Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

SEGMENTATION AND CLASSIFICATION OF ANIMAL BEHAVIOUR FROM ACCELERATION DATA

Alejandro Lendínez Padilla

Director: Ignasi Cos Aguilera
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 13 de juny de 2022

Abstract

The study of animal behaviour is, even today, an unknown field due to the difficulty involved. Most of the time, it is unfeasible to be present to observe and analyse animal behaviour in a situation of freedom, and other study methods such as laboratory study condition the behaviour of the animal and do not allow us to study it in depth.

It has been shown that by analysing time series of the acceleration of the animal this problem can be solved, as it provides very detailed information about the movement of the animal with a high resolution over time, allowing to determine with great precision what the animal was doing at a specific time, without altering its behaviour or the need for human presence.

This work studies a new algorithm for segmenting and classifying animal acceleration data into different behaviours using tri-axial acceleration data (for each Cartesian axis), recorded using an accelerometer and placed in the red-billed tropicbird (Phaethon aethereus). This seabird lives in Cape Verde and is distinguished by flying long distances over the sea. The algorithms explained below are divided into three major blocks: segmentation, to be able to extract different behaviours from the data; grouping, to be able to cluster similar behaviours; and classification, using a recurrent network neuronal (RNN) to be able to classify previously untreated behaviours into one of the groups we found above.

Resum

L'estudi del comportament animal és, encara avui en dia, un camp molt desconegut a causa de la dificultat que comporta. La major part del temps és inviable estar presents per observar i analitzar el comportament animal en situació de llibertat, i altres mètodes d'estudi com l'estudi al laboratori condicionen el comportament de l'animal i no ens permeten estudiar-lo en profunditat.

S'ha demostrat que gràcies a l'anàlisi de sèries temporals de l'acceleració de l'animal es pot arribar a solucionar aquest problema, ja que aporta informació molt detallada sobre el moviment de l'animal amb una gran resolució en el temps, permetent determinar amb gran precisió que estava fent l'animal en un moment concret, sense alterar el seu comportament ni la necessitat de presència humana.

Aquest treball és un estudi sobre un nou algorisme de segmentació i classificació de dades d'acceleració animal en diferents comportaments utilitzant dades d'acceleració tri-axials (per cada eix cartesià), enregistrades mitjançant un acceleròmetre, col·locat en el cua de jonc bec-roig (Phaethon aethereus). Aquest ocell marí habita a Cap Verd i es distingeix per volar grans distàncies sobre el mar. Els algorismes que s'expliquen tot seguit es divideixen en tres grans blocs, la segmentació, per ser capaç d'extreure de les dades els diferents comportaments, l'agrupació, per poder identificar els diferents grups de comportaments, i la classificació, fent servir una xarxa neuronal recurrent (RNN) per ser capaç de classificar comportaments no tractats prèviament en un dels grups que hem trobat anteriorment.

Resumen

El estudio del comportamiento animal es, todavía hoy, un campo muy desconocido debido a la dificultad que comporta. La mayor parte del tiempo es inviable estar presentes para observar y analizar el comportamiento animal en situación de libertad, y otros métodos de estudio como el estudio en el laboratorio condicionan el comportamiento del animal y no nos permiten estudiarlo en profundidad.

Se ha demostrado que gracias al análisis de series temporales de la aceleración del animal se puede llegar a solucionar este problema, puesto que aporta información muy detallada sobre el movimiento del animal con una gran resolución en el tiempo, permitiendo determinar con gran precisión que estaba haciendo el animal en un momento concreto, sin alterar su comportamiento ni la necesidad de presencia humana.

Este trabajo es un estudio sobre un nuevo algoritmo de segmentación y clasificación de datos de aceleración animal en diferentes comportamientos utilizando datos de aceleración tri-axiales (por cada eje cartesiano), registradas mediante un acelerómetro, colocado en el rabo de junco pico rojo (Phaethon aethereus). Este pájaro marino habita en Cabo Verde y se distingue por volar grandes distancias sobre el mar. Los algoritmos que se explican a continuación se dividen en tres grandes bloques, la segmentación, para ser capaz de extraer de los datos los diferentes comportamientos, la agrupación, para poder identificar los diferentes grupos de comportamientos, y la clasificación, usando una red neuronal recurrente (RNN) por ser capaz de clasificar comportamientos no tratados previamente en uno de los grupos que hemos encontrado anteriormente.

Acknowledgements

First of all, I would like to thank my family and Selena for supporting me during the development of this project. They have encouraged me and helped in whatever they could.

Secondly, I would like to acknowledge my tutor Ignasi Cos for letting me work on this project and for the constant guidance given throughout its elaboration. Without him, this project would not be possible.

Contents

Figures

Chapter 1

Introduction

1.1. Motivation

Ethology is the scientific study of animal behaviour, especially under natural conditions [2]. Behaviour is how an animal or person behaves in response to a particular situation or stimulus [3].

When we think about ethology, what comes to our minds are scientists observing animals in their natural habitat or a laboratory, trying to understand what they are doing and why they are acting that way. Still, this method has a lot of limitations.

Little is known about the behaviour of many animal species. That is because of the difficulty of studying them. Most of the time, ethologists cannot observe and analyse animals in their natural habitat, either because their habitat is inaccessible or because their presence changes the behaviour of the animals. Because of this last reason, studying animal behaviour in laboratories is also insufficient for understanding the nature of some animals thoroughly.

Several studies indicate that this lack of understanding of the behaviour of some animals can be partly supplied with acceleration data. Raw acceleration data values recorded in each acceleration channel can be used to determine animal posture and movement [6] and, therefore, can help us identify which behaviour the subject species is performing through time.

1.2. Subject species

The red-billed tropicbird (Phaethon aethereus) is a pelagic seabird that weighs around 700 g, measures 90–107 cm from beak to tail and has a wingspan of 99–106 cm. It mainly inhabits a huge range across tropical waters of the Atlantic Ocean, the northwest Indian Ocean and the eastern Pacific. It usually feeds on small fish caught by plunge diving. However, surface dives can also be expected since one of its primary food sources consists in flying fish (Exocoetus volitans), which are sometimes caught in flight.

Although the conservation status of this species is ‘Least Concern’ according to the Red List of Threatened Species by the International Union for Conservation of Nature (IUCN), the colony from Cape Verde has significantly decreased in its population. As a result, researchers from the Seabird Ecology Lab of the University of Barcelona are working in Cape Verde to understand better these overlooked birds by studying different colonies located on Boavista island and in Ilhéu de Cima.

Figure 1. Red-billed tropicbird. Source: Sharif Uddin

1.3. The data

The data was collected by the members of the Seabird Ecology Lab using the Axy-trek Marine data logger. This tiny waterproof device includes an accelerometer, a GPS pressure sensor and a temperature sensor. Its size is 40 x 20 x 8 mm and weighs 14 g with battery and casing. These devices were attached to the lower back of the Red-Billed Tropicbirds using waterproof tape that does not damage the bird's feathers (TESA tape) while the animals were resting at their respective colonies.

In this study, we only work with the acceleration data, which has a 25Hz frequency, is measured in g (1 g = 9.8 m/s2), and is given for each axis: X (head-tail), Y (right-left) and Z (dorso-ventral). The data is distributed in different files of different individuals.

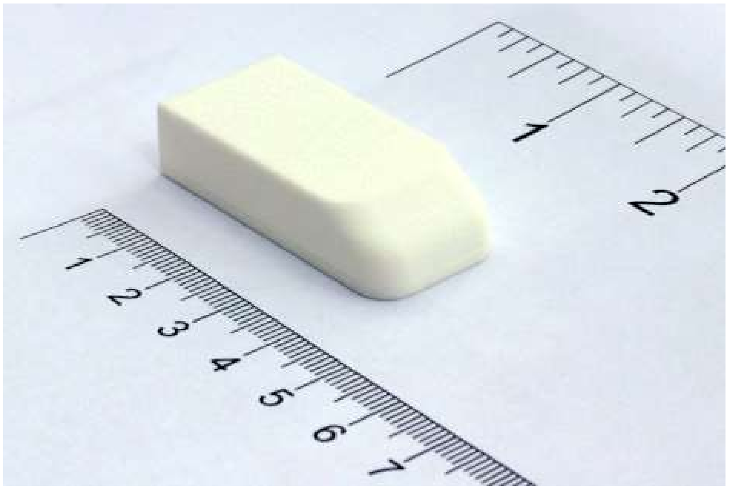


Figure 2. Axy-trek device (Technosmart Europe srl [4])

It is inviable to analyse such large amounts of data manually, so we need to use IT tools.

1.4. State of the art

Knowing that accelerometry is a potent tool for this objective, many studies have already conducted behaviour classification from acceleration data.

Despite this, most of them conduct this classification as a supervised learning method (such as Liang Wang et al. [7]), that although giving good results for later unobserved behaviours, requires a manual labelling and previous observing of the subject species, which, as commented in section 1.1, is not always possible. Other approaches that use unsupervised learning methods tend to divide the data into fixed-length segments.

A study from 2019 by Patterson et al. [8] compared different techniques for classifying behaviour from accelerometers for two seabird species. This work demonstrates that general behaviours of seabirds can be classified from acceleration profiles using a range of techniques and a small number of predictor variables and that the classification method has a negligible effect on accuracy, so simple classification methods would be adequate.

Although we could base our project only on studies about animal behaviour classification, we can also use references that work with time series unrelated to accelerometry. K-Shape [9] is an algorithm presented in 2015 by Paparrizos and Gravano that performs an efficient and accurate time-series classification. Their study states that, unlike the trend in the time-series classification field of using distance measures such as the Euclidean distance or Dynamic Time Warping and its variants, the cross-correlation measure is a lot more efficient and gives very similar results and even slightly better ones.

Finally, this project is based on previous work [1] to create an automatic method for segmenting and classifying acceleration data recordings into different groups that could be easily interpreted as animal behaviours of ecological interest. This work consisted of three main blocks: segmentation, grouping and classification.

The segmentation algorithm takes inspiration from the ADWIN algorithm [10], selecting a series of points over a certain threshold and applying a window to them to merge them with other segments if they overlap.

The grouping algorithm uses the maximum normalized cross-correlation between each segment as a distance measure, grouping the ones with a higher correlation.

The classification is based on the Reservoir computing framework (RC) [11], a framework for computation derived from recurrent neural networks (RNN) that constructs a random recurrent topology and only trains a single linear readout layer. That makes it a lot more efficient and gives similar results. We will discuss the details of this framework in section 4.5.

Chapter 2

Objectives

The objectives of this project are the following:

- To create a pipeline to segment the acceleration temporal series and classify the resulting behaviours in different groups.
- To create a segmentation algorithm that divides the data into different behaviours of the red-billed tropicbird.
- To create a clustering algorithm capable of grouping the before segmented behaviours.

Chapter 3

Planning

3.1. Original planning

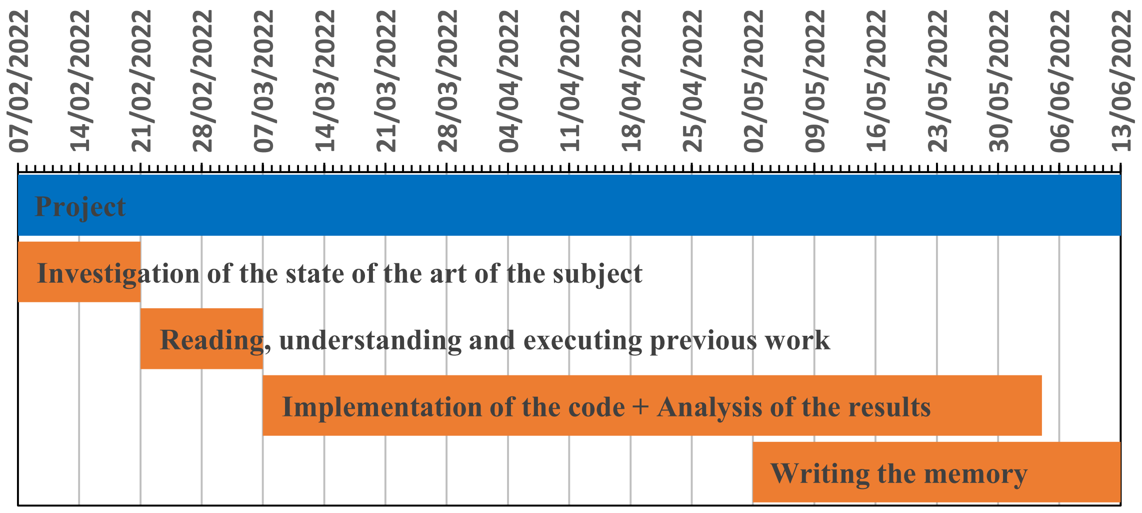


Figure 3. Original planning.

3.2. Final planning

Due to technical factors, we had to wait to have access to a VPN to connect with a remote computer. That delayed the whole project for about a month.

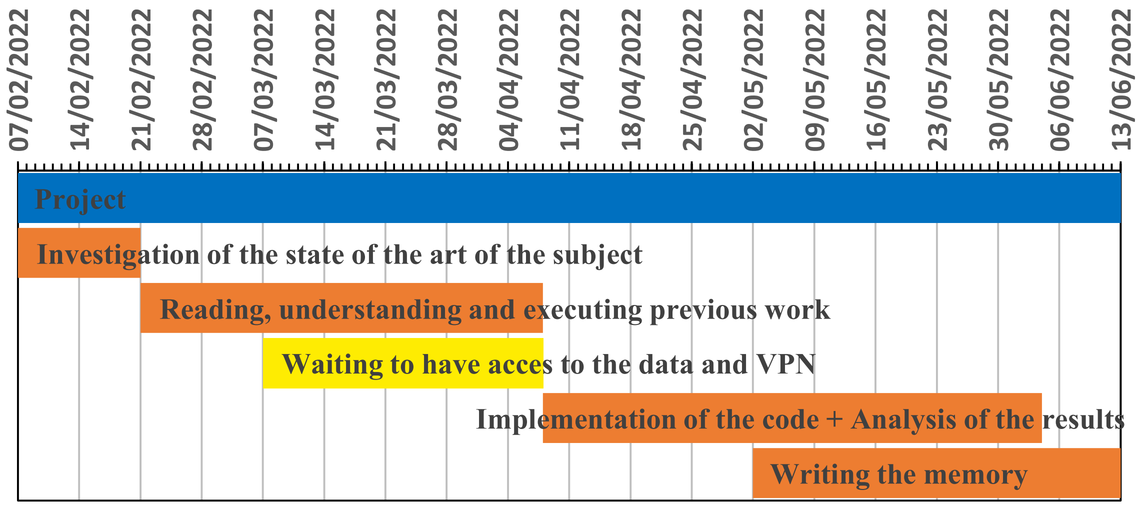


Figure 4. Final planning.

Chapter 4

Implementation

In this chapter, in section 4.1, we will introduce the whole pipeline that we present in this project. In each of the other sections, we will explain the implementation of each of the steps of the pipeline.

4.1. Pipeline overview

The pipeline consists of four main steps: Data pre-processing, Segmentation, Clustering and Testing.

The data pre-processing step, as its name suggests, is the first step in which we prepare the data to be able to treat it. Right after, the segmentation step is needed to distinguish the different behaviours the subject is doing that are later clustered in groups in the clustering step.

Finally, to check if the clustering is good enough, we use a final testing step that consists of a machine learning model based on the Reservoir Computing framework that predicts the group of unobserved behaviours. We can loop over it, correcting the wrong classified segments.

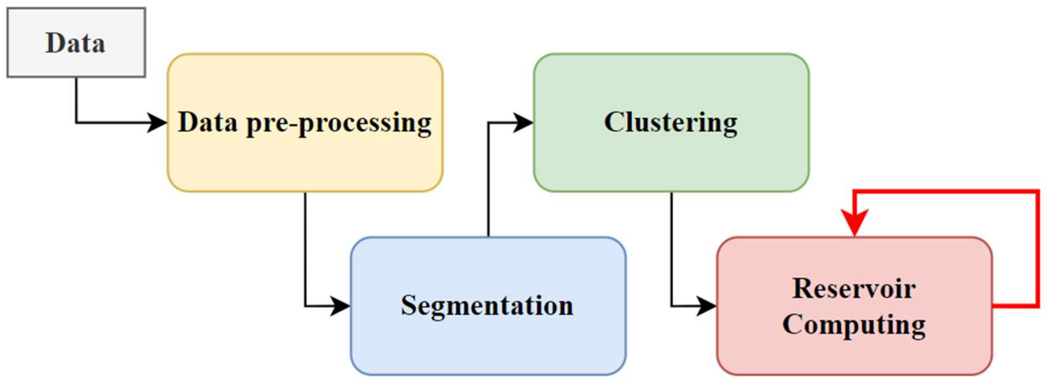


Figure 5. Schematic representation of the pipeline proposed in the present work.

4.2. Data pre-processing

As explained in section 1.3, the data was collected using the Axy-trek Marine data-logger, a device attached to the lower back of the subject species. This data is generated in a specific file format with the extension ".ard" that needs to be extracted with the X

Manager [5]. This program was provided to us by the members of the Seabird Ecology Lab.

Using the X Manger, we can extract the data into CSV files that include the following information:

- Tag ID: to identify the subject
- Timestamp: date and time
- X, Y and Z-axis acceleration value, measured in *g* (*1g = 9.8m/s*^{2})
- Activity: whether the animal was above or below the movement threshold and above or below the 6m threshold in depth.
- Pressure: in mBar
- Temperature: in Celsius degrees
- Latitude and longitude: degrees, minutes and decimal
- Altitude: meters above mean sea level
- Ground speed: real-time speed of the device in km/h
- Satellite count: number of satellites used to take the fix.
- Hdop: value of horizontal dilution of precision for that fix (to determine accuracy, lower values have higher accuracy).
- Maximum-signal-strength: the value of satellite reception power. The higher the value, the better the satellite reception. This is the GSV value.
- Sensor raw: this is the value obtained from the analog sensor.
- Battery Voltage: the voltage of the battery

We could also generate KML files with the position data, a file type that can be opened with Google Earth, but as we are only interested in the acceleration data, we won't use it.

With this data already to be treated, the only pre-processing modification our proposed pipeline applies to the acceleration signals before starting with the segmentation algorithm is a 4th order Butterworth filter.

The objective of applying this filter is to smooth the signal and reduce its noise.

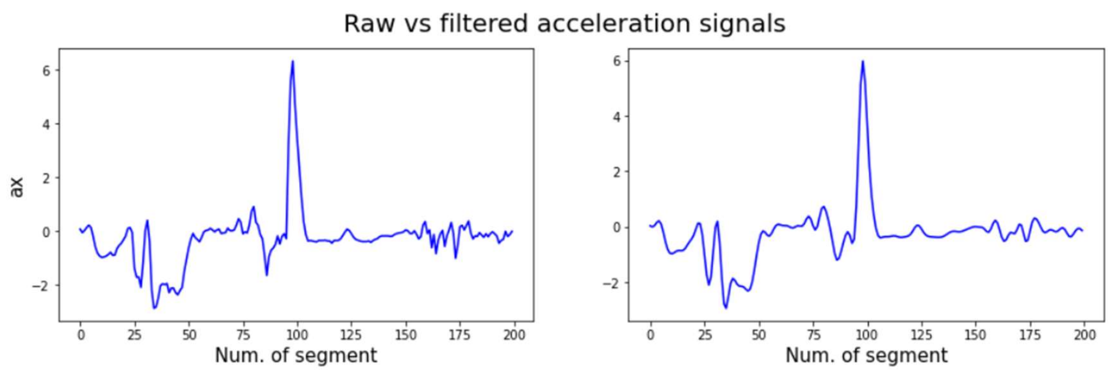


Figure 6. Example of raw signal (left) vs filtered signal with a 4th order Butterworth filter (right).

4.3. Segmentation algorithm

The data segmentation in different segment behaviours is a complex problem that we need to perform very precisely since having the wrong output segments would make the rest of our pipeline useless.

As our resources are limited, our first approach to this problem is detecting the "transition" behaviours. We refer as "transition behaviours" to those with high variance, or what is the same, that are not constant. Behaviours like "resting" or "constant flying" will have constant values and a large length. We do not have enough computation power to process these large segments in the next steps of the algorithm, and they should be easier to identify as they have constant values, so we try to detect the "transition" behaviours, such as "landing" or "diving", that are shorter and more difficult to differentiate.

Our algorithm takes inspiration from the ADWIN algorithm [10] and the implementation used in the previous work [1]. The main idea is simple, we have two variable parameters: w (window size) and σ (threshold).

We will iterate over the data applying a window of size w to each point. Then for each data point, we will calculate the standard deviation of the window, so the result will be an array of the same size of the data populated with the standard deviations.

