Lab-3
Group-5

Dennis D'Amico - s343841, Pierre Perrier - s350300,
Manuele Sforzini - s349555

0 Goal of the laboratory

The main goal of this laboratory is to apply Shallow and Deep Learning anomaly detection techniques to create an Intrusion Detection Systems (IDS). In particular, to achieve the objective, a dataset consisting of network traffic data is used. The traffic is labeled as either normal or one of several types of attacks (DoS, Probe, R2L).

In this lab, the labels will not be used to directly train a supervised model. Instead, the objective is to evaluate whether unsupervised algorithms can automatically detect anomalous patterns and to analyze how their performance evolves as the knowledge of the data increases.

1 Task 1: Dataset Characterization and Preprocessing

The first Task of the laboratory is about exploring and preprocessing the various features in the dataset.

Q: What are the dataset characteristics? How many categorical and numerical attributes do you have? How are your attack labels and binary label distributed? In the dataset there are 39 numerical attributes and 4 categorical attributes, including *label* and *binary_label*. The labels and the binary labels are quite unbalanced:

Label	Count	Percentage
Normal	13448	71.41%
DoS	2913	15.47%
Probe	2289	12.16%
R2L	181	0.96%

(a) Distribution of Labels

Binary Label	Count	Percentage
Normal (0)	13448	71.41%
Attack (1)	5383	28.59%

(b) Distribution of Binary Labels

We can see from the tables above that about 3 out of 4 samples from the dataset are labeled as normal traffic, while among the samples labeled as attacks, we have a prevalence of DoS and Probe, while R2L is only in a few samples.

Q: How do you preprocess categorical and numerical data? In the preprocessing phase, after having dropped all the duplicate rows in the dataset, we used a different approach for numerical and categorical features: the former were preprocessed using a Standard Scaler, which is perfect to correctly handle such features, while for the latter we used One Hot Encoding, to avoid introducing biases in the data.

The Mean Heatmap, the Standard Deviation Heatmap and the Median Heatmap are reported in Figures 1, 2 and 3, respectively. **Q: Looking at the different heatmaps, do you find any main characteristics that are strongly correlated with a specific attack?** Yes, we can see some examples of such correlations for each attack:

- The R2L attack is strongly correlated with the features *hot* and *is_guest_login*, which are related to specific access attempts;
- DoS is related instead with high values in counting features like *count* and *srv_count*, which is coherent with that kind of attack;
- Probe has instead a unique behavior in port-related features like *dst_host_same_src_port_rate*, with an high variability in the connection duration.

