10: Confusion matrices on validation, training, and test sets based on reconstruction error.

3.4 Autoencoder Bottleneck + OC-SVM

We extract compressed representations from the encoder (bottleneck) and use them to train a One-Class SVM on bottleneck features derived from normal training data, then evaluate its performance on the corresponding features from the test set. Figure 11 shows the confusion matrices for this approach.

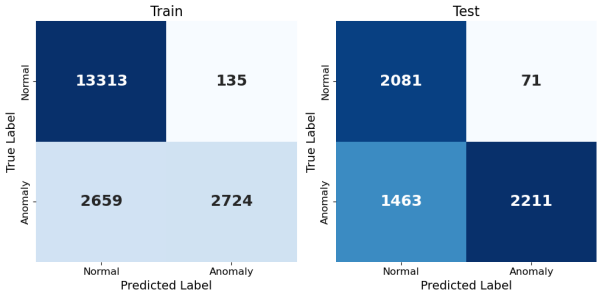

Figure 11: Confusion matrices of OC-SVM on Autoencoder bottleneck representations.

Q: Compare the results with the best original OC-SVM.
We compare the F1-macro scores of the original OC-SVM and the Autoencoder-based OC-SVM. The OC-SVM trained on raw features achieves 0.9368 on training and 0.7888 on test, while the model based on the Autoencoder's bottleneck features achieves 0.7140 on training and 0.7578 on test. Although the bottleneck-based model generalizes slightly better on test, it shows reduced training score, likely due to information loss during compression.

3.5 PCA + OC-SVM

We apply Principal Component Analysis (PCA) to reduce the dimensionality of the dataset. Based on the explained variance curve, we select the first 25 principal components, which together account for at least 95% of the total variance.
After transforming the normal training data using PCA, we train an OC-SVM model on the reduced representation. We then apply this model to PCA-transformed test data to evaluate its anomaly detection performance.
Figure 12 shows the confusion matrices that result from this approach, allowing a direct comparison with the previous models.

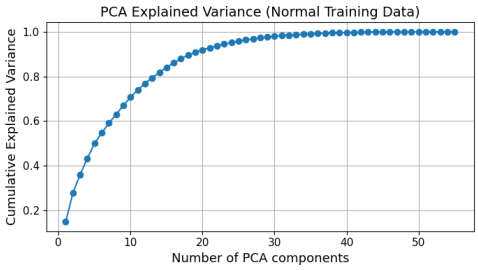

Figure 12: Confusion matrices of OC-SVM on PCA-transformed data.

Q: Compare results with original OC-SVM and AE bottleneck OC-SVM.

Model	F1-macro (Train)	F1-macro (Test)
OC-SVM (raw features)	**0.9368**	**0.7888**
OC-SVM (AE bottleneck)	0.7140	0.7578
OC-SVM (PCA features)	0.7562	0.7006

The raw feature OC-SVM performs best. PCA and AE provide useful dimensionality reduction, but slightly reduce detection accuracy. The PCA-based model performs better than the AE bottleneck on training, but slightly worse on test data.
While PCA and AE reduce complexity and may improve robustness, the OC-SVM trained on raw, scaled features achieves the highest performance across all metrics in this task.

4 Task 4: Unsupervised Anomaly Detection and Interpretation

In many real-world scenarios, labels for anomalies are not available. In this task, we apply unsupervised learning techniques to detect and interpret anomalies in the dataset without relying on label information.

4.1 K-means with Limited Domain Knowledge and Interpretation

We assume the dataset contains four main types of network traffic (e.g., normal and three attack classes), but the labels are unknown. We apply K-means clustering with `n_clusters = 4` on the full training set (normal + anomalies).

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

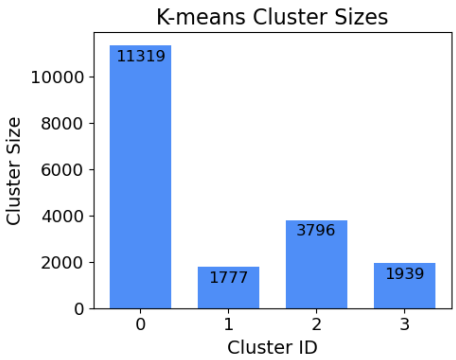

Figure 13: Cluster sizes.

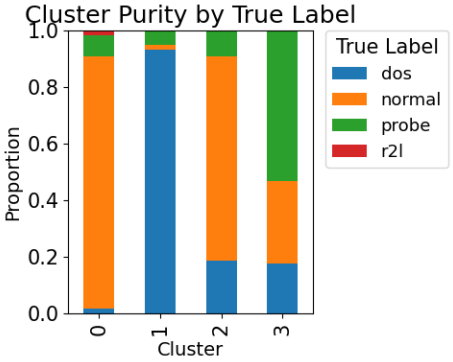

Figure 14: Label distribution per cluster.

Figure 13 presents the size of each K-means cluster, showing highly imbalanced cluster sizes. Figure 14 shows the distribution of true attack labels within each cluster, revealing that some clusters are highly pure (e.g., Cluster 1: 93% DoS), while others contain a mix of attack and normal traffic.

