Tutor/s

Dr. Albert Cortijos i Aragonès
Departament de ciències de materials i Química Física

José Mauricio Regalado Aguilar
Departament de Química Orgànica I Inorgànica

Grau de Química

Treball Final de Grau

Novel Machine Learning Tools for data treatment in STM Break-Junction Technique

Noves eines d'aprenentatge automàtic pel tractament de dades en la Tècnica STM Break-Junction

Sara Cuscó Rovira

June 2024

UNIVERSITAT DE BARCELONA

B:KC Barcelona Knowledge Campus
Campus d'Excel·lència Internacional

El camí del progres no és ni ràpid ni fàcil.

Marie Curie

En primer lloc, voldria expressar el meu agraïment al meu tutor, Albert Cortijos, per guiar-me al llarg del camí, orientar-me i, sobretot, per introduir-me a un món que desconeixia i que m'ha captivat. També agraeixo a Mauricio Regalado la seva disponibilitat i assistència en tot moment. Gràcies als seus consells i ajuda, aquest treball s'ha realitzat èxit i ha complert amb els objectius esperats.

D'altra banda, m'agradaria donar especialment les gràcies a la meva família per estar sempre al meu costat, donar-me suport i per animar-me durant aquests anys. Finalment, vull agrair a la meva parella, Bernat Luco, per ser el meu pilar imprescindible de cada dia i per poder finalitzar aquesta carrera plegats com a químics.

En resum, família, parella, amics, companys i professors gràcies de tot cor per formar part d'aquesta etapa tan especial.

REPORT

IDENTIFICATION AND REFLECTION ON THE SUSTAINABLE DEVELOPMENT GOALS (SDG)

The 2030 Agenda for Sustainable Development, adopted by all UN Member States in 2015, is a global plan for peace and prosperity centered on 17 Sustainable Development Goals. These goals call for joint action to eliminate poverty, improve health and education, reduce inequality, stimulate economic growth, and protect the climate, oceans, and forests. This academic project supports several of these goals, which are Quality Education (SDG 4), Industry, Innovation, and Infrastructure (SDG 9), and Partnerships for the Goals (SDG 17).

Within each section, the following points are supported:

- **SDG 4 (target 4.4):** The incorporation of Machine Learning algorithms to analyze experimental data helps people learn advanced technology and analytical skills. This is important for training staff and giving them valuable work experience.
- **SDG 9 (target 9.5):** The proposal of new analysis methodologies based on Machine Learning represents a significant advancement in scientific and technological research, contributing to innovation in industry and scientific infrastructure.
- **SDG 17 (target 17.6):** The realization and advancement of the project require collaboration between experts in physics, chemistry, computer science, and other disciplines, promoting strategic alliances for the advancement of knowledge and technology.

CONTENTS

1. SUMMARY 3
2. RESUM 5
3. INTRODUCTION 7
3.1. Measurement of single molecule conductance: molecular electronics 7
3.1.1. Break junction techniques 8
3.1.1.1. Mechanically Controllable Break Junctions 8
3.1.1.2. Scanning tunneling Microscope Break-Junction 9
3.1.1.3. Conducting Probe Atomic-Force-Microscope Break-Junction 10
3.1.2. Single-molecule histogram-based analysis 11
3.2. Machine learning 12
3.2.1. Supervised learning 12
3.2.2. Unsupervised learning 13
3.2.2.1. Dimensionality reduction 14
3.2.3. Semi-supervised learning 15
4. OBJECTIVES 16
5. METHODS 16
5.1. Clustering Algorithms 16
5.1.1. Hard clustering algorithms 17
5.1.1.1. K-means 17
5.1.1.2. Agglomerative hierarchical clustering 18
5.1.1.3. Density-Based Spatial Clustering of Applications with Noise 20
5.1.2. Soft clustering algorithms 21
5.1.2.1. Fuzzy C-means 22
5.2. Cluster evaluation methods 23
5.2.1. Elbow method 24
5.2.2. Average silhouette method 24

5.2.3. Davies Bouldin Index 25
5.2.4. Calinski-Harabasz index 25
6. MACHINE LEARNING APPLIED TO STM-BJ DATA 26
6.1. Analyzed data's description 26
6.2. Data processing 27
6.3. Features extraction 29
6.4. Clustering evaluation 32
6.4.1. Dimensionality reduction 35
6.5. Validation 38
7. CONCLUSIONS 41
8. REFERENCES AND NOTES 42
9. ACRONYMS 44
APPENDICES 45
Appendix 1: STM-BJ tool (Fuzzy-c means) 46

1. SUMMARY

The study of electronic properties of individual molecules have become a reality thanks to the use of advanced techniques such as the Break Junction approach. These techniques enable the creation of single-molecule junctions by contacting electrically a molecule between two electrodes with sub-nanometer precision. This study focuses on the Scanning Tunneling Microscope Break-Junction (STM-BJ) technique, which allows the formation of thousands of single-molecule junctions by repeatedly approaching and retracting a tip electrode against a surface electrode, generating current curves that relate conductance to displacement. These curves show steps associated with the formation atomic and molecular contacts.