Once we have calculated the standard deviation, we get to the second step of the algorithm. We get the standard deviations array indexes that have a value over σ. Then, we join the immediately consecutive indexes and apply the third step of the algorithm for each of them.

For each segment, we apply a window of size w (understanding applying a window as expanding both ends of the segment by $w/2$) and look if it overlaps with another segment. If it is overlapping, we merge them, recheck if it is overlapping with other segments to merge them again until they are not overlapping with any new segments, so we reapply the window again at the end from which the segment was merged. If it is not overlapping, we stop the window application and get on with the next segment.

Finally, we also join the segments with a distance inferior to $w/2$ because they are close enough to be part of the same behaviour.

The algorithm parameters must be tuned to get significant behaviours. A high w would merge different behaviours into a bigger one, and a low w would result in a behaviour subdivided into smaller ones. Also, a high σ would leave out some important segments, but a low σ would identify some subsections of large constant behaviours as segments.

```
Segmentation algorithm

Given three acceleration data signals $A_x$, $A_y$, $A_z$ of size N and the parameters w (even
number bigger than 0) and $\sigma$ (bigger than 0).

1. For each signal A, iterate over it and apply a window w, calculating the standard
   deviation of the points in the window.
2. Find the points of each signal A with a standard deviation value > $\sigma$.
3. Concatenate the consecutive points, forming segments.
4. Merge the segments of the three A signals and sort them by starting point.
5. For each segment:
   5.1. If it is not overlapping with the previous segment:
        5.1.1. Apply the window to the left
   5.2. If it is overlapping with the previous segment:
        5.2.1. Merge the segments
   5.3. If it is not overlapping with the next segment:
        5.3.1. Apply the window to the right
   5.4. If it is overlapping with the next segment:
        5.4.1. Merge the segments
        5.4.2. If it is not overlapping with the next segment:
               5.4.2.1. Apply the window to the right
        5.4.3. Go back to 5.4
6. Join the segments that are at a distance < w/2.

Output: List of segments

Note: Steps from 5.1 to 5.4 are not part of an if-else statement, but sequential if
statements.
```

Figure 7. Segmentation algorithm pseudocode.

4.4. Clustering algorithm

Our clustering algorithm is based on the K-Shape algorithm [9] and its implementation by Tslearn [12]. This algorithm is very similar to a typical K-means algorithm but establishes its distance measure on the maximum normalized cross-correlation, specifically in the coefficient normalization, resulting in normalizing the cross-correlation by dividing it by the product of norms of the two segments. This metric is demonstrated in the study that gives better results.

One difference in the use case between the presented in the K-Shape study and the present work is the length of the segments. K-Shape supposes and is only executable having fixed-length segments as input, while our segmentation algorithm produces variable-length segments. To adapt K-Shape to our requirements, we need to change the time-series shape extraction algorithm that it uses, or in other words, change the way we select the centroids of our algorithm.

As we are working with a lot of segments, we use a heuristic supposing that the centroid may be very similar to a real segment, so we select as a centroid the segment that maximizes the squared sum of coefficient normalized cross-correlations between all the segments of the cluster and itself. For k the cluster number, μ_k a centroid candidate for cluster k and x a segment:

$$\mu_k^* = argmax \sum_{x_i \in P_k} NCC_C(x_i, \mu_k)^2$$

The computation of the NCC$_c$ is quite expensive, so to speed up the process, we will use the Tslearn implementation of the normalized cross-correlation [12]. To make this implementation work properly, we will need to pass the longer segment of both we want to calculate the NCC$_c$ as the first.

Moreover, as the calculation of the NCC$_c$ is very computationally demanding, and we need to calculate it between more than 20000 segments, we used the python library “multiprocessing” to parallelize this process and take advantage of all the CPUs of our machine, reducing the amount of time needed by 16 times.

We applied these modifications, creating a new model based on the KShape implementation of Tslearn [12] that we named KShapeVariableLength. The fitting part of the model is computed like the following:

```
KShapeVariableLength

Given N the number of segments, an N x N matrix of distances between segments, k
(number of clusters), max_iter (max. number of iterations) and n_init (number of
random seeds to compute).

1. For each random seed:
   1.1. Get random centroids.
   1.2. Assign each segment to the centroid at less distance.
   1.3. While iteration < max_iter:
        1.3.1. Update the centroids by selecting the segment that maximizes the sum
               of NCC_c squared for all the segments in the cluster.
        1.3.2. Assign each segment to the centroid at less distance.
        1.3.3. If the assignment is the same as the last iteration, exit the loop.
2. Save the result of the best seed

Output: List of group labels for each segment.
```

Figure 8. KShapevariableLength fit pseudocode.

Once we solved the compatibility problems, our algorithm only takes as input the segments generated by the segmentation algorithm and the number of clusters we want to generate:

```
Clustering algorithm

Given a list of segments of size N and a parameter k (number of clusters).

3. Calculate a matrix of distances of size N x N, with each cell corresponding to the
   NCC_c of each pair of segments.
4. Train the KShapeVariableLength model with k clusters.
5. Extract the groups from the model result.

Output: Groups generated.
```

Figure 9. Clustering algorithm pseudocode.

4.5. Reservoir Computing testing model

The pipeline's last step consists of testing our final clustering with the help of an artificial neural network (ANN). In particular, we need a specific type of ANN, the recurrent neural network (RNN). RNNs are a class of ANNs that differ in that they have, as their name indicates, recurrent connections between their nodes, in addition to the feed-forward connections that use simpler ANNs.

We need to use RNN because they are very powerful and used for solving complicated time-series problems, thanks to these recurrent connections that provide it with "memory". The information from previous inputs has an impact on the present and future ones, and that makes it able to process variable-length sequences of inputs, that is what we are trying to classify.

The Reservoir Computing framework [11] is derived from RNNs. The difference is that it constructs a random recurrent topology and only trains a single linear readout layer. That makes it much more efficient, faster and computationally cheaper to train without losing a significant amount of performance, so it is a good option since we do not have much computation power and time.

The implementation of the RC model used is the same as that used in the previous work [1], and its structure can be described like:

- An input layer of K nodes. For our specific case, we use three nodes, one for each acceleration axis.
- A hidden layer of N nodes. These are the only nodes that will conform to the model, apart from those from the input and readout layer.
- A readout layer of L nodes. L is the number of groups that we want to be able to classify.

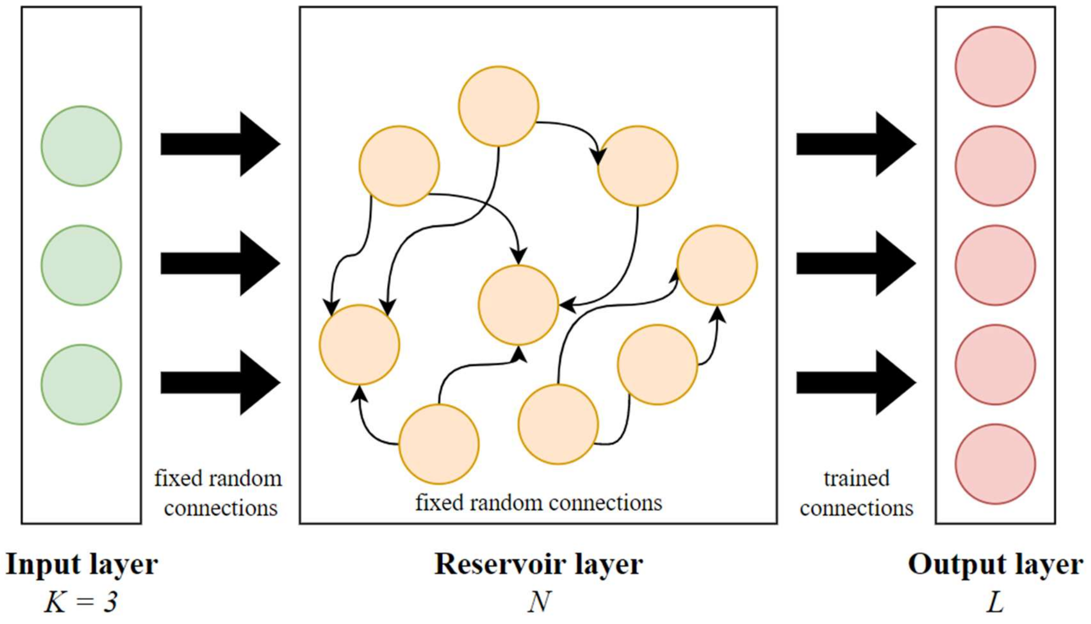


Figure 10. Reservoir Computing topology example.

We also take three input parameters:

- The *input probability,* that indicates the probability of establishing a connection between a node from the input layer and a node from the hidden layer. It takes a value between 0 and 1.
- The *reservoir probability*, that indicates the probability of establishing a connection between two nodes from the hidden layer. It takes a value between 0 and 1.
- The *classifier*, that allow us to set the regressor method to obtain the weights of the connections from the internal units of our network to the output layer. We selected the logistic regressor for this work.

To finalize with the definition of the RC model, the activation functions of each node are hyperbolic tangent functions (*tanh*), which are one of the most used activation functions for RNNs:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

After defining the implementation of the RC model, we can now proceed to explain the data preparation before the training. The groups we will train the model with do not have the same length. That is predictable because there are more common behaviours than others, and without a treatment of the dataset, we could get the predictions biased towards the most common behaviours.

The treatment used to prevent this bias is to apply data augmentation to all the groups, creating new segment variants of the ones that already are part of the groups until they reach the size of the largest group.

We used the Tsaug python library [13] to apply this data augmentation. Specifically, for each new segment that is going to be created, we choose randomly between one of these two methods:

- Adding random noise to the acceleration signals.
- Applying a random and smooth drift to the acceleration signals.

The parameters used to obtain the different copies of the segments are the same used in the previous work [1].

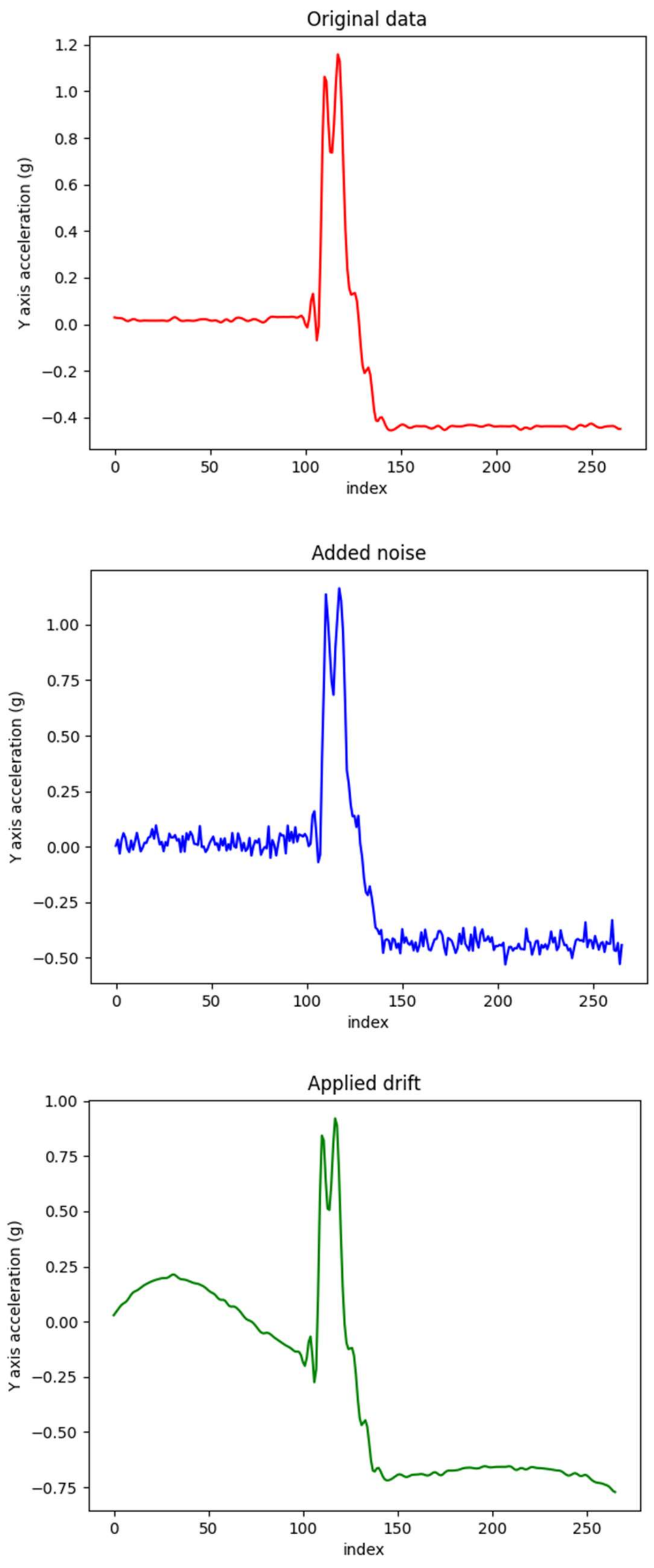


Figure 11. Comparison between an original segment and modifications of it after performing data augmentation.

Another treatment given to the dataset is that we intercalate the segments by their group once the data augmentation is done. For example, if we have 6 groups, we will set the first 6 segments with one of each group: 0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5…

Once we have defined the structure of our model, after training it, we can create a confusion matrix to see the results visually:

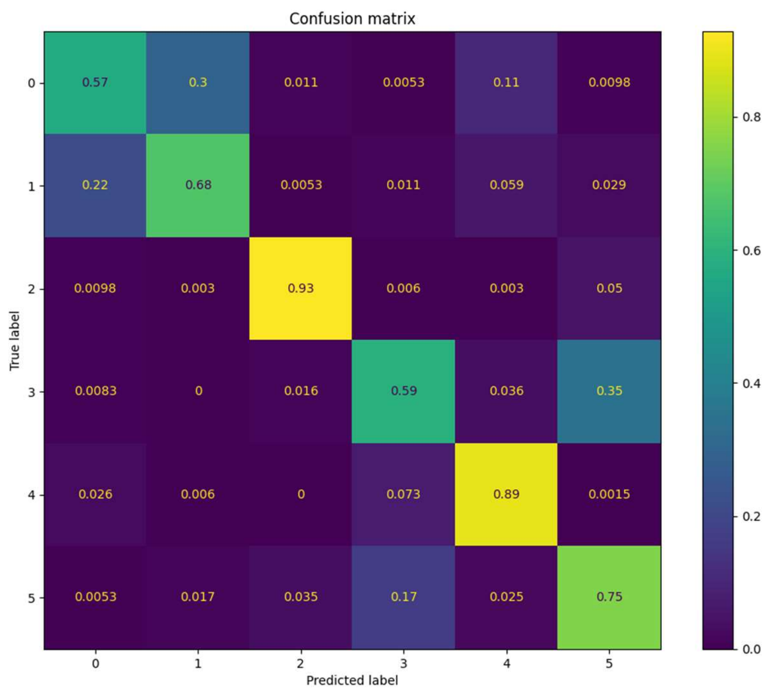


Figure 12. Confusion matrix example of a model with 6 clusters over test data. Performance of 72% of success.

As this is a testing model, it classifies some segments into other groups because they probably belong to it.

Due to the fact that our clustering algorithm is not perfect, we can perform various iterations on it, retraining a new model over again after changing to the predicted group the segments misclassified to improve the performance.

Chapter 5

Results

In this chapter, we are going to analyse the result for each step of the pipeline individually.

5.1. Segmentation results

In this section, we will revise the results that the segmentation algorithm has produced. The result of the segmentation algorithm is a list of segments of variable length that represent different behaviours of the subject species where, at some point, the standard deviation of a slice of size *w* is above the threshold *σ*.

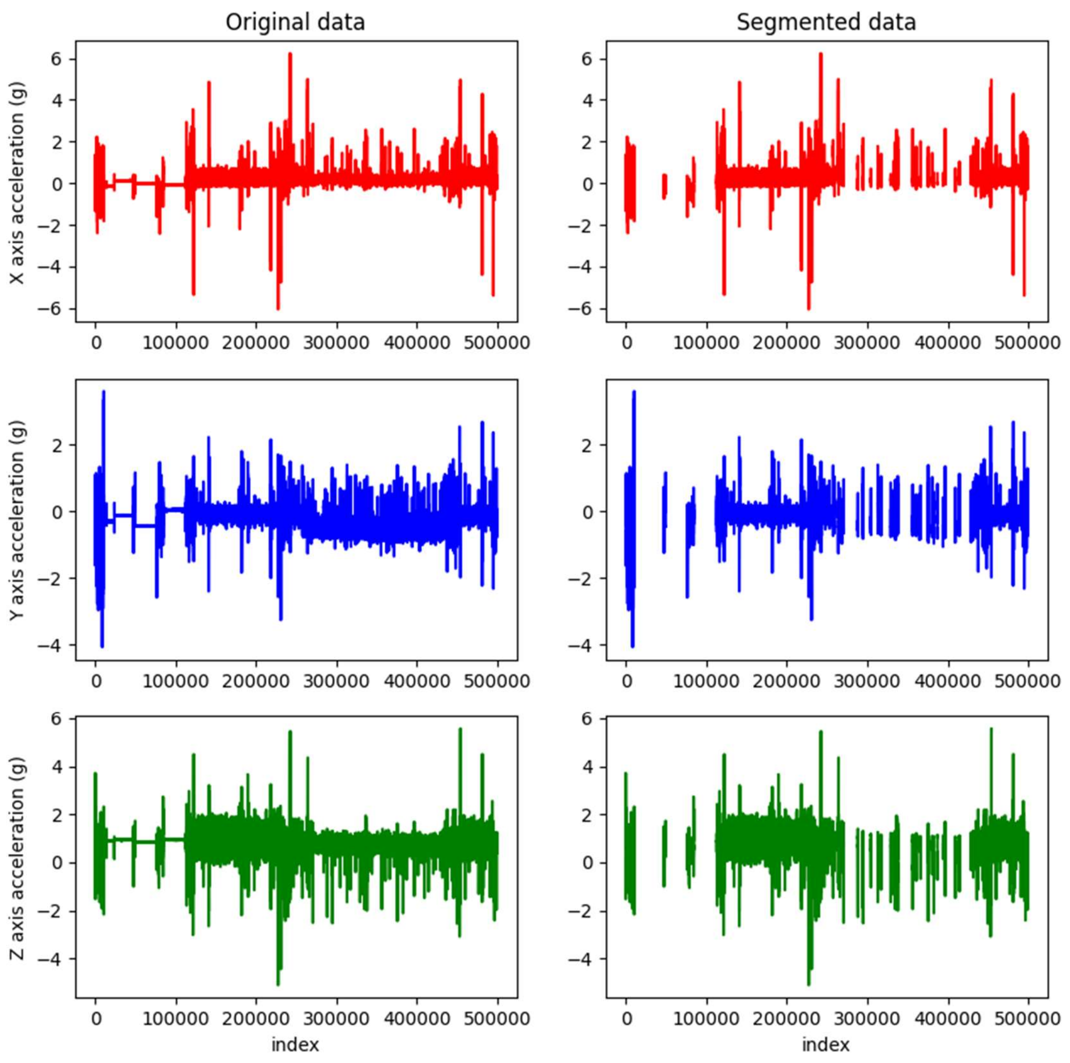


Figure 13. Result of the segmentation algorithm for σ=0.3 and w=150. Original data (left) vs Segmented data (right).

Several interpretations can be obtained from the results produced by the segmentation algorithm. First, as our first objective was, the algorithm tends to leave out the large constant segments that we would recognize as “resting” or “flying”. On the other hand, the results differ depending on the parameters on the behaviours we are trying to identify.

The metrics we considered to decide which segmentation was the more accurate are:

- Number of segments
- Median segment length
- Manual identification of important segments that should be identified

And the parameters modified, as explained in section 4.3 are:

- w – window length
- σ – threshold

A high window length tends to lower the number of segments and increment their size, merging nearby segments. Otherwise, a low window length will make segments very near each other, interpreted as different behaviours.

Also, a high threshold tends to leave out segments with a low standard deviation that could be of biological significance. In contrast, a low threshold could identify slight increments of standard deviation that could not be interpreted as an independent behaviour.

Whatever our interpretations are, it is a fact that as we increase w and σ, our number of segments decreases and our distance between the segments increases. On the other hand, the segment length increases with the increase of w but is independent of σ.

The final tests we made in the final version of the algorithm are the combination of the following values for both parameters:

- w – 50, 70, 80, 100, 150, 200
- σ – 0.3, 0.4, 0.5, 0.6

A total of 24 combinations.