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?

To evaluate the cohesion and separation of the K-means clusters, we compute the silhouette score for each cluster. This metric quantifies how well each data point fits within its assigned cluster relative to others.

Figure 15 presents the silhouette scores for all four clusters. Clusters 2 and 3 exhibit low values (below 0.3), suggesting that they lack clear separation and contain data points near cluster boundaries.

Figure 16 further investigates the content of these low-silhouette clusters. Cluster 2 is composed mostly of normal traffic (72%) but includes notable portions of DoS (19%) and Probe (9%). Cluster 3 is dominated by Probe (53%), along with normal (29%) and DoS (18%). These mixed compositions explain the low silhouette values, indicating ambiguity in cluster assignments.

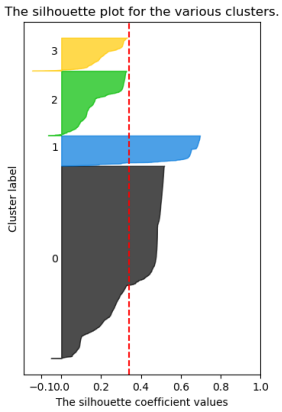

Figure 15: Silhouette scores per K-means cluster.

Figure 16: True label distribution in low-silhouette clusters.

Q: Use t-SNE to visualize clusters and labels. What are the differences? Which points are misinterpreted?

We explore three perplexity values for t-SNE: 10, 30, and 50. Figure 17 shows that perplexity 30 gives the most informative visualization.

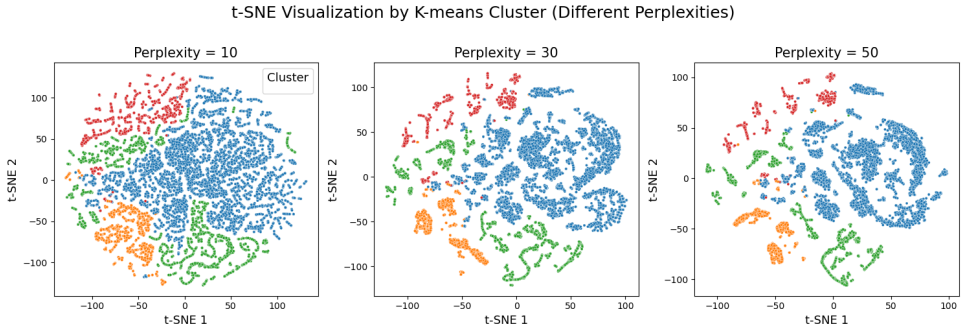

Figure 17: t-SNE with varying perplexity values (10, 30, 50).

Using the best perplexity (30), Figure 18 compares K-means cluster assignments (left) with true labels (right). Cluster 1 corresponds quite well to DoS attacks, while others, such as Cluster 2, blend normal and Probe traffic. This highlights the difficulty K-means has in separating similar or overlapping traffic types.

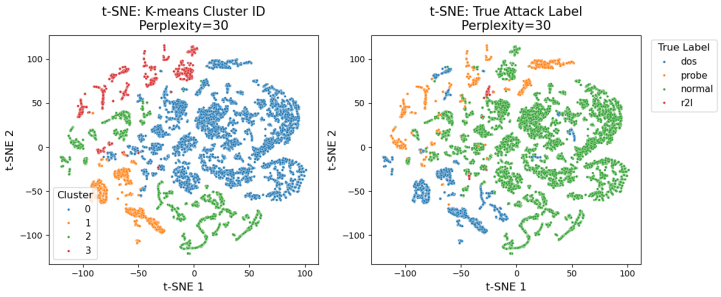

Figure 18: t-SNE: Left—K-means clusters, Right—true labels.

4.2 DB-Scan anomalies are anomalies?

Q: Create the clustering results using the entire training set (normal + anomalous) using the parameters min_points and ϵ. Does the DB-Scan noise cluster (cluster -1) consist only of anomalous points (cross-reference with real attack labels)?
We apply DBSCAN to the full training set using parameters selected from the elbow method (Figure 19). We choose the following values: ϵ `= 7.0` and `min_samples = 5`.

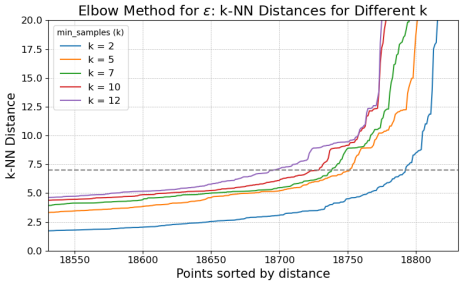

Figure 19: Elbow method to select ϵ.

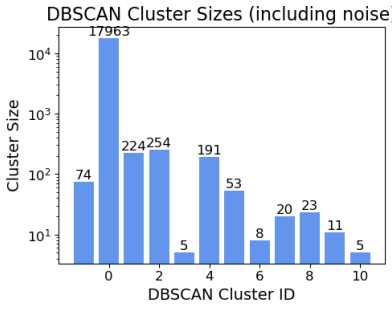

Figure 20: DBSCAN cluster sizes (w/ noise).