To better understand these kind of measurements, 1D and 2D histograms are commonly used to accumulate thousands of traces to characterize the single-molecule junction evolutions. Traditionally, these histograms have been built using selection criteria imposed by the researchers, commonly, leading to biased interpretations and limitations in identifying complex patterns. Our study proposes new methodologies to classify single-molecule current traces, unveiling-details that could go unnoticed by human-based analysis. The proposed methodologies are based on Machine Learning algorithms, a sub-domain of Artificial Intelligence, which enables computers to understand and identify patterns in large datasets and make predictions. Specifically, *unsupervised learning* has been used, which classifies unlabeled data into groups based on predefined variables. Various types of algorithms have been tested, ranging from hard clustering to the more sophisticated soft clustering.

In this project we developed a Python-based Machine Learning framework aimed to revolutionize the interpretation of single-molecule junction datasets. It is expected that this advancement will improve the rationalization of single-molecule phenomenology, facilitating a more accurate understanding of experimental data.

Keywords: Scanning Tunneling Microscope Break-Junction, atomic junctions, Machine Learning, unsupervised learning, hard clustering, soft clustering

2. RESUM

L'estudi de les propietats electròniques de molècules individuals s'ha convertit en una realitat gràcies a l'ús de tècniques avançades. Aquestes tècniques, anomenades *Break-Junction*, permeten la creació d'unions unimoleculars posant en contacte una molècula individual entre dos elèctrodes amb una precisió sub-nanomètrica. Aquest estudi se centra en la tècnica de l'*Scanning Tunneling Microscopy Break-Junction* (STM-BJ), que permet la formació de milers d'unions unimoleculars apropant i retraient repetidament dos elèctrodes (punta contra superfície), generant corbes de corrent que relacionen la conductància amb el desplaçament.

Per comprendre millor aquest tipus de mesures, s'utilitzen comunament histogrames 1D i 2D els quals acumulen milers de corbes per caracteritzar les evolucions d'aquestes unions. Tradicionalment, aquests histogrames s'han construït utilitzant criteris de selecció decidits pels investigadors, cosa que sovint pot conduir a interpretacions esbiaixades i molt limitades en la identificació de patrons complexos en la racionalització de resultats. Ell nostre estudi proposa noves metodologies per classificar corbes de corrent d'unions moleculars, proporcionant uns detalls que podrien passar desapercebuts per l'anàlisi humà. Aquestes metodologies es basen en algoritmes d'*Aprenentatge Automàtic*, un subdomini de la Intel·ligència Artificial, que permeten als ordinadors entendre i identificar patrons en grans conjunts de dades i fer prediccions. Així doncs, s'ha utilitzat l'*Aprenentatge No Supervisat*, que organitza dades no classificades en grups basats en variables pre-definides. Per tal d'obtenir el millor resultat, s'han provat diversos tipus d'algoritmes, amb diferents nivells de classificació, de mes a menys flexibles.

En aquest projecte, hem desenvolupat un programa d'Aprenentatge Automàtic basat en llenguatge *Python* amb l'objectiu de revolucionar la interpretació dels conjunts de dades d'unions unimoleculars. El nostre avanç millorarà la racionalització de la fenomenologia unimolecular, facilitant una comprensió més precisa de les dades experimentals.

Paraules clau: Microscopi de Túnel d'Escaneig *Break-Junction*, Unions atòmiques, Aprenentatge Automàtic, Aprenentatge no-supervisat, Agrupament dur, Agrupament suau

3. INTRODUCTION

3.1. MEASUREMENT OF SINGLE-MOLECULE CONDUCTANCE: MOLECULAR ELECTRONICS

In recent years, a new branch has emerged in nanotechnology: the ability to measure electronic properties of molecules in their characteristic length scale, known as molecular electronics. A key concept within this field is the single-molecule junction, which its prediction dates back to 1974, when Aviram and Ratner proposed a hypothetical molecular diode based on a simulated asymmetric current-voltage (I-V) behavior obtained from a model molecule.[1] Since then, there have been significant advances leading to the development of several devices and methods that can achieve this feat, providing a unique opportunity to understand charge transport-phenomenon, undoubtedly crucial for physical, chemical and biological processes. [2]

The conductance of a single-molecule can be determined by accurately positioning a target molecule between two electrodes, creating a single-molecule junction. Additionally, three requirements must be taken into account: (a) provide a signature to confirm that the measured conductance comes from a individual molecule, not a group of molecules in the interelectrode nanogap, (b) ensure that the molecule is strongly attached to the two terminal electrodes, and (c) perform the measurement in a well-defined environment to prevent possibles alterations from factors such as solvent molecules, pH or ions. [3]

The type of material used for the electrodes in conductance determination, typically metal, is a crucial component in the formation of a single-molecule junction. These are highly relevant in charge transport due to their electrical conductivity, which allows charge carriers to be transmitted through the single-molecule junction, and the ability to form such junctions stable enough to ensure that the molecule remains rigidly attached to the electrodes during the measurement.[4]

The most used metal is gold (Au), a noble metal, due to its high probability of junction formation, chemical stability, and electrical conductivity.[5] Au's inert nature ensures consistent and reliable results over extended experimental durations, as its lack of chemical reactivity prevents undesired changes during the measurements.[5] In addition, it enables the use of the

quantum of conductance (1 G_o), with a value of equivalent to 77.5 μS, as an internal conductivity reference.[6] However, other metals can also be used as electrodes, including Cu, Pt and Si.[7]

