Machine learning for Networking: Projects by Group 17

Project 9: SSH shell attacks

Matteo Scursatone, Fabio Lorenzato, Alessandro Milani, Vincenzo Ricciardelli

First Semester 2023/2024

Indice

1 Data exploration and pre-processing

The first part of the project was all about trying to understand how the supplied dataset is structured and what kind of data is present in it.
The analysis was carried out by visualizing the data via the use of libraries such as `seaborn` and the `mathplotlib` utilities, which are the ones that has been explained in the laboratories. The dataset was handled using a `pandas` dataframe.

1.1 Temporal series analysis

For a first, quick glance at the temporal series, that being when the attacks occurs temporarily, we chose to represent it with some **line plots** to have a simple representation of how the numbers of attacks changes over time. To do so, we have extracted the date and the hour of the attacks from the timestamps present in the dataframe.

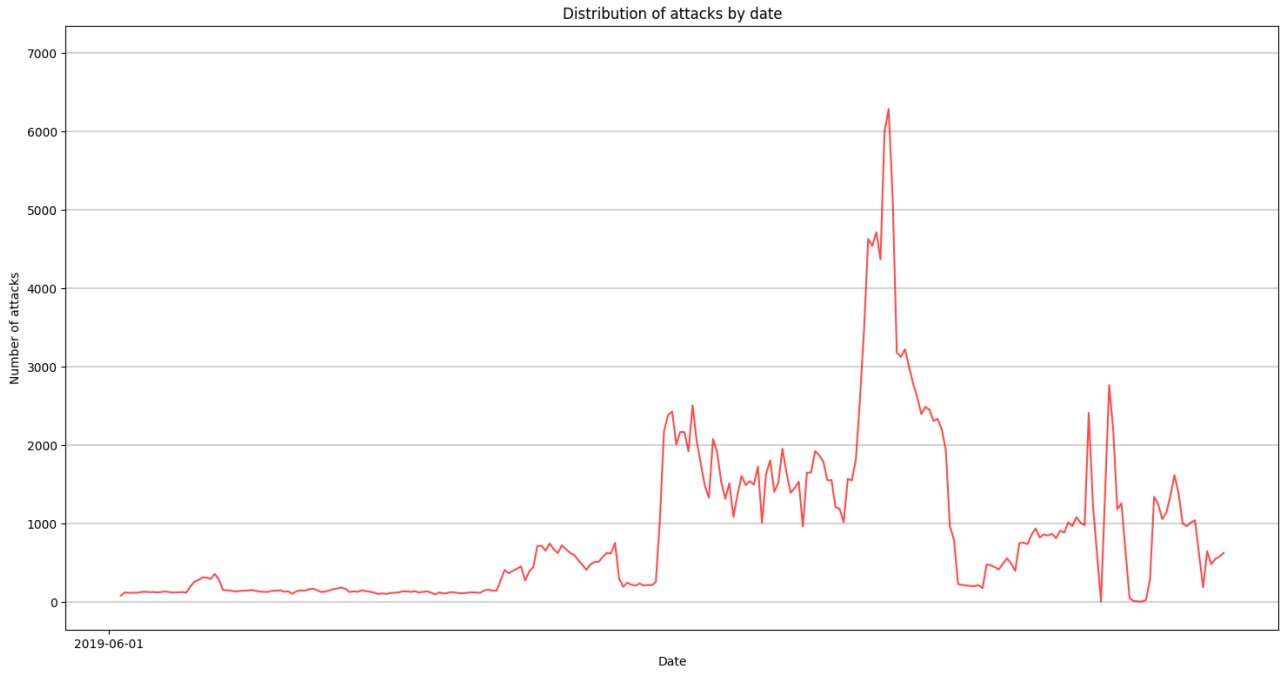


Figura 1.1: Line plot showing the number of attacks over time

As shown in picture 1.1, you can see that, in the first period of time, from June 2019 to October of the same year, there hasn't been that much attacks in general, which are, on contrary, concentrated in the period from around mid October until the end of December of the same year.

That is even more evident when looking at the plot for each individual month, as shown in picture 1.2. You can also notice how the number of attacks has increased starting by October 13th and has only started decreasing, at least in a meaningful way, by the 11th of December. This two dates have been marked using some vertical lines.

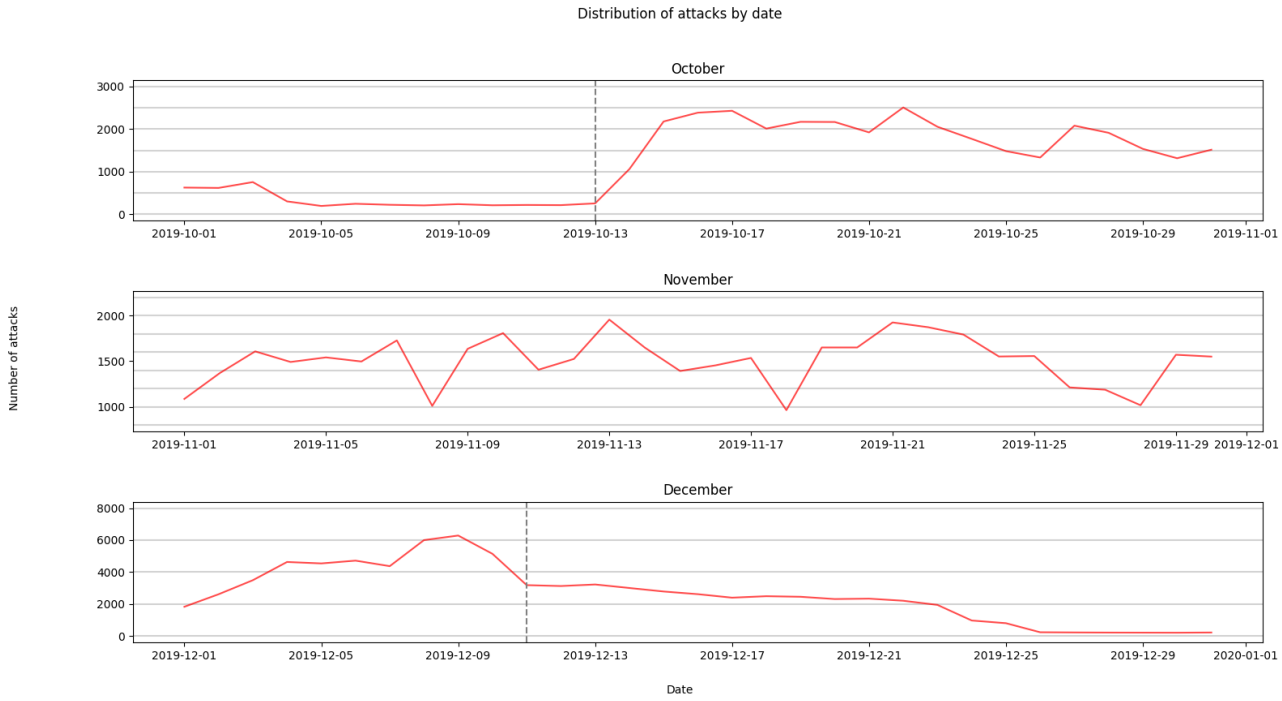


Figura 1.2: Line plot for each individual month

After looking at the distribution of the attacks by date, out of curiosity we took a look at how the attacks are distributed during the day. As shown in picture 1.3, you may notice that most of the commands are sent between 11pm and 12pm(the image is shown using the 24 hour format).

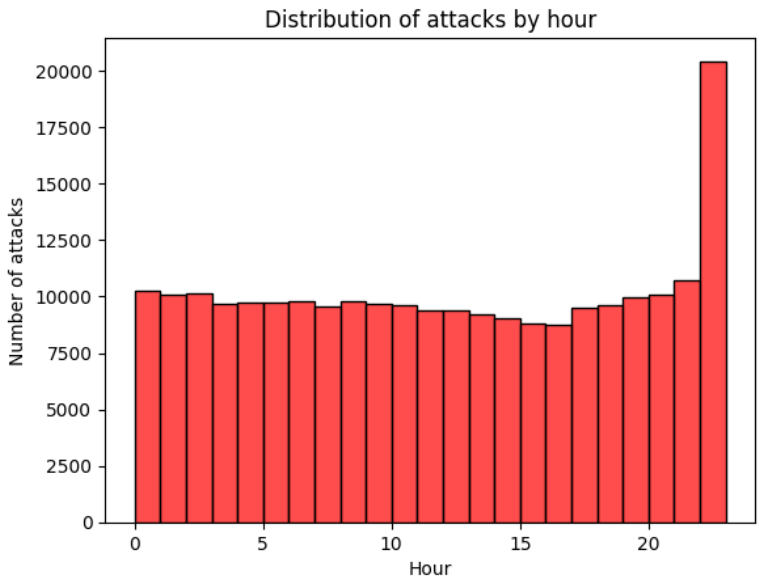


Figura 1.3: Hist plot of number of attacks during each hour of the date

Taking a closer look to the distribution of commands during the day for each month, it seems like it is also true for each month, while remaining constant during the whole day, as shown in picture 1.4. So its not the result of some divergent data.

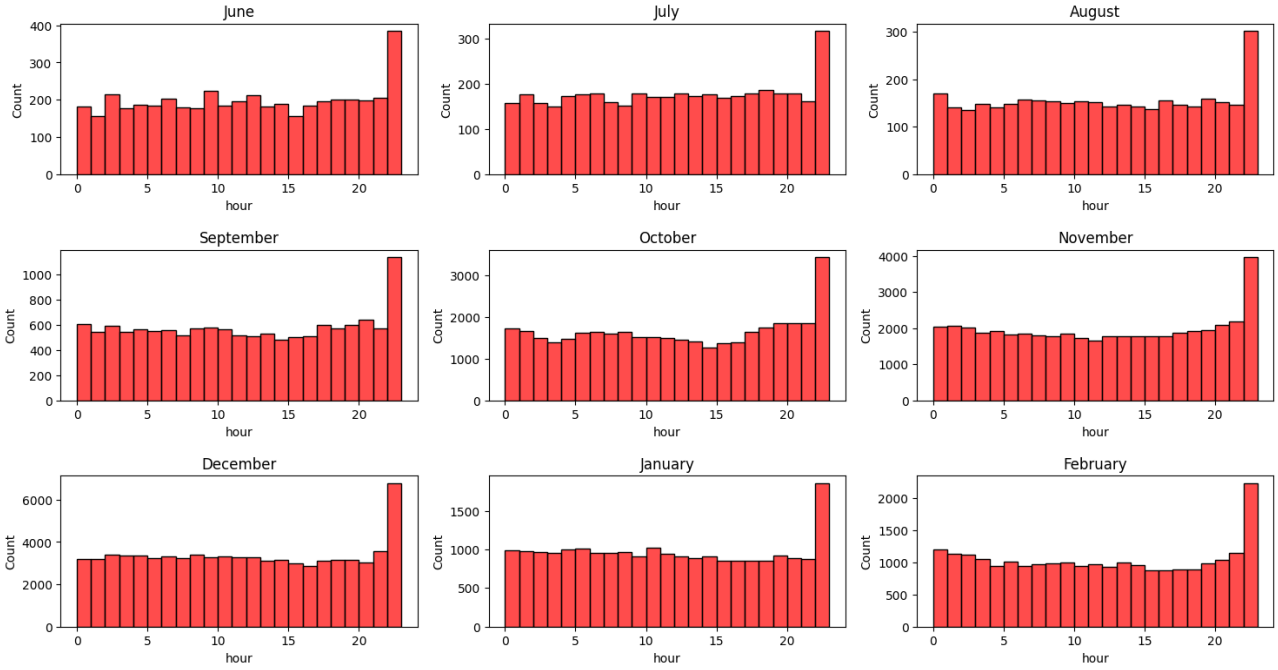


Figura 1.4: Hist plot of number of attacks during each hour of the date by month

1.2 Empirical Distribution of the words

Taking a look at the empirical distribution of the character count of each record, you may notice that most of the records have a small character count(which is smaller than 1934 for 99% of the records, which is evident by looking at the 99th percentile of the distribution, as also shown by the table 1.1). This is quite noticeable by looking at the picture 1.5. Furthermore, only 1% of the records have a character count greater than 1934, by a mile, which is quite evident by looking at the third subplot, which highlight how some divergent values can have an huge impact on this kind of plots.

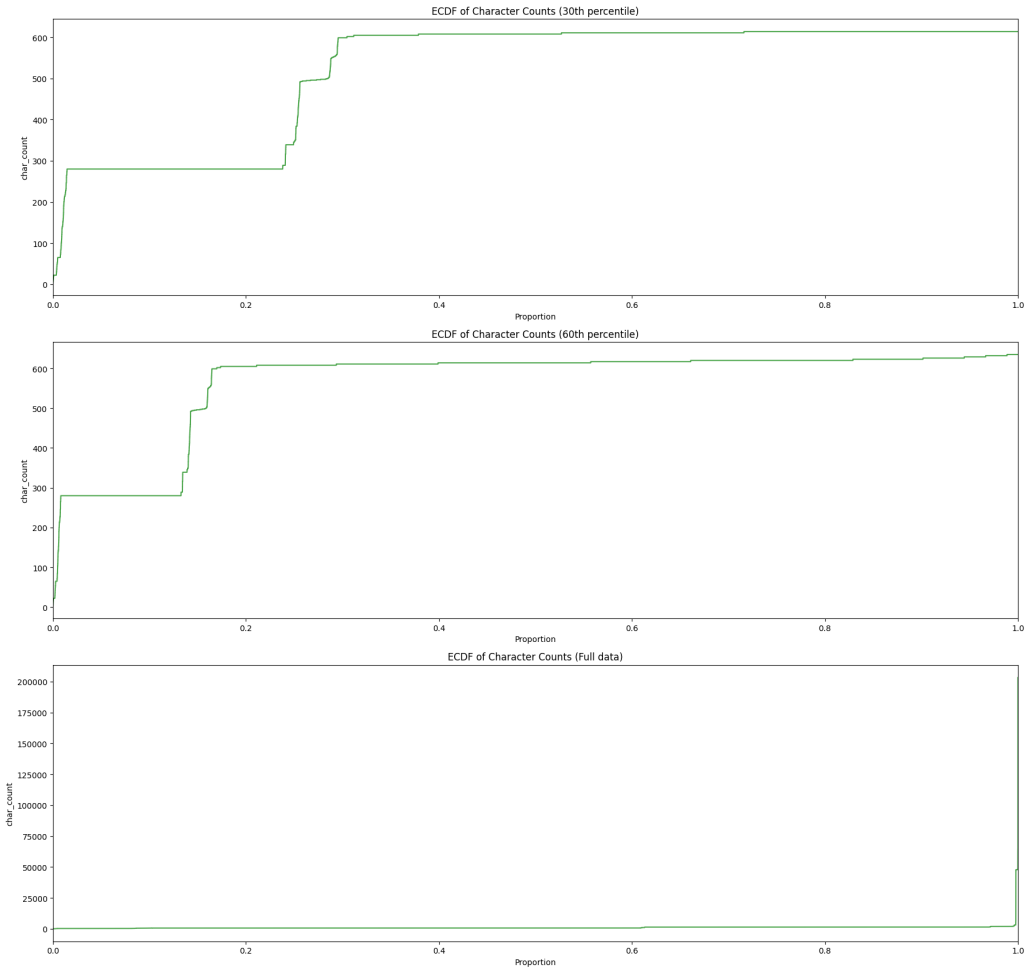


Figura 1.5: Three plots figuring the ECDF of the number of character per session.