With these parameters, DBSCAN detects 74 noise points. Figure 20 shows the cluster sizes. The noise cluster (label = -1) contains mostly normal traffic (85.1%) and some Probe samples (14.9%), suggesting that DBSCAN does not isolate anomalies effectively.
Q: Next, consider the 10 largest clusters by size - DO NOT consider cluster -1. How are the labels distributed across these clusters, i.e. how are they composed of a single label?
Figure 21 shows the distribution of true labels in the top DBSCAN clusters, excluding noise. While the requirement was to consider the 10 largest clusters, our analysis found 11 clusters total (excluding noise),

and their sizes were comparable. Therefore, we include all of them for completeness. Clusters 5, 6, 7, 9, and 10 are entirely pure, containing only normal or DoS samples. Other clusters, such as 0, 1, 2, 3, 4, and 8, are more mixed.

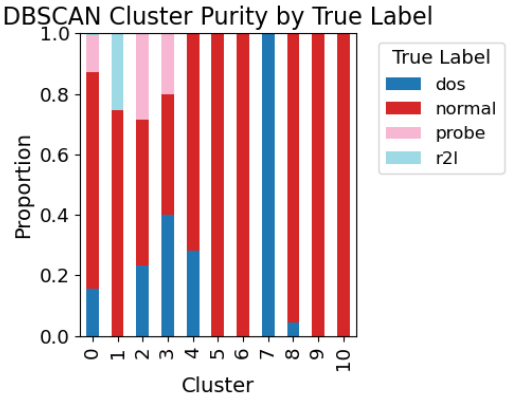

Figure 21: Label distribution in DBSCAN's 11 largest clusters.

Cluster	Normal	DoS	Probe	R2L
0	0.716	0.155	0.123	0.007
1	0.746	0.000	0.000	0.254
2	0.484	0.232	0.280	0.004
3	0.400	0.400	0.200	0.000
4	0.717	0.283	0.000	0.000
5	**1.000**	0.000	0.000	0.000
6	**1.000**	0.000	0.000	0.000
7	0.000	**1.000**	0.000	0.000
8	0.957	0.043	0.000	0.000
9	**1.000**	0.000	0.000	0.000
10	**1.000**	0.000	0.000	0.000

Table 2: True label distribution in top 11 DBSCAN clusters (excluding noise).

Table 2 presents the true label proportions for the 11 largest DBSCAN clusters, excluding the noise cluster. Among them, several clusters are pure or nearly pure: Clusters 5, 6, 7, 9, and 10 consist entirely of a single class (either normal or DoS), while Cluster 8 is dominated by normal traffic (95.7%) with a small fraction of DoS (4.3%). These indicate that DBSCAN can isolate dense homogeneous regions effectively.

In contrast, other clusters are more heterogeneous. Cluster 0 is mostly normal (71.6%) but contains notable proportions of DoS, Probe, and R2L traffic. Cluster 1 is mainly normal (74.6%) but contains 25.4% R2L, while Cluster 4 splits between normal and DoS. Cluster 2 is more evenly mixed with normal (48.4%), Probe (28.0%), and DoS (23.2%). Cluster 3 is balanced among DoS, normal, and Probe.

This pattern suggests that while DBSCAN is capable of identifying pure clusters for well-separated classes like normal or DoS, it struggles when attack types share similar densities or features. As a result, mixed clusters emerge where traffic types overlap in feature space.

Q: Use t-SNE to visualize DBSCAN clusters. What are the differences with ground truth labels?

Figure 22 shows a t-SNE comparison using the same perplexity. Clusters mix true labels, and many borderline points are misinterpreted. DBSCAN captures dense regions (e.g., DoS), but struggles with scattered or overlapping samples, particularly Probe and R2L.

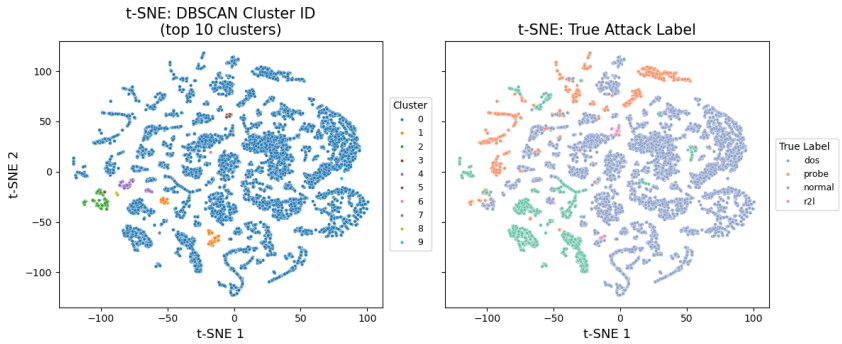

Figure 22: t-SNE: DBSCAN cluster ID vs true labels.

Q: Why does DBSCAN fail to cleanly separate anomalies?

DBSCAN depends on local density and distance. In high-dimensional data, these metrics become less meaningful. Some normal samples appear isolated and are marked as noise, while dense attack regions are absorbed as normal. The curse of dimensionality and density overlap limit DBSCAN's performance in this context.