3.1.1. Break junction techniques

The techniques used to measure individual atoms or molecules conductance by creating a electrode-electrode nanogaps are called Break-Junction (BJ) techniques.[1] BJ techniques rely in the formation of electrode-electrode nanogaps with a sub-nanometer precision, allowing the atomic and molecular trapping.[3] There are three major approaches for the BJ technique that will be described below: (1) mechanically controllable break junctions (MCBJ); (2) scanning tunneling microscope break junction (STM-BJ); and (3) conducting probe atomic-force-microscope break junction (CP-AFM-BJ).[6]

3.1.1.1. Mechanically Controllable Break Junction

In 1985, Moreland and Ekin developed a method for mechanically forming tunnelling junctions[1] which evolved towards the existing MCBJ technique.[8] They used a thin wire of Nb-Sn filament mounted on a flexible glass bead.[8] Using this configuration, they measured electron tunnelling characteristics of superconductors. In 1992, Muller et al. pioneered the technique to create metallic quantum point contacts, to prove the conductance quantization via a narrow conductor of a length of the order of electron's elastic mean free path.[3]

MCBJ's three-point adjustable bending mechanism, as shown in Fig. 1a, uses a pushing rod that applies an upward force causing the progressive bending of the substrate, which results in the continuous stretching of the metallic wire. As the metallic wire elongates, the cross-section at the central region decreases until it fractures, and then, two atomic-level metal electrodes with clean surfaces are simultaneously formed. Conversely, when the substrate relaxes, the inevitable contraction of the metallic wire causes the two electrodes to revert to re-forming a metallic wire. The nanogap between the two opposing electrodes can be precisely regulated during the bending and relaxation process (Fig. 1b) mediated by a piezoelectric transducer (PZT) coupled to the pushing rod.[9]

The MCBJ sample can be prepared manually by attaching a metal wire to a flexible substrate using epoxy glue and notching the central part of the wire to create a constriction point.[1] More recently, an alternative method to create the nanobridge is based on microfabrication techniques, to define suspended metallic bridges. [8]

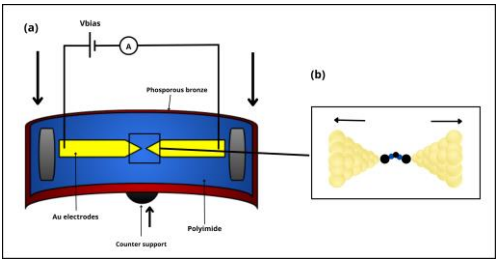

Figure 1. Schematic representation of the MCBJ technique.

3.1.1.2. Scanning Tunneling Microscope Break-Junction

The scanning tunneling microscope (STM) was developed by Gerd Binning and Heinrich Rohrer in 1981 at the IBM Zurich Research Laboratory. STM was the first device capable to provide the first three-dimensional (3D) images of solid conductive and semiconductive surfaces. Nowadays, the STM is widely used to study the electronic structure and properties of metallic and semimetallic surfaces at atomic level and has further evolved to incorporate various techniques, one of which is the STM-based break junction (STM-BJ) technique.[10] The STM-BJ technique was created by Profs. Xu and Tao in 2003. STM-BJ enables the creation of thousands of single-molecule junctions by repeatedly moving a STM tip electrode into and out of contacts with the substrate electrode. In this technique the molecules of interest can be solved in (i) the working solution surrounding the tip and the substrate electrodes or (ii) adsorbed to the surface electrodes.[11] This technique typically (but not necessarily) uses a Au substrate and Au STM tip to form single-molecule junctions.[12]

STM-BJ process can be divided in four steps (see Fig. 2 sequence): (a) A PZT drives the Au tip towards the Au substrate until it contacts the molecule, which causes the formation of atomic junction. [8] (b) The tip is pulled away from the substrate, forming an Au-Au atomic junction formed by few metallic atoms. (c) As the tip continues retracting, the metallic atomic junction progressively stretches, disconnecting metallic atoms sequentially, until eventually it breaks. The formed nanogap enables the trapping and electrical contact of the target molecules in the interelectrode nanogap, i.e. the single-molecule junction,[1] until spontaneously (d) such single-molecule junction breaks. The entire cycle is repeated thousands of times to gather enough data for subsequent statistical analysis. [3] During this cycle a fixed bias potential is set between the tip and the substrate, and the tunneling current is monitored to create conductance-displacement curves.[13]

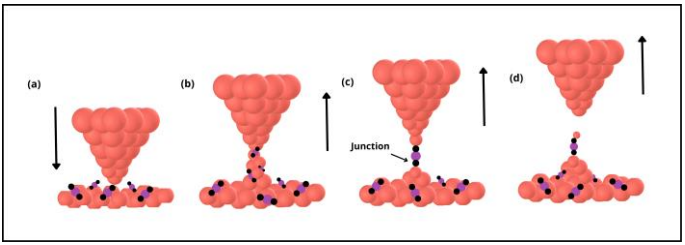

Figure 2. Schematic representation of the STM-BJ technique working principle.