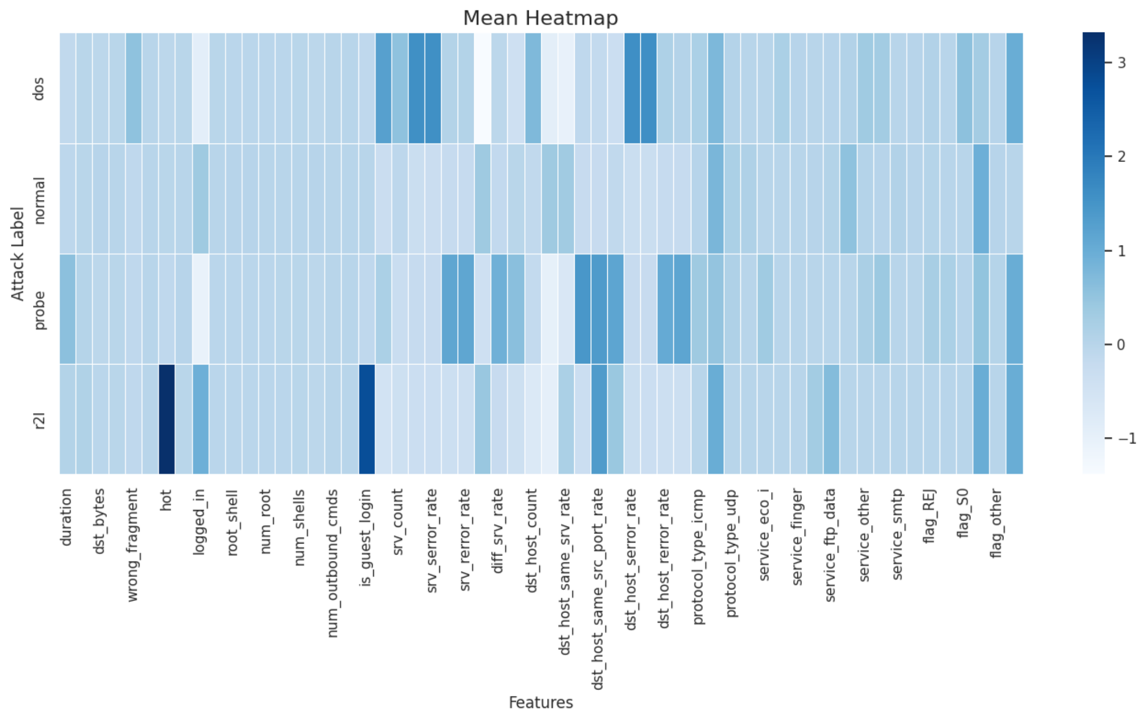


Figure 1: Mean heatmap

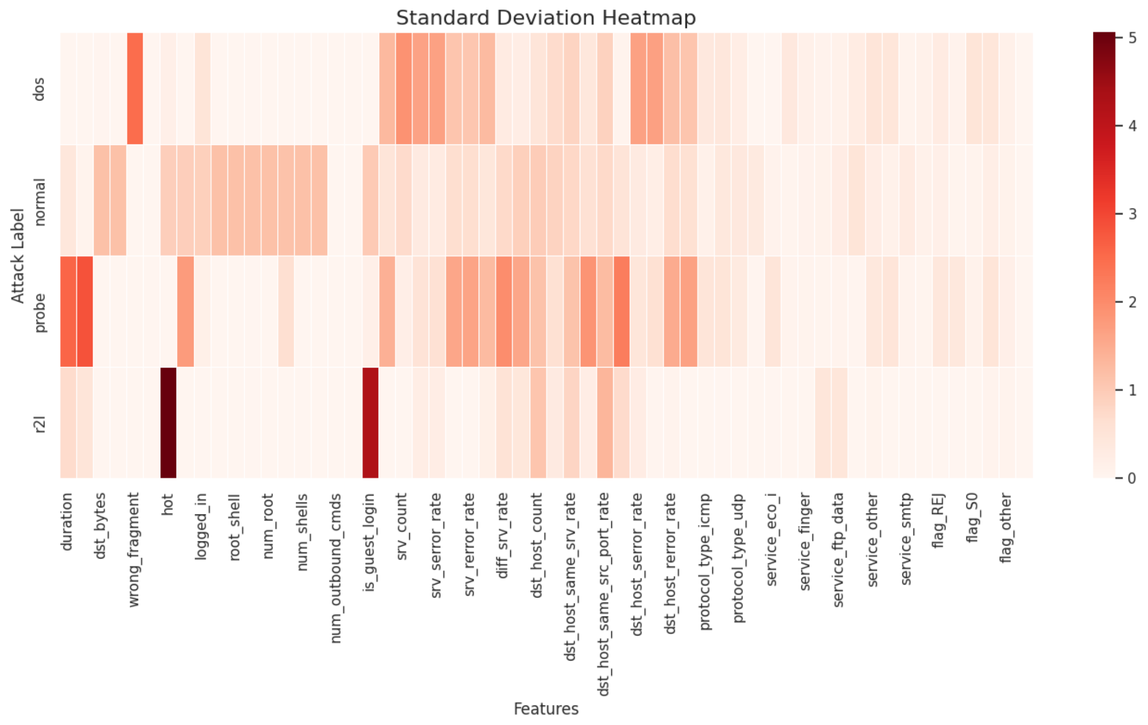


Figure 2: Standard Deviation heatmap

2 Task 2: Shallow Anomaly Detection - Supervised vs Unsupervised

In this section of the laboratory, the tasks were all about One-Class-SVMs and how they can be used for anomaly detection both in a Supervised and in an Unsupervised scenario.