Percentile	Character Value	Words Value
0th	3	1
10th	599	76
20th	611	78
30th	614	78
40th	620	78
50th	620	78
60th	635	78
70th	1414	92
80th	1416	92
90th	1418	92
99th	1934	107
99.9th	47807	168
100th	203442	22438

Tabella 1.1: Percentile Values

As expected, the same thing is true for the number of words in each record, as shown by the plots in picture 1.6. But, again the number of words per session is quite small for 99% of the records, getting similar results to the empirical distribution of characters, resulting in similar plots. The distribution of words and characters remains quite constant between the 10th and the 60th percentile and the 70th and the 90th. The criteria for discerning each word will be explained in next section.

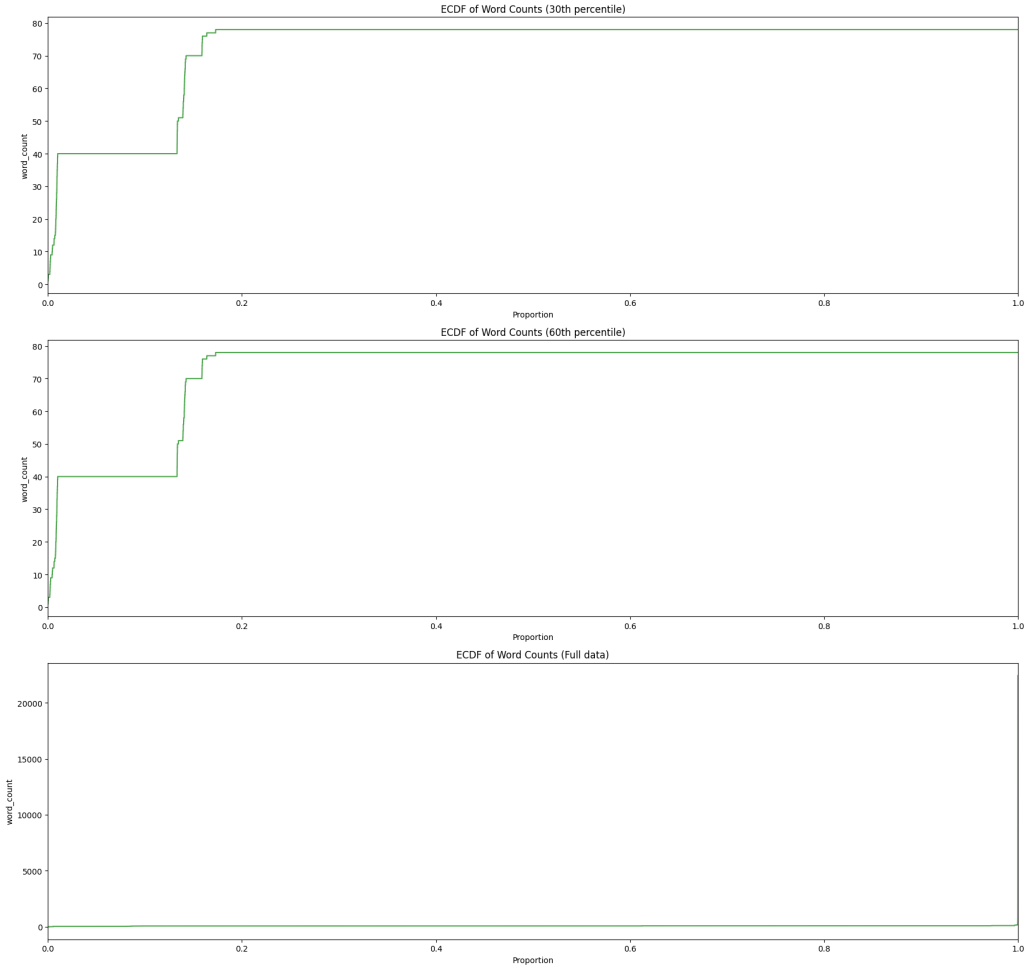


Figura 1.6: Three plots figuring the ECDF of the number of words per session.

1.3 Most common words

Another aspect that we were required to analyze are the most common words in each session. We have identified as words each sequence of symbols, containing at least one letter, separated by a space characters, mainly for simplicity.

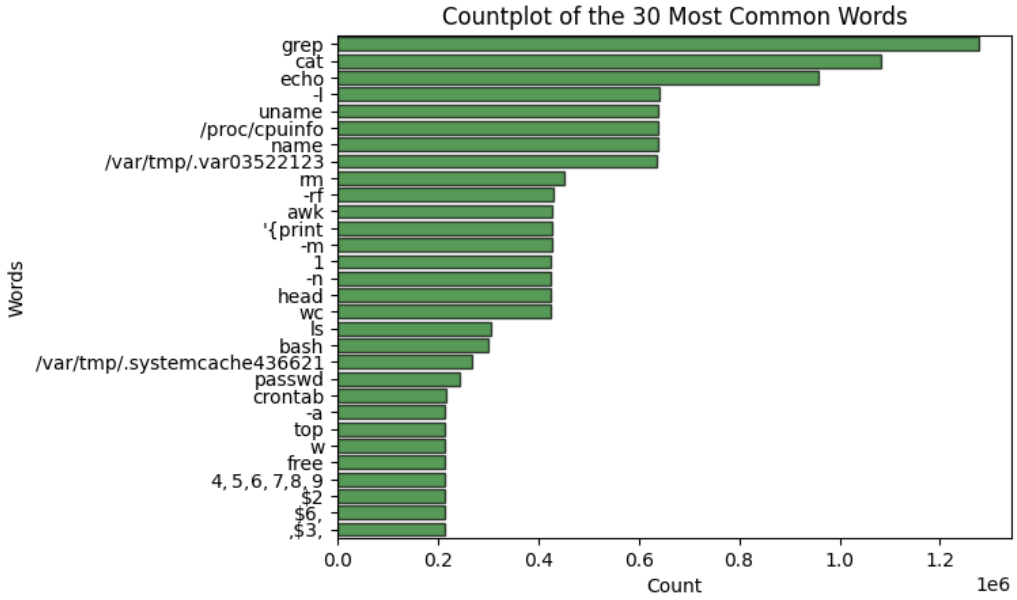


Figura 1.7: Count plot of the words present in the dataset.

To identify them most common words a simple count plot has been used, which is represented in picture 1.7. We found out that in the dataset are present 18836453 words, of which 400821 unique ones.
Furthermore, some of the most common words are actually command used to read file, such as cat, or to filter its content(grep). During this part of the analysis, we also assigned each word with a unique integer identifier, which has been useful later on to make part of our code run faster.

1.4 Session intent analysis

Another interesting aspect that we have been required to observe is the distribution of the intents of each session.

First of all, we have decided to find out which are the most common intents, which were stored in the *set_fingerprint* field of the dataset. Via a simple count plot, shown in picture 1.8, it was evident that the Discover and Persistence intents were the most common by far. The number of labels such as Impact of Harmless is irrelevant when compared to the others.

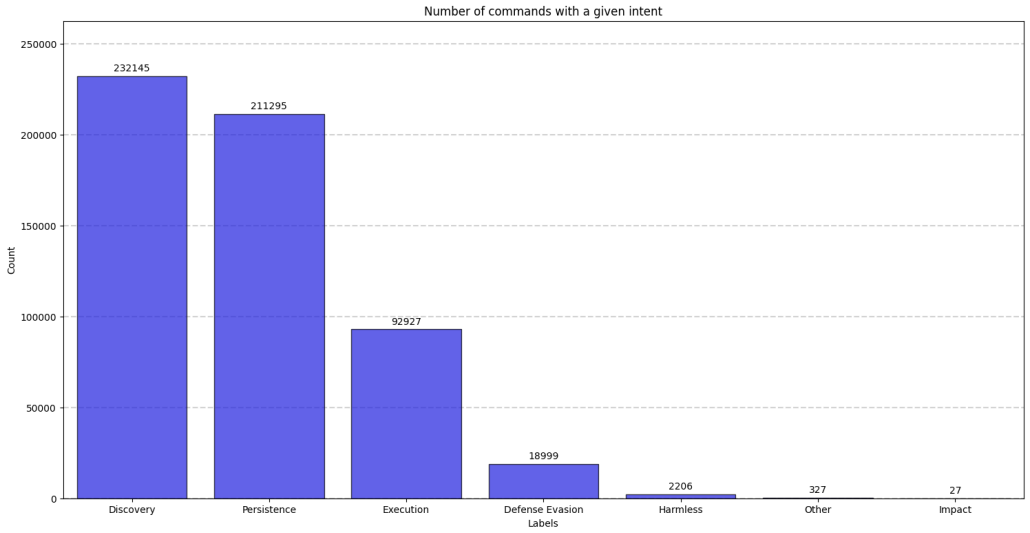


Figura 1.8: Count plot of intents of all sessions.

When it comes to the number of intents per entry, by creating a single count plot we were able to find out that most of the session have two distinct intents, sometimes 3, as shown by picture 1.9, with an average of 2.39 intents per session.

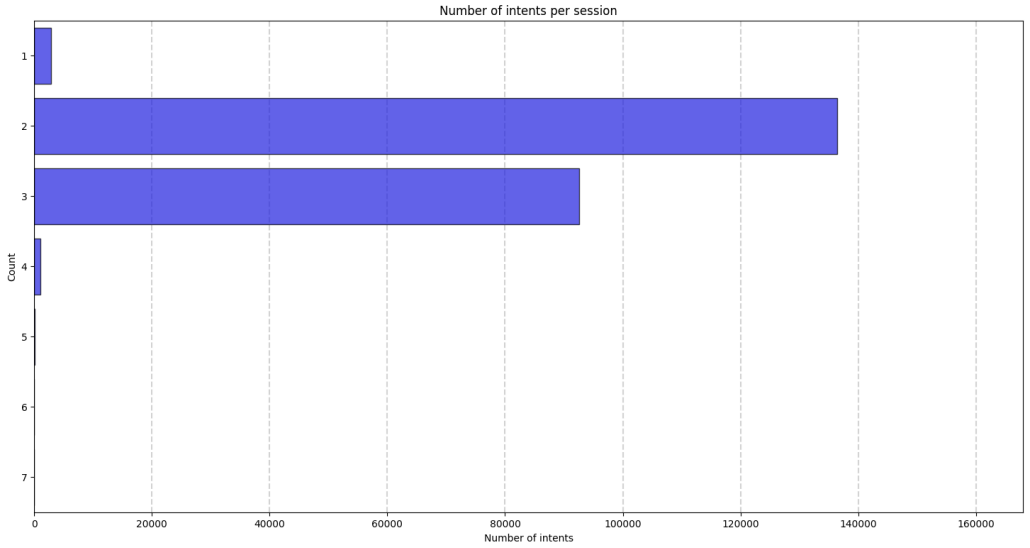


Figura 1.9: Count plot of the number of intents per session.

Taking a look at the line plot we made that shows how the number of intents changes over time it is possible to observe how the session intent count spikes up in the period of time from the 4th of September to the 12th of December. As shown in the first plot of picture 1.10, in which this period of time has been highlighted in red. To observe if there is any evident correlation with the period of time with most sessions per day(that being the period in-between October 13th and December 11th) we have highlighted this period in the second subplot of the same picture, without noticing any evident correlation. other that being in the period of time previously shown. We chose to remove the line shadowing from those plots because it didn't show any meaningful information, and the mean value for each day was fine.

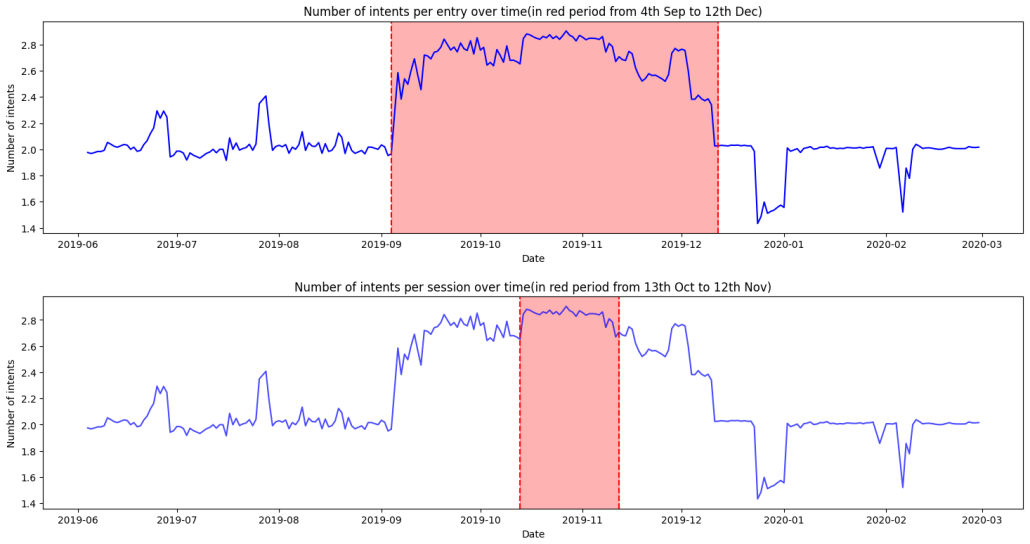


Figura 1.10: The plots showing how the intents per session changes over time

At last, we tried to observe how the number of sessions with each label changes over time, by making a simple line plot shown in picture 1.11. It is quite evident how there is some kind of correlation between the session with label *Persistence* and *Discovery* for most part of the plot.

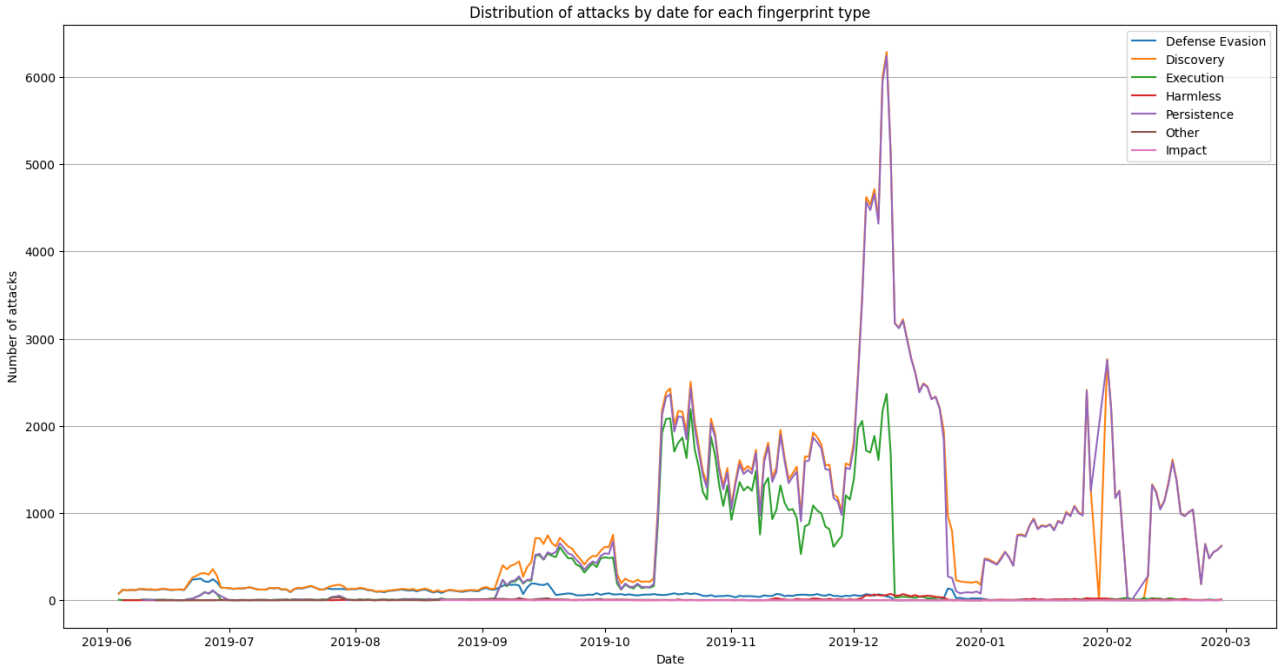


Figura 1.11: Line plot that shows the distribution of session with each label changes over time.