Typically, the conductance–displacement curves show a short step feature at 1 G_0 corresponding to the quantum conductance of the atomic metal junction, below such value the step feature that appears at a later stage can be assigned to the formation of single-molecule junctions.[11]

As an added value, since the STM-BJ technique is based on STM, it can be employed to imaging the substrate surface before performing the electron transport measurement,[13] enabling us to place the tip on an atomically flat area or move it laterally to a fresh are of the substrate during the measurement.[3]

3.1.1.3. Conducting Probe Atomic-Force-Microscope Break-Junction

Xu et al. successfully measured the electronic properties of a junction using an Atomic Force Microscope (AFM). In the CP-AFM-BJ technique, a single-molecule junction is created by placing a conducting AFM tip in contact with a metal-supported molecular film, such as a self-assembled monolayer (SAM) on Au, as illustrated in Fig. 3.[14] The AFM's normal force feedback circuit controls the mechanical load on the microcontact while the current-voltage (I-V) characteristics are recorded. Each abrupt conductance decrease is accompanied by an abrupt decrease in the monitored force, corresponding to the breakdown of a molecule from contacting the electrodes.[3]

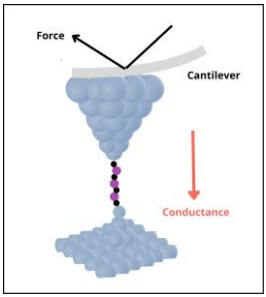

Figure 3. Schematic representation of the CP-AMF-BJ technique.

This technique's ability to adjust the load on the microcontact is unique, allowing researchers to study the relationship between the mechanical deformation of molecules and their transport properties. Moreover, the load-dependent contact area between the tip and the SAM in these junctions is small (approximately 10 nm²), meaning the junction properties reflect transport through a limited number of molecules.[14]

3.1.2. Single-molecule histogram-based analysis

The techniques described before represented different ways to measure atomic and single-molecule junctions. When data is analyzed from these junctions' current traces, a single trace is insufficient to determine the sample's conductance in a statistically reliable way, requiring a large volume of accumulated data. Thus, in this context, histogram-based analysis becomes the most suitable statistical method for analysis.[15]

Conductance histograms are typically created by accumulating junctions' conductance traces during the contact rupture process, and the mean conductance is then determined from the peak positions of the histogram. The most used histograms in STM-BJ are 1D and 2D histograms. The 1D histograms represent the frequency of occurrence of specific conductance values, providing a straightforward statistical distribution of conductance measurements (Fig. 4a).[1]

On the other hand, 2D histograms, display conductance on the vertical axis and elongation distance on the horizontal axis, which allows for a visualization of the statistical behavior of the conductance as a function of the time or tip displacement (Fig. 4b). These 2D histograms are a valuable statistical tool for analyzing junction evolution parameters related to junction stability, including the length of the plateau of the junction, retention time, and snapback distance—the distance traveled by the electrode immediately after its collapse.[15]

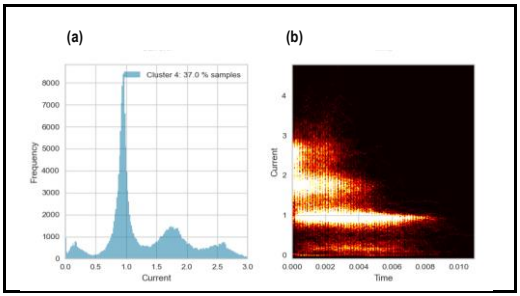

Figure 4. Examples of single-molecule 1D and 2D histograms.

3.2. MACHINE LEARNING

As previous mentioned, single-molecule measurements are typically rationalized using histograms-based analysis. However, such histograms are constructed based on the discrimination criteria imposed by the researcher, which often leads to biased interpretations and limitations to identify complex patterns. By implementing new tools such as Machine Learning (ML) based on clustering with multidimensional features, we can enhance the accuracy of discrimination and reveal details that may be missed by simple human-based analysis. [15]

ML is a sub-domain of artificial intelligence (AI) that gives computers the ability to learn without being explicitly programmed.[16] ML goal is usually to understand the structure of the data and to match that data to models that can be understood and used by humans. Although ML and AI are frequently paired, they represent distinct concepts. AI encompasses a wide range of capabilities, including decision-making, skill acquisition, and problem-solving. Conversely, ML serves as a subset of AI, empowering intelligent systems to autonomously acquire new knowledge from data.

ML employs a series of algorithms to learn from a dataset. These algorithms can be classified based on their purpose and the main categories they follow. The classification of ML algorithms is as follows: (i) Supervised learning, (ii) Unsupervised learning, and (iii) Semi-supervised learning.[17]

3.2.1. Supervised learning

Supervised learning is a category of ML that uses labelled datasets to train algorithms to predict outcomes and recognize patterns. In this model the predictors (input variables) and outputs (response or target variables) values are already known, and the algorithm learns the mapping function from input to output.[18] Mathematically, given a set of N training examples of the form $\{(x_1,y_1),...,(x_N, y_N)\}$ such that x is the feature vector and y its label, the learning algorithm tries to find the best mapping functions, f, such that Y= f(X), where Y is the output space and X is the input space.(Fig. 5) [19]

Supervised learning problems can be split further into classification or regression problems according to type of output variables. When the output variable represents a numerical value, regardless of whether it's discrete or continuous, the task is considered a regression problem. Conversely, if the output variable represents categories or classes, the problem falls into the classification category.