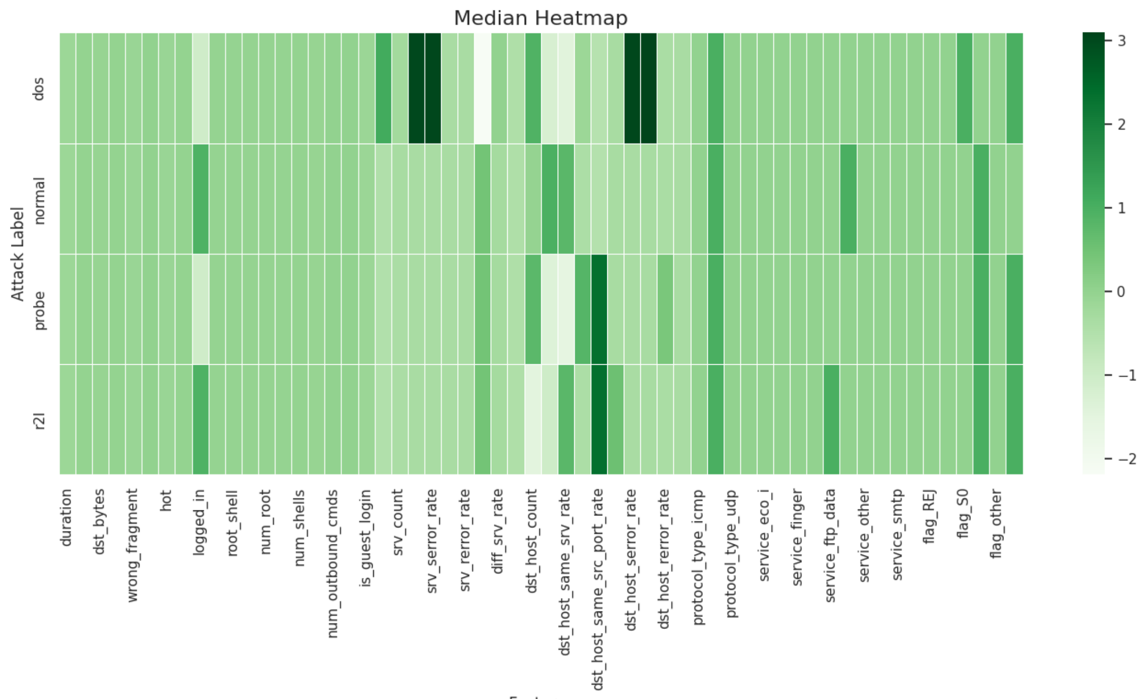


Figure 3: Median heatmap

2.1 OC-SVM with Normal Data only

First, we used a OC-SVM trained using normal traffic only and an rbf kernel.
Q: Considering that you are currently training only on normal data, which is a good estimate for the parameter nu? What is the impact on training performance? Try both your estimate and the default value of nu
The nu value has an important impact on the performance, as it "tells" the svm how many anomalies are expected to be in the training set. As we're using only normal data, nu should be very small to have a good performance, while the default value is nu = 0.5, so the svm will expect that half the points are anomalies. As we expected, the performance is good with a small nu (we chose nu = 0.001) and worse with nu = default. Here the classification report for the training is reported, for both cases:

(a) Performance with nu = 0.001

nu = 0.001	precision	recall	f1-score
0	0.91	1.00	0.95
1	0.98	0.76	0.86
accuracy			0.93
macro avg	0.95	0.88	0.90
weighted avg	0.93	0.93	0.92

(b) Performance with default nu

nu = 0.5	precision	recall	f1-score
0	0.99	0.50	0.66
1	0.44	0.99	0.61
accuracy			0.64
macro avg	0.72	0.74	0.64
weighted avg	0.83	0.64	0.65

Table 2: Performance with the default nu and the estimated nu, on normal data only