1.5 Bag of words

At this point, we were asked to assign a numerical representation and generate the bag of words by those ones. We have previously assigned a unique integer value as stated in section 1.3, by using two different hash maps to take advantage of the uniqueness of the key and get fast lookup times.

When it comes to the bag of words itself, one with every words in the dataset would have resulted in a program with a excessive space complexity (to run the program from the beginning to this point, it would have required around 640 GB of main memory, which any of us hardly had) so we settled for the 100 most common words, which we have computed using the `Counter` library. After all, words with only a few instances wouldn't have much impact overall. That library was also used to assign the value of each word into the bag of words itself, which in the end was a simple `Pandas` dataframe.

1.6 TF-IDF values

To compute the Term Frequency-Inverse Document Frequency of each word in the top 100, we used the library supplied by the `sklearn` module, an in particular the `TfidfTansformer` class, which did most of the heavy lifting. We just had to set the index as the original dataset.

2 Supervised learning

In this chapter, we explore the application of supervised learning and classification techniques to analyze attack session data.
The primary objective is to categorize each session according to its tactics, utilizing textual content as a key feature. To achieve this goal, models from the `scikit-learn` library were employed, alongside `mathplotlib` for visualizing results in an easily understandable format.

2.1 Preparation

To make the dataset suitable to be analyzed by the various models, we start from splitting the column Set_Fingerprint in 7 different binary columns (with value 1 if the label is present in *Set_Fingerprint*, otherwise 0). In this way, we decompose the problem into seven binary classification problems.
Following this, the dataset was divided into training and testing sets with a 70-30 proportion.

2.2 Analyze normalized data

2.2.1 Choosing models

The first main section of this analysis is based on the use of a normalized dataset for perform the classification, but before that we need to choose what model utilized. To achieve this, a preliminary test was conducted with all base models available in the `scikit-learn` library. Models that exhibited errors or required excessive time were discarded.
After the end four models were selected: `LogisticRegression, RandomForestClassifier, AdaBoostClassifier, LinearSVC`.
At this point, were created 4 different pipeline for concatenate the vectorization of the training set and the transformation in a normalized Tf-idf rappresentation with the use of the class `TfidfVectorizer`, followed by the call to the class `MultiOutputClassifier` utilized for extend all the model finked for a binary classification in multi label.

2.2.2 General results

After the execution of the different pipeline and a first run with both, training and test dataset, the average results obtained are as follows:

At this general level it can be observed that, with the exception of the Random Forest model, all other models have achieved high accuracy and low loss in both training and test set predictions (the considerable difference of the Random Forest model, is leagued to the fact that

Training Set	**Accuracy**	**Hamming Loss**
Logistic Regression	0.980978	0.003020
Random Forest	0.966437	0.007986
ADA Boost	0.984607	0.002328
Linear SVC	0.997542	0.000369

Tabella 2.1: average results of default models

Test Set	**Accuracy**	**Hamming Loss**
Logistic Regression	0.982707	0.002932
Random Forest	0.483887	0.077701
ADA Boost	0.982907	0.002628
Linear SVC	0.984595	0.002374

Tabella 2.2: average results of default models

it is the only model that, due to time constraints, has not utilized the default parameters).

2.2.3 Matrix and Reports results

But it's not all good, because by a more accurate analysis of the various Confusion Matrix and Classification report, we can find a very strong imbalance for classes Impact and Harmless:

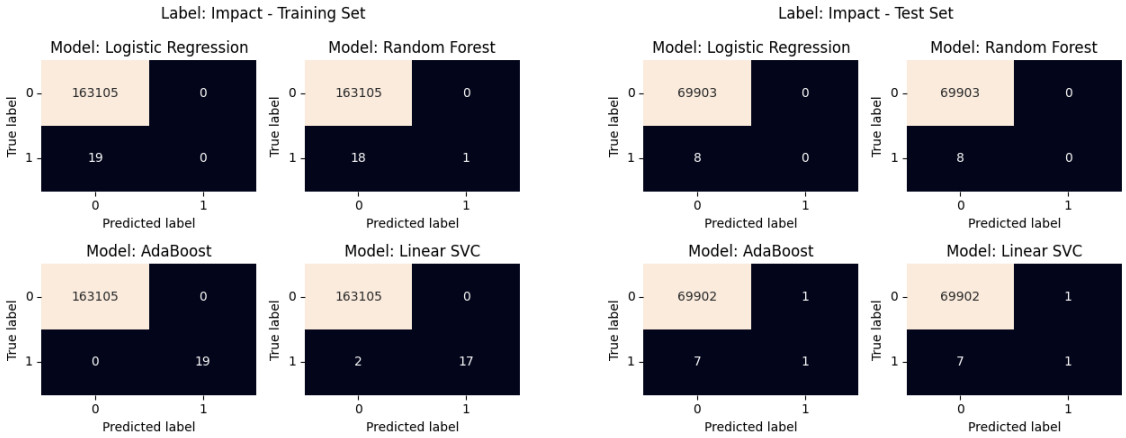


Figura 2.1: Confusion Matrix for label Impact

From this excerpt of the various confusion matrices, it is evident that for the Impact label, depending on the model used, there are issues attributable to a situation of overfitting (ex. AdaBoost or Linear SVC, that have an high recall for the training set and an extremely low recall for the test set) or underfitting (ex. Logistic Regression that miss all the correct label for both, training and test set). This discrepancy is attributed to the limited presence of only 27 samples with this particular label, in the dataset of over 160000 elements.
We have also a similar situation for the Harmless label:

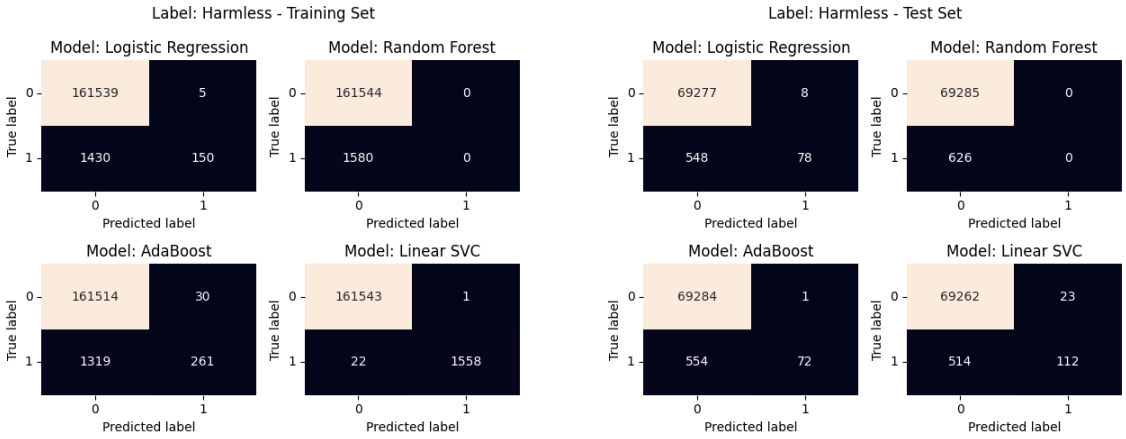


Figura 2.2: Confusion Matrix for harmless Impact

As evident from the graphs, there is limited recognition of this label. However, the situation is comparatively better than that of the Impact label. Therefore, it might be more informative to examine the data provided by the classification report.

Tabella 2.3: Results of the LR model

Logistic Regression				
Class	**Precision**	**Recall**	**F1-score**	**Support**
Training Set				
Defense Evasion	0.99	0.97	0.98	13254
Discovery	1.00	1.00	1.00	162492
Execution	1.00	0.99	0.99	64958
Harmless	0.97	0.09	0.17	1580
Persistence	1.00	1.00	1.00	147935
Other	1.00	0.95	0.98	217
Impact	0.00	0.00	0.00	19
Micro avg	1.00	0.99	1.00	390455
Macro avg	0.85	0.71	0.73	390455
Weighted avg	1.00	0.99	0.99	390455
Samples avg	1.00	0.99	1.00	390455
Test Set				
Defense Evasion	0.99	0.96	0.98	5745
Discovery	1.00	1.00	1.00	69653
Execution	1.00	0.99	0.99	27969
Harmless	0.91	0.12	0.22	626
Persistence	1.00	1.00	1.00	63360
Other	0.98	0.95	0.96	110
Impact	0.00	0.00	0.00	8
Micro avg	1.00	0.99	1.00	167471
Macro avg	0.84	0.72	0.74	167471
Weighted avg	1.00	0.99	0.99	167471
Samples avg	1.00	0.99	1.00	167471

Tabella 2.4: Results of the RF model

Random Forest				
Class	**Precision**	**Recall**	**F1-score**	**Support**
Training Set				
Defense Evasion	0.99	0.95	0.97	13254
Discovery	1.00	1.00	1.00	162492
Execution	0.97	0.97	0.97	64958
Harmless	0.00	0.00	0.00	1580
Persistence	0.98	1.00	0.99	147935
Other	1.00	0.94	0.97	217
Impact	1.00	0.05	0.10	19
Micro avg	0.99	0.99	0.99	390455
Macro avg	0.85	0.70	0.71	390455
Weighted avg	0.98	0.99	0.99	390455
Samples avg	0.99	0.99	0.99	390455
Test Set				
Defense Evasion	0.99	0.95	0.97	5745
Discovery	1.00	1.00	1.00	69653
Execution	0.44	0.99	0.61	27969
Harmless	0.00	0.00	0.00	626
Persistence	0.98	1.00	0.99	63360
Other	0.98	0.94	0.96	110
Impact	0.00	0.00	0.00	8
Micro avg	0.82	0.99	0.90	167471
Macro avg	0.63	0.70	0.65	167471
Weighted avg	0.89	0.99	0.93	167471
Samples avg	0.82	0.99	0.89	167471

Tabella 2.5: Results of the ADA model

ADA boost				
Class	**Precision**	**Recall**	**F1-score**	**Support**
Training Set				
Defense Evasion	0.99	0.97	0.98	13254
Discovery	1.00	1.00	1.00	162492
Execution	1.00	0.99	0.99	64958
Harmless	0.90	0.17	0.28	1580
Persistence	1.00	1.00	1.00	147935
Other	1.00	1.00	1.00	217
Impact	1.00	1.00	1.00	19
Micro avg	1.00	0.99	1.00	390455
Macro avg	0.98	0.88	0.89	390455
Weighted avg	1.00	0.99	1.00	390455
Samples avg	1.00	0.99	1.00	390455
Test Set				
Defense Evasion	0.99	0.97	0.98	5745
Discovery	1.00	1.00	1.00	69653
Execution	1.00	0.99	0.99	27969
Harmless	0.99	0.12	0.21	626
Persistence	1.00	1.00	1.00	63360
Other	0.99	0.96	0.98	110
Impact	0.50	0.12	0.20	8
Micro avg	1.00	0.99	1.00	167471
Macro avg	0.92	0.74	0.77	167471
Weighted avg	1.00	0.99	0.99	167471
Samples avg	1.00	0.99	1.00	167471

Tabella 2.6: Results of the LSVC model

Linear SVC				
Class	**Precision**	**Recall**	**F1-score**	**Support**
Training Set				
Defense Evasion	1.00	0.98	0.99	13254
Discovery	1.00	1.00	1.00	162492
Execution	1.00	1.00	1.00	64958
Harmless	1.00	0.99	0.99	1580
Persistence	1.00	1.00	1.00	147935
Other	1.00	1.00	1.00	217
Impact	1.00	0.89	0.94	19
Micro avg	1.00	0.99	1.00	390455
Macro avg	1.00	0.98	0.99	390455
Weighted avg	1.00	1.00	1.00	390455
Samples avg	1.00	1.00	1.00	390455
Test Set				
Defense Evasion	0.99	0.97	0.98	5745
Discovery	1.00	1.00	1.00	69653
Execution	1.00	0.99	1.00	27969
Harmless	0.83	0.18	0.29	626
Persistence	1.00	1.00	1.00	63360
Other	0.99	0.96	0.98	110
Impact	0.50	0.12	0.20	8
Micro avg	1.00	0.99	1.00	167471
Macro avg	0.90	0.75	0.78	167471
Weighted avg	1.00	0.99	1.00	167471
Samples avg	1.00	0.99	1.00	167471

It's indeed evident from the results that the "Harmless" and "Impact" labels exhibit notably different performance compared to the other five labels.[1] The precision for the "Harmless" label remains consistently high, indicating that when the model predicts an instance as "Harmless," it's usually correct. However, the recall for "Harmless" is relatively low, suggesting that the model tends to miss many instances of this class.
On the other hand, the "Impact" label shows poor performance in terms of both precision and recall, especially in the test set.
Overall, these observations highlight the need for further analysis and potentially model improvement, especially for the "Impact" label, to enhance the model's ability to accurately recognize instances of this class.

2.2.4 Hyper-parameters optimization

After this first execution we tried to improve the results by optimizing the hyper-parameters.
To comprehensively explore a wide range of parameter combinations, we employed the `GridSearchCV` class from the `scikit-learn` library. This class, equipped with a table listing parameters and their potential values, systematically executes all conceivable combinations, allowing for an exhaustive search across the parameter space.
In addition to the parameter list (called *param_grid*), the value of the variable *cv* was also set to the value 3, which is used to determines the cross-validation splitting strategy (in this case a 3-fold cross validation).

[1]Note: For this reason, in the following sections of this chapter their analysis will be neglected (full reports and confusion matrix will still be available in the Jupyter Notebook)

Hyper-parameters Results

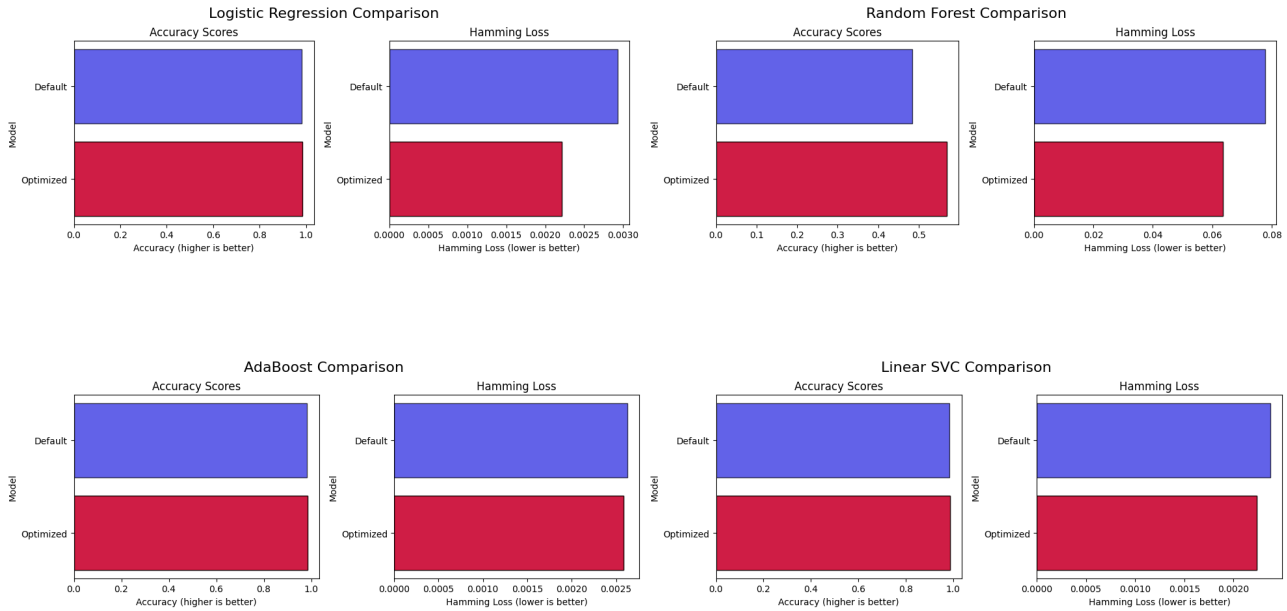


Figura 2.3: Post-optimization model improvement

Comparing the various models before and after optimization, marginal overall improvement is observed in all models, with significant differences in loss for Logistic Regression and in both accuracy and loss for Random Forest.
However, no noteworthy improvements are evident in the previously discussed problem labels. The only modest improvement is seen in the recall of the Impact label in the Logistic Regression model, which goes from 0 to 0.79 (with an accuracy of 1.0) for the training set and from 0 to 0.12 for the test set, but with an accuracy of only 0.25.