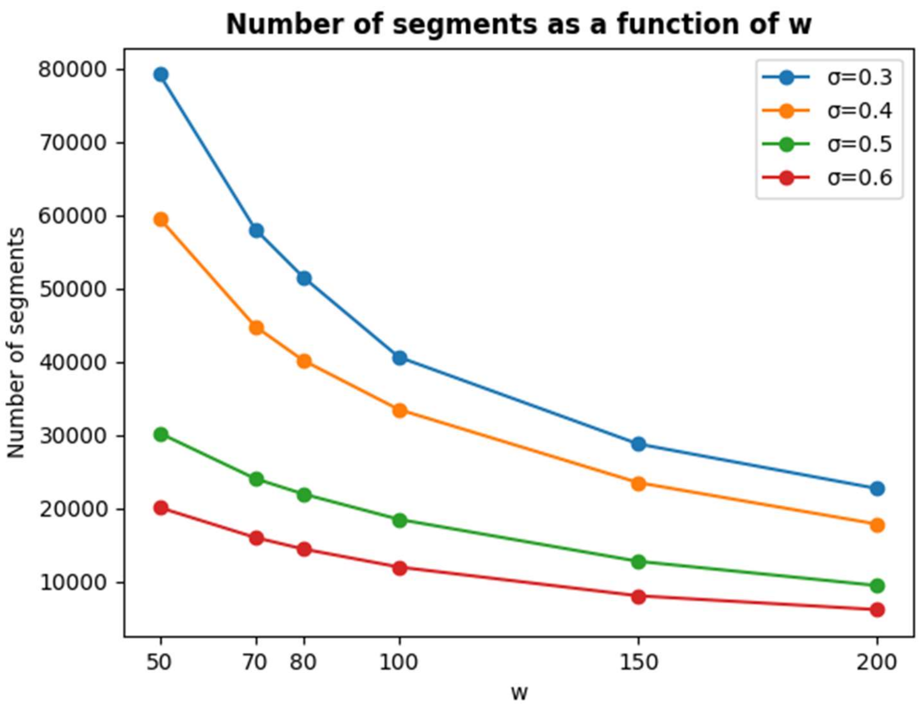


Figure 14. Number of segments as a function of *w* for each *σ*.

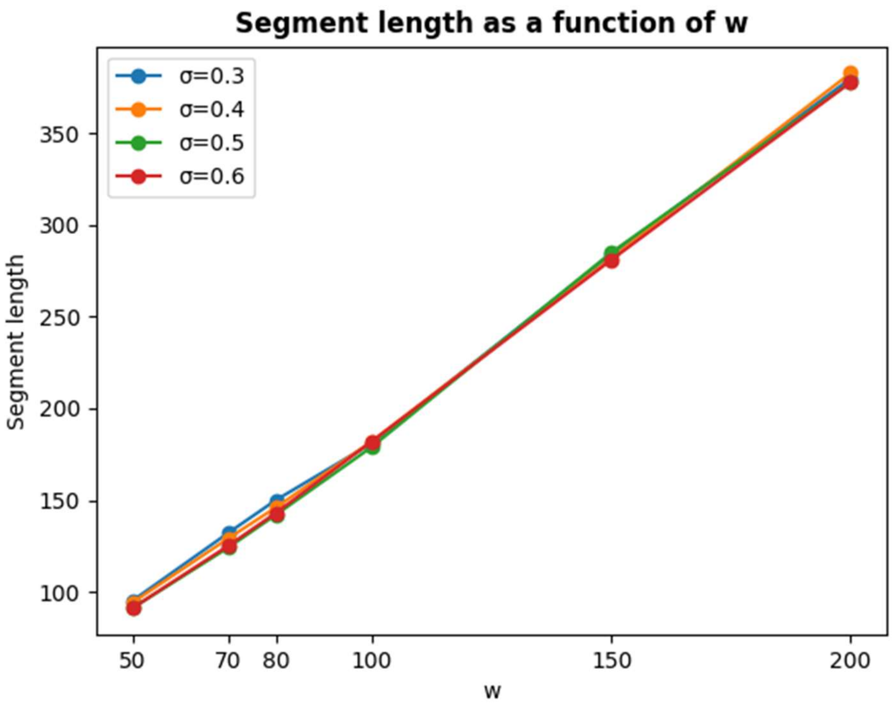


Figure 15. Median segment length as a function of *w* for each *σ*.

From the results of figures 10 and 11, we can extract the following conclusions:

- The segment length is independent of the σ value.
- For a $w \leq 100$, the median segment length is < 200 and the number of segments for $\sigma < 0.5$ is ≥ 40000 approximately. Such values indicate that some segments may be part of the same behaviour. On the contrary, for a $w = 200$, we get a low number of segments and a high median segment length, which indicates that some independent behaviours are being merged.
- For a $\sigma \geq 0.5$ the number of segments is considerably lower than with lower values, sign that we are losing a significant number of behaviours.

That leaves us with two options to check manually: $w = 150$ and $\sigma = 0.3/0.4$.

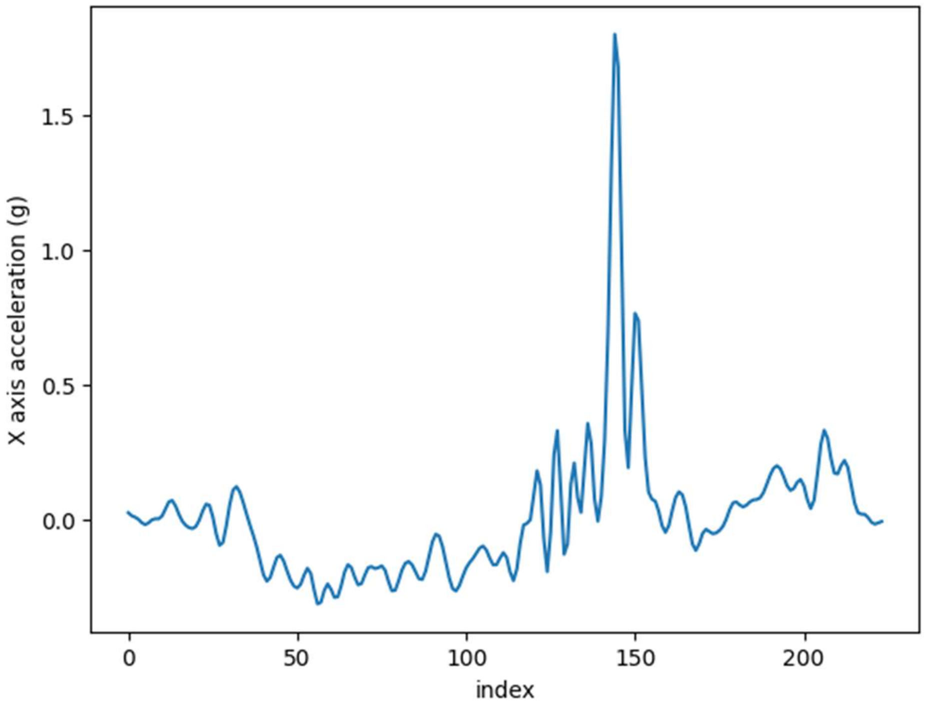


Figure 16. Segment detected with w=150, σ=0.3 and not with σ=0.4

In figure 12 we can visualize the X-axis of a segment detected only with σ=0.3. This segment is a clear acceleration event that for values of $\boldsymbol{\sigma} > 0.3$ is not detected, so we concluded that the optimal value for this parameter was 0.3.

After comparing all the results of the different combinations and discussing their differences, we selected as the optimal parameters to move on with the pipeline the following ones:

$$\boldsymbol{w}=150 \qquad \boldsymbol{\sigma}=0.3$$

5.2. Clustering results

In this section, we will look at the results produced by the clustering algorithm. As explained in section 4.4, we need to indicate manually how many clusters we want to generate. The model calculates the inertia, the sum of the squared distances between the segments, to get the best centroid of the clusters. To get an optimal number of groups we can compare the inertias given by the models:

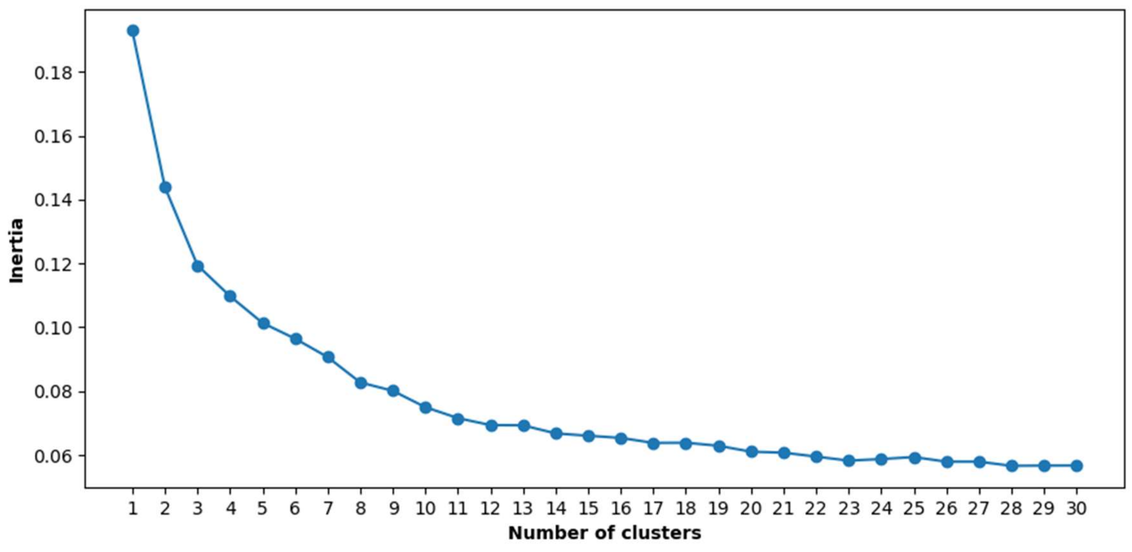


Figure 17. Loss function of the clustering algorithm.

We can observe how, as we increment the number of clusters, the inertia tends to decrease. Despite this, having lower inertia does not indicate that the number of clusters is optimal because is possible that a group of segments could be divided into two or more clusters of the more similar segments inside it.

The optimal number of clusters must be one where the loss function starts to be near asymptotic.

5.3. Reservoir Computing results

In this section, we will present the results from the Reservoir Computing model. As the clustering algorithm output depends on the initial number of clusters given by parameter, we are unaware of how many groups of behaviours are in our dataset, so we trained new models with a different number of clusters several times.

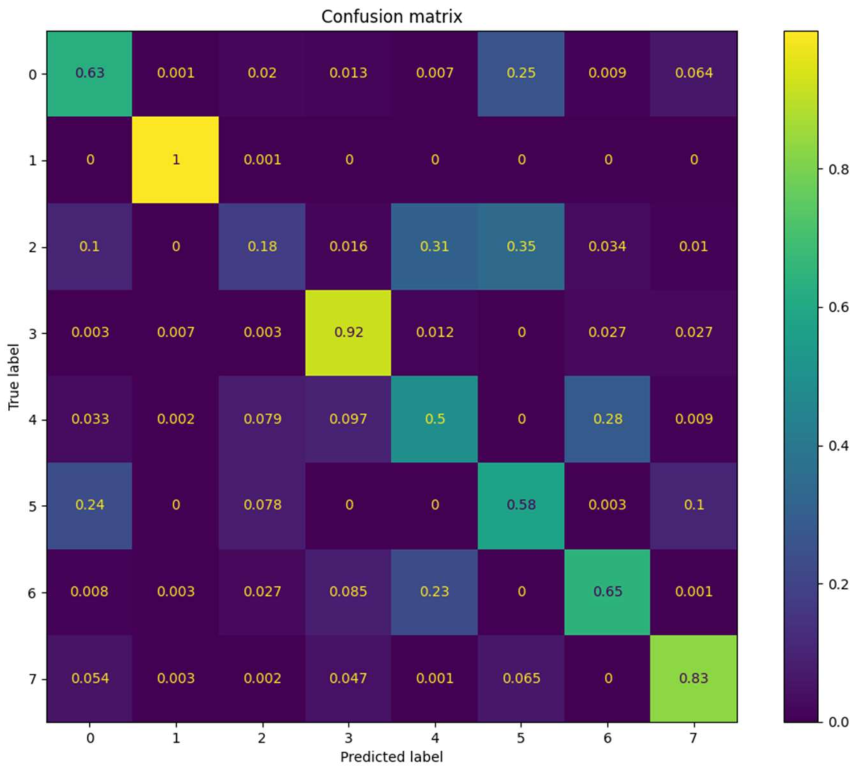


Figure 18. Confusion matrix of a model with 8 clusters over test data. Performance of 66% of success.

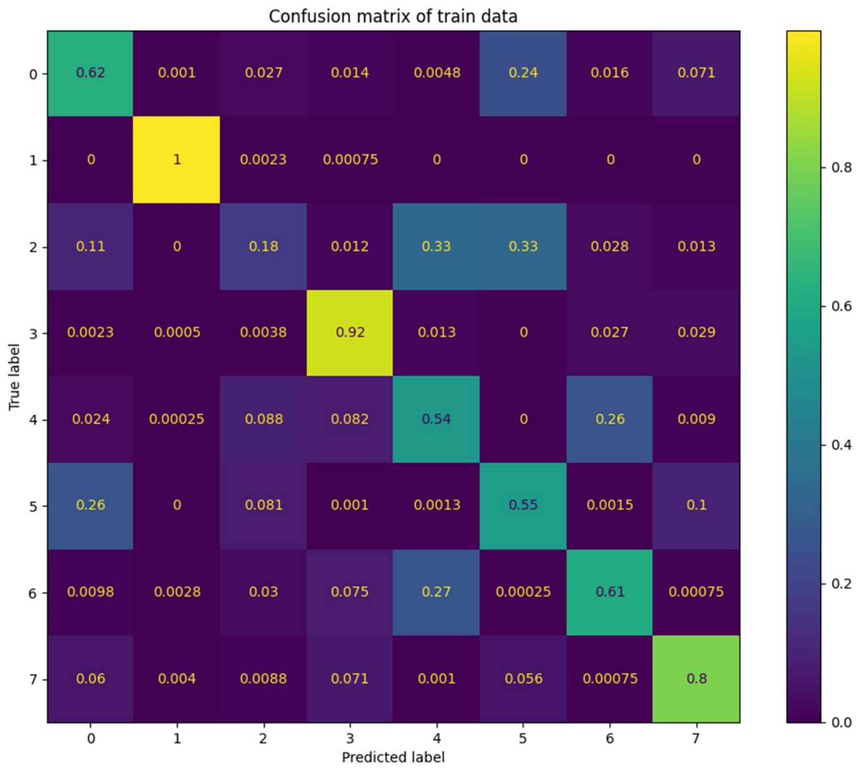


Figure 19. Confusion matrix of a model with 8 clusters over train data. Performance of 65% of success.

First, we can see how the results of test and train data are very similar, which is a good indication that we are not overfitting our model, thanks to the measures we took to avoid biases in our model.

Also, we can see how there are groups that are clearly confused with other ones, so we can reassign the labels of the segments misclassified to the predicted ones and retrain a new model again.

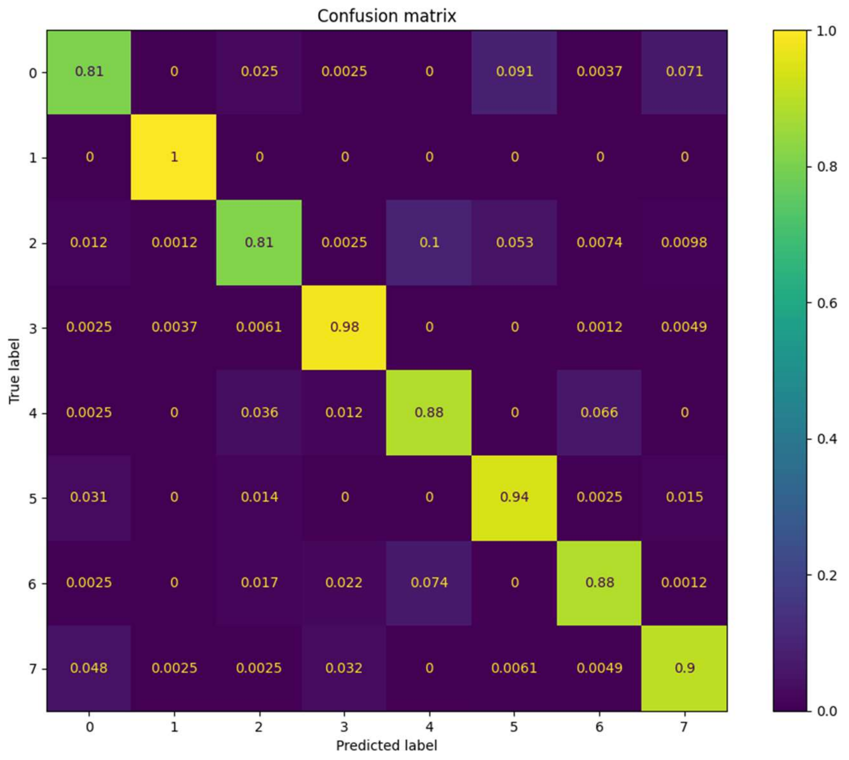


Figure 20. Confusion matrix of a model with 8 clusters over test data, after reassigning the labels. Performance of 90% of success.

We can see how the performance increases to 90% after reassigning the labels only once. We can keep reassigning the labels and retraining the model repeatedly until we reach the best possible accuracy.

A 100% accuracy is doubtful to be possible, as some identified behaviours could not pertain to any group, as they could be random movements not usually done by the subject species.

Here are the results for other numbers of clusters (see the Annex C for more results):

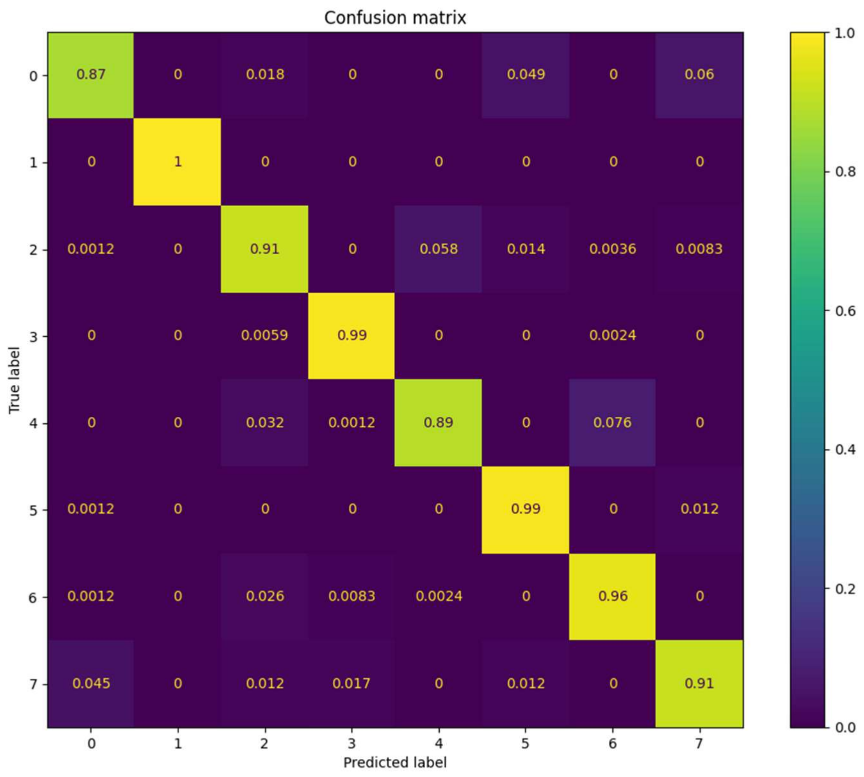


Figure 21. Confusion matrix of a model with 8 clusters over test data, after reassigning the labels two times. Performance of 94% of success.