2.2 OC-SVM with all the Training Data

Then, we trained another OC-SVM using all the training data, estimating nu as the fraction of anomalies in the training set. As we can see in table 1b, the fraction of anomalies in the training set is 28.59%, so we considered nu = 0.29. **Q: Which model performs better? Why do you think that?** Compared to the model with only normal data, it performs much worse. SVM is typically designed for anomaly detection based on the "normal" class, introducing anomalous data makes the model less effective at distinguishing normal behavior from anomalies. Here, the classification report on the training set is reported:

nu = 0.29	precision	recall	f1-score
0	0.85	0.85	0.85
1	0.63	0.63	0.63
accuracy			0.79
macro avg	0.74	0.74	0.74
weighted avg	0.79	0.79	0.79

Table 3: Performance with nu = 0.29, on all the training data

2.3 OC-SVM with some anomalies

After that, we trained several OC-SVMs using the training set again, but each OC-SVM with a different fraction of anomalies. In particular, the percentage of anomalies considered (with the corresponding nu estimated) are reported in the table below:

Percentage	0%	10%	20%	50%	100%
Estimated nu	0.001	0.0385	0.0741	0.1667	0.29

Table 4: Percentage of anomalies with the related nu estimated

Q: Plot the f1-macro score for each scenario. How does the increasing ratio of anomalies affect the results? The f1-score increases as the number of anomalies considered increases, as we can see in the following table:

Percentage	0%	10%	20%	50%	100%
f1-score	0.90	0.63	0.68	0.73	0.74

Table 5: F1-score with different percentage of anomalies considered

2.4 Evaluation of different OCSVMs

To conclude this Task, we evaluated on the test set three OCSVMs among those we trained in the previous sections. In particular, we used the one trained with no anomalies, the one with 100% of the anomalies and the one with only 10% of anomalies.
Q: Is the best-performing model in the training set also the best here? Does it confuse normal data with anomalies? Which attack is the most confused? Yes, the model trained without anomalies is again the one that performs better. Here are shown the classification reports generated on the test set. We can see that the first model achieves the best results in all the considered metrics: precision, recall and f1-score.

(a) Performance of model trained on normal data only

nu = 0.001	precision	recall	f1-score
0	0.62	0.75	0.68
1	0.84	0.73	0.78
accuracy			0.74
macro avg	0.73	0.74	0.73
weighted avg	0.76	0.74	0.75

(b) Performance of model trained on all training data

nu = 0.29	precision	recall	f1-score
0	0.55	0.60	0.58
1	0.75	0.72	0.74
accuracy			0.67
macro avg	0.65	0.66	0.66
weighted avg	0.68	0.67	0.68

(c) Performance of model trained on 10% of anomalies

nu = 0.038	precision	recall	f1-score
0	0.38	0.75	0.51
1	0.67	0.30	0.41
accuracy			0.46
macro avg	0.52	0.52	0.46
weighted avg	0.56	0.46	0.45

Table 6: Performance of the three models on the Test set

The best model confuses some normal data with anomalies, as we can see in the confusion matrix reported here:

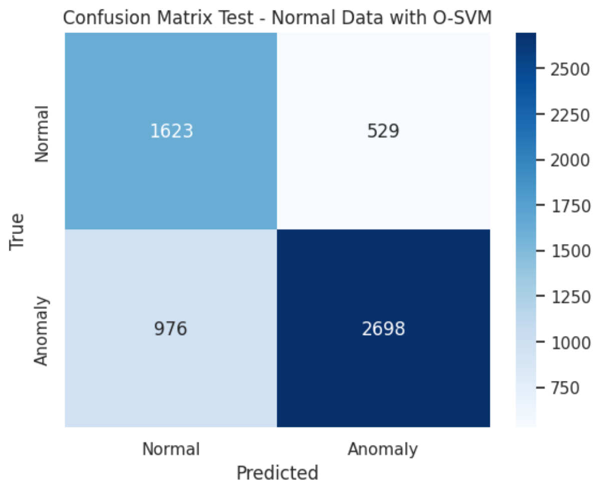


Figure 4: Confusion Matrix of the model trained on normal data only

In particular, by computing the detection accuracy for each attack, we can see that DoS is the most confused attack:

Attack	Accuracy
DoS	68.96%
Probe	83.96%

Table 7: Attack detection accuracy in the best model

The R2L attack is not shown as it is not present in the test set.