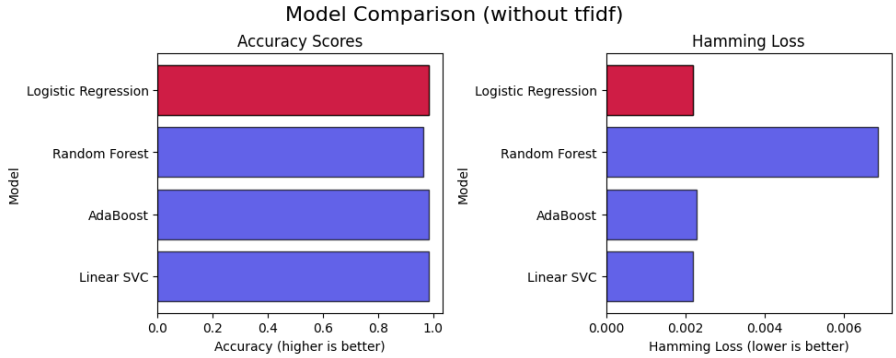
2.3 Analyze non-normalized data

At this point, an attempt was made to fit a model using an un-normalized dataset.
To implement this modification, we adjusted the pipeline by replacing the `TfidfVectorizer` class with `CountVectorizer`. This alteration ensures that only the vectorization of the input strings is executed, omitting the transformation into a Tf-idf representation.

2.3.1 Base model

Consistent with our previous approach, we initially ran various models with default parameters. The execution of these models yielded the following results:

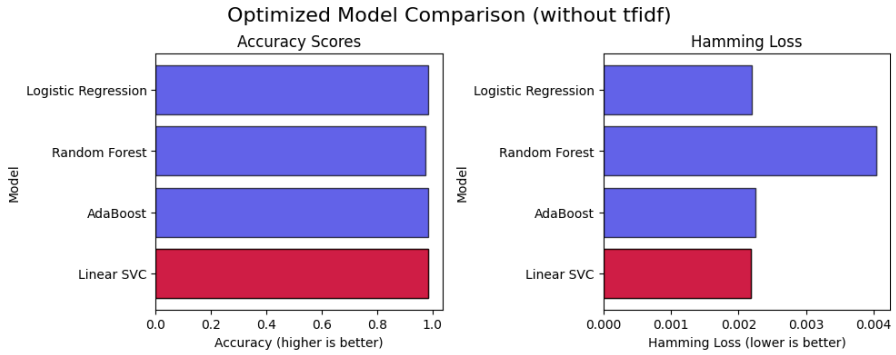
Upon comparing the current situation with previous iterations, it's notable that the Random Forest model attains significantly higher accuracy without the implementation of the tf-idf function. Furthermore, the loss has decreased by an order of magnitude, dropping from approximately 0.08 to 0.007.
Additionally, the Logistic Regression model now is the best performer for both accuracy and loss, even with default parameters.
Despite these changes, there are no discernible improvements concerning the problematic labels Harmless and Impact.



2.3.2 Hyper-parameters Optimization

Subsequently, we conducted an optimization of the parameters while maintaining non-normalized data as input to explore whether this alteration could yield different outcomes.
In contrast to the initial execution, certain parameter values had to be omitted during the optimization process. This was crucial to prevent the associated model from exceeding the time constraints and resulting in excessively long run times compared to the available time.

Hyper-parameters Results



From a preliminary overview, it is evident that the Random Forest hamming loss has further decreased to 0.004, although it still remains notably higher compared to other models, which hover slightly above 0.02.
Moreover, the best-performing model among those with optimized hyperparameters is now the Linear SVC for both loss and accuracy. For a more detailed analysis of individual label evolution, we observe the following:

- **Logistic Regression**: No notable changes are observed.
- **Random Forest**: While the overall situation appears somewhat static, there is a remarkable improvement in the recall for the Impact label, rising from 0.05 and 0 to 0.89 and 0.12 in the training and test sets, respectively. Additionally, there are slight improvements in the recall for the Harmless label, moving from 0 and 0 to 0.08 and 0.07.
- **ADA Boost**: No notable changes are observed.
- **Linear SVC**: No notable changes are observed.

Test Set	**Accuracy**	**Hamming Loss**
Ensemble	0.984938	0.002336
Ensemble (without TF-IDF	0.985639	0.002203

Tabella 2.7: average results of ensemble models

To close this chapter, employing `GridSearchCV` to optimize hyperparameters in this context did not yield substantial improvements for most models.
Exploring alternative techniques or redefining the set of possible hyperparameters might offer better results.

2.4 Other tests

At this point, considering the less than ideal results achieved with the previous methods, we opted to explore the implementation of two additional approaches.

2.4.1 Ensemble

The ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.
At the source code level, we implemented the `VotingClassifier` class, which, given an array of models, provides a result through a voting mechanism among them. The results follow the same line as the previous ones, so we find better results working without normalization.
Anyway, the problems of Impact and Harmless remain, and also the optimized Linear SVC witouth TF-IDF remain the best model for accuracy and loss (0.985739 - 0.002182).

2.4.2 Class weights

The final test involves implementing different weights for the various classes.
This approach aims to assign greater significance to samples that exhibit under-represented labels, potentially addressing issues stemming from class imbalances.
however, all the tests led to worse results than those obtained previously, so we will not go further into the analysis.

2.5 Final comment

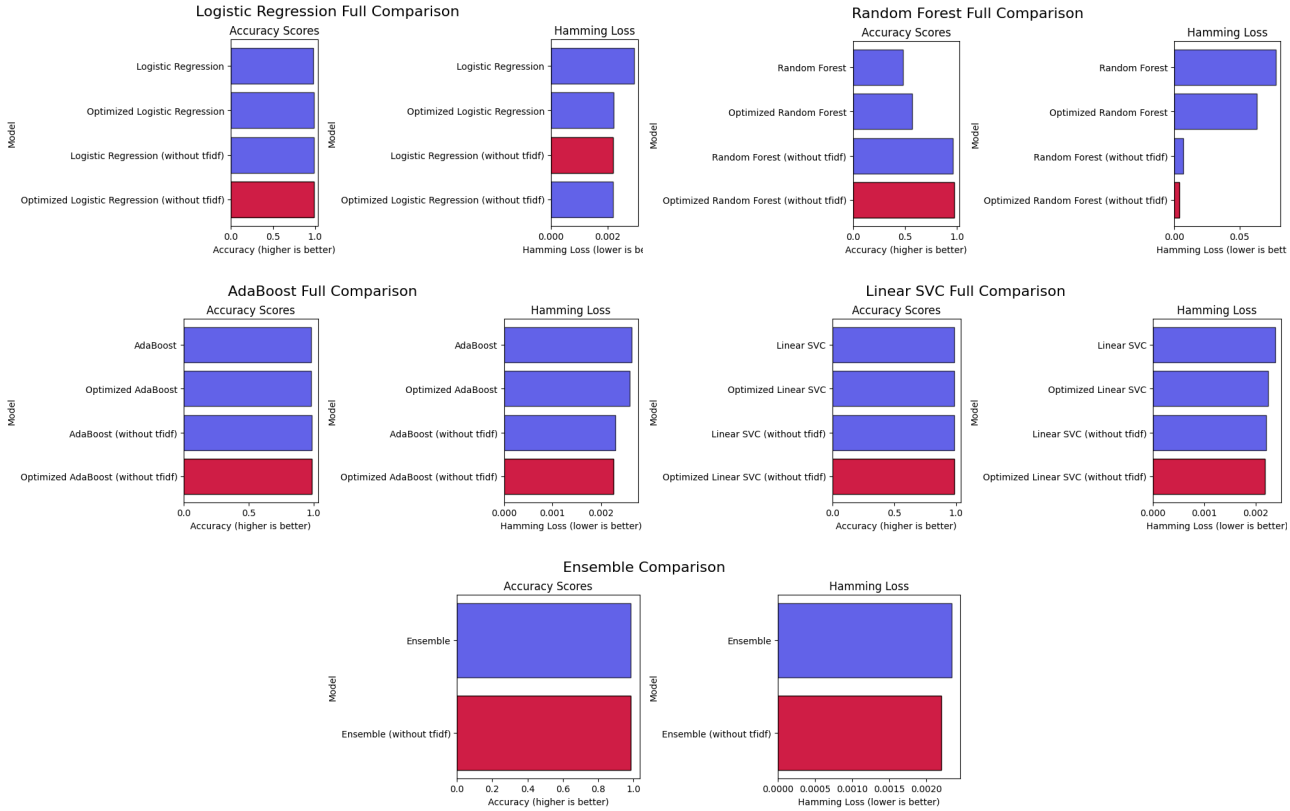


Figura 2.4: Models Evolution

At the end, we can say that the best result was obtained by optimizing the model *Linear SVC* and by fitting it with non normalized data. Despite this, none of the tested methods proved effective to compensate for the Impact and Harmless class imbalance.
In hindsight, it appears that adopting a different optimization strategy or employing oversampling techniques for the underrepresented classes might have been more effective in addressing the imbalance issue. However, it's essential to acknowledge that exploring these possibilities would have necessitated more time than was available for the testing process.

3 Unsupervised learning – clustering

In this chapter we will explore the concept of unsupervised learning and clustering, which is a way to divide the dataset in subsets of homogeneous or similar data. This subsets are the clusters. The clustering could be:

- Hard Clustering (ex. k-Means)
- Soft Clustering (ex. GMM)

3.1 K-Mean algorithm

Given a dataset made of $x^{(i)}$ features, and given the cluster index of the i-th data point $y^{(i)} = \{1, \ldots, k\}$, we define the Hard clustering as the methods that compute predicted cluster indices $\hat{y}$ based solely on $x^{(i)}$.
In hard-clustering, each data point belongs exclusively to one cluster.

K-Means is an hard clustering algorithm, characterized by the usage of the number of clusters **k** parameter. Each cluster is associated with a point called **Centroid** or **Mean**.
K-Means iteratively minimize clustering error.

3.1.1 Determine the number of clusters

After we scaled, the properly filtered dataset, we can proceed to apply the k-mean algorithm. Next we want to analyze the clustering error obtained with the k-means algorithm, and to do so we use two methods:

- Elbow method
- Silhouette method

Elbow method

Starting with the *elbow method*, looking at the graph, we have to take the value just before the "elbow", that's because just after that, the clustering error is going to stabilize. In this case the value **k** of number of clusters is equal to **6**.

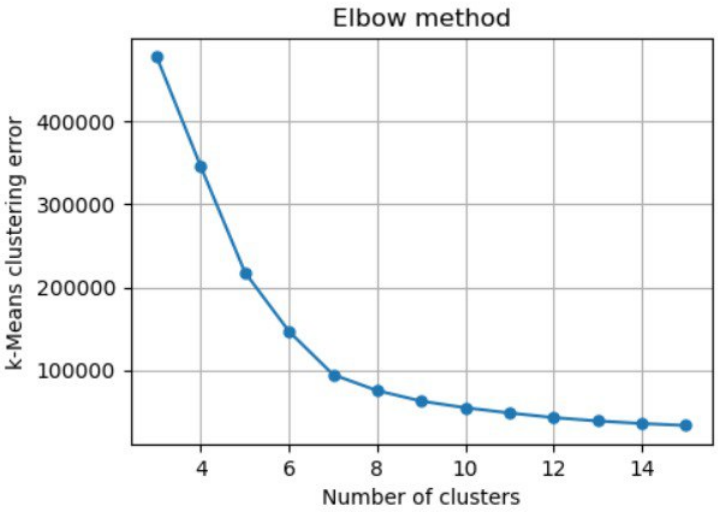


Figura 3.1: The elbow method graph

Silhouette method

Continuing with the *silhouette method*, we use silhouette to measure how similar a data point is to its own cluster and compared to other clusters, so it measures consistency. The silhouette is defined like this:

$$s = \frac{b - a}{\max(a, b)}$$

where a is the mean distance between a sample and all the other points in the same cluster, while b is the mean distance between a sample and all other points in the next nearest cluster. For our exercise it's important to know which is the highest silhouette value, because it corresponds to the best number of clusters k. In our case the highest silhouette value is 0.5796691001521069 and the corresponding best **k** is again **6**. This information is clear also from the following graph.

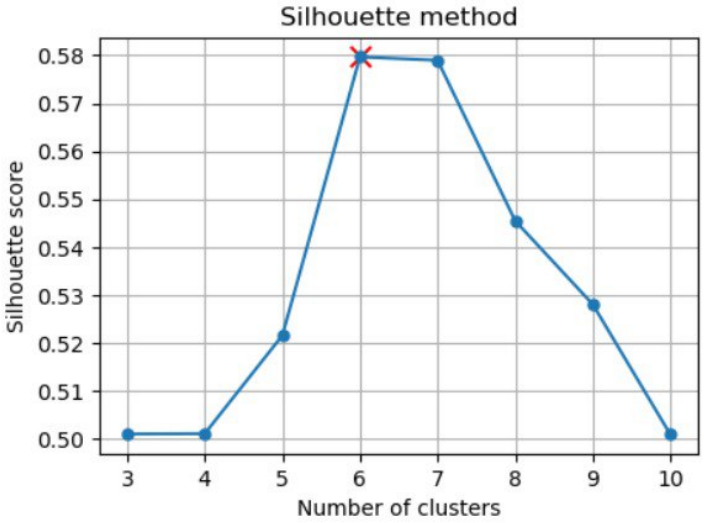


Figura 3.2: The silhouette method graph

3.1.2 Tune other hyper-parameters

The hyper-parameters are values that control the overall behavior of a machine learning algorithm rather than the actual model weights.
K-Means has different hyper-parameters:

- Number of clusters K
- Initialization method
- Random state
- N-init

Tune the number of clusters

We first try to see the differences between two k-Means tuning the k hyper-parameter. We choose two values of k: one is the best k found in the previous section, while the second value is k=3.