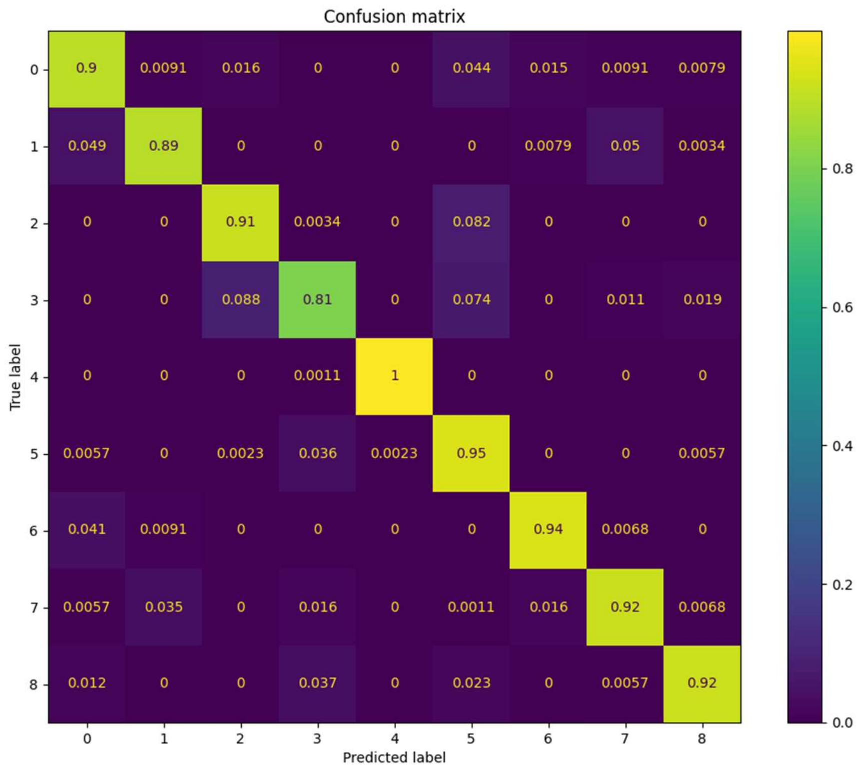


Figure 22. Confusion matrix of a model with 9 clusters over test data, after reassigning the labels two times. Performance of 92% of success.

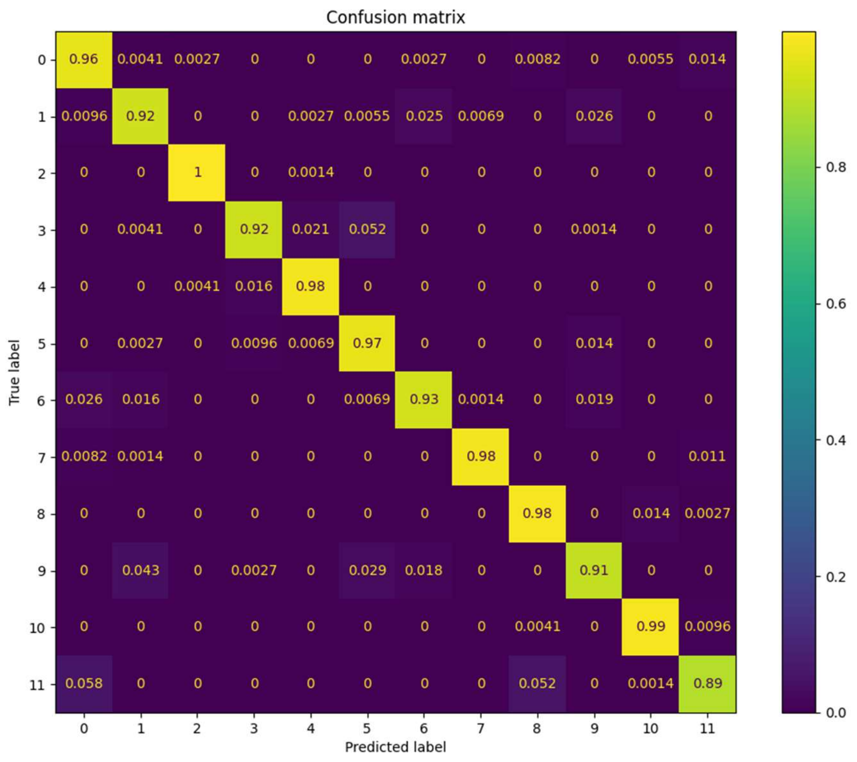


Figure 23. Confusion matrix of a model with 12 clusters over test data, after reassigning the labels three times. Performance of 95% of success.

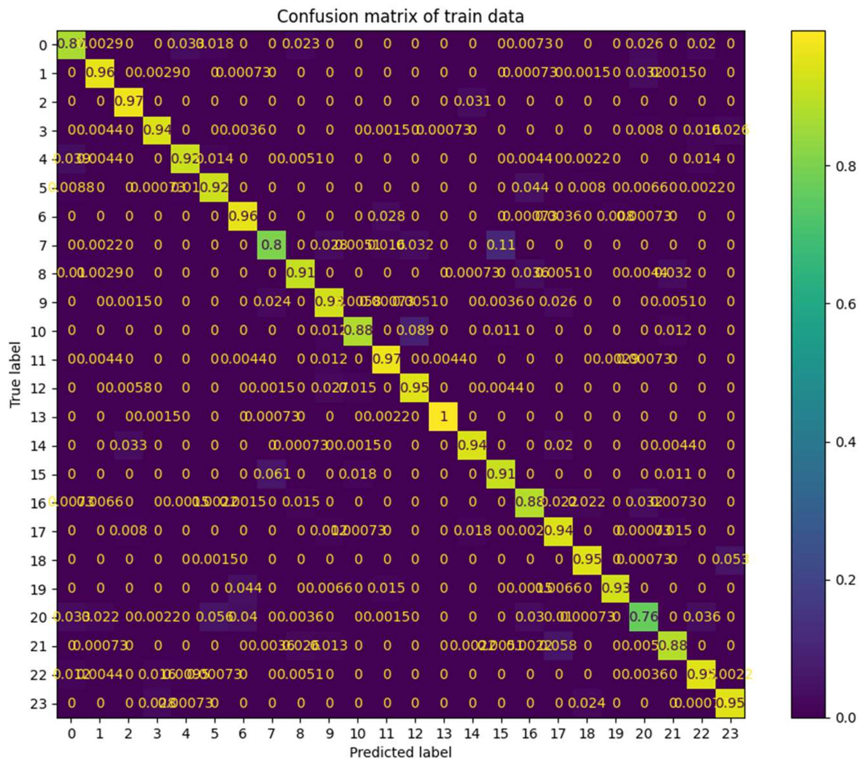


Figure 24. Confusion matrix of a model with 24 clusters over test data, after reassigning the labels three times. Performance of 92% of success.

We can see how we can obtain a remarkably high performance for any number of clusters. This could indicate that there are groups made of two similar groups in the classification with a small number of clusters, or there are groups divided into different clusters in the classification with a high number of clusters.

After trying different methods such as density plots of the correlation to the centroids for each group, comparing the medians and others, we could not identify if one of the two hypothetical conditions explained in the previous paragraph in any of the groups for any of the number of clusters.

It is worth noting that, due to the high memory requirements of the RC model, the 5% largest segments needed to be discarded to train the model.

Chapter 6

Conclusions

In the present work, we have proposed a pipeline that can classify different behaviours into groups from raw data, generating an ethogram.

The objectives of this project, as defined in chapter 3, were the creation of this pipeline, including a segmentation algorithm that divides the data into different behaviours of the red-billed tropicbird and a clustering algorithm capable of grouping the before segmented behaviours. This pipeline actually includes segmentation and clustering algorithms, so we can conclude that the objectives were successfully achieved, although there is still room for improvement.

The segmentation algorithm gets raw acceleration data as input and, with only two parameters (w and σ), can divide the data into different variable-length behaviours that the subject species has performed.

The clustering algorithm gets as input the segments generated by the segmentation algorithm and the number of groups we want to generate, and it does it by minimizing the distances between the segments in each group.

Finally, using the Reservoir Computing framework, we are able to check how good the clustering is and loop over it, correcting the labels of the misclassified segments and retraining the model, in order to get the classification up to 95% of accuracy.

Due to the lack of time, although the initial objectives were met, we could not wholly finalize the study and create a usable tool, as this is a very extensive and not trivial problem. The future work that should be done to continue this project is detailed in chapter 7.

Once the project is finalized, it could be used with other species. The algorithms treat the acceleration data as generic time series, with only the need to tune the parameters.

Chapter 7

Future work

Once analysed the results and finished our project, we thought of possible changes in our pipeline that could lead to better results. In this section, we will comment on these changes that should be considered to be implemented.

Changes on the segmentation step

When analysing the segmentation algorithm, we thought of two changes on it that could lead to a better segmentation.

The first one is trying to use different thresholds when detecting the "transition" behaviours for each of the three axes. Even though our first thought was that all the axes are equally important, so we should put the same threshold on the three of them, after working with the data, we realised that the signals were significantly different in shape, so it could be a good option to consider that each of the axes has its own requirements to detect change points on them.

The second one is to add the "constant" segments between the "transition" segments to the pipeline, as they could also be considered as different behaviours and be classified.

Changes on the clustering step

The main flaw of our clustering step is that we are not able to identify which is the optimal number of clusters to classify, or what is the same, we do not know how many types of behaviours our subject species has. This should be the main line of work to follow to continue this project.

Another change that could be interesting to explore is to find a way to adapt the "Shape extraction" algorithm from the "KShape" model [9] for variable-length segments, to get more accurate centroids for each cluster. In this work, we used a heuristic approximating the optimal centroid as one of the segments we already have, but this can lead to misclassifying some segments as it is not the actual optimal centroid.

Bibliography

1. Aguirrezábal, A. (2021). Segmentation and classification of animal behavior from tri-axial accelerometry recordings.
2. "Definition of ethology". Merriam-Webster. Retrieved 7 May 2022.
3. "Definition of behaviour". Lexico. Retrieved 14 May 2022.
4. Axy-Trek Marine. Technosmart Europe srl. Available online: https://www.technosmart.eu/axy-trek-marine (accessed on 12 June 2022).
5. X-Manager. Technosmart Europe srl. Available online: https://www.technosmart.eu/downloads/ (accessed on 12 June 2022).
6. Shepard ELC, Wilson RP, Quintana F, Gómez Laich A and others (2008) Identification of animal movement patterns using tri-axial accelerometry. Endang Species Res 10:47-60. https://doi.org/10.3354/esr00084
7. Wang, L., Arablouei, R., Alvarenga, F. A. P. &amp; Bishop-Hurleya, G. J. (2021). Animal Behavior Classification via Accelerometry Data and Recurrent Neural Networks. https://arxiv.org/pdf/2111.12843.pdf
8. Patterson, A., Gilchrist, H. G., Chivers, L., Hatch, S., &amp; Elliott, K. (2019). A comparison of techniques for classifying behavior from accelerometers for two species of seabird. Ecology and Evolution, 9(6), 3030–3045. https://doi.org/10.1002/ece3.4740
9. Paparrizos, J. & Gravano, L. (2016). k-Shape: Efficient and Accurate Clustering of Time Series. ACM SIGMOD Record, 45(1), 69–76. http://dx.doi.org/10.1145/2723372.2737793
10. Bifet, Albert & Gavaldà, Ricard. (2007). Learning from Time-Changing Data with Adaptive Windowing. Proceedings of the 7th SIAM International Conference on Data Mining. 7. http://dx.doi.org/10.1137/1.9781611972771.42
11. Schrauwen, Benjamin & Verstraeten, David & Campenhout, Jan. (2007). An overview of reservoir computing: Theory, applications and implementations. Proceedings of the 15th European Sympsosium on Artificial Neural Networks. 471-482.
12. Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., … Woods, E. (2020). Tslearn, A Machine Learning Toolkit for Time Series Data. Journal of Machine Learning Research, 21(118), 1–6. http://jmlr.org/papers/v21/20-091.html
13. Wen, T. tsaug. 2019. Available online: https://tsaug.readthedocs.io/en/stable/ (accessed on 12 June 2022).

Annex

A. Proposals of modification of the previous work algorithms

In this annex, we included proposals for modifying the algorithms of the previous work [1]. The final clustering algorithm is not included as it is an entirely new different algorithm.

Modification of the segmentation algorithm

Given A_x, A_y and A_z the acceleration signals for each axis, w and $\boldsymbol{\sigma}$

Actual algorithm:

```
1. Find points where $a_n \geq mean(A_n) + \boldsymbol{\sigma}*stdev(A_n)$ or $a_n \leq -mean(A_n) + \boldsymbol{\sigma}*stdev(A_n)$
2. Concatenate consecutive points into the same segment
3. Order the segments based on the starting point of each of them
4. For each segment:
    4.1. Apply the window by adding a w/2 margin to its starting and ending point
    4.2. If it overlaps with another segment:
        4.2.1. Merge them
        4.2.2. Go back to 5.1
    4.3. Else go to the next segment
```

```
1. Find the points of each signal A with a standard deviation value > $\boldsymbol{\sigma}$.
2. Join the consecutive points, forming segments.
3. Merge the segments of the three A signals and sort them by starting point.
4. For each segment:
    4.1. If it is not overlapping with the previous segment:
        4.1.1. Apply window to the left
    4.2. If it is overlapping with the previous segment:
        4.2.1. Merge the segments
    4.3. If it is not overlapping with the next segment:
        4.3.1. Apply window to the right
    4.4. If it is overlapping with the next segment:
        4.4.1. Merge the segments
        4.4.2. If it is not overlapping with the next segment:
            4.4.2.1. Apply window to the right
        4.4.3. Go back to 5.4
5. Join the segments that are at a distance < w/2.
```

Modification proposal 1:

```
1. Find points where $a_n \geq mean(A_n) + \boldsymbol{\sigma}*stdev(A_n)$ or $a_n \leq -mean(A_n) + \boldsymbol{\sigma}*stdev(A_n)$
2. Concatenate consecutive points into the same segment
3. Order the segments based on the starting point of each of them
4. For each segment:
   4.1. Apply the window by adding a w/2 margin to its starting and ending point
   4.2. If it overlaps with another segment:
        4.2.1. Merge them
        4.2.2. Apply the window in the direction where the merged segment was
        4.2.3. Go back to 4.2
   4.3. Else go to the next segment
```

Modification proposal 2 (see in section 4.3)**:**

```
1. For each signal A, iterate over it and apply a window w, calculating the standard
   deviation of the points in the window.
2. Find the points of each signal A with a standard deviation value > $\boldsymbol{\sigma}$.
3. Join the consecutive points, forming segments.
4. Merge the segments of the three A signals and sort them by starting point.
5. For each segment:
   5.1. If it is not overlapping with the previous segment:
        5.1.1. Apply window to the left
   5.2. If it is overlapping with the previous segment:
        5.2.1. Merge the segments
   5.3. If it is not overlapping with the next segment:
        5.3.1. Apply window to the right
   5.4. If it is overlapping with the next segment:
        5.4.1. Merge the segments
        5.4.2. If it is not overlapping with the next segment:
               5.4.2.1. Apply window to the right
        5.4.3. Go back to 5.4
6. Join the segments that are at a distance < w/2.
```

<u>Modification of the grouping algorithm</u>

Given A_x, A_y and A_z the acceleration signals for each axis and thresholds a_x, a_y and a_z

Actual algorithm:

1. Compute the normalized cross-correlation between each of the segments for each acceleration axis and save its maximum value and the lag at which this value occurs.
2. For each segment:
   - 2.1. Find the segments in which the maximum normalized cross-correlation in each axis is higher or equal than thresholds a_x, a_y and a_z, respectively
   - 2.2. Save them in a different list and remove them from the actual one

Modification proposal:

1. Compute the normalized cross-correlation between each of the segments for each acceleration axis and save its maximum value and the lag at which this value occurs.
2. **Find the maximum value of the normalized cross-correlation values:**
   - **2.1. Find the segments in which the maximum normalized cross-correlation in each axis is higher or equal than thresholds a_x, a_y and a_z, respectively with both segments**
   - **2.2. Save them in a different list and remove them from the actual one**

B. Logs

Segments statistics w=100 and σ=0.3 (without last joining step)

```
Data loaded
Total number of segments loaded: 40061
Acceleration data set
Number of segments: 40061

Min distance: 1
5 percent distance: 11
1st quantile distance: 77
Median distance: 357
3rd quantile distance: 1731
Max distance: 2658007

Min length: 101
1st quantile length: 200
Median length: 257
3rd quantile length: 395
95 percent length: 1759
Max length: 184800

Number of 1-axis segments: 22690
Max 1-axis segment length: 29183
Min 1-axis segment length: 101
Mean 1-axis segment length: 398.4961657117673
Median 1-axis segment length: 203.0

Number of 2-axis segments: 7088
Max 2-axis segment length: 40120
Min 2-axis segment length: 152
Mean 2-axis segment length: 529.3766930022573
Median 2-axis segment length: 257.0

Number of 3-axis segments: 10283
Max 3-axis segment length: 184800
Min 3-axis segment length: 162
Mean 3-axis segment length: 1140.7857629096568
Median 3-axis segment length: 280.0
```

Segments statistics *w*=150 and *σ*=0.3 (without last joining step)

```
Data loaded
Total number of segments loaded: 28316
Acceleration data set
Number of segments: 28316

Min distance: 1
5 percent distance: 20
1st quantile distance: 154
Median distance: 676
3rd quantile distance: 2570
Max distance: 2658349

Min length: 151
1st quantile length: 299
Median length: 380
3rd quantile length: 569
95 percent length: 2633
Max length: 223536

Number of 1-axis segments: 15123
Max 1-axis segment length: 62772
Min 1-axis segment length: 151
Mean 1-axis segment length: 587.8466574092442
Median 1-axis segment length: 301.0

Number of 2-axis segments: 5986
Max 2-axis segment length: 65321
Min 2-axis segment length: 227
Mean 2-axis segment length: 771.3434680922152
Median 2-axis segment length: 382.0

Number of 3-axis segments: 7207
Max 3-axis segment length: 223536
Min 3-axis segment length: 234
Mean 3-axis segment length: 1707.5294852227003
Median 3-axis segment length: 466.0
```

Segments statistics *w*=200 and *σ*=0.3 (without last joining step)

```
Data loaded
Total number of segments loaded: 22307
Acceleration data set
Number of segments: 22307

Min distance: 1
5 percent distance: 28
1st quantile distance: 237
Median distance: 954
3rd quantile distance: 3278
Max distance: 2658338

Min length: 169
1st quantile length: 397
Median length: 503
3rd quantile length: 744
95 percent length: 3504
Max length: 224271

Number of 1-axis segments: 11654
Max 1-axis segment length: 87736
Min 1-axis segment length: 169
Mean 1-axis segment length: 754.2649733996911
Median 1-axis segment length: 399.0

Number of 2-axis segments: 5184
Max 2-axis segment length: 75579
Min 2-axis segment length: 301
Mean 2-axis segment length: 1027.2139274691358
Median 2-axis segment length: 506.0

Number of 3-axis segments: 5469
Max 3-axis segment length: 224271
Min 3-axis segment length: 312
Mean 3-axis segment length: 2298.5774364600475
Median 3-axis segment length: 663.0
```

Segments statistics *w*=150 and *σ*=0.3 (with joining step)

```
Data loaded
Total number of segments loaded: 24196
Acceleration data set
Number of segments: 24196

Min distance: 76
5 percent distance: 107
1st quantile distance: 317
Median distance: 1025
3rd quantile distance: 3181
Max distance: 2658349
Number of segments with dist < w/2 0
Number of segments with dist < w 2517

Min length: 151
1st quantile length: 298
Median length: 313
3rd quantile length: 586
95 percent length: 3303
Max length: 224199
```

Reassigning labels

8 clusters

```
Initial groups lengths: [5845, 378, 1500, 3488, 2406, 1899, 4977, 2494]
Final groups lengths: [5028, 413, 1014, 4068, 3053, 2756, 3954, 2701]
Segments misclassified: 13909
Augmented segments misclassified: 6064

Initial groups lengths: [5845, 378, 1500, 3488, 2406, 1899, 4977, 2494]
Final groups lengths: [5187, 417, 973, 4134, 3055, 2683, 3871, 2667]
Segments misclassified: 13951
Augmented segments misclassified: 6119
```

8 clusters - 2nd iteration

```
Initial groups lengths: [5187, 417, 973, 4134, 3055, 2683, 3871, 2667]
Final groups lengths: [4721, 429, 1142, 4224, 2928, 3026, 3808, 2709]
Segments misclassified: 3048
```

Augmented segments misclassified: 846

8 clusters - 3rd iteration

Initial groups lengths: [4721, 429, 1142, 4224, 2928, 3026, 3808, 2709]
Final groups lengths: [4305, 437, 1349, 4245, 2758, 3237, 3841, 2815]
Segments misclassified: 1849
Augmented segments misclassified: 484

9 clusters

Initial groups lengths: [1216, 4750, 1388, 4667, 378, 1633, 2064, 4351, 2540]
Final groups lengths: [674, 4643, 3169, 4271, 420, 437, 2503, 4372, 2498]
Segments misclassified: 19909
Augmented segments misclassified: 9897

9 clusters - 2nd iteration

Initial groups lengths: [674, 4643, 3169, 4271, 420, 437, 2503, 4372, 2498]
Final groups lengths: [801, 4407, 2949, 3776, 427, 1264, 2667, 4289, 2407]
Segments misclassified: 4856
Augmented segments misclassified: 1796