3 Task 3: Deep Anomaly Detection and Data Representation

This task required to use Deep Learning methodologies, in particular an Autoencoder, with a shrinking Encoder and an expansion Decoder. We chose to use 8 neurons for the latent dimension.

3.1 Training and Validating the Autoencoder, Reconstruction error threshold estimation

The Autoencoder was trained on normal data only, using a training set and a validation set obtained from the normal data in the original training set. The best results were achieved with 0.001 as the learning rate and 100 epochs, as the loss curves remained stable, as we can see in Figure 5. We also used a custom loss to correctly handle the numerical and categorical features. After training the Autoencoder,

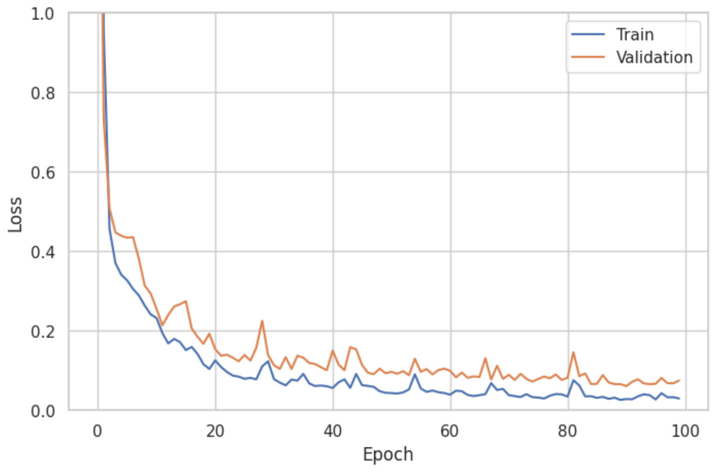


Figure 5: Training and validation losses for the Autoencoder

a reconstruction threshold was estimated. The idea was that if the model had a higher reconstruction error than the threshold, that point would be an anomaly. To estimate the threshold, the ECDF curve of the reconstruction error in the validation data was plotted (Figure 6). **Q: How did you pick the threshold? What is its value?** We selected the threshold at the 95th percentile of the reconstruction error distribution on the validation set. Since the validation set contains only normal traffic, this choice implies acceptance of a 5% False Positive Rate on normal data, which is a trade-off to ensure sensitivity to anomalies while keeping false alarms manageable. The threshold value is 0.1185.

3.2 Anomaly Detection with reconstruction error

Now the trained model is used to compute the reconstruction errors for each point in the full training set and test set. **Q: Plot and report the ECDF of the reconstruction errors for each point i) in the validation set; ii) in the full training set; iii) in the test set. Why the reconstruction errors higher on the full training set than on the validation one? And why are the reconstruction errors in the test set even higher?** The ECDF curves are reported in Figure 6. The reconstruction error is higher in the full training with respect to the validation set because the latter contains only normal data and was also used to train the Autoencoder, while the former contains also anomalies, which are difficult to be reconstructed effectively by the Autoencoder. Regarding the test set, since it contains data that were never seen by the Autoencoder, the errors are higher than the ones in the training set, which also contained data used during training. **Q: Use the threshold identified in the previous point to classify anomalies. How is the performance in the training, validation and test set?** The classification report for these three sets is shown in Table 8. Looking at 8a, we can see that the model is able to classify anomalies in a nice way (Recall 0.90, F1 0.90), generating a reconstruction error for them that is greater than the error in the normal data. Regarding the validation set (Table 8b), as we expected, we have Recall 0.95 for class 0, which is consistent with the chosen threshold, as it is exactly the same set we used to estimate the threshold. In the test set (Table 8c), we can see a drop in precision and recall for class 0, meaning that the model is having higher reconstruction error for such class (which can be possible, as the test data are unknown and may follow different patterns). Moreover, the model keeps good values in class 1 in the different metrics, meaning that it is still able to detect attacks in an efficient way.