We can observe at first sight that both the labels and the centroids assume different values.

```
The clustered labels of best k are:
 [4 4 4 ... 0 0 0]

The centroids are:
 [[ 8.72696487e-01 -1.96523124e-01 -1.50785600e-01 -7.92744995e-01]
 [-8.63654060e-01  1.80988200e-01  2.07882478e-01  1.16530118e+00]
 [-4.88957705e-02  2.07132657e+01  1.55513283e+00  2.66097350e+00]
 [ 8.84326798e-01  1.81749689e-01  2.11539407e-01  1.17754067e+00]
 [-8.63338807e-01 -1.95602331e-01 -1.46398946e-01 -7.92589209e-01]
 [-9.24136534e-01  8.86839734e+01  3.89355420e+02 -2.66244386e+00]]

The clustered labels of k=3 are:
 [0 0 0 ... 0 0 0]

The centroids are:
 [[-6.25211710e-03 -1.96056930e-01 -1.48564653e-01 -7.92666121e-01]
 [ 9.61059250e-03  1.81285208e-01  2.09331668e-01  1.17145708e+00]
 [-4.81331694e-02  2.08234687e+01  2.39675614e+00  2.63972233e+00]]
```

Figura 3.3: Labels and Centroids values tuning number of clusters

This difference is more evident from the following plots:

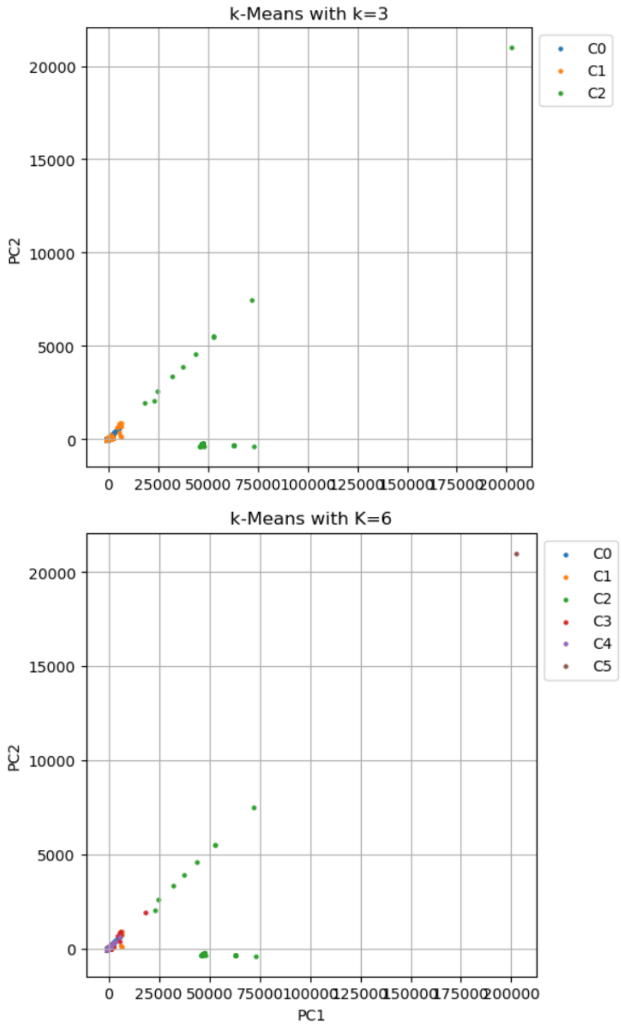


Figura 3.4: Plot of clustering with different values of k

Tune the Initialization method

The next hyper-parameter that we tune is the initialization strategy for the centroid *init* that can be:

- *Random*: centroids will be initialized randomly in the valid data region.
- *k-means++*: centroids will be equally distant from each other, leading to provably better results than random initialization.

We can see, from the below image, that both the labels and the centroids assumes different values.

```
The clustered labels of centroids randomly initialized are:
 [2 2 2 ... 4 4 4]

The centroids randomly initialized are:
 [[-0.85494325  0.40066397  0.23210312  1.18093516]
 [ 0.71685852 -0.19709987 -0.15367137 -0.79381791]
 [-0.21656572 -0.19678757 -0.1523894  -0.7928184 ]
 [-1.21350391 -0.19513357 -0.14404493 -0.7925593 ]
 [ 1.42335603 -0.19501459 -0.1430514  -0.79102782]
 [ 0.88431593  0.1813559   0.20982091  1.17766464]]

The clustered labels of centroids equally distant are:
 [1 1 1 ... 5 5 5]

The centroids equally distant are:
 [[-7.95275512e-01  1.81062389e-01  2.07838653e-01  1.16465195e+00]
 [-7.93633652e-01 -1.95822797e-01 -1.47489203e-01 -7.93023437e-01]
 [-4.69929230e-02  2.06847131e+01  7.04372528e-01  2.70399101e+00]
 [-4.12325822e-01  3.71341153e+01  1.63340460e+02 -2.66244386e+00]
 [ 9.54142397e-01  1.81728002e-01  2.11904763e-01  1.17935322e+00]
 [ 9.44940605e-01 -1.96339774e-01 -1.49863844e-01 -7.92234467e-01]]
```

Figura 3.5: Labels and Centroids values tuning the initialization method

Tune the random state

Tuning the hyper-parameter *random state* may change the result due to a different centroid initialization.
This concept is easy to assume watching the below results:

```
The clustered labels of kmeans with a random state are:
 [0 0 0 ... 2 2 2]

The centroids are:
 [[-0.14710848 -0.1968525  -0.15251685 -0.79335251]
 [ 0.51196231  0.17007663  0.19336753  1.16266377]
 [ 1.08611569 -0.19619933 -0.14915373 -0.79207779]
 [ 1.37241052  0.80323701  0.30739542  1.2474386 ]
 [-1.21350391 -0.19513357 -0.14404493 -0.7925593 ]
 [-0.93465481  0.18222806  0.20866364  1.16489395]]

The clustered labels of kmeans without a random state are:
 [1 1 1 ... 0 0 0]

The centroids are:
 [[ 0.94493352 -0.19629306 -0.14965614 -0.7922641 ]
 [-0.79363365 -0.1958228  -0.1474892  -0.79302344]
 [ 1.08784824  0.18149963  0.21094138  1.18237243]
 [-0.15279961  0.1815176   0.20869055  1.16397578]
 [-0.04813317 20.82346865  2.39675614  2.63972233]
 [-1.21387042  0.18066137  0.2074747   1.16554392]]
```

Figura 3.6: Labels and Centroids values tuning the random state

Tune the n-init

Last but not least we can tune *n init* which is the number of times the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n init consecutive runs in terms of clustering error.
Once again the differences are visible through this result:

```
The clustered labels of kmeans with a n_init = 1 are:
 [0 0 0 ... 2 2 2]

The centroids are:
 [[-0.14710848 -0.1968525  -0.15251685 -0.79335251]
 [ 0.51196231  0.17007663  0.19336753  1.16266377]
 [ 1.08611569 -0.19619933 -0.14915373 -0.79207779]
 [ 1.37241052  0.80323701  0.30739542  1.2474386 ]
 [-1.21350391 -0.19513357 -0.14404493 -0.7925593 ]
 [-0.93465481  0.18222806  0.20866364  1.16489395]]

The clustered labels of kmeans without a n_init = 5 are:
 [2 2 2 ... 3 3 3]

The centroids are:
 [[ 0.71685852 -0.19709987 -0.15367137 -0.79381791]
 [ 0.88431593  0.1813559   0.20982091  1.17766464]
 [-0.21656572 -0.19678757 -0.1523894  -0.7928184 ]
 [ 1.42335603 -0.19501459 -0.1430514  -0.79102782]
 [-0.85494325  0.40066397  0.23210312  1.18093516]
 [-1.21350391 -0.19513357 -0.14404493 -0.7925593 ]]
```

Figura 3.7: Labels and Centroids values tuning n-init

3.1.3 t-SNE Visualization

t-SNE stands for t-distributed stochastic neighbor embedding, it's a nonlinear dimensionality reduction technique for embedding high-dimensional data for visualization. This embedding works like this, similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

While t-SNE doesn't directly perform clustering, it can be used as part of a broader data analysis process to explore the data structure and identify any clusters or patterns.
We apply this technique to our dataset and the following is the result:

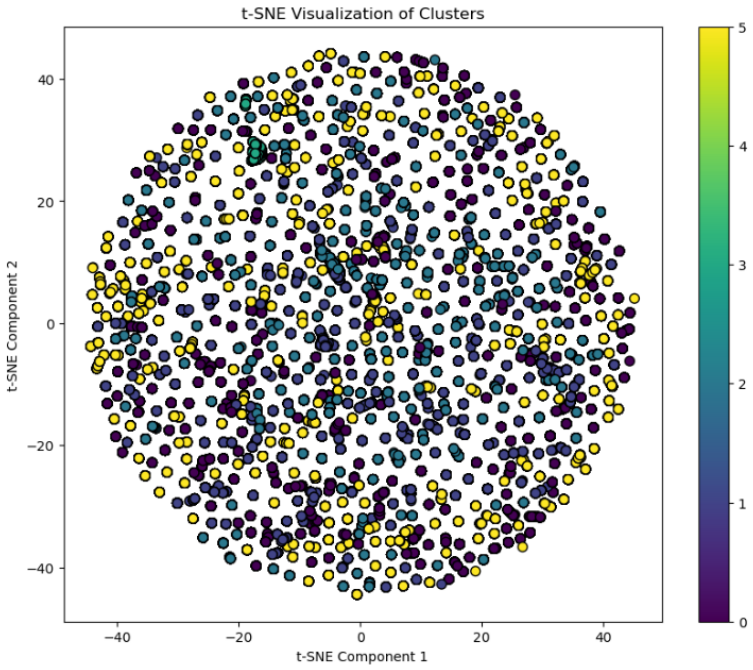


Figura 3.8: k-Means and t-SNE

We can observe how t-SNE is able to recover well-separated clusters, and it approximates a simple form of *spectral clustering*. This last one it's another clustering technique based on the euclidean distance between the dataset's points.

3.1.4 Cluster analysis

In this section we are able to analyze the cluster just obtained, using k-Mean with k=7. In particular we examine the most frequent words in each cluster in order to understand which are the most representative ones, and this is possible through **word cloud**.

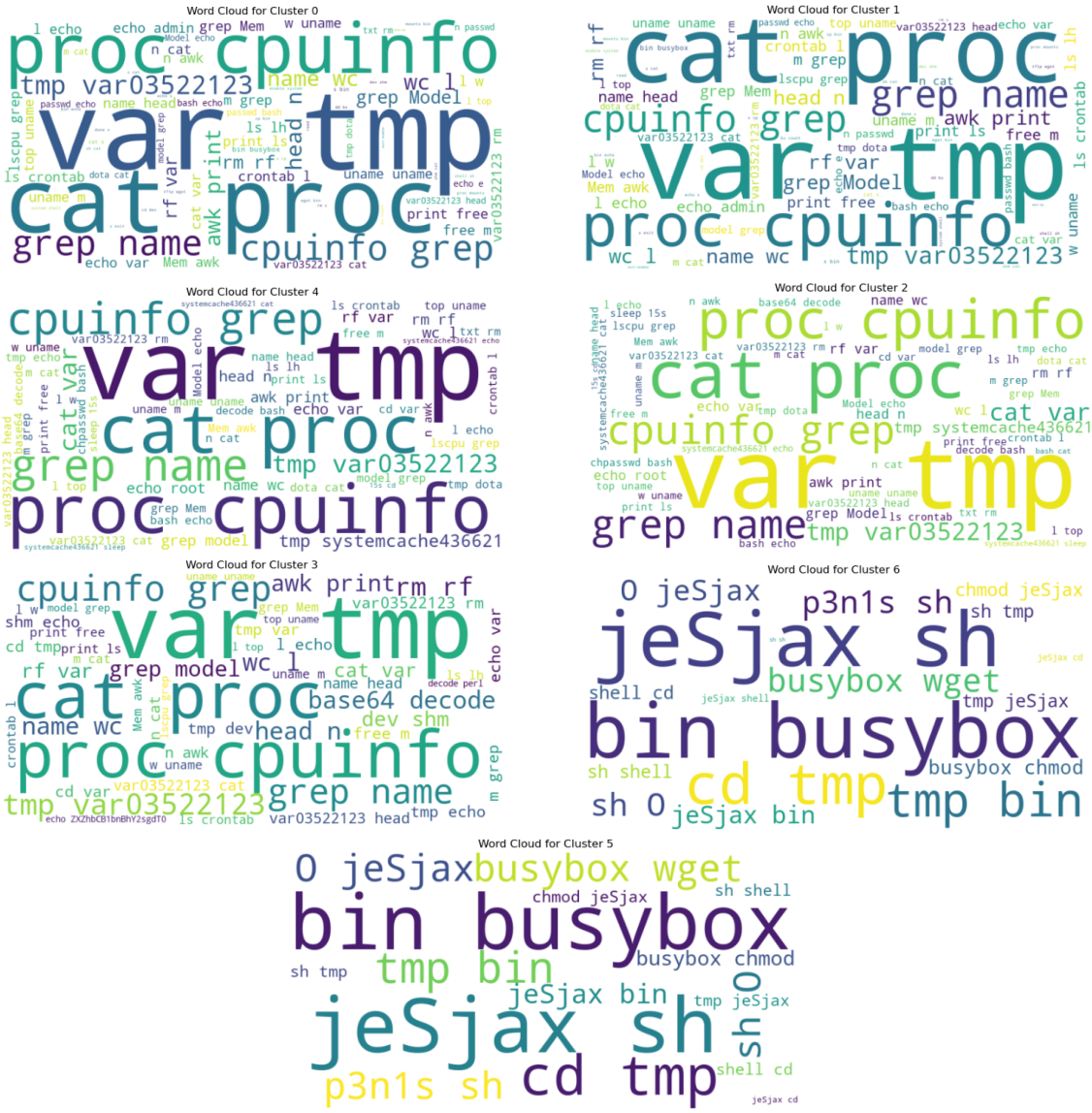


Figura 3.9: Most frequent words for each cluster

Among the results we can say for sure that the most present words in each cluster are *var tmp*, *cd tmp*, *cat proc*, *bin busybox*, *proc cpuinfo* and *jeSjax sh*.
In detail we notice that clusters 0 through 4 have similar most used words, such as **var tmp**, **cat proc**, **proc cpuinfo**. While the cluster 5 and 6 present as similar most used words, words like **bin busybox** and **jeSjax.sh**.

So we can assert that clusters 0 through 4 contains similar command lines and the same consideration for the clusters 5 and 6.

3.1.5 Intent Division and Homogeneity

Trying to expand the analysis made in the previous section, we can analyze how strong is the intent division and how much homogeneous are the data in the clusters.

Cluster 0:

	full_session	Set_Fingerprint
0	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
1	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
2	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
3	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
4	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
...	...	...
232698	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
232699	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
232700	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
232701	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
232702	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]

[70559 rows x 2 columns]

Cluster 1:

	full_session	Set_Fingerprint
127	solokey ; nan ; sh ; nan ; shell ; nan ; enabl...	[Discovery, Execution]
568	sh ; shell ; help ; busybox ; cd /tmp || cd /r...	[Defense Evasion, Discovery, Execution]
623	sh ; shell ; cd /tmp || cd /var/run || cd /mnt...	[Defense Evasion, Execution]
745	sh ; shell ; help ; busybox ; cd /tmp || cd /d...	[Defense Evasion, Discovery, Execution]
754	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
...	...	...
232075	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Execution, Persistence]
232473	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Persistence]
232498	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Persistence]
232562	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Persistence]
232682	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Execution, Persistence]

[46624 rows x 2 columns]

Cluster 2:

	full_session	Set_Fingerprint
13	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
14	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
15	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
16	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
17	enable ; system ; shell ; sh ; cat /proc/mount...	[Defense Evasion, Discovery]
...	...	...
233030	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
233031	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
233032	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
233033	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]
233034	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Persistence]

[68804 rows x 2 columns]

Cluster 3:

	full_session	Set_Fingerprint
29	sh ; shell ; cd /tmp || cd /var/run || cd /mnt...	[Defense Evasion, Execution]
766	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
788	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
791	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
819	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
...	...	...
232878	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Persistence]
232891	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Execution, Persistence]
232929	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Execution, Persistence]
232934	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Execution, Persistence]
232960	cat /proc/cpuinfo | grep name | wc -l ; echo -...	[Discovery, Harmless, Persistence]

[46548 rows x 2 columns]

Cluster 4:

	full_session	Set_Fingerprint
6259	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Execution, Persistence]
6264	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Persistence, Defense Evasion, Harm...
6308	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Persistence, Defense Evasion, Harm...
6323	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Persistence, Defense Evasion, Harm...
6346	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Discovery, Persistence, Defense Evasion, Harm...
...	...	...
125753	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Defense Evasion, Discovery, Execution, Persis...
125942	cd /var/tmp ; echo "IyEvYmluL2Jhc2gKY2QgL3RtcA...	[Defense Evasion, Discovery, Execution, Persis...
194101	w ; ls ; nproc ; sudo su ; su ; admin ; ^C ; w...	[Discovery, Persistence, Defense Evasion, Harm...
227682	echo "ZXZhbCB1bnBhY2sgdT0+cXtfIkZVWSgiMVA8Rl1D...	[Execution]
229994	echo "ZXZhbCB1bnBhY2sgdT0+cXtfIkZVWSgiMVA8Rl1D...	[Execution]

[499 rows x 2 columns]

Cluster 5:

	full_session	Set_Fingerprint
16880	sh ; shell ; cd /tmp ; /bin/busybox wget http:...	[Execution]

Figura 3.10: Cluster visualization

After we use k-Means with k number of clusters equals to 6, we obtain the above clusters. Even if we don't visualize all the entries, except of cluster 5, is clear that the clusters are not so much homogeneous, since there are at least two different intents in each cluster. The only cluster that results to be homogeneous is, of course, the cluster 5 which contains only one data entry.
Speaking of intent division is clear that is not so strong, for example if we compare cluster 0 and cluster 2 they look very similar at first sight.