9 clusters – 3rd iteration

Initial groups lengths: [801, 4407, 2949, 3776, 427, 1264, 2667, 4289, 2407]
Final groups lengths: [1112, 4092, 3040, 3299, 434, 1780, 2639, 4232, 2359]
Segments misclassified: 3352
Augmented segments misclassified: 1035

12 clusters

Initial groups lengths: [2002, 2817, 332, 564, 604, 1394, 1904, 3109, 1356, 4523, 2748, 1634]
Final groups lengths: [927, 2780, 362, 662, 749, 2642, 1913, 3737, 2329, 3055, 2402, 1429]
Segments misclassified: 22876
Augmented segments misclassified: 12712

12 clusters - 2nd iteration

Initial groups lengths: [927, 2780, 362, 662, 749, 2642, 1913, 3737, 2329, 3055, 2402, 1429]
Final groups lengths: [1120, 2683, 369, 548, 790, 2900, 1888, 3672, 2416, 2947, 2417, 1237]
Segments misclassified: 4968
Augmented segments misclassified: 2439

12 clusters – 3rd iteration

Initial groups lengths: [1120, 2683, 369, 548, 790, 2900, 1888, 3672, 2416, 2947, 2417, 1237]
Final groups lengths: [1226, 2690, 379, 468, 831, 3011, 1833, 3644, 2440, 2849, 2525, 1091]
Segments misclassified: 2903
Augmented segments misclassified: 1412

16 clusters

Initial groups lengths: [1591, 2441, 258, 1216, 1444, 310, 1426, 1510, 2491, 1962, 859, 1646, 1201, 784, 2007, 1841]
Final groups lengths: [1594, 1448, 422, 2200, 1515, 331, 1261, 166, 2738, 2418, 2584, 540, 764, 1070, 1250, 2686]
Segments misclassified: 18846
Augmented segments misclassified: 6377

16 clusters - 2nd iteration

Initial groups lengths: [1594, 1448, 422, 2200, 1515, 331, 1261, 166, 2738, 2418, 2584, 540, 764, 1070, 1250, 2686]
Final groups lengths: [1494, 1644, 483, 2040, 1597, 337, 1227, 420, 2479, 2085, 2575, 690, 1014, 1235, 1265, 2402]
Segments misclassified: 6125
Augmented segments misclassified: 2628

16 clusters – 3rd iteration

Initial groups lengths: [1494, 1644, 483, 2040, 1597, 337, 1227, 420, 2479, 2085, 2575, 690, 1014, 1235, 1265, 2402]
Final groups lengths: [1492, 1755, 512, 2028, 1648, 343, 1272, 446, 2185, 2001, 2581, 750, 1153, 1430, 1250, 2141]
Segments misclassified: 2970

Augmented segments misclassified: 1099

24 clusters

Initial groups lengths: [1998, 743, 616, 412, 1247, 2359, 700, 23, 503, 989, 1658, 262, 952, 317, 1008, 782, 457, 1230, 460, 1907, 1874, 770, 1347, 373]
Final groups lengths: [2303, 15, 1122, 568, 1390, 1402, 881, 317, 926, 471, 2114, 577, 303, 334, 1829, 45, 822, 187, 1323, 2129, 1809, 958, 333, 829]
Segments misclassified: 30597
Augmented segments misclassified: 17569

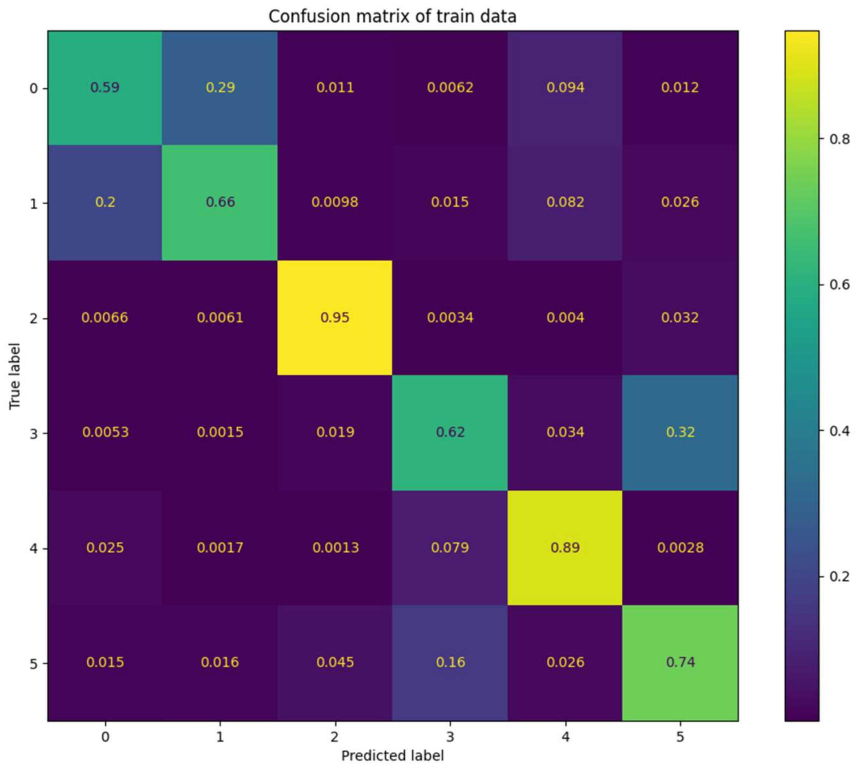
24 clusters -2nd iteration

Initial groups lengths: [2303, 15, 1122, 568, 1390, 1402, 881, 317, 926, 471, 2114, 577, 303, 334, 1829, 45, 822, 187, 1323, 2129, 1809, 958, 333, 829]
Final groups lengths: [2048, 73, 1197, 539, 1028, 1655, 1032, 295, 861, 615, 1812, 692, 499, 338, 1630, 93, 926, 502, 1610, 1874, 1609, 836, 424, 799]
Segments misclassified: 8485
Augmented segments misclassified: 4146

24 clusters -3rd iteration

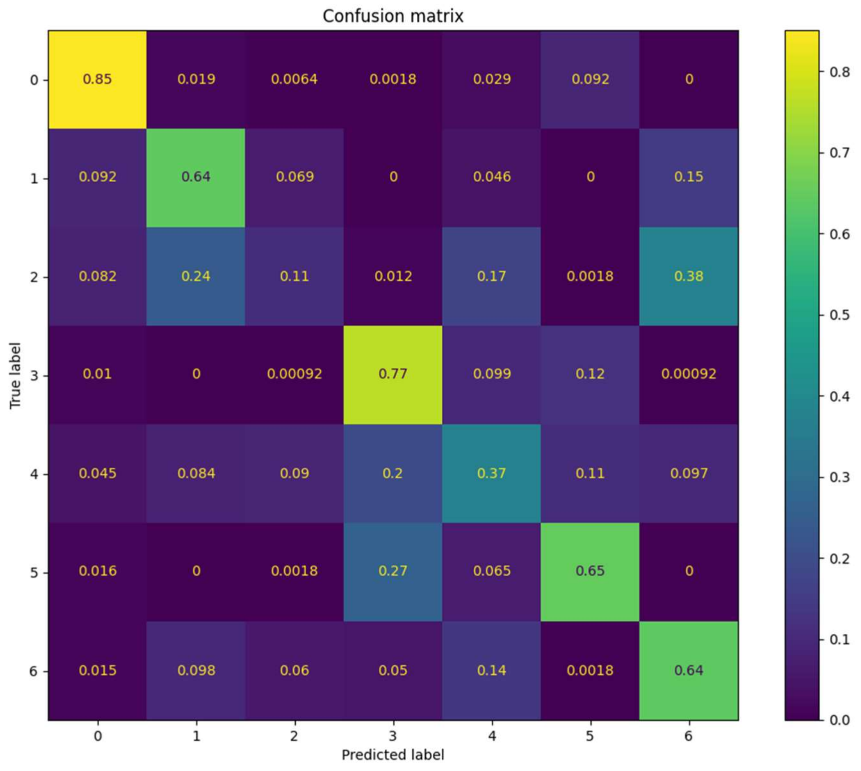
Initial groups lengths: [2048, 73, 1197, 539, 1028, 1655, 1032, 295, 861, 615, 1812, 692, 499, 338, 1630, 93, 926, 502, 1610, 1874, 1609, 836, 424, 799]
Final groups lengths: [1941, 110, 1265, 513, 958, 1672, 1184, 287, 839, 646, 1550, 718, 669, 340, 1537, 146, 1025, 685, 1593, 1713, 1402, 801, 497, 896]
Segments misclassified: 4421
Augmented segments misclassified: 1854

C. More Reservoir Computing results



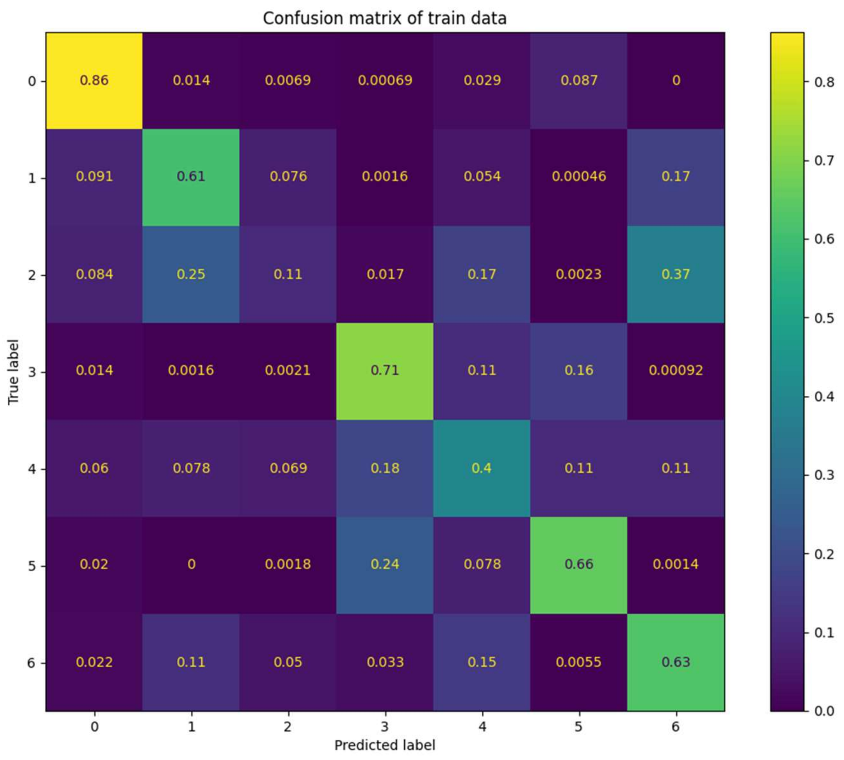
True \ Predicted	0	1	2	3	4	5
0	0.59	0.29	0.011	0.0062	0.094	0.012
1	0.2	0.66	0.0098	0.015	0.082	0.026
2	0.0066	0.0061	0.95	0.0034	0.004	0.032
3	0.0053	0.0015	0.019	0.62	0.034	0.32
4	0.025	0.0017	0.0013	0.079	0.89	0.0028
5	0.015	0.016	0.045	0.16	0.026	0.74

Figure 25. Confusion matrix of a model with 6 clusters over train data. Performance of 74% of success.



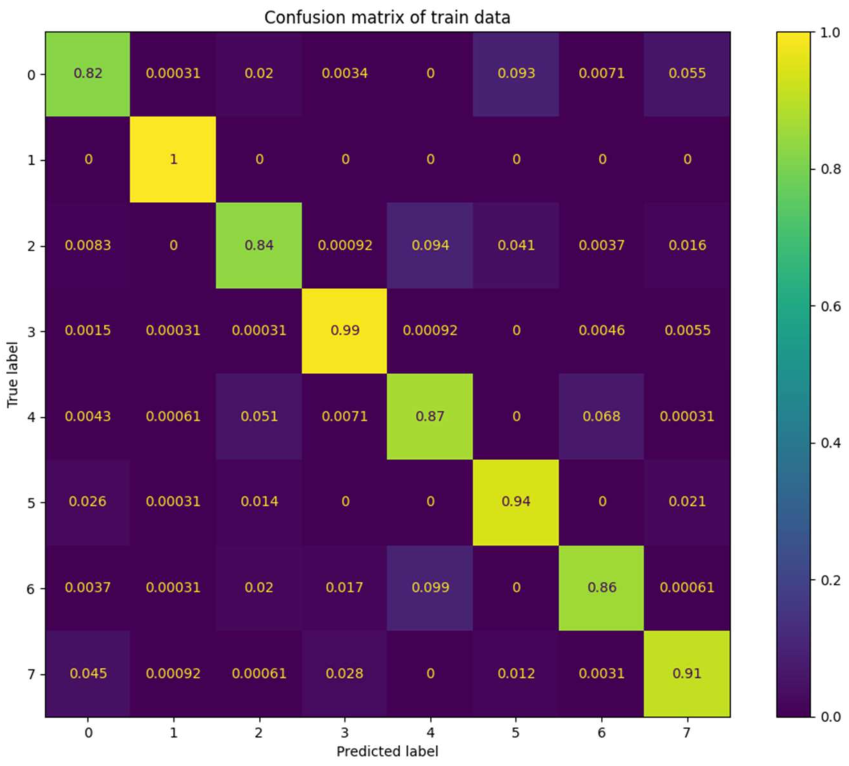
True \ Predicted	0	1	2	3	4	5	6
0	0.85	0.019	0.0064	0.0018	0.029	0.092	0
1	0.092	0.64	0.069	0	0.046	0	0.15
2	0.082	0.24	0.11	0.012	0.17	0.0018	0.38
3	0.01	0	0.00092	0.77	0.099	0.12	0.00092
4	0.045	0.084	0.09	0.2	0.37	0.11	0.097
5	0.016	0	0.0018	0.27	0.065	0.65	0
6	0.015	0.098	0.06	0.05	0.14	0.0018	0.64

Figure 26. Confusion matrix of a model with 7 clusters over test data. Performance of 57% of success.



	0	1	2	3	4	5	6
0	0.86	0.014	0.0069	0.00069	0.029	0.087	0
1	0.091	0.61	0.076	0.0016	0.054	0.00046	0.17
2	0.084	0.25	0.11	0.017	0.17	0.0023	0.37
3	0.014	0.0016	0.0021	0.71	0.11	0.16	0.00092
4	0.06	0.078	0.069	0.18	0.4	0.11	0.11
5	0.02	0	0.0018	0.24	0.078	0.66	0.0014
6	0.022	0.11	0.05	0.033	0.15	0.0055	0.63

Figure 27. Confusion matrix of a model with 7 clusters over train data. Performance of 57% of success.



	0	1	2	3	4	5	6	7
0	0.82	0.00031	0.02	0.0034	0	0.093	0.0071	0.055
1	0	1	0	0	0	0	0	0
2	0.0083	0	0.84	0.00092	0.094	0.041	0.0037	0.016
3	0.0015	0.00031	0.00031	0.99	0.00092	0	0.0046	0.0055
4	0.0043	0.00061	0.051	0.0071	0.87	0	0.068	0.00031
5	0.026	0.00031	0.014	0	0	0.94	0	0.021
6	0.0037	0.00031	0.02	0.017	0.099	0	0.86	0.00061
7	0.045	0.00092	0.00061	0.028	0	0.012	0.0031	0.91

Figure 28. Confusion matrix of a model with 8 clusters over train data, after reassigning the labels. Performance of 90% of success.

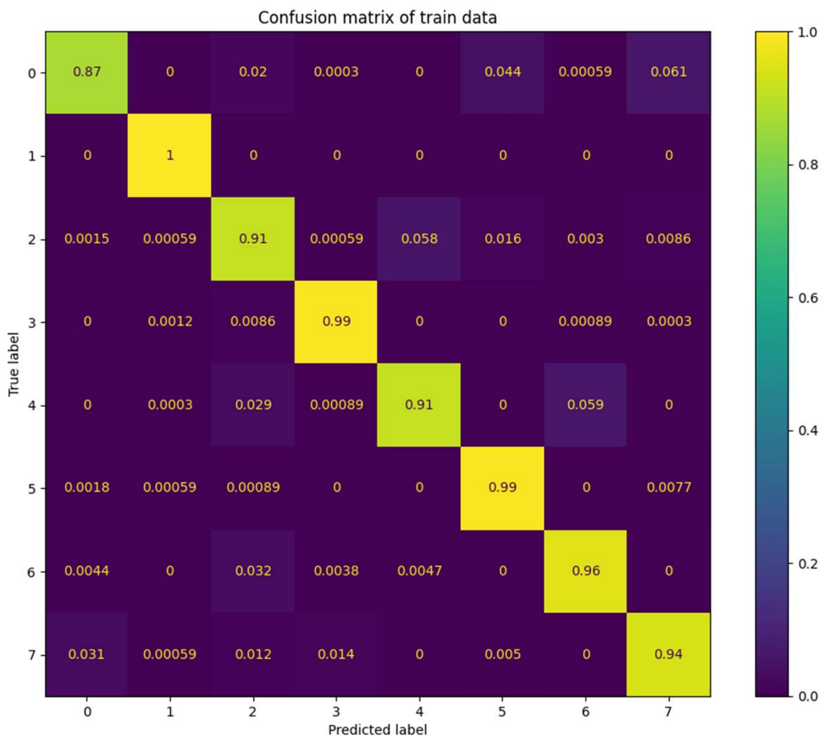


Figure 29. Confusion matrix of a model with 8 clusters over train data, after reassigning the labels two times. Performance of 95% of success.

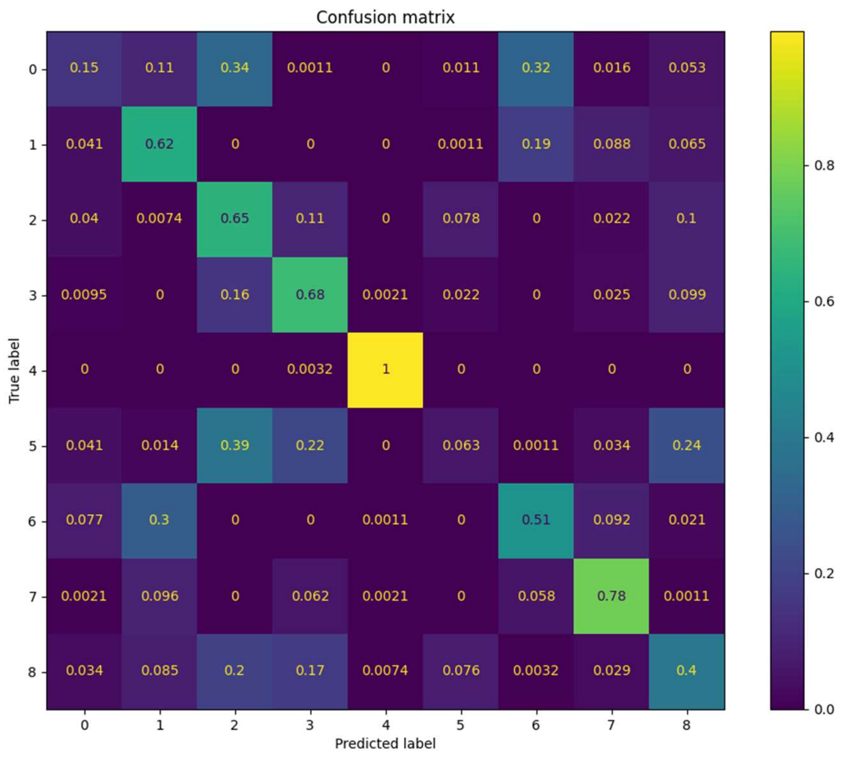


Figure 30. Confusion matrix of a model with 9 clusters over test data. Performance of 54% of success.