Otherwise, we have a type of models that combines the previous ones when data contains both continuous and categorical variables. Classification and regression algorithms combine the learning function to predict continuous values (as in regression) with classification to assign elements to specific categories.[20]

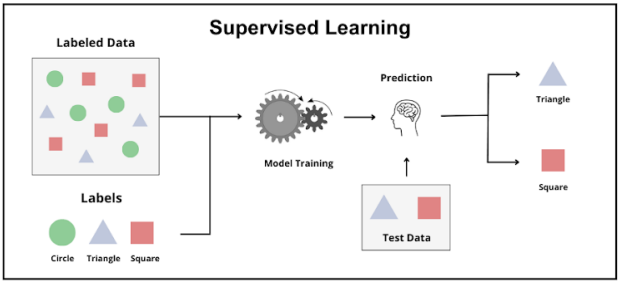

Figure 5. Schematic representation of supervised learning procedure.

3.2.2. Unsupervised learning and Clustering

In unsupervised learning, the model's task is to identify patterns within the training dataset, and subsequently react based on the presence or absence of these patterns in new data introduced to the model. Unlike supervised algorithms, these types of algorithms utilize an unlabeled training set. Implying that, these data points do not have a predefined expected output, and the system must be able to identify or determine this output. To enable the model to learn to discern hidden patterns within the dataset, it is necessary to expose it to large volumes of data. (Fig. 6) [16] Mathematically, given a set of N training without labels dataset $\{(x_1,y_?),...,(x_N, y_?)\}$, the learning algorithm return a model with k (number of clusters) and centroids—points representing the center of each cluster— for each cluster.[19]

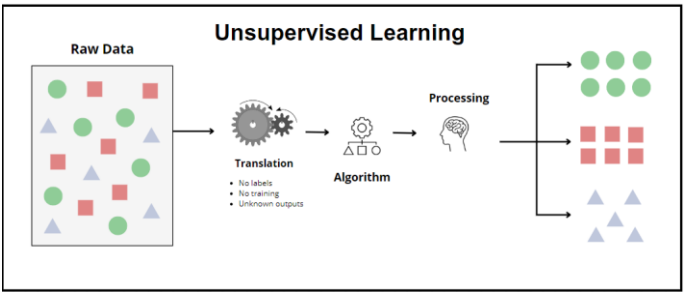

Figure 6. Schematic representation of unsupervised learning procedure.

Clustering is a protocol which enables us partitioning the dataset into groups, called clusters in an unsupervised manner.[21] The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different. The resulting structured data is termed as *data-concept*. [22] Clustering algorithms are used to process raw, unclassified objects (i.e. data) into groups represented by structures or patterns in the information.

There are two primary types of clustering: hard and soft clustering. Hard clustering involves grouping objects where each object can only belong to one group. Commonly used methods for hard clustering include k-means, hierarchical clustering, and DBSCAN.[23] On the other hand, soft clustering involves grouping data items in a way that allows an item to exist in multiple clusters. The most typical method for soft clustering is Fuzzy C-means. [23]

3.2.2.1. Dimensionality reduction

Clustering, as discussed above, provides a structured approach to partitioning datasets into meaningful groups. However, as datasets grow in complexity and dimensionality, understanding and interpreting these clusters can become challenging. This is where dimensionality reduction techniques step into offers valuable insights.

Dimensionality reduction is a methodology of unsupervised learning that reduces the number of data inputs to a manageable size while also preserving the integrity of the dataset as much as possible.[24] There are a few different methods that can be used, such as *Principal Component Analysis* (PCA), a linear representation method, or *t-distributed Stochastic Neighbor Embedding* (t-SNE) and *Uniform Manifold Approximation and Projection* (UMAP), non-lineal representation methods.

- PCA is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. In this the dimension of the data is reduced to make the computations faster and easier. It is used to explain the variance-covariance structure of a set of variables through linear combinations.[16]
- t-SNE is a nonlinear dimensionality reduction technique used to represent high-dimensional data in two or three dimensions. This method constructs a probability distribution over pairs of high-dimensional objects and then tries to find a similar distribution in a lower-dimensional space.[24] It minimizes the *Kullback-Leibler* (KL) divergence between the two distributions, which is a measure of the difference

between them. The KL divergence quantifies how much information is lost when one probability distribution is used to approximate another. t-SNE excels at preserving the local structure of the data while also uncovering global patterns, such as clusters at multiple scales.[25]

- UMAP is an innovative manifold learning technique for dimensionality reduction, grounded in a theoretical framework of Riemannian geometry—a branch of mathematics concerned with curved surfaces and spaces—and algebraic topology. This results in a practical and scalable algorithm applicable to real-world data. UMAP competes with t-SNE in visualization quality, arguably preserving more of the global structure while offering superior runtime performance.[26]