3.1.6 Find clusters of similar attacks

Because of this not so strong intent division, we can try to find clusters of similar attacks.

First of all we limit the search to the clusters 1, 3, 5, then we try to fine grained categorize the attacks. To do so we assign a specific regex to a specific attack, these are:

- Bin Download+Run
- IoT Busybox
- Passwd change
- DOTA tgz
- DOTA perl

After that we search inside each cluster what is the most common attack and at last we compare the results.

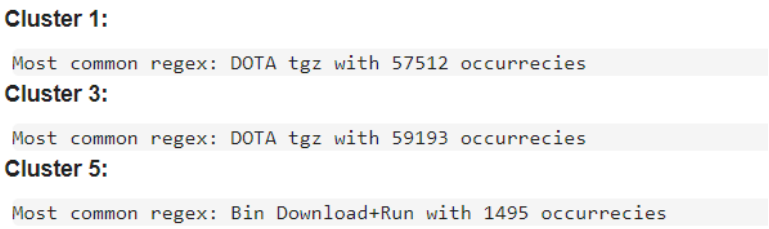


Figura 3.11: Selected clusters with their most common attack

Observing the results we can say that cluster 1 and cluster 3 are clusters of a similar attack: **DOTA tgz**.

3.2 Gaussian Mixture Method algorithm

Given a dataset made of $x^{(i)}$ features, and given the i-th data point characterized by k label values $y^{(i)} = \{y_1^{(i)}, \ldots, y_k^{(i)}\}$, we interpret the $y_c^{(i)}$ as **probability** of a i-th data point belongs to the cluster c, with $0 < y_c^{(i)} < 1$ and $\sum_{c=1}^{k} y_c^{(i)}$.
The Soft clustering is the method that allows to obtain the *predicted degrees of belonging* $\hat{y}^{(1)}, \ldots, \hat{y}^{(m)}$ from the feature vectors $x^{(1)}, \ldots, x^{(m)}$.
A peculiarity of Soft clustering is that that data points have probabilities of belonging to each cluster, providing more nuanced information about cluster membership, so it depends crucially on initialization mean.

GMM, the short for *Gaussian mixture model*, is a soft clustering algorithm characterized by this trait: each cluster produces data based upon random draws from a multi-dimensional Gaussian distribution and also each Gaussian cluster has its own mean and standard deviation. The soft-clustering error is iteratively optimized to get the best values of $y_c^{(i)}$.

3.2.1 Determine the number of clusters

As with k-means, after we scaled, the properly filtered dataset, we can proceed to apply the GMM algorithm.
We again want to analyze the clustering error obtained with the GMM algorithm, and to do so, this time, we use only the *Silhouette method*.

Silhouette method

Also with GMM our goal is to know which is the highest silhouette value, because it corresponds to the best number of clusters k. In our case the highest silhouette value is 0.5056172595992869 and the corresponding best **k** is **5**. This information is clear also from the following graph.

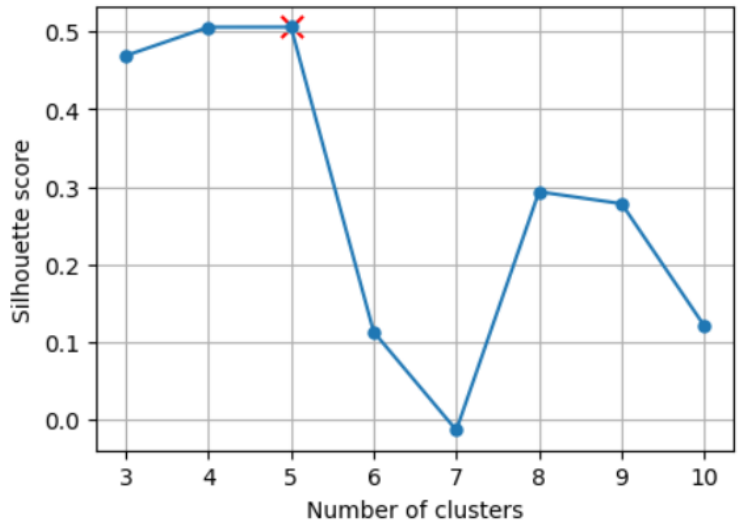


Figura 3.12: The silhouette method graph

3.2.2 Tune other hyper-parameters

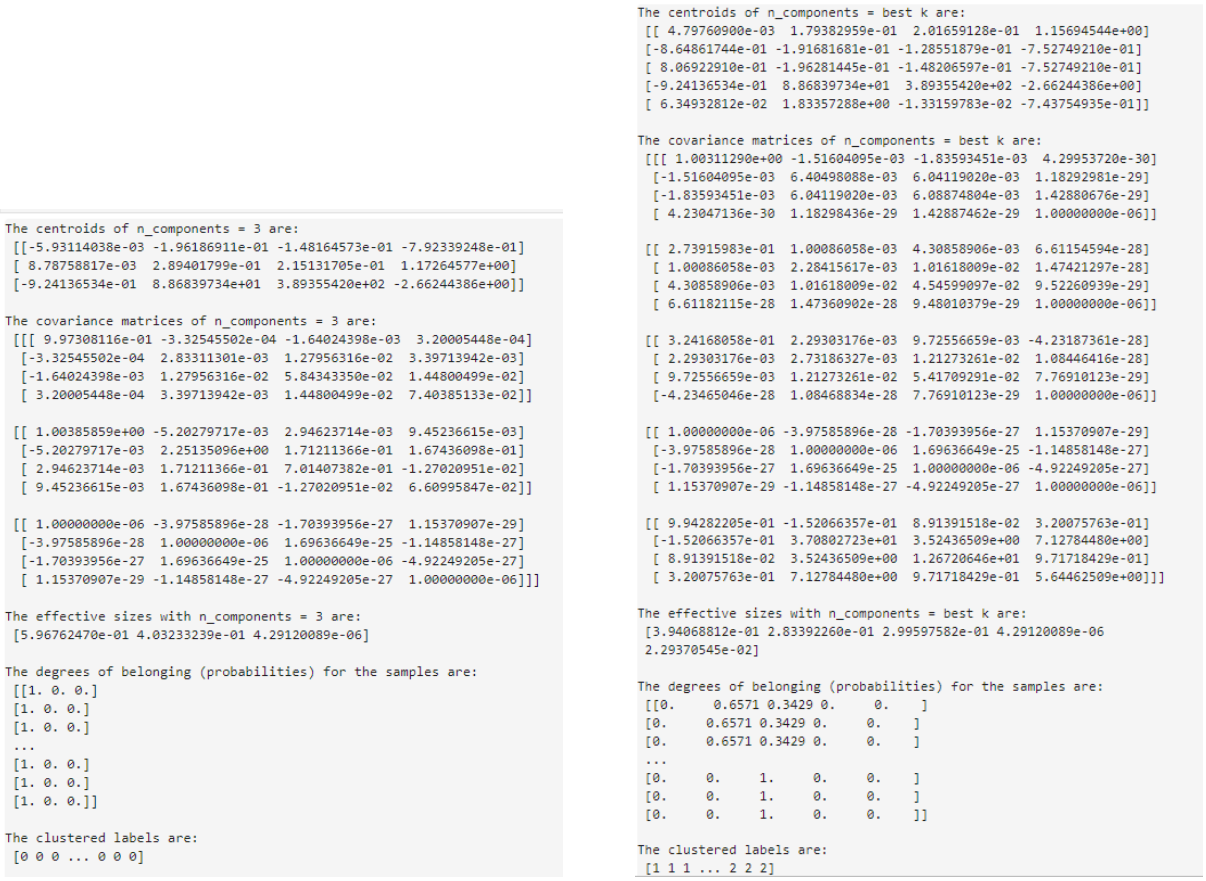
GMM has hyper-parameter likewise, these are:

- Number of components
- Init params

Tune the number of components

We first try to see the differences between two GMM tuning the *n-components* hyper-parameter. We choose two values of n-components: one is the best k found in the previous section, while the second value is n-components=3.

We can observe at first sight that labels, centroids, covariance matrix, effective sizes and of course the degrees of belonging assume different values.



```
The centroids of n_components = 3 are:
 [[-5.93114038e-03 -1.96186911e-01 -1.48164573e-01 -7.92339248e-01]
 [ 8.78758817e-03  2.89401799e-01  2.15131705e-01  1.17264577e+00]
 [-9.24136534e-01  8.86839734e+01  3.89355420e+02 -2.66244386e+00]]

The covariance matrices of n_components = 3 are:
 [[[ 9.97308116e-01 -3.32545502e-04 -1.64024398e-03  3.20005448e-04]
  [-3.32545502e-04  2.83311301e-03  1.27956316e-02  3.39713942e-03]
  [-1.64024398e-03  1.27956316e-02  5.84343350e-02  1.44800499e-02]
  [ 3.20005448e-04  3.39713942e-03  1.44800499e-02  7.40385133e-02]]

 [[ 1.00385859e+00 -5.20279717e-03  2.94623714e-03  9.45236615e-03]
  [-5.20279717e-03  2.25135096e+00  1.71211366e-01  1.67436098e-01]
  [ 2.94623714e-03  1.71211366e-01  7.01407382e-01 -1.27020951e-02]
  [ 9.45236615e-03  1.67436098e-01 -1.27020951e-02  6.60995847e-02]]

 [[ 1.00000000e-06 -3.97585896e-28 -1.70393956e-27  1.15370907e-29]
  [-3.97585896e-28  1.00000000e-06  1.69636649e-25 -1.14858148e-27]
  [-1.70393956e-27  1.69636649e-25  1.00000000e-06 -4.92249205e-27]
  [ 1.15370907e-29 -1.14858148e-27 -4.92249205e-27  1.00000000e-06]]]

The effective sizes with n_components = 3 are:
 [5.96762470e-01 4.03233239e-01 4.29120089e-06]

The degrees of belonging (probabilities) for the samples are:
 [[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 ...
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

The clustered labels are:
 [0 0 0 ... 0 0 0]
```

```
The centroids of n_components = best k are:
 [[ 4.79760900e-03  1.79382959e-01  2.01659128e-01  1.15694544e+00]
 [-8.64861744e-01 -1.91681681e-01 -1.28551879e-01 -7.52749210e-01]
 [ 8.06922910e-01 -1.96281445e-01 -1.48206597e-01 -7.52749210e-01]
 [-9.24136534e-01  8.86839734e+01  3.89355420e+02 -2.66244386e+00]
 [ 6.34932812e-02  1.83357288e+00 -1.33159783e-02 -7.43754935e-01]]

The covariance matrices of n_components = best k are:
 [[[ 1.00311290e+00 -1.51604095e-03 -1.83593451e-03  4.29953720e-30]
  [-1.51604095e-03  6.40498088e-03  6.04119020e-03  1.18292981e-29]
  [-1.83593451e-03  6.04119020e-03  6.08874804e-03  1.42880676e-29]
  [ 4.23047136e-30  1.18298436e-29  1.42887462e-29  1.00000000e-06]]

 [[ 2.73915983e-01  1.00086058e-03  4.30858906e-03  6.61154594e-28]
  [ 1.00086058e-03  2.28415617e-03  1.01618009e-02  1.47421297e-28]
  [ 4.30858906e-03  1.01618009e-02  4.54599097e-02  9.52260939e-29]
  [ 6.61182115e-28  1.47360902e-28  9.48010379e-29  1.00000000e-06]]

 [[ 3.24168058e-01  2.29303176e-03  9.72556659e-03 -4.23187361e-28]
  [ 2.29303176e-03  2.73186327e-03  1.21273261e-02  1.08446416e-28]
  [ 9.72556659e-03  1.21273261e-02  5.41709291e-02  7.76910123e-29]
  [-4.23465046e-28  1.08468834e-28  7.76910123e-29  1.00000000e-06]]

 [[ 1.00000000e-06 -3.97585896e-28 -1.70393956e-27  1.15370907e-29]
  [-3.97585896e-28  1.00000000e-06  1.69636649e-25 -1.14858148e-27]
  [-1.70393956e-27  1.69636649e-25  1.00000000e-06 -4.92249205e-27]
  [ 1.15370907e-29 -1.14858148e-27 -4.92249205e-27  1.00000000e-06]]

 [[ 9.94282205e-01 -1.52066357e-01  8.91391518e-02  3.20075763e-01]
  [-1.52066357e-01  3.70802723e+01  3.52436509e+00  7.12784480e+00]
  [ 8.91391518e-02  3.52436509e+00  1.26720646e+01  9.71718429e-01]
  [ 3.20075763e-01  7.12784480e+00  9.71718429e-01  5.64462509e+00]]]

The effective sizes with n_components = best k are:
 [3.94068812e-01 2.83392260e-01 2.99597582e-01 4.29120089e-06
 2.29370545e-02]

The degrees of belonging (probabilities) for the samples are:
 [[0.     0.6571 0.3429 0.     0.    ]
 [0.     0.6571 0.3429 0.     0.    ]
 [0.     0.6571 0.3429 0.     0.    ]
 ...
 [0.     0.     1.     0.     0.    ]
 [0.     0.     1.     0.     0.    ]
 [0.     0.     1.     0.     0.    ]]

The clustered labels are:
 [1 1 1 ... 2 2 2]
```