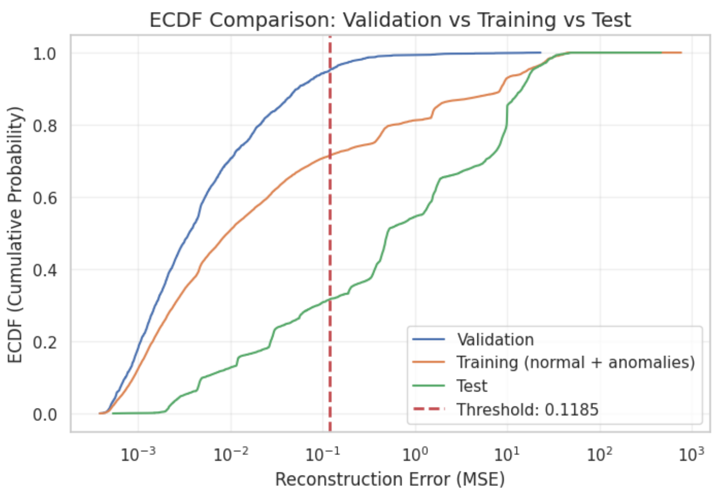


Figure 6: ECDF curves for the validation set, the full training set and the test set

(a) Performance on the full training set

	precision	recall	f1-score
0	0.96	0.96	0.96
1	0.90	0.90	0.90
accuracy			0.94
macro avg	0.93	0.93	0.93
weighted avg	0.94	0.94	0.94

(b) Performance on the validation set

	precision	recall	f1-score
0	1.00	0.95	0.97
1	0.00	0.00	0.00
accuracy			0.95
macro avg	0.50	0.47	0.49
weighted avg	1.00	0.95	0.97

(c) Performance on the test set

	precision	recall	f1-score
0	0.73	0.63	0.68
1	0.80	0.87	0.83
accuracy			0.78
macro avg	0.77	0.75	0.75
weighted avg	0.77	0.78	0.77

Table 8: Performance of the Autoencoder on training, validation and test set

3.3 Auto-Encoder's bottleneck and OC-SVM

For this section, we used instead the encoder of the Autoencoder we trained before to extract the bottleneck embeddings and use them to train a One Class SVM. Then, we used again the encoder to extract the embeddings of test data and used the trained SVM for anomaly detection. **Q: Compare the results with the best original OC-SVM and with the Autoencoder with reconstruction error. Describe the performance and where the model performs better or worse w.r.t. the original OC-SVM.** In Table 9 the performance of the OC-SVM trained using the embeddings is reported. We can compare it with the performance of the best OC-SVM by looking at Table 6a and with the performance of the Autoencoder by looking at Table 8c. We can see that the OC-SVM trained using the embeddings has a performance which is very similar to the one of the Autoencoder: the latter has a slightly better f1-score in the normal class, but the former has higher precision. A similar statement can be expressed looking at the anomaly class, where the Autoencoder has better precision but a slightly worse recall, but the models in the end present the same f1-score.
The original OC-SVM, instead, has a worse f1-score but a better precision in the anomaly class than the OC-SVM trained with the Autoencoder's bottleneck, while it has a slightly better recall and f1-score in the normal class.

	precision	recall	f1-score
0	0.77	0.54	0.64
1	0.77	0.91	0.83
accuracy			0.77
macro avg	0.77	0.72	0.74
weighted avg	0.77	0.77	0.76

Table 9: Performance of the SVM obtained from the Autoencoder's embeddings

3.4 PCA and OC-SVM

For the last part of this task, we used the Principal Component Analysis (PCA) to represent the data. We used PCA on the normal data in the training set and found the *elbow point* with respect to the explained variance, to fit and transform the data using the best estimated number of components. The 95% of the variance was expressed by 21 components.
Then, we transformed the test set using the same number of components. Finally, we used the transformed training set to train an OC-SVM to be used in the transformed test set. **Q: Compare results with the original OC-SVM and the OC-SVM trained using the Encoder embeddings. Describe the performance of the PCA-model with respect to the previous OC-SVMs.** The performance of this OC-SVM is reported in Table 10. Comparing it with the results in Tables 9 and 6a, we can see that with PCA we can achieve the good results for Anomaly Detection, with a f1-score value very close the ones of the OC-SVM trained with the embeddings and higher than the one of the original OC-SVM. On the other hand, this model outperforms the OC-SVM trained with the embeddings in the recognition of the normal class, with a slightly better f1-score, but it does not reach the value obtained with the original OC-SVM.