3.2.2. Semi-supervised learning

We identified two main types of ML, supervised and unsupervised learning. However, there exists a third category called semi-supervised learning that integrates features from both categories.[15] Semi-supervised learning is a branch of ML which can train models with small amount of labeled data and effectively label unknown data (Fig.7). These methods are especially relevant in situations where obtaining enough labeled data is significantly difficult or computationally demanding, but large amounts of unlabeled data are relatively easy to acquire. Mathematically, given a set of N training examples with partial labels dataset as an input $\{(x_1,y_2),...,(x_N, y_P)\}$, the learning algorithm return a training model with probabilistic approach as an output.[19]

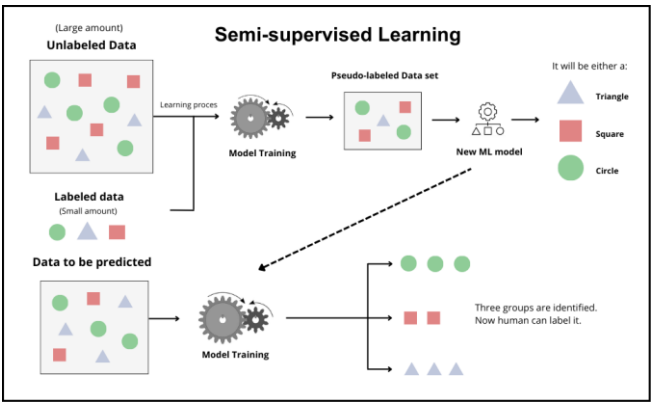

Figure 7. Schematic representation of semi-supervised learning procedure.

4. OBJECTIVES

The aim of this work is to create and implement a new tool using Python that utilizes ML, specifically focusing on unsupervised learning techniques, to organize and classify data obtained from STM-BJ measurements. This tool will employ advanced algorithms capable of learning and adapting autonomously, enabling a more detailed and comprehensive interpretation of the data with an objective criterion.

The focus on unsupervised learning will enable the tool to identify patterns and group the data without the need for predefined labels, making the classification process unbiased, more flexible and adaptable to new data. This method aims to overcome the limitations of manual classification, which is often influenced by individual biases and may overlook subtle details.

Several algorithmic methods and techniques will be tested to achieve the best analysis results, demonstrating the potential of ML to transform how complex datasets are interpreted and understood. By implementing and evaluating these methods, we aim to provide a robust tool that enhances the accuracy and depth of data analysis, ultimately contributing to the field of molecular electronics.

5. METHODS

5.1. CLUSTERING ALGORITHMS

This section provides a concise overview of the clustering algorithms utilized in this study, which will be later evaluated to ensure optimal data analysis. Cluster algorithms are divided into two categories already presented in previous sections: hard clustering algorithms and soft clustering algorithms. With a focus on unsupervised learning and the objective of determining the optimal number of clusters (k), many algorithms will be combined with cluster evaluation methods (detailed in the following section) to refine the data selection. Each method will be presented with a brief description, followed by an explanation of its associated procedure.

5.1.1. Hard clustering algorithms

Clustering algorithms can be categorized into two groups: hard and soft clustering. This study primarily concentrates on hard clustering, where data exclusively belong to a single cluster. Below, we provide explanations of the three algorithms employed in this work: (i) K-means, (ii) Agglomerative hierarchical clustering, and (iii) DBSCAN, which are widely recognized and implemented in various fields for their effectiveness in data analysis and pattern recognition.

5.1.1.1. K-means

K-means is one of the simplest of the hard clustering algorithms and as such is widely used and depurated.[27] This algorithm follows a straightforward way to classify a given data set through a certain number of clusters (assume k clusters) fixed *a priori*.[23] K-means goal is to produce groups of cases with a high degree of similarity within each group and a low degree of similarity between groups. To achieve this, k centroids are defined and placed far apart. Each data point is assigned to the nearest centroid, then new centroids are calculated based on the resulting clusters. This process repeats until the centroids no longer move (Fig. 8).[27]

Different metrics can be used to calculate this similarity, the Euclidian distance being the most common (1).[28]

(1) $$\boldsymbol{dE} = \sqrt{\sum_{i=1}^{n}(\boldsymbol{c_i} - \boldsymbol{x_i})^2}$$

Here, *c* is the cluster center, x is the case it is compared to, i is the dimension of x (or *c*) being compared and *k* is the total number of dimensions.[27]

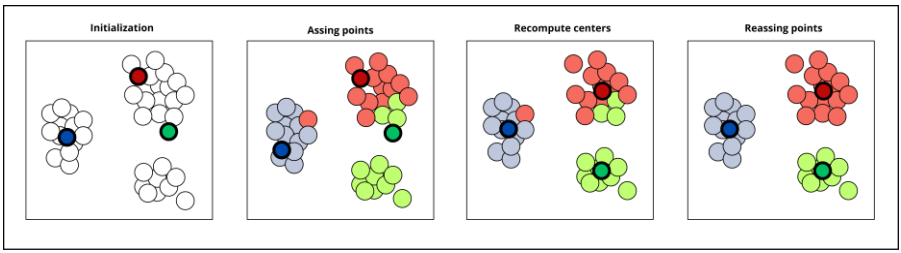

Figure 8. Schematic representation of K-means algorithm.