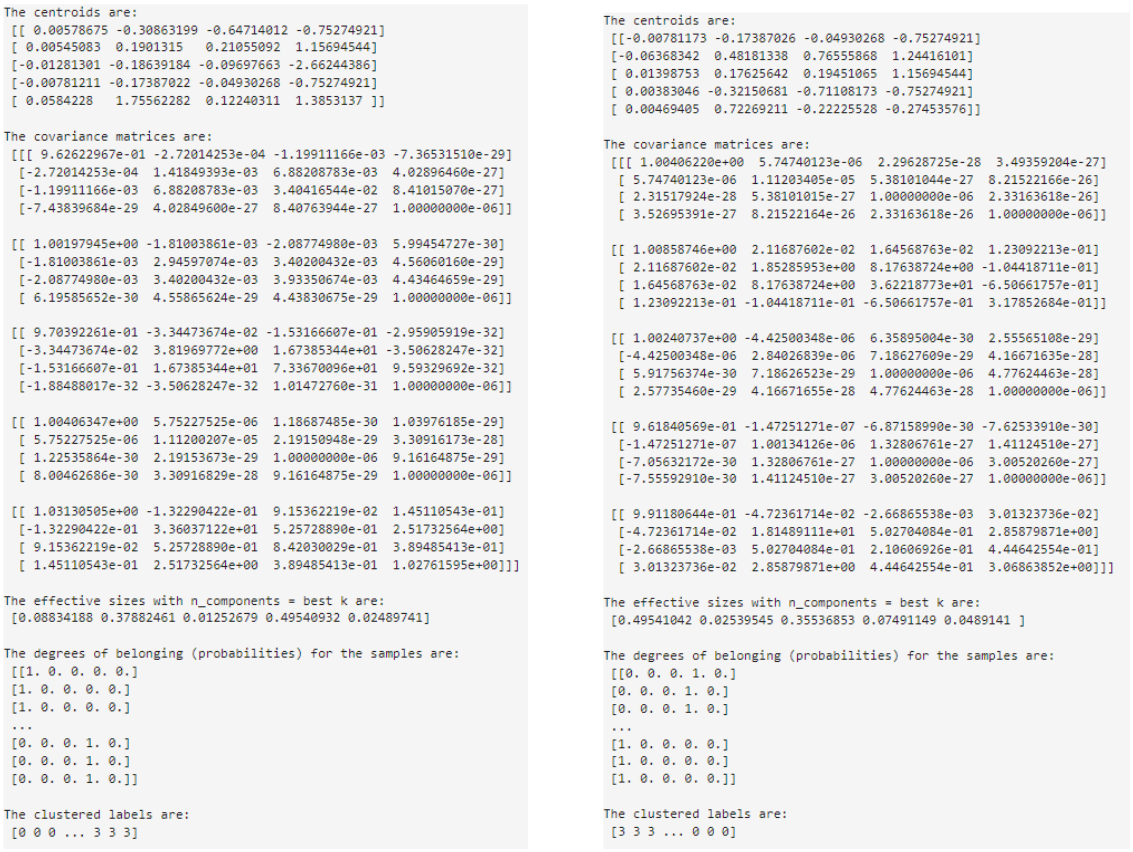
Figura 3.13: Labels, Centroids, Covariance matrix, Effective sizes and Degrees of belonging values tuning number of components

Tune the Initialization parameters

The next hyper-parameter that we tune is the initialization parameters *init-params* that can be:

- *kmeans*: responsibilities are initialized using k-Means.
- *random*: responsibilities are initialized randomly

We can observe that labels, centroids, covariance matrix, effective sizes and of course the degrees of belonging assume different values.



```
The centroids are:
 [[ 0.00578675 -0.30863199 -0.64714012 -0.75274921]
 [ 0.00545083  0.1901315   0.21055092  1.15694544]
 [-0.01281301 -0.18639184 -0.09697663 -2.66244386]
 [-0.00781211 -0.17387022 -0.04930268 -0.75274921]
 [ 0.0584228   1.75562282  0.12240311  1.3853137 ]]

The covariance matrices are:
 [[[ 9.62622967e-01 -2.72014253e-04 -1.19911166e-03 -7.36531510e-29]
  [-2.72014253e-04  1.41849393e-03  6.88208783e-03  4.02896460e-27]
  [-1.19911166e-03  6.88208783e-03  3.40416544e-02  8.41015070e-27]
  [-7.43839684e-29  4.02849600e-27  8.40763944e-27  1.00000000e-06]]

 [[ 1.00197945e+00 -1.81003861e-03 -2.08774980e-03  5.99454727e-30]
  [-1.81003861e-03  2.94597074e-03  3.40200432e-03  4.56060160e-29]
  [-2.08774980e-03  3.40200432e-03  3.93350674e-03  4.43464659e-29]
  [ 6.19585652e-30  4.55865624e-29  4.43830675e-29  1.00000000e-06]]

 [[ 9.70392261e-01 -3.34473674e-02 -1.53166607e-01 -2.95905919e-32]
  [-3.34473674e-02  3.81969772e+00  1.67385344e+01 -3.50628247e-32]
  [-1.53166607e-01  1.67385344e+01  7.33670096e+01  9.59329692e-32]
  [-1.88488017e-32 -3.50628247e-32  1.01472760e-31  1.00000000e-06]]

 [[ 1.00406347e+00  5.75227525e-06  1.18687485e-30  1.03976185e-29]
  [ 5.75227525e-06  1.11200207e-05  2.19150948e-29  3.30916173e-28]
  [ 1.22535864e-30  2.19153673e-29  1.00000000e-06  9.16164875e-29]
  [ 8.00462686e-30  3.30916829e-28  9.16164875e-29  1.00000000e-06]]

 [[ 1.03130505e+00 -1.32290422e-01  9.15362219e-02  1.45110543e-01]
  [-1.32290422e-01  3.36037122e+01  5.25728890e-01  2.51732564e+00]
  [ 9.15362219e-02  5.25728890e-01  8.42030029e-01  3.89485413e-01]
  [ 1.45110543e-01  2.51732564e+00  3.89485413e-01  1.02761595e+00]]]

The effective sizes with n_components = best k are:
 [0.08834188 0.37882461 0.01252679 0.49540932 0.02489741]

The degrees of belonging (probabilities) for the samples are:
 [[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 ...
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]]

The clustered labels are:
 [0 0 0 ... 3 3 3]
```

```
The centroids are:
 [[-0.00781173 -0.17387026 -0.04930268 -0.75274921]
 [-0.06368342  0.48181338  0.76555868  1.24416101]
 [ 0.01398753  0.17625642  0.19451065  1.15694544]
 [ 0.00383046 -0.32150681 -0.71108173 -0.75274921]
 [ 0.00469405  0.72269211 -0.22225528 -0.27453576]]

The covariance matrices are:
 [[[ 1.00406220e+00  5.74740123e-06  2.29628725e-28  3.49359204e-27]
  [ 5.74740123e-06  1.11203405e-05  5.38101044e-27  8.21522166e-26]
  [ 2.31517924e-28  5.38101015e-27  1.00000000e-06  2.33163618e-26]
  [ 3.52695391e-27  8.21522164e-26  2.33163618e-26  1.00000000e-06]]

 [[ 1.00858746e+00  2.11687602e-02  1.64568763e-02  1.23092213e-01]
  [ 2.11687602e-02  1.85285953e+00  8.17638724e+00 -1.04418711e-01]
  [ 1.64568763e-02  8.17638724e+00  3.62218773e+01 -6.50661757e-01]
  [ 1.23092213e-01 -1.04418711e-01 -6.50661757e-01  3.17852684e-01]]

 [[ 1.00240737e+00 -4.42500348e-06  6.35895004e-30  2.55565108e-29]
  [-4.42500348e-06  2.84026839e-06  7.18627609e-29  4.16671635e-28]
  [ 5.91756374e-30  7.18626523e-29  1.00000000e-06  4.77624463e-28]
  [ 2.57735460e-29  4.16671655e-28  4.77624463e-28  1.00000000e-06]]

 [[ 9.61840569e-01 -1.47251271e-07 -6.87158990e-30 -7.62533910e-30]
  [-1.47251271e-07  1.00134126e-06  1.32806761e-27  1.41124510e-27]
  [-7.05632172e-30  1.32806761e-27  1.00000000e-06  3.00520260e-27]
  [-7.55592910e-30  1.41124510e-27  3.00520260e-27  1.00000000e-06]]

 [[ 9.91180644e-01 -4.72361714e-02 -2.66865538e-03  3.01323736e-02]
  [-4.72361714e-02  1.81489111e+01  5.02704084e-01  2.85879871e+00]
  [-2.66865538e-03  5.02704084e-01  2.10606926e-01  4.44642554e-01]
  [ 3.01323736e-02  2.85879871e+00  4.44642554e-01  3.06863852e+00]]]

The effective sizes with n_components = best k are:
 [0.49541042 0.02539545 0.35536853 0.07491149 0.0489141 ]

The degrees of belonging (probabilities) for the samples are:
 [[0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0.]
 ...
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]]

The clustered labels are:
 [3 3 3 ... 0 0 0]
```

Figura 3.14: Labels, Centroids, Covariance matrix, Effective sizes and Degrees of belonging values tuning initialization parameters (respectively *kmeans* and *random*)

3.2.3 t-SNE Visualization

We apply again this technique to our dataset, but this time after we use GMM with n-components = best k = 5, and the following is the result:

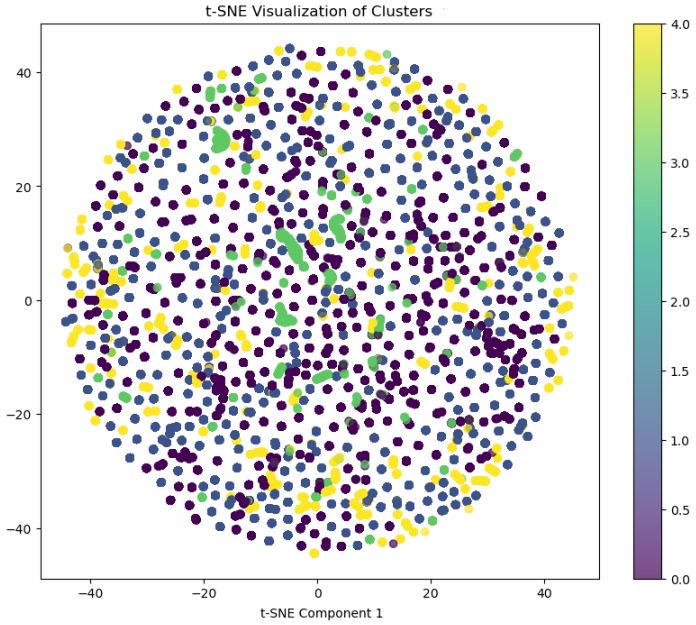


Figura 3.15: GMM and t-SNE

We can observe how t-SNE, also in this case, is able to recover well-separated clusters, and it approximates a simple form of *spectral clustering*.

3.2.4 Cluster analysis

Also after GMM algorithm we are able to analyze the cluster just obtained. In the specific we examine the most frequent words in each cluster, using GMM with n-components=8, through **word cloud**.

Figura 3.16: Most frequent words for each cluster

Again we can say for sure that the most present words in each cluster are *var tmp*, *cd tmp*, *cat proc*, *bin busybox*, *proc cpuinfo* and *jeSjax sh*.

In detail we notice that all clusters except the cluster 4 have similar most used words, such as **var tmp**, **cat proc**, **proc cpuinfo**. While only cluster 4 has as most used words, words like **bin busybox**, **jeSjax.sh**, **tmp bin**, **cd tmp**.

This time we can assert that all cluster, except cluster 4, contains similar command lines.

3.2.5 Intent Division and Homogeneity

Trying, like we have done for k-mean, to expand the analysis made in the previous section, we can analyze how strong is the intent division and how much homogeneous are the data in the clusters.

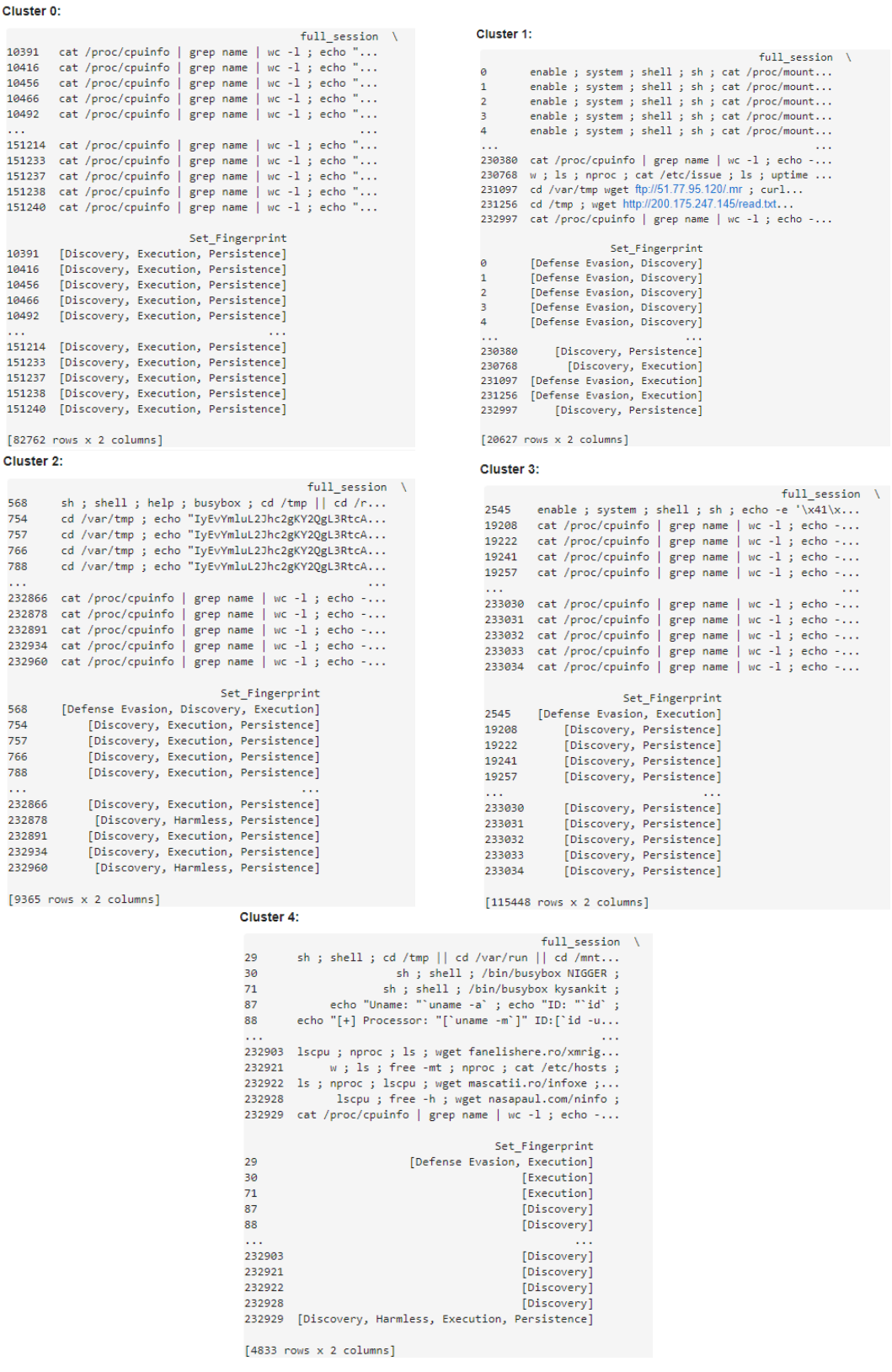


Figura 3.17: Cluster visualization

After we use GMM with n-components number equals to 5, we obtain the above clusters. Even if, also this time, we don't visualize all the entries, we can make some assumptions. With GMM the majority of clusters seems to be very homogeneous, like for example cluster 0 or cluster 3, but on the other hand there are cluster, like cluster 4, that appear very inhomogeneous. Intent division is clear that is again not strong, for example if we compare cluster 0 and cluster 3 they look the same at first sight, except for one row.

3.2.6 Find clusters of similar attacks

Lastly, we try to find clusters of similar attacks.

First of all, this time, we limit the search to the clusters 1, 2, 4, then, using the same categories of attack as before, we search inside each cluster what is the most common attack and at last we compare the results.

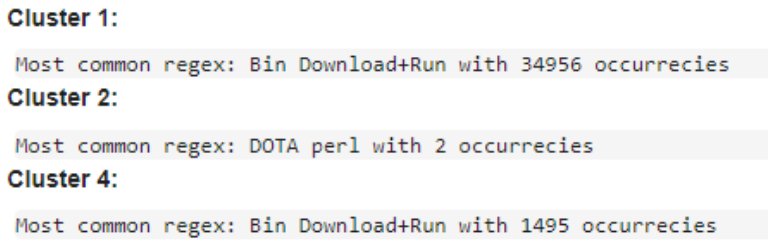


Figura 3.18: Selected clusters with their most common attack

Observing the results we can say that cluster 1 and cluster 4 are clusters of a similar attack: **Bin Download+Run**.