	precision	recall	f1-score
0	0.71	0.62	0.66
1	0.79	0.85	0.82
accuracy			0.77
macro avg	0.75	0.74	0.74
weighted avg	0.76	0.77	0.76

Table 10: Performance of the SVM obtained from PCA

4 Task 4: Unsupervised Anomaly Detection and Interpretation

This section focuses on the use of Unsupervised Learning algorithms to classify data.

4.1 K-means cluster interpretation

We started by fitting K-Means with 4 clusters and the full training set, including anomalies. **Q: How big are the clusters? How are the attack labels distributed across the clusters? Are the clusters pure (i.e., they consist of only one attack label)?** In Table 11, the distribution of labels across the clusters, along with the dimension of each cluster, is reported. We can see that there are not

Cluster	Normal	DoS	Probe	R2L	Dimension
0	11455	233	1148	181	13017
1	37	1657	83	0	1777
2	551	343	1038	0	1932
3	1405	680	20	0	2105

Table 11: Label distribution across K-Means clusters

pure clusters, but we can also notice that R2L is contained only inside the biggest cluster, which also contains the majority of the normal samples.

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? In Figure 7, the Silhouette plot per cluster is reported. We can see that there is a cluster with a much higher silhouette (Cluster 1), while the other 3 clusters show poorer performances. Specifically, the silhouette plots for Cluster 0 and Cluster 2 reveal negative values, implying that many data points in those clusters are misclassified. From Table 11, we can see which attack labels are in those clusters: in particular, both in Cluster 0 and Cluster 2 the two most common labels are *Normal* and *Probe*, while in Cluster 1 (the one with the highest silhouette) almost all the points are labeled as *DoS*. **Q: Use the t-SNE algorithm to obtain a 2D visualization of your points. Plot and report: i) t-SNE using all training data and as label the cluster ID. To do this, try different values (max 3) of perplexity to determine the best visualization. ii) Use the t-SNE with the best-looking perplexity and plot all training data with the attack label. Can you find a difference between the two visualizations? What are the misinterpreted points?** We tried three different perplexity values (10, 30, 50), the one that offered the best visualization was 30. In Figure 8 the t-SNE plots are shown:

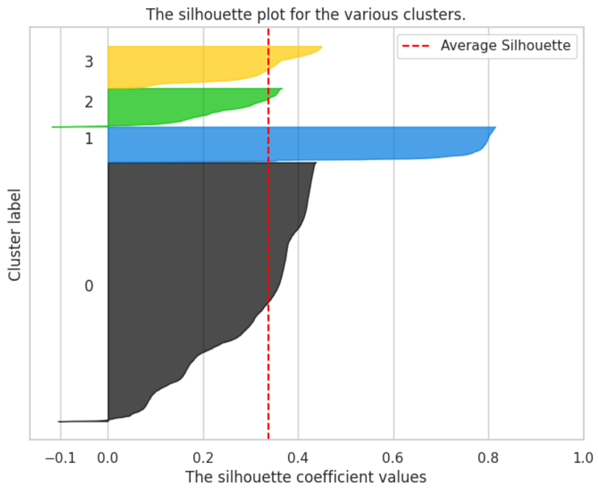
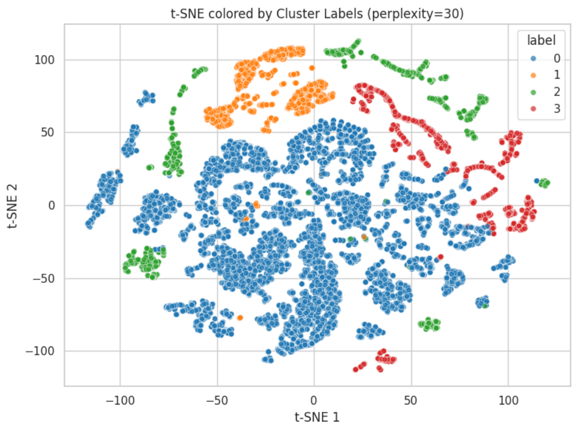
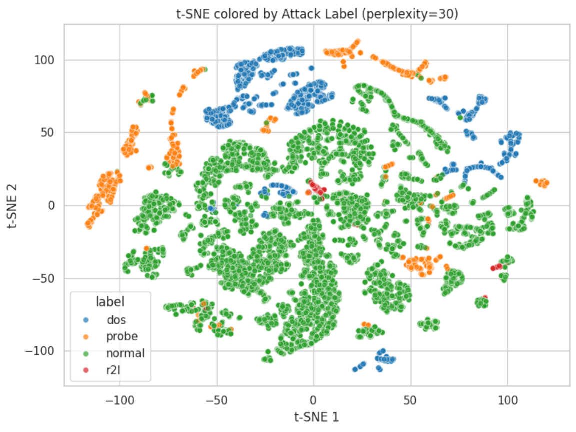