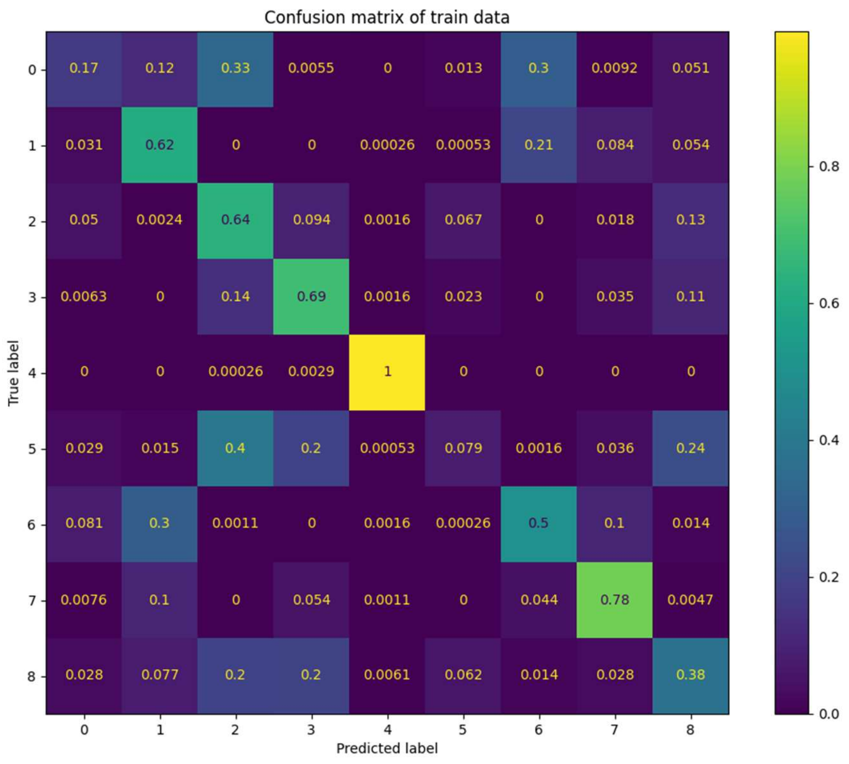


Figure 31. Confusion matrix of a model with 9 clusters over train data. Performance of 54% of success.

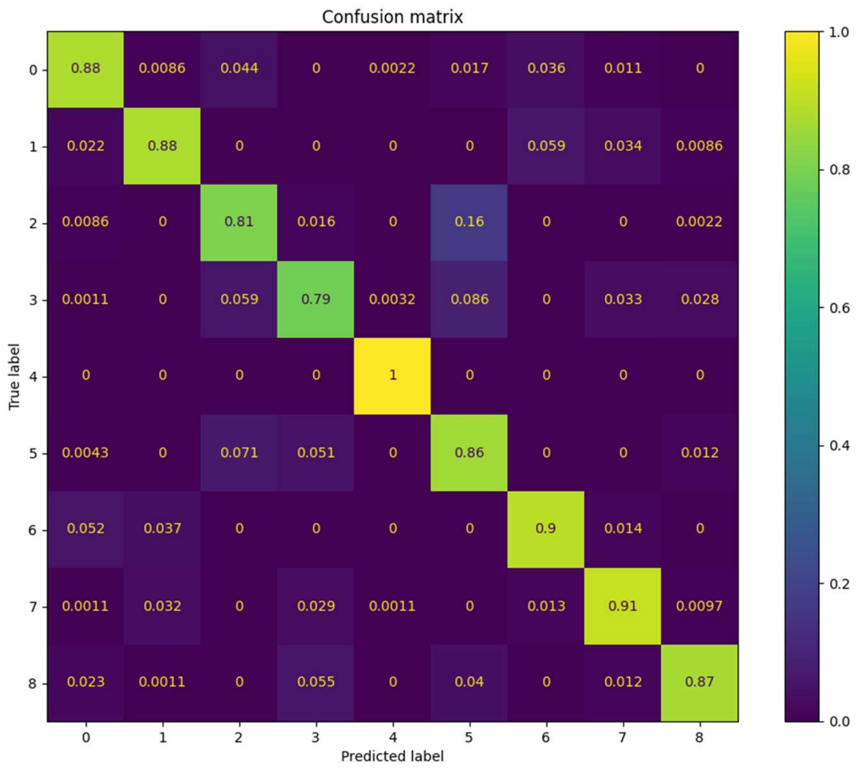
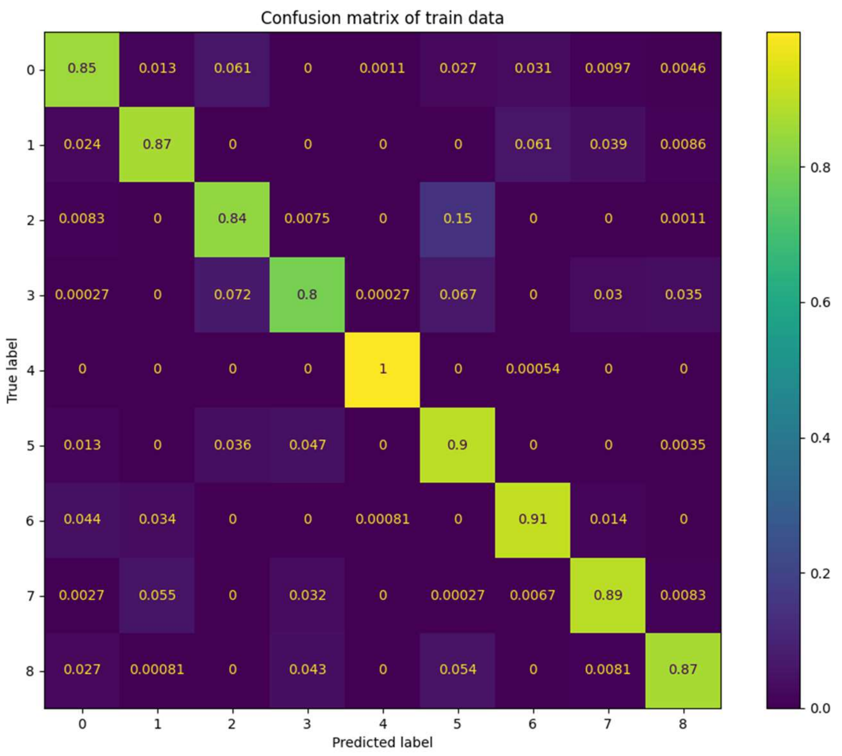
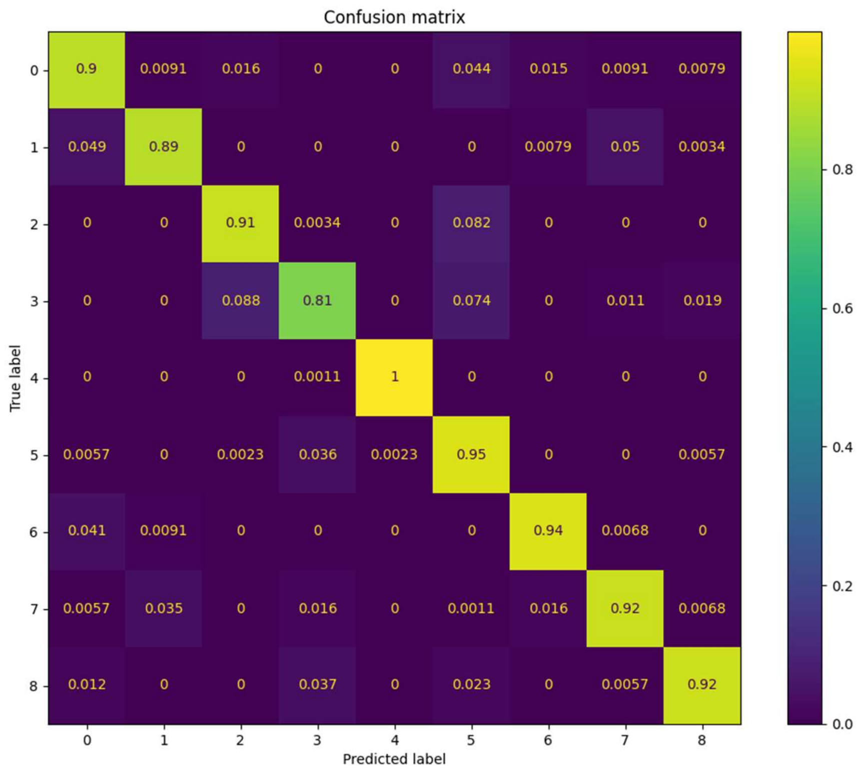


Figure 32. Confusion matrix of a model with 9 clusters over test data, after reassigning the labels. Performance of 88% of success.



True label \ Predicted label	0	1	2	3	4	5	6	7	8
0	0.85	0.013	0.061	0	0.0011	0.027	0.031	0.0097	0.0046
1	0.024	0.87	0	0	0	0	0.061	0.039	0.0086
2	0.0083	0	0.84	0.0075	0	0.15	0	0	0.0011
3	0.00027	0	0.072	0.8	0.00027	0.067	0	0.03	0.035
4	0	0	0	0	1	0	0.00054	0	0
5	0.013	0	0.036	0.047	0	0.9	0	0	0.0035
6	0.044	0.034	0	0	0.00081	0	0.91	0.014	0
7	0.0027	0.055	0	0.032	0	0.00027	0.0067	0.89	0.0083
8	0.027	0.00081	0	0.043	0	0.054	0	0.0081	0.87

Figure 33. Confusion matrix of a model with 9 clusters over train data, after reassigning the labels. Performance of 88% of success.



True label \ Predicted label	0	1	2	3	4	5	6	7	8
0	0.9	0.0091	0.016	0	0	0.044	0.015	0.0091	0.0079
1	0.049	0.89	0	0	0	0	0.0079	0.05	0.0034
2	0	0	0.91	0.0034	0	0.082	0	0	0
3	0	0	0.088	0.81	0	0.074	0	0.011	0.019
4	0	0	0	0.0011	1	0	0	0	0
5	0.0057	0	0.0023	0.036	0.0023	0.95	0	0	0.0057
6	0.041	0.0091	0	0	0	0	0.94	0.0068	0
7	0.0057	0.035	0	0.016	0	0.0011	0.016	0.92	0.0068
8	0.012	0	0	0.037	0	0.023	0	0.0057	0.92

Figure 34. Confusion matrix of a model with 9 clusters over test data, after reassigning the labels two times. Performance of 92% of success.

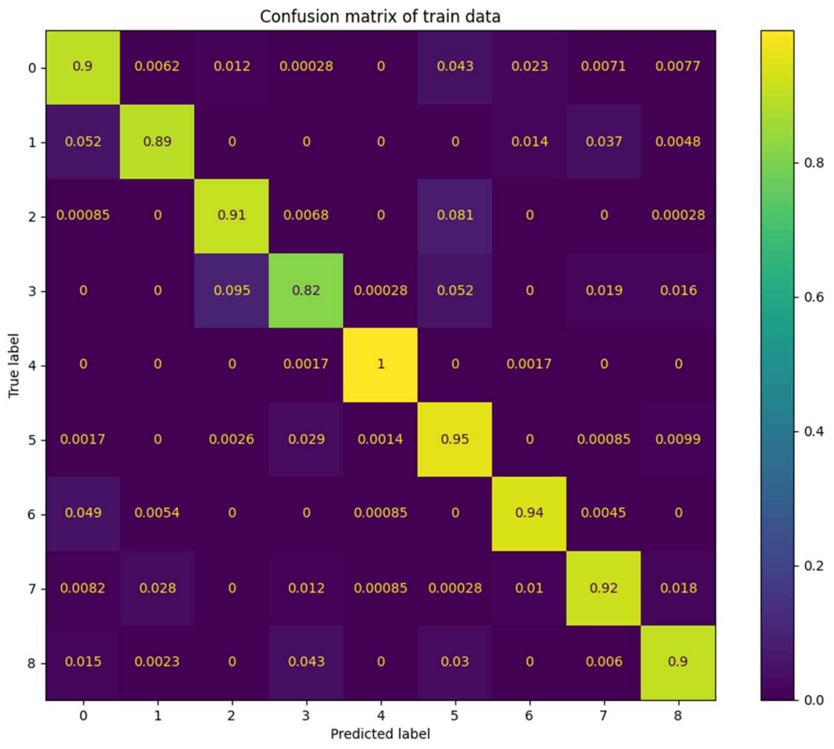


Figure 35. Confusion matrix of a model with 9 clusters over train data, after reassigning the labels two times. Performance of 92% of success.

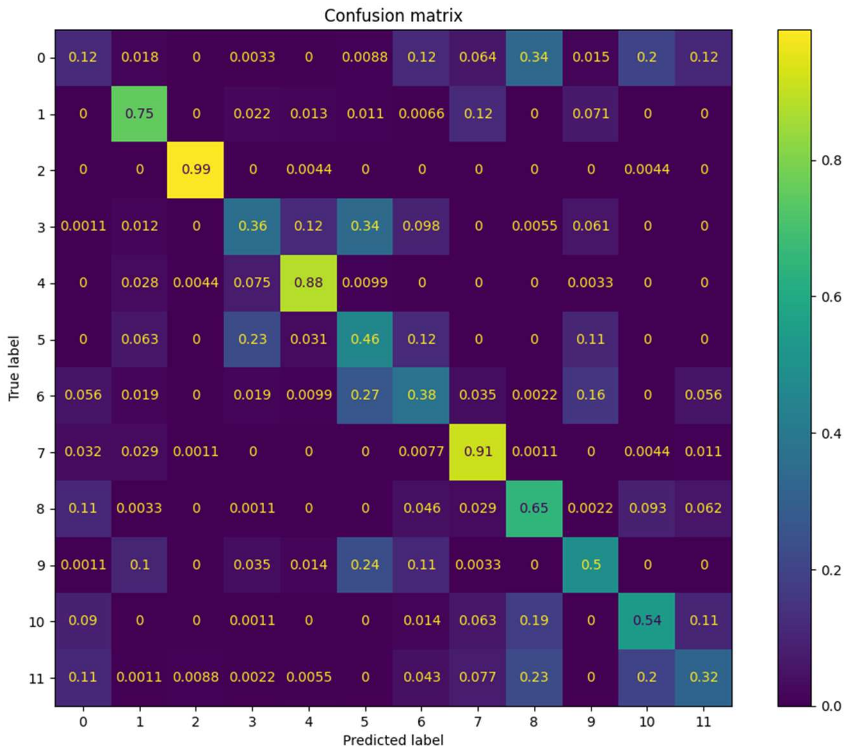


Figure 36. Confusion matrix of a model with 12 clusters over test data. Performance of 57% of success.

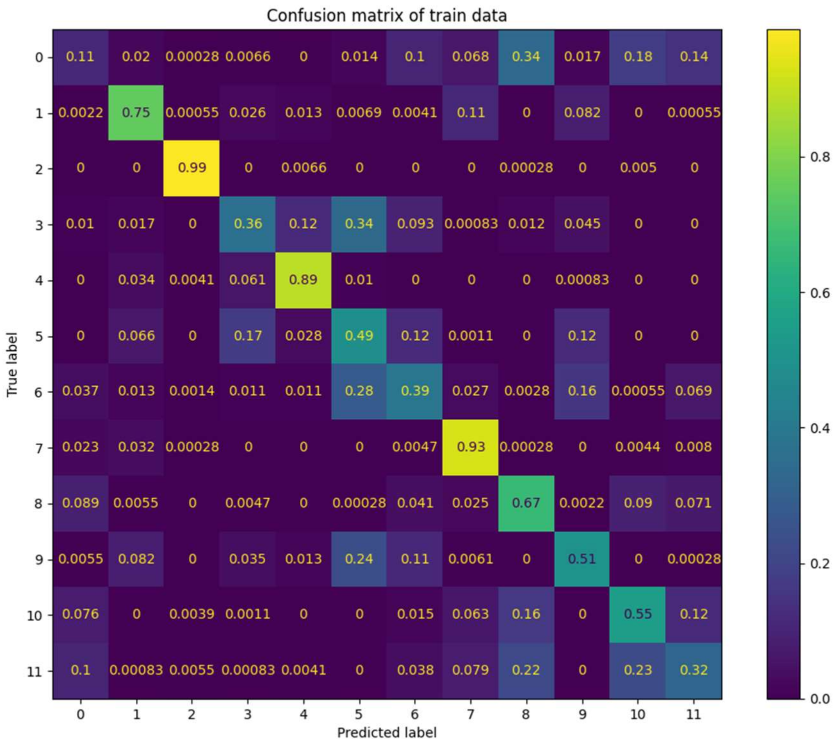


Figure 37. Confusion matrix of a model with 12 clusters over train data. Performance of 58% of success.

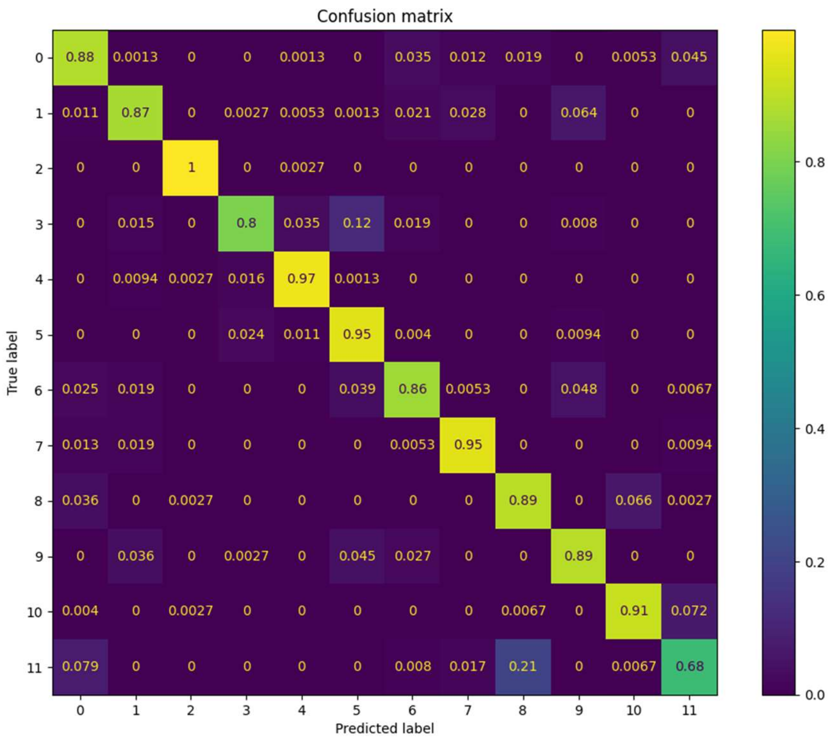


Figure 38. Confusion matrix of a model with 12 clusters over test data, after reassigning the labels. Performance of 89% of success.

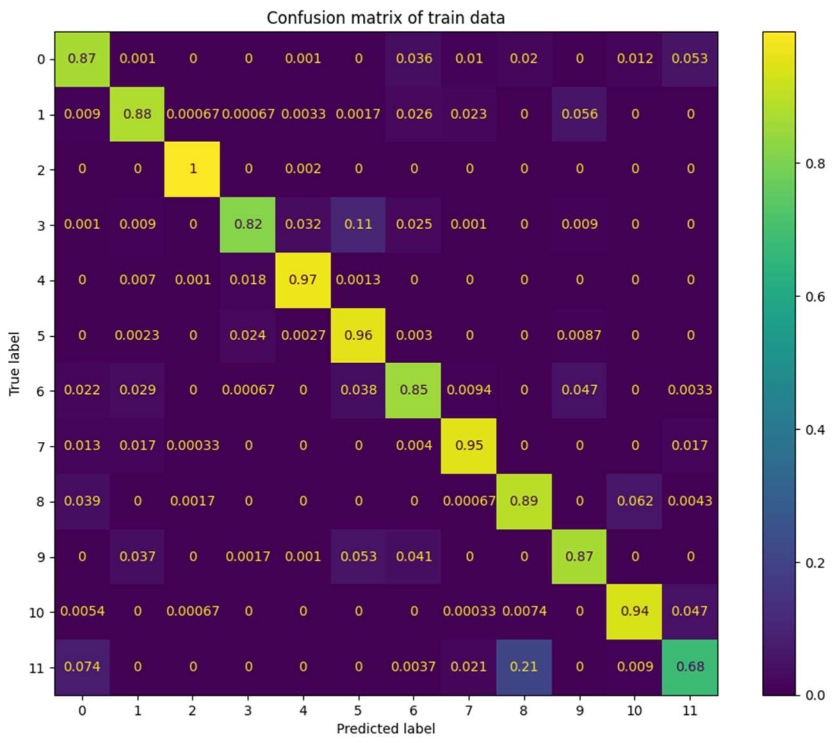


Figure 39. Confusion matrix of a model with 12 clusters over train data, after reassigning the labels. Performance of 89% of success.