3.3 Final comment

To conclude, after all this analysis it appears that **k-means** algorithm is a better solution for our dataset, in terms of homogeneity and in terms of intent division.

4 Supervised learning - BERT

4.1 BERT

BERT, Bidirectional Encoder Representations from Transformers, is a language model based on the transformer architecture. It represents the state of the art in the field of Natural Language Processing. It has two functionalities:

- The ability to use a pre-trained model. As stated in the Hugging Face documentation: "Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch."
- Transforming all sentences into vectors of integers to leverage more data.

4.1.1 Tokenizer

To transform the sentences into vectors, we have to use the BertTokenizer, specifically the 'bert-base-uncased' pretrained model. Every pre-trained model has its own tokenizer and model. The tokenizer we are using takes the sentence to tokenize as input, with some variables and one important hyperparameter, which we will discuss later (the max_length number).

4.2 Pretrained Model

After that, we can split our data and load the pre-trained model 'bert-base-uncased' with the expected label as output and the problem type defined as 'multi-label-classification.' Changing the problem type means that Bert will use a different loss function; in this case, the loss function will be BCEWithLogitsLoss, namely Binary Cross Entropy with Logits Loss. The model is a 12-layer architecture with dimensions 768x768 and an output layer on top that has 7 result labels (editable in the declaration of the model). The model is as follows:

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False))
    (encoder): BertEncoder(
```

```
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False))
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)))
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation())
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False))))
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()))
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=7, bias=True)
)
```

After the logits, results obtained from the model , we use a sigmoid function to scale the results between 0 and 1. It brings all the values over 0.5 to 1 and all the values under 0.5 to 0. Then we can compare the real label and the predicted one to have our results.

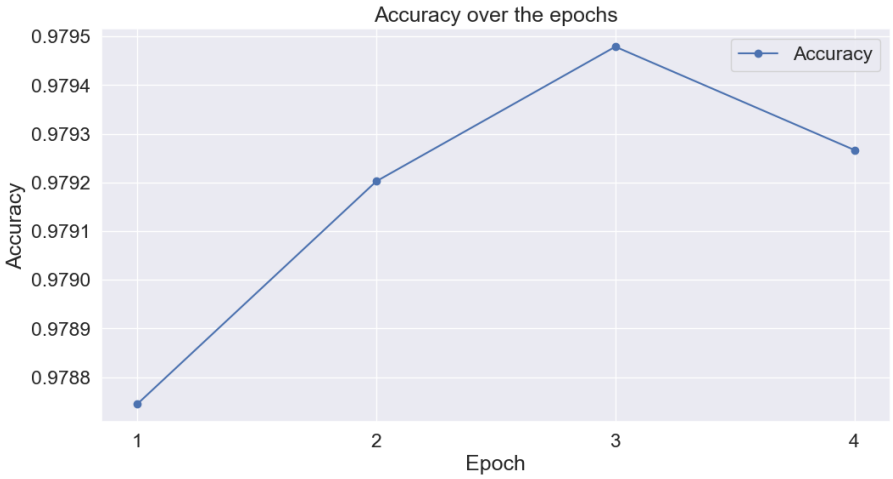
4.2.1 Training and Validation

After that, we need to set up the optimizer and the scheduler with standard values. However, the learning rate and the adam_epsilon can be modified to have a different impact on the backward pass. The effective training can now begin and it follows a standard procedure: the loss is accumulated to compute the average loss over the epochs. The number of epochs is set to 4 because BERT authors recommend between 2 and 4, and every time in the code the fourth epoch has a validation loss higher than the previous. In addition, if we want to add more epochs, we must consider the amount of time needed, as it is the biggest challenge. Without a GPU, the entire training can take 3 to 4 hours, and with a GPU, approximately 50 minutes to one hour, but with the GPU at 100% load.

After a while, the results are shown in Figure 4.1. Analyzing the scores, we are confident

that the best model, with the lowest result in validation loss and the highest in accuracy, is the one from the third epoch. Note that automatically, the model with the lowest validation loss will be saved in a different path and taken to be used to evaluate the test set.

(a) Training and Validation loss



(b) Accuracy over the epochs

Figura 4.1: Results of the Training to choose the model for the Test set

4.2.2 Test

After the best model loaded in memory we can put it in evaluation mode and give it the test set. The accuracy score is: 98.05% and the other results are in Figure 4.2.

	precision	recall	f1-score	support
Defense Evasion	0.99	0.97	0.98	9544
Discovery	1.00	1.00	1.00	116042
Execution	0.99	0.99	0.99	46404
Harmless	0.78	0.19	0.30	1126
Persistence	1.00	1.00	1.00	105595
Other	1.00	0.95	0.98	171
Impact	0.00	0.00	0.00	16
micro avg	1.00	0.99	1.00	278898
macro avg	0.82	0.73	0.75	278898
weighted avg	1.00	0.99	0.99	278898
samples avg	1.00	0.99	1.00	278898

Figura 4.2: Result for the test set

'Impact' is the label with the lowest score, indicating 0% for every metric. This implies that the model cannot recognize sentences with that label at all. 'Harmless' is another problematic label, as it has a relatively high precision (78%) but a low level of recognition, as evident from the recall and F1-score. The other five labels have high scores, with the lowest being 95% for the recall of the label 'Other'. More specifically, in Figure 4.3, we can observe the misclassifications as false positives and true negatives. [1].

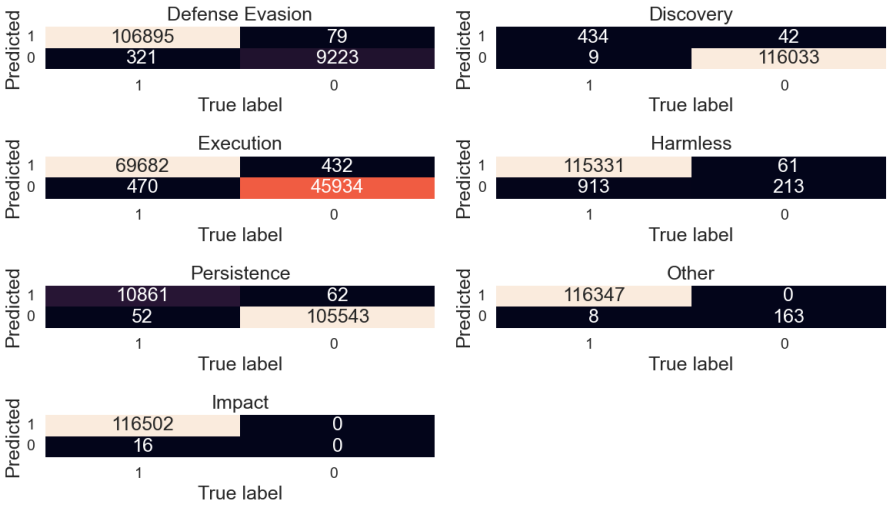


Figura 4.3: Result by label

4.3 Dense Layer

The next steps involve adding a dense layer and an output layer on top of Bert, using them instead of those implemented in the pre-trained model. Firstly, we add a dropout of 0.3, then we create two linear layers. The first takes as input the last layer of the previous model (768) and outputs half of the perceptrons (384). The output layer reduces the number to 7, which are our output labels. We use the Tanh activation function.

The loss function remains the same, but to try to improve the identification for the label with a lower population, we add weights to every label. These weights are calculated as the inverse of the frequency.

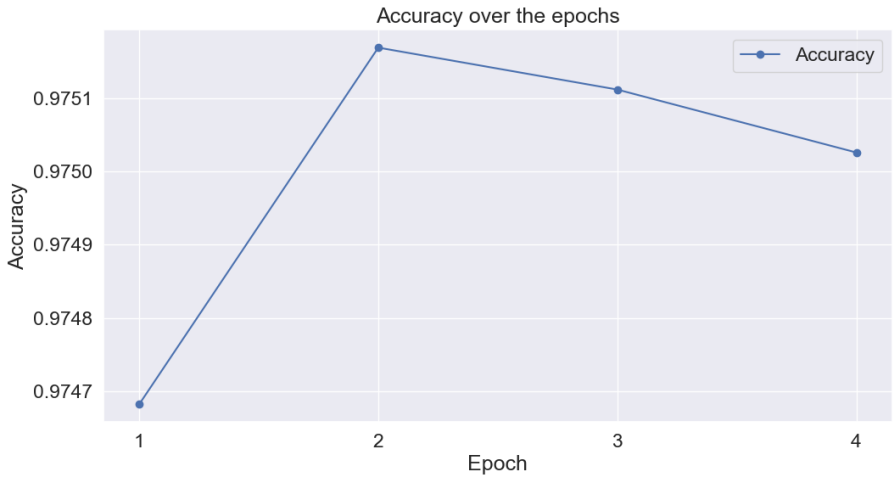
Similar to the previous test part, we load the best model, and we are ready to train the new layer.

4.3.1 Training and Validation

The parameters and the data are the same as in the previous case. The results shown in Figure 4.4 indicate that after only 2 epochs, we have the best model before overfitting..

[1]In all table the value of the 0 and 1 are inverted

(a) Training and Validation loss



(b) Accuracy over the epochs

Figura 4.4: Results of the Training to choose the model for the Test set

4.3.2 Test

The test set will be evaluated on the model taken from the second epoch. The results (Figure 4.5) are similar to the previous ones, with an accuracy of 97.50%. Other scores are pretty similar, except for 'Harmless', which lowers its score in precision but increases its results in recall and F1-score. In fact, we can see in Figure 4.6 that 1 correct classification has risen (from 213 to 286), but also the false positives have decreased (from 61 to 328). For 'Impact', instead, there is no change, with 0 out of 16 positive classifications and 0 false positives.

	precision	recall	f1-score	support
Defense Evasion	0.99	0.97	0.98	9544
Discovery	1.00	1.00	1.00	116042
Execution	0.99	0.99	0.99	46404
Harmless	0.47	0.25	0.33	1126
Persistence	1.00	1.00	1.00	105595
Other	1.00	0.95	0.98	171
Impact	0.00	0.00	0.00	16
micro avg	1.00	0.99	0.99	278898
macro avg	0.78	0.74	0.75	278898
weighted avg	0.99	0.99	0.99	278898
samples avg	1.00	0.99	0.99	278898

Figura 4.5: Result for the test set

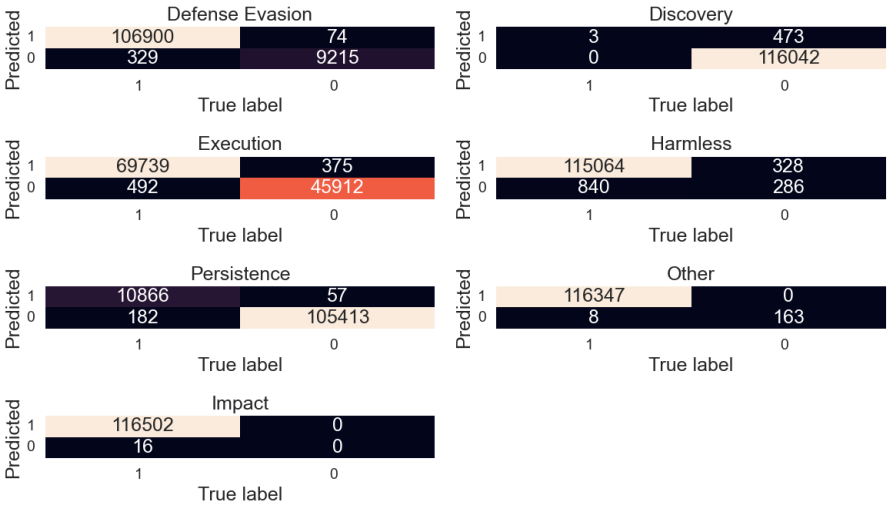


Figura 4.6: Result by label

4.4 Hyperparameter

The hyperparameters for this type of model are numerous and varied, starting with the length of the tokenizer, set to 90 in this example. If the sentence is longer, it will be truncated, and some important information may be lost. We chose 90 because we need to strike a trade-off between computational time and the length of the sentences. As shown in boxplot 4.7, both 'impact' and 'harmless' have most or all of the sentences below 90, so the problem of misclassification cannot be attributed to this parameter.

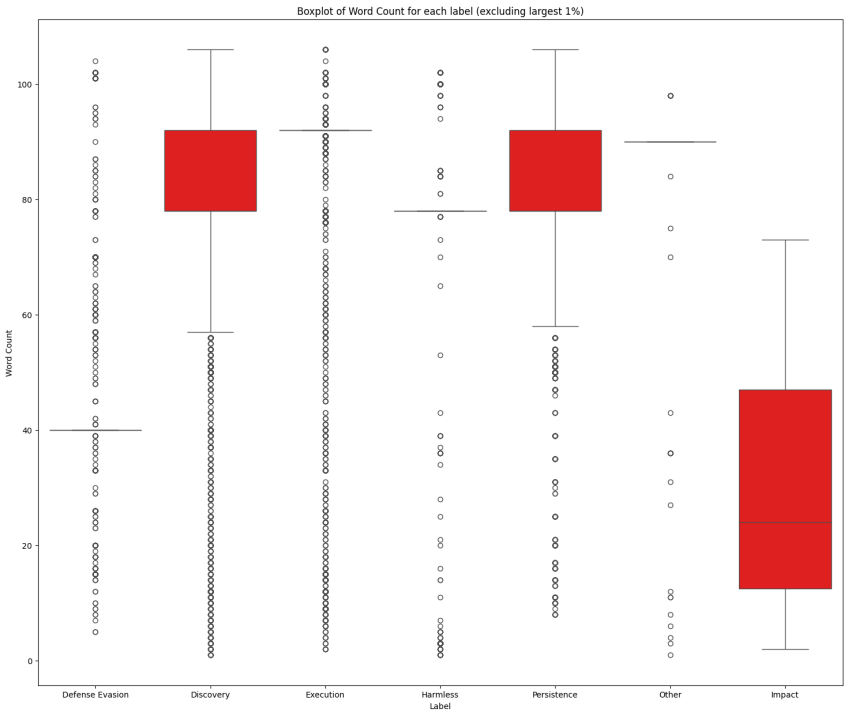


Figura 4.7: Boxplot for length of the sentences

The percentage of the training, validation, and test sets can be different from what we used. Increasing the number of sentences in training may help to achieve better performance on the 'harmless' label, but the time needed for performing rises tremendously.

Two other important factors are the batch size and the sampler for the train, validation, and test sets. Due to the imbalance of the dataset, we decide not to sample the branch because every instance of the smallest label is important. In certain trials, when we sampled the dataset, 'Defense evasion' and 'other' became critical due to the different sampling. A batch size of 32 is taken from Bert's suggestion for fine-tuning on a specific task.

Learning rate and Adam epsilon starting values are taken from the example, but they seem to work well because we have to go through 3 epochs before overfitting. Probably, the values can be lowered in the second training because the training loss falls too quickly. The number of epochs set to 4 is good because we can observe a clear bottom before overfitting in all training times.

4.5 Conclusion

In conclusion, the results seem to be excellent, but there are some areas for improvement, starting with the data. Bert does not have the capability to use weighted loss, and working with an unbalanced dataset is not ideal. With more samples, all the labels could potentially be recognized.

Another area to address is computational power: processing more data and exploring different approaches and hyperparameters require substantial GPU capabilities. GPU computation far surpasses CPU performance for this type of task, providing a 6x faster processing speed. Ho-

wever, the high electric consumption of GPUs, especially when used at 100% load for extended periods throughout the day, poses challenges.

The most significant issue is time. Each entire run, assuming no errors, takes about 2 to 5 hours, depending on various hyperparameters. The number of epochs is particularly impactful in this regard.