Figure 7: Silhouette plot for each cluster



(a) t-SNE plot with Cluster ID as label



(b) t-SNE plot with Attack Labels as label

Figure 8: t-SNE plots (perplexity = 30)

The difference between the plots is that in Figure 8a we can see how clusters are positioned in the space, while in Figure 8b we can see which is the attack label of each point. By comparing the plots, we can identify the misinterpreted points: in particular, we can see that the points with label *R2L* are always in Cluster 0, together with most of the points labeled *Normal*. We can also see that in the rightmost part of the plot there are some points labeled as *Normal* that were considered part of Cluster 3 and some other points labeled as *DoS* that instead were split in clusters 2 and 3, while most of the points with

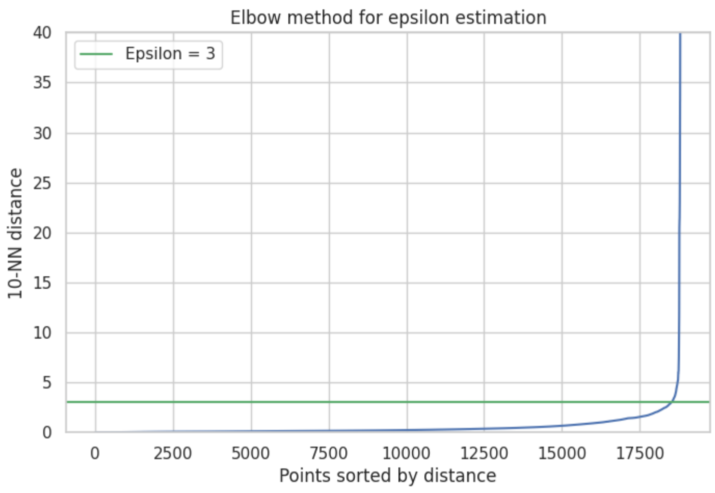


Figure 9: Plot used to estimate ϵ according to the *elbow rule*

such label are in Cluster 1.

4.2 DB-Scan anomalies are anomalies?

In this last section of the laboratory, we used DB-SCAN for anomaly detection. This clustering algorithm required two parameters: ϵ and *min_points*. The latter was estimated by evaluating the k-means result and looking for the smallest cluster consisting only of normal data. After trying different values for k, we found that with k=8 a cluster consisting of 10 normal points and no anomalies was generated, so we fixed min_points=10.
Then, we used the *elbow rule* to estimate the ϵ parameter required by DB-SCAN, based on the increasing distance between each point and its 10 neighbors. Based on the results in Figure 9, we chose $\epsilon = 3$. **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)?** After having evaluated DB-SCAN with the chosen parameters in the full training set, we extracted the content of the noise cluster, which is reported in the table below:

Label	Count
Normal	169
Probe	24
R2L	4
DoS	1

Table 12: Content of the noise cluster in DB-SCAN

We can see that in the noise cluster not only there are normal points, they are even the majority.