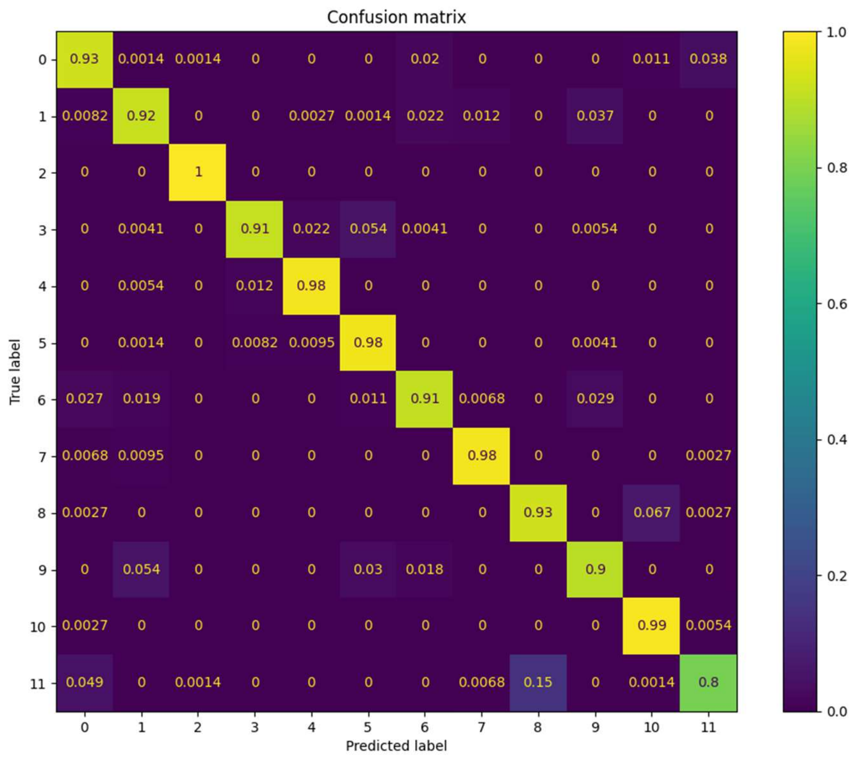
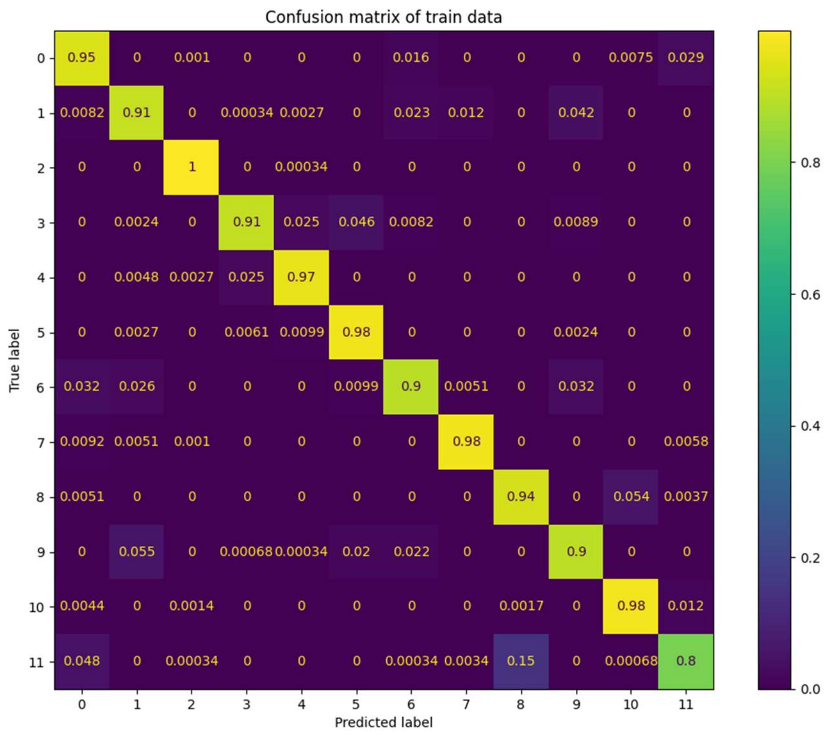
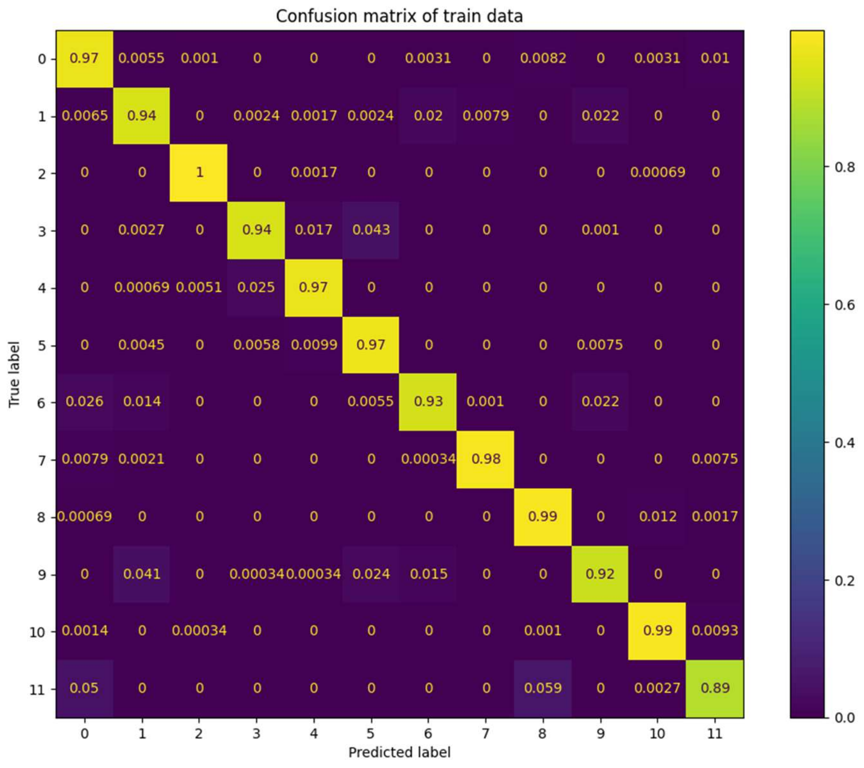


Figure 40. Confusion matrix of a model with 12 clusters over test data, after reassigning the labels two times. Performance of 93% of success.



	0	1	2	3	4	5	6	7	8	9	10	11
0	0.95	0	0.001	0	0	0	0.016	0	0	0	0.0075	0.029
1	0.0082	0.91	0	0.00034	0.0027	0	0.023	0.012	0	0.042	0	0
2	0	0	1	0	0.00034	0	0	0	0	0	0	0
3	0	0.0024	0	0.91	0.025	0.046	0.0082	0	0	0.0089	0	0
4	0	0.0048	0.0027	0.025	0.97	0	0	0	0	0	0	0
5	0	0.0027	0	0.0061	0.0099	0.98	0	0	0	0.0024	0	0
6	0.032	0.026	0	0	0	0.0099	0.9	0.0051	0	0.032	0	0
7	0.0092	0.0051	0.001	0	0	0	0	0.98	0	0	0	0.0058
8	0.0051	0	0	0	0	0	0	0	0.94	0	0.054	0.0037
9	0	0.055	0	0.00068	0.00034	0.02	0.022	0	0	0.9	0	0
10	0.0044	0	0.0014	0	0	0	0	0	0.0017	0	0.98	0.012
11	0.048	0	0.00034	0	0	0	0.00034	0.0034	0.15	0	0.00068	0.8

Figure 41. Confusion matrix of a model with 12 clusters over train data, after reassigning the labels two times. Performance of 93% of success.



	0	1	2	3	4	5	6	7	8	9	10	11
0	0.97	0.0055	0.001	0	0	0	0.0031	0	0.0082	0	0.0031	0.01
1	0.0065	0.94	0	0.0024	0.0017	0.0024	0.02	0.0079	0	0.022	0	0
2	0	0	1	0	0.0017	0	0	0	0	0	0.00069	0
3	0	0.0027	0	0.94	0.017	0.043	0	0	0	0.001	0	0
4	0	0.00069	0.0051	0.025	0.97	0	0	0	0	0	0	0
5	0	0.0045	0	0.0058	0.0099	0.97	0	0	0	0.0075	0	0
6	0.026	0.014	0	0	0	0.0055	0.93	0.001	0	0.022	0	0
7	0.0079	0.0021	0	0	0	0	0.00034	0.98	0	0	0	0.0075
8	0.00069	0	0	0	0	0	0	0	0.99	0	0.012	0.0017
9	0	0.041	0	0.00034	0.00034	0.024	0.015	0	0	0.92	0	0
10	0.0014	0	0.00034	0	0	0	0	0	0.001	0	0.99	0.0093
11	0.05	0	0	0	0	0	0	0	0.059	0	0.0027	0.89

Figure 42. Confusion matrix of a model with 12 clusters over train data, after reassigning the labels three times. Performance of 96% of success.

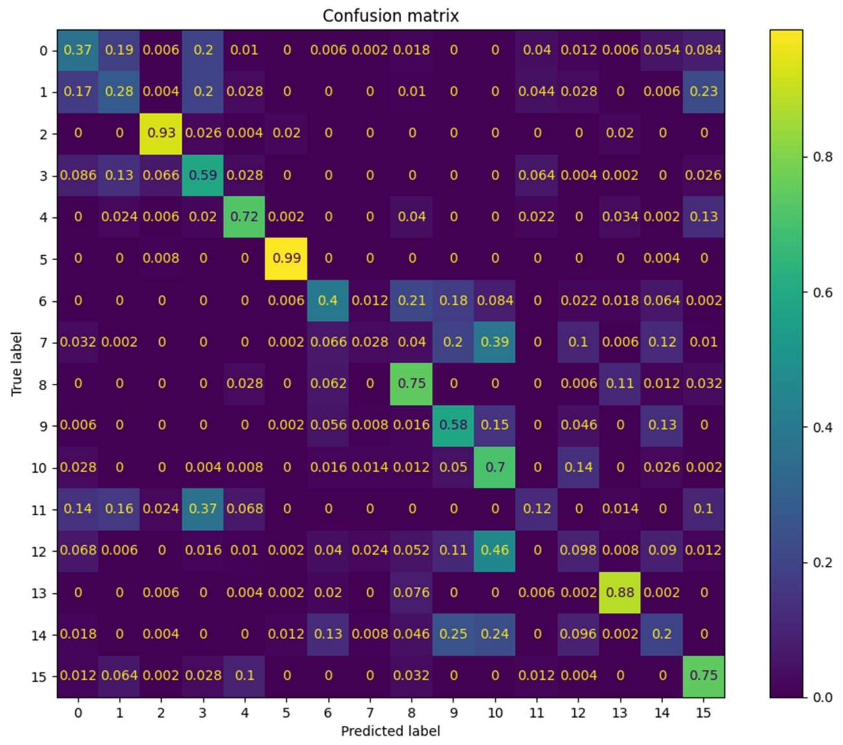


Figure 43. Confusion matrix of a model with 16 clusters over test data. Performance of 52% of success.

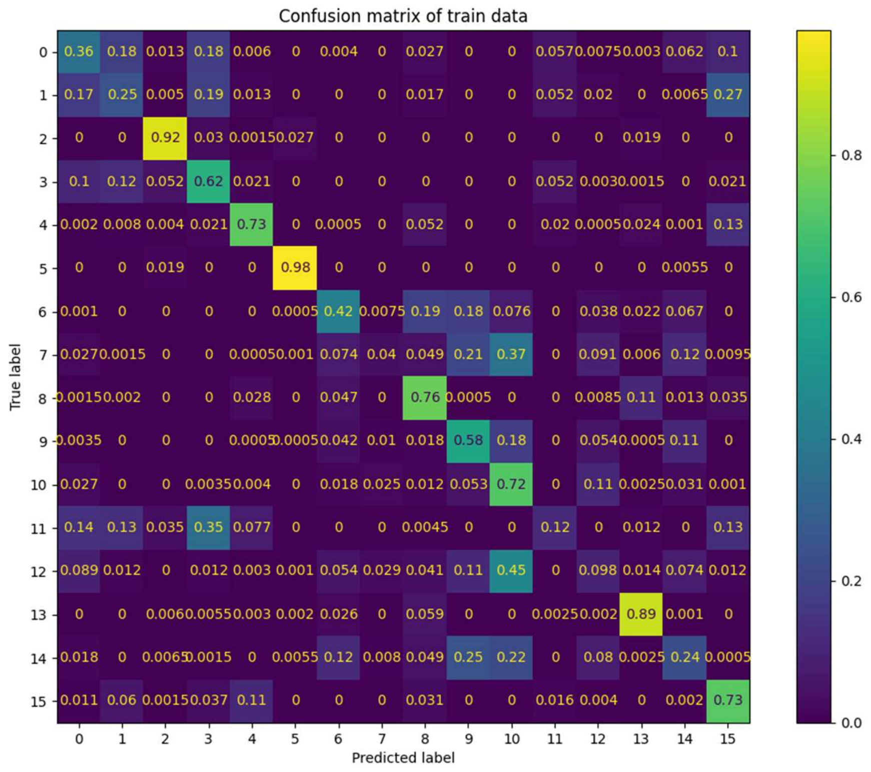


Figure 44. Confusion matrix of a model with 16 clusters over train data. Performance of 53% of success.

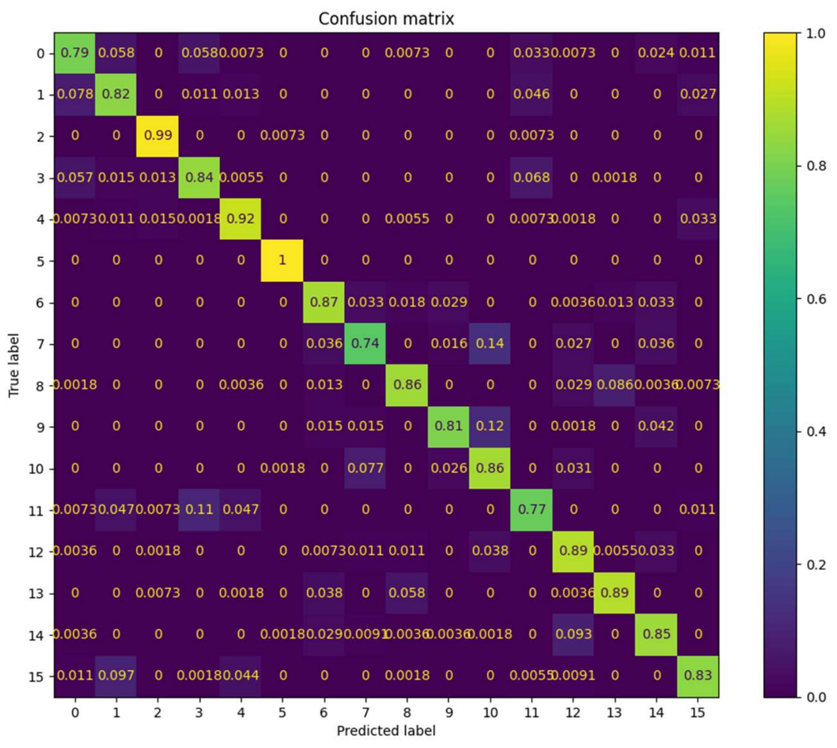


Figure 45. Confusion matrix of a model with 16 clusters over test data, after reassigning the labels. Performance of 86% of success.

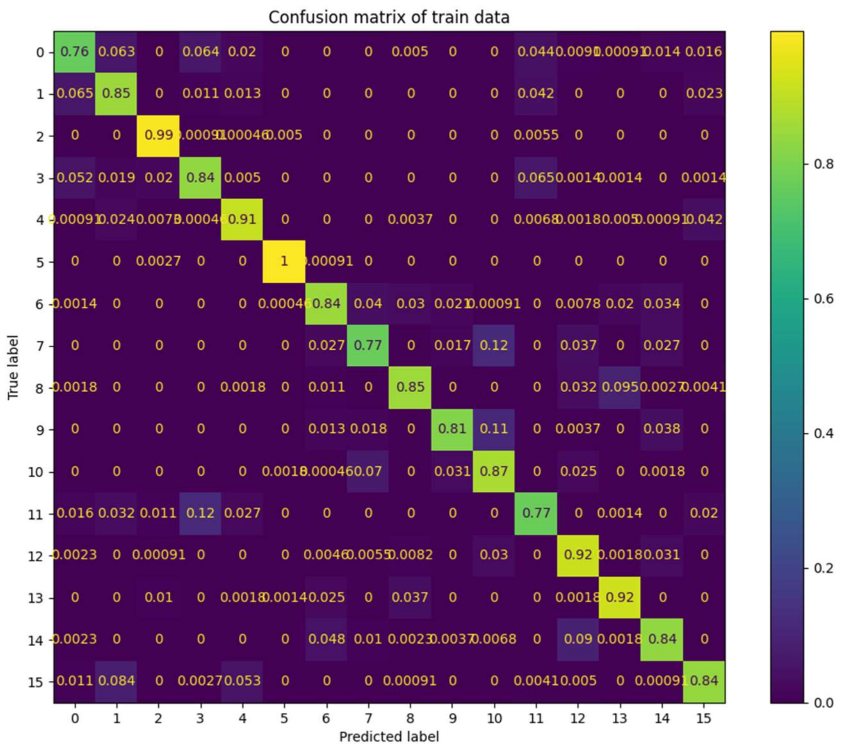


Figure 46. Confusion matrix of a model with 16 clusters over train data, after reassigning the labels. Performance of 86% of success.

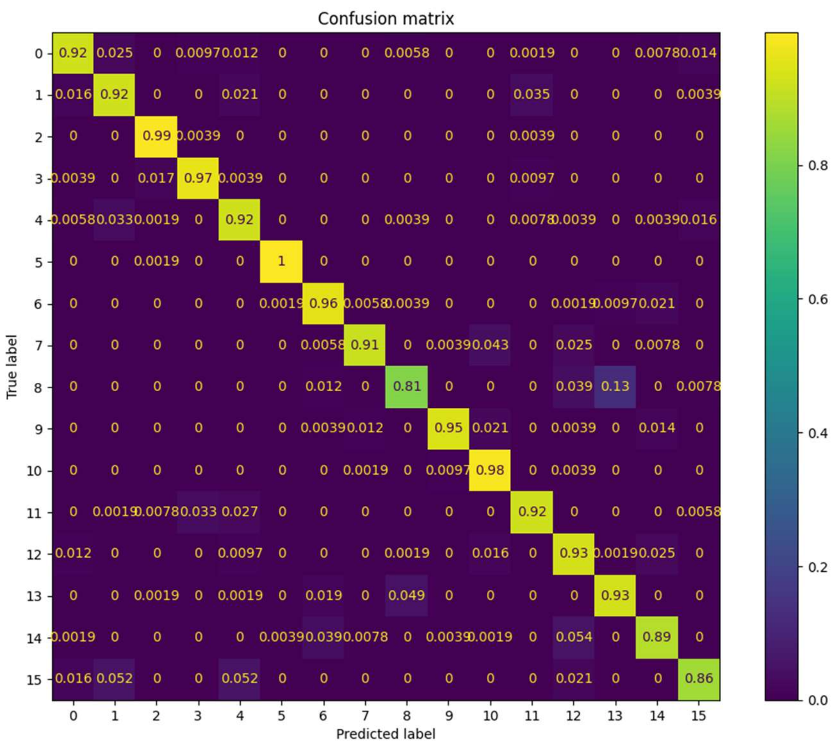


Figure 47. Confusion matrix of a model with 16 clusters over test data, after reassigning the labels two times. Performance of 93% of success.

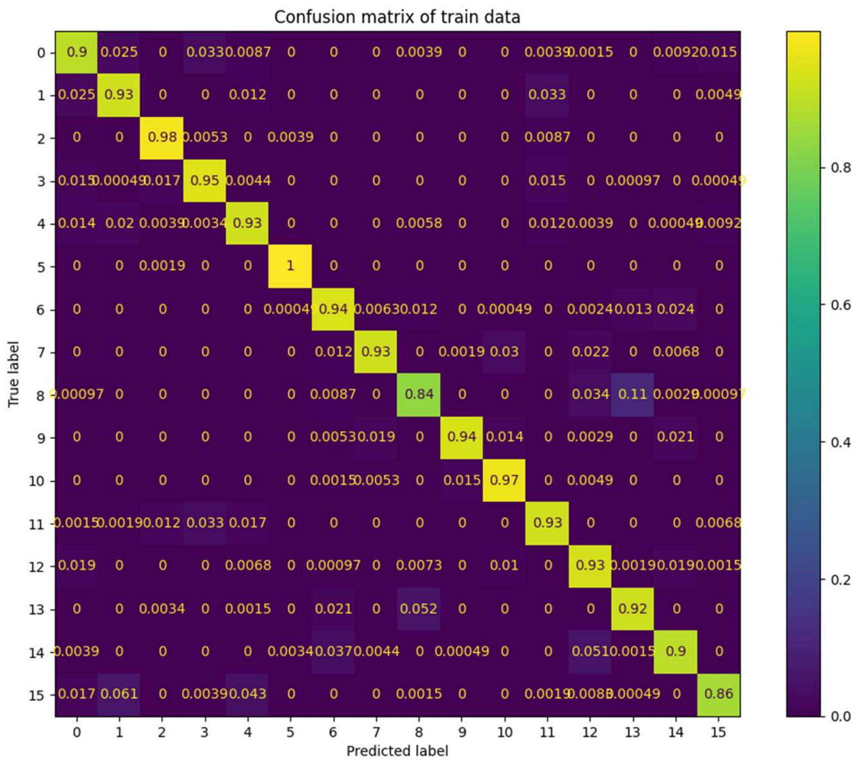


Figure 48. Confusion matrix of a model with 16 clusters over train data, after reassigning the labels two times. Performance of 93% of success.