K-means algorithm is composed of the following steps:[28]

K-MEANS (S, k)

Input: a dataset of points $S = \{x_1,\ldots, x_n\}$, a number of clusters k
Output: centers $\{c_1, \ldots, c_k\}$ implicitly dividing S into k clusters

Step 1: Choose k numbers of clusters to be determined
Step 2: Choose C_k centroids randomly as the initial centers of the clusters
Step 3: Iteration
3.1: Assing each object to their closest cluster center using Euclidean distance
3.3: Compute new cluster center by calculating mean points
Step 4: Until
4.1: No changes in cluster center OR No object changes its cluster

5.1.1.2. Agglomerative hierarchical clustering

Agglomerative hierarchical clustering has been the dominant approach to constructing embedded classification schemes.[29] Agglomerative clustering schemes start from the partition of the data set into singleton nodes and merge step by step the current pair of mutually closest nodes into a new node until there is one final node left, which comprises the entire data set (Fig. 9).[30]

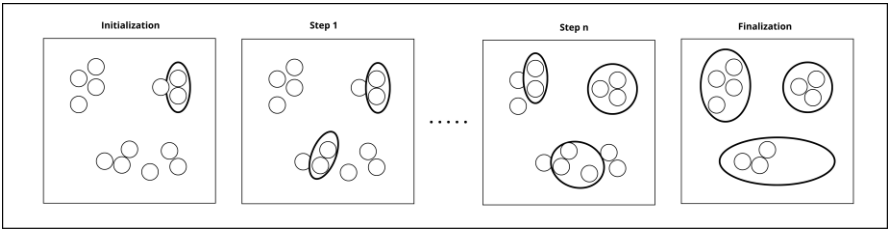

Figure 9. Schematic representation of agglomerative hierarchical clustering algorithm.

Various clustering schemes share this procedure as a common definition but differ in the way in which the measure of inter-cluster dissimilarity is updated after each step. [31] The most used methods are: (i) Single linkage, (ii) Average linkage, (iii) Complete linkage and (iv) Ward's methods.

- **Single linkage:** Also known as the nearest neighbor or minimum method. This measure defines the distance between two clusters as the minimum distance found between any pair of points, one from each cluster.

(2) $$\boldsymbol{d_{single}}(C_i, C_j) = \boldsymbol{min}\{d(x_a, x_b) | x_a \in C_i, x_b \in C_j\}$$

Here $d(x_a, x_b)$ is the distance between point x_a in cluster C_i and x_b in cluster C_j.

- **Complete linkage:** Also known as the furthest neighbor or maximum method. This method is close to Single linkage described above, but instead of searching for the minimum distance between pairs of cases, it considers the furthest distance between pairs of points from each cluster.

(3) $$\boldsymbol{d_{complete}}(C_i, C_j) = \boldsymbol{max}\{d(x_a, x_b) | x_a \in C_i, x_b \in C_j\}$$

Here $d(x_a, x_b)$ is the distance between point x_a in cluster C_i and x_b in cluster C_j.

- **Average linkage:** Also known as the Unweighted Pair-Group Method using Arithmetic averages (UPGMA). For Average linkage, the distances between each case in the first cluster and every case in the second cluster are calculated and then averaged.

(4) $$\boldsymbol{d_{average}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x_a \in C_i} \sum_{x_b \in C_j} d(x_a, x_b)$$

Here $|C_i|$ and $|C_j|$ are the number of points in clusters C_i and C_j respectively. The formula calculates the average of all distances between points in the two clusters.

- **Ward's method:** Defines the distance between two clusters as the increase in the total within-cluster variance after merging the two clusters. It minimizes the sum of squared differences within all clusters.

(5) $$\boldsymbol{d_{ward}}(C_i, C_j) = \frac{|C_i| \cdot |C_j|}{|C_i| + |C_j|} \left\| \bar{x}_i - \bar{x}_j \right\|^2$$

Here $\overline{x_i}$ is the mean (average) of the points in cluster C_i, $\overline{x_j}$ is the mean of the points in cluster C_j, and $\|\overline{x_i} - \overline{x_j}\|$ is the Euclidean distance between these means.[32]

Agglomerative clustering algorithm is composed of the following steps:[30]

AGGLOMERATIVE CLUSTERING (S, k)

Input: A dataset S containing data points $\{x_1, x_2, \ldots, x_n\}$, number of clusters k
Output: A dendrogram showing the hierarchy of clusters for k=1 to n

Step 1: Initialization
1.1: Each data point x_i starts as its own cluster
1.2: Define $C_i=\{x_i\}$ for each data point x_i
Step 2: Loop from n clusters to 1 cluster
2.1: At each iteration, there are k clusters. Construct the corresponding dendrogram k with the current clusters $\{C_1, C_2, \ldots, C_k\}$
2.2: Compute the distance $d\ (i, j)$ between all pairs of clusters C_i and C_j. The distance $d\ (C_i, C_j)$ is computed based on the chosen linkage criterion
2.3: Identify the two clusters C_l and C_m with the smallest distance
2.4: Merge these two clusters into a new cluster
2.5: Remove cluster C_m from the list of current clusters
Step 3: Termination of the loop
3.1: Continue the iterative process until only one cluster remains, which contains all data points

5.1.1.3. Density-Based Spatial Clustering of Applications with Noise

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm designed to identify core points and form clusters based on these points.[33] Before running the algorithm, the user must set two parameters: neighborhood distance (ε), which defines the radius around a sample point, and the threshold. These parameters are constants set before the program starts and remain unchanged.[32]