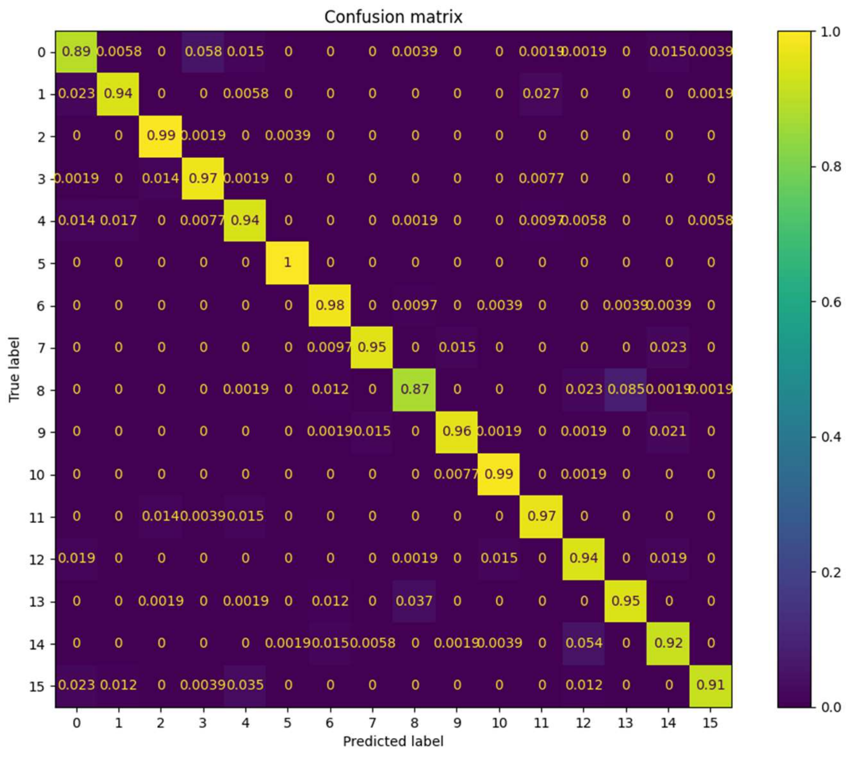


Figure 49. Confusion matrix of a model with 16 clusters over test data, after reassigning the labels three times. Performance of 95% of success.

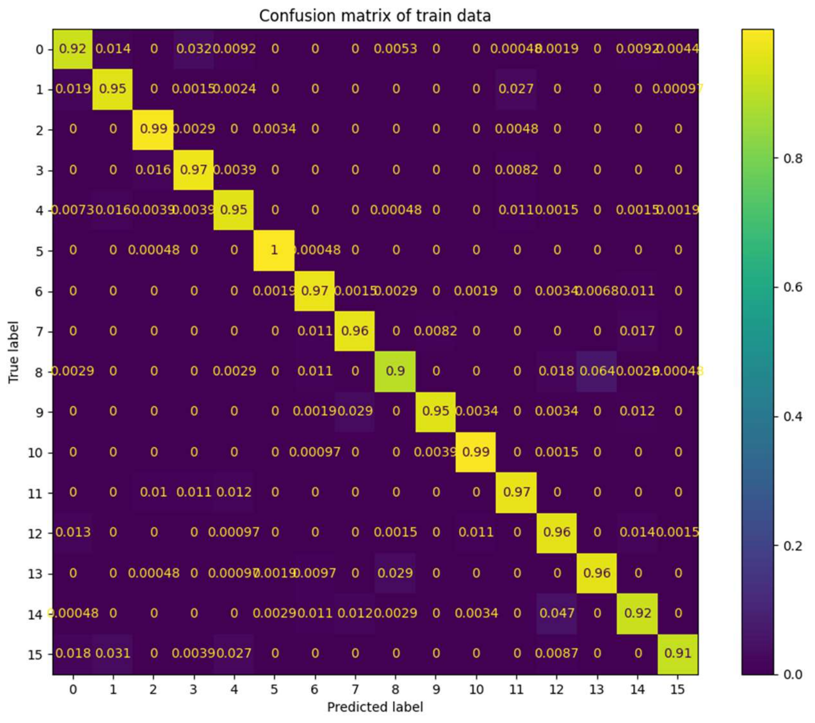


Figure 50. Confusion matrix of a model with 16 clusters over train data, after reassigning the labels three times. Performance of 96% of success.

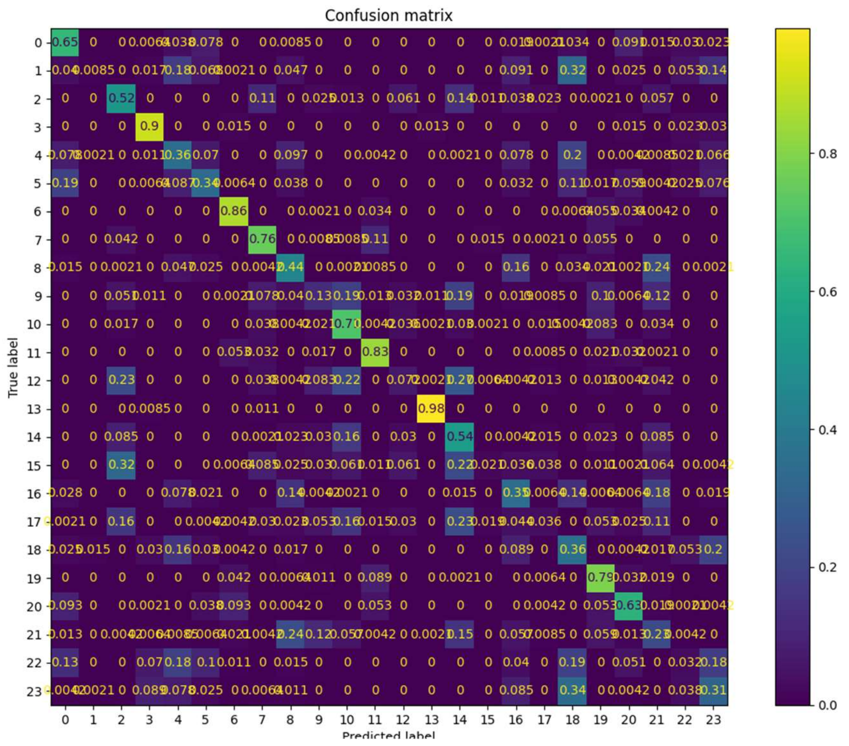


Figure 51. Confusion matrix of a model with 24 clusters over test data. Performance of 45% of success.

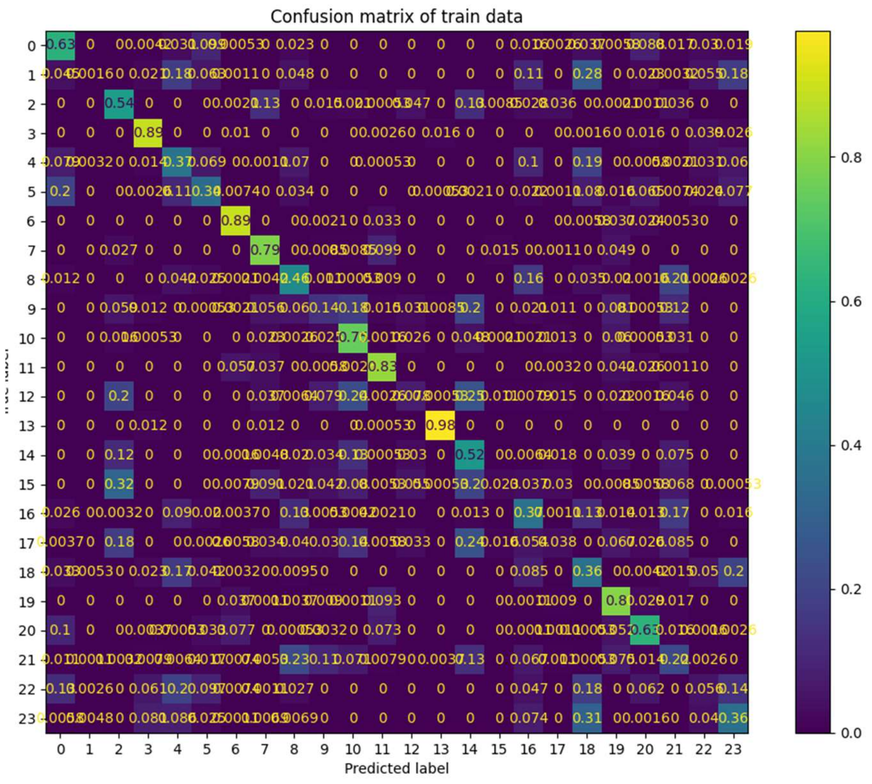


Figure 52. Confusion matrix of a model with 24 clusters over train data. Performance of 46% of success.

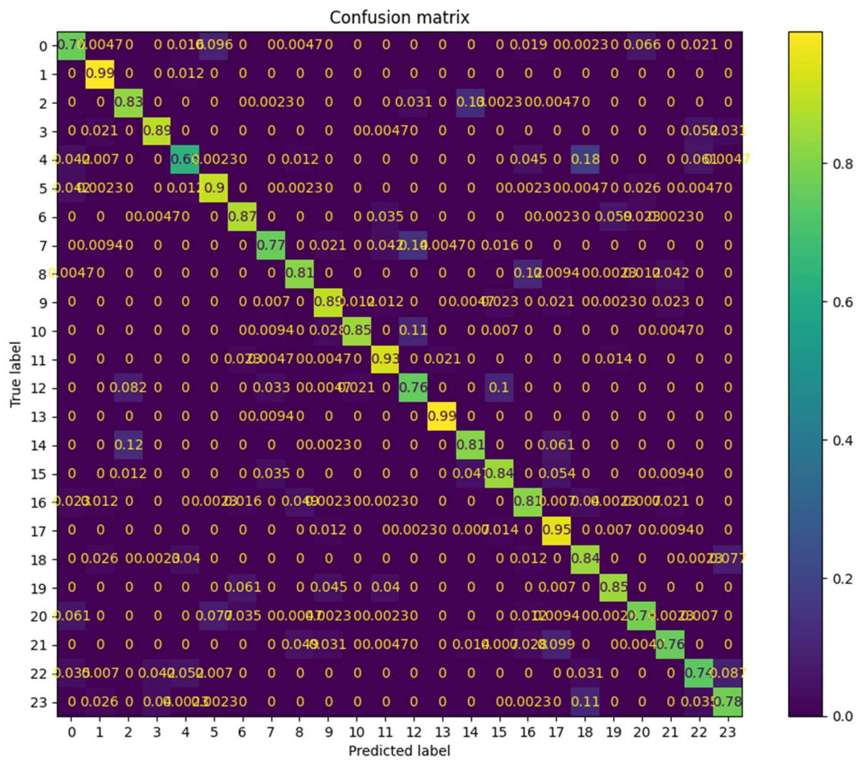


Figure 53. Confusion matrix of a model with 24 clusters over test data, after reassigning the labels. Performance of 84% of success.

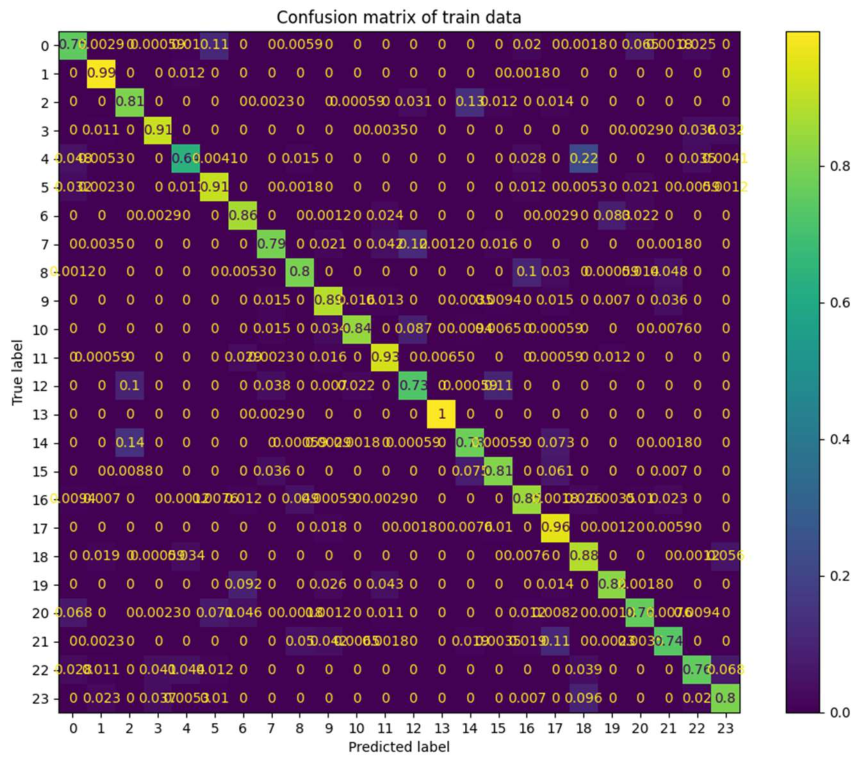


Figure 54. Confusion matrix of a model with 24 clusters over train data, after reassigning the labels. Performance of 83% of success.

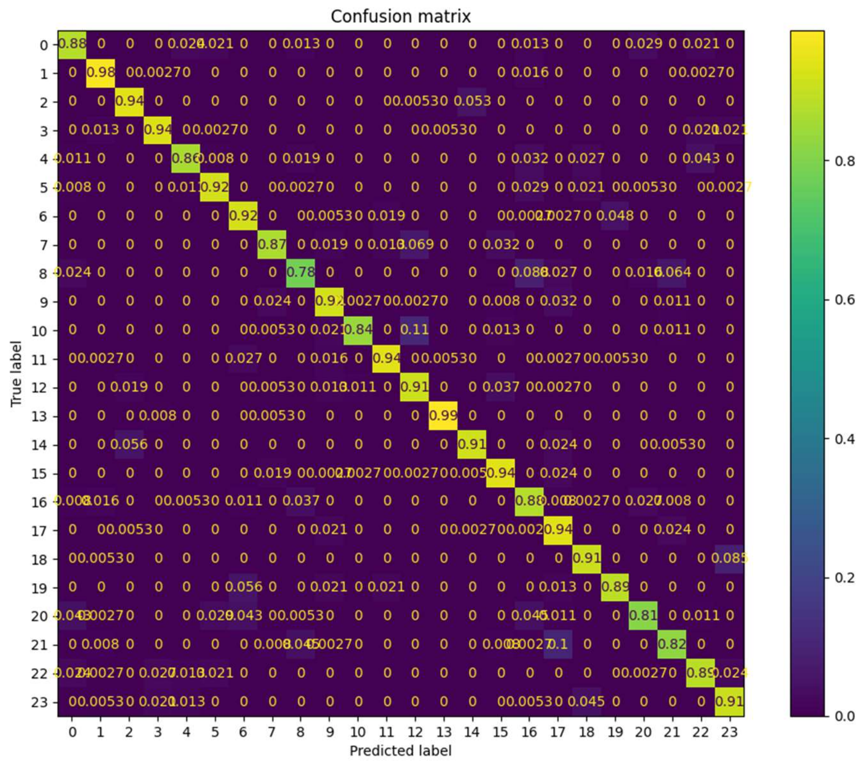


Figure 55. Confusion matrix of a model with 24 clusters over test data, after reassigning the labels two times. Performance of 90% of success.

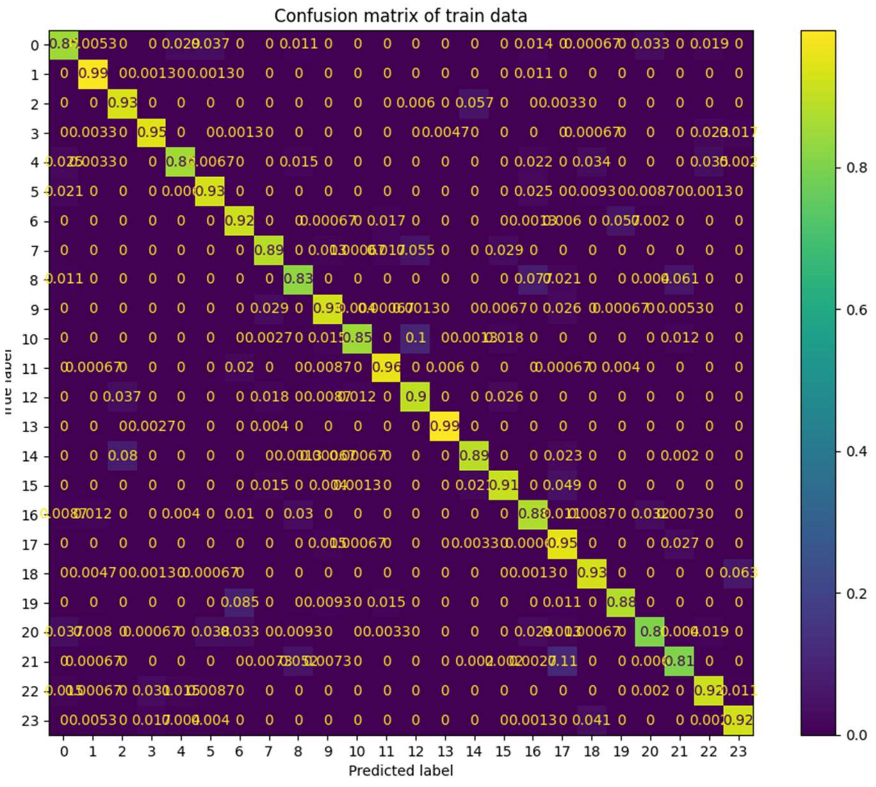


Figure 56. Confusion matrix of a model with 24 clusters over train data, after reassigning the labels two times. Performance of 90% of success.

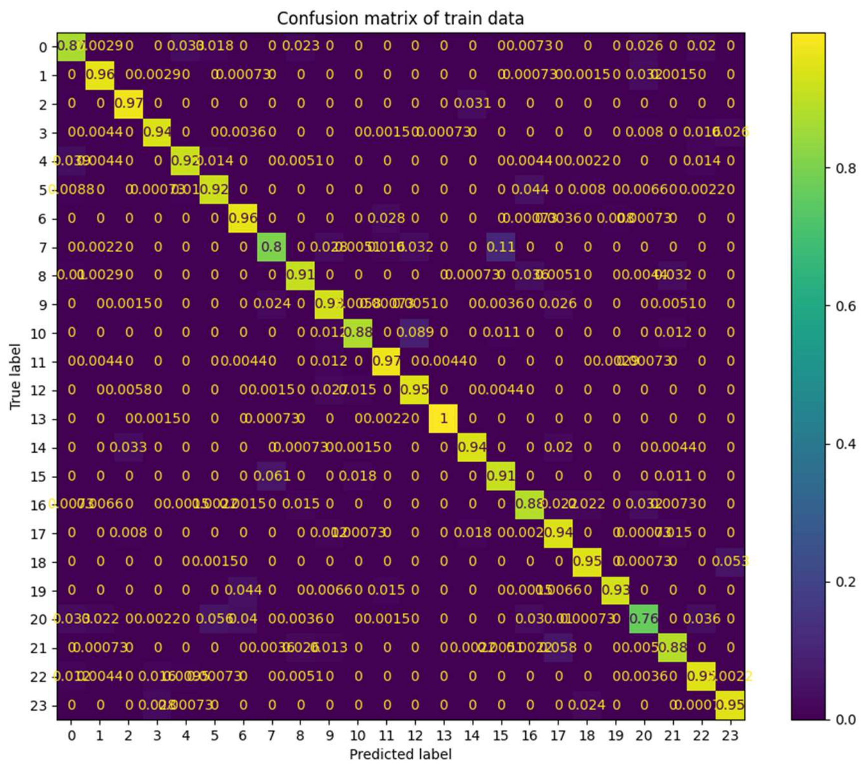


Figure 57. Confusion matrix of a model with 24 clusters over train data, after reassigning the labels three times. Performance of 92% of success.