A core point is defined as a point that has enough neighboring points within a specified radius, indicating a dense region. The algorithm starts by identifying all core points and then forms clusters by connecting these core points with their neighboring points. Points that do not have enough neighbors to be considered core points are labeled as noise. Consequently, border points emerge as those point which, although not qualifying as core points themselves, reside within the ε radius of at least one core point, making them adjacent to dense regions (Fig. 10).[33]

DBSCAN offers a notable advantage by eliminating the need to predefine the number of clusters, enhancing its adaptability across various datasets. By effectively filtering out noise points, DBSCAN significantly improves the precision and clarity of the resulting clusters.[32]

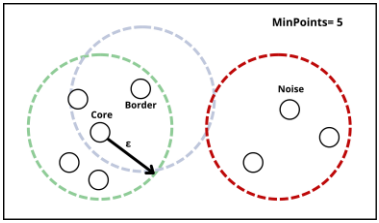

Figure 10. Schematic representation of DBSCAN clustering algorithm.

DBSCAN algorithm is composed of the following steps:[34]

DBSCAN (S, ε, MinPts)

Input: Dataset $S = \{x_1, x_2 \ldots, x_n\}$; Parameters: ε (radius of a neighborhood), MinPts (minimum number of points to form a dense region)
Output: Cluster labels for each point in dataset, noise points

Step 1: Initialization
1.1: Mark all points as unvisited
1.2: Initialize empty lists for clusters and noise points
Step 2: Iteration: for each point x do
2.1: Mark x as visited
2.2: Count points within ε-radius
2.3: If count ≥ MinPts, mark x as a core point
Step 3: Form clusters
3.1: start a new cluster with each unvisited core point x
3.2: Add all points within the ε-radius of x to this cluster
Step 4: Expand cluster
4.1: For each core point q in the cluster, add all its ε-neighbors if they are not already in the cluster
Step 5: Label noise points
5.1: Points that are not part of any cluster and are not core points are marked as noise

5.1.2. Soft clustering algorithms

Soft clustering is a technique that allows data items to belong to multiple clusters simultaneously. This work focuses exclusively on one specific type: Fuzzy C-Means, which is one of the most widely used methods in the field of soft clustering. By using this type of clustering, the aim is to obtain a different approach to data treatment.

5.1.2.1. Fuzzy C-means

Fuzzy C-Means (FCM) is a data clustering technique where each data point's membership in a cluster is defined by a degree of membership. The core idea of FCM is to determine the cluster centers, which represent the average location of each cluster. The algorithm assigns membership values to each data point for each cluster center based on the distance between the data point and the cluster center. The closer a data point is to a cluster center, the higher its membership value for that cluster.[23]

The summation of membership values for each data point across all clusters equals one. Initially, the cluster centers are imprecise, but through iterative refinement, both the cluster centers and the membership values are updated according to the following formulas:[35]

1. Update cluster Centers:

(6) $$G_k = \frac{\sum_{i=1}^{n} u_{ik}^{w} \cdot x_i}{\sum_{i=1}^{n} u_{ik}^{w}}$$

Here G_k is the center of cluster k, u_{ik} is the degree of membership of data point i in cluster k, w is the weighting exponent, and x_i represents the data point number i.

2. Update membership values:

(7) $$u_{ik} = \left(\sum_{j=1}^{c} \left(\frac{\|x_i - G_k\|}{\|x_i - G_j\|} \right)^{\frac{2}{w-1}} \right)^{-1}$$

Here $\|\cdot\|$ denotes the Euclidean distance, and c is the number of clusters.

Through these iterations, the cluster centers move towards their optimal and final positions. The primary advantage of FCM clustering is that it allows each data point to belong to multiple clusters simultaneously. Fig. 11 demonstrates how FCM operates compared to hard clustering.

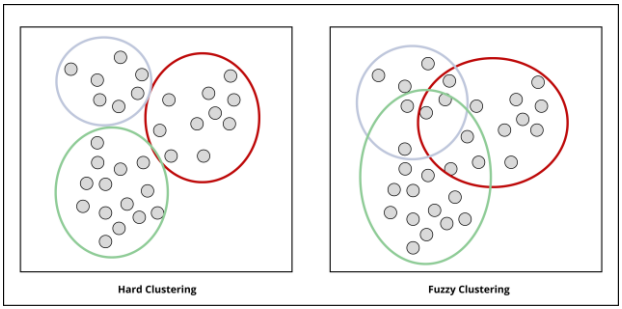

Figure 11. Hard and soft (fuzzy) clustering.

Fuzzy c-means algorithm is composed of the following steps:[23]

FUZZY C-MEANS (X, w, max_inter, *K*)

Input: data matrix X (n x m in size), with n = the amount of data to be clustered and m = the number of criteria (variables), number of clusters ($K \geq 2$), weighting rank (w >1), and maximum iteration (max_inter)
Output: Cluster centroids G_k for k=1, 2…, K, and membership matrix U indicating the degree of membership of each data point to each cluster

Step 1: Randomly initialization
1.1: Randomly initialize the membership matrix U, where each entry u_{ij} represents the degree of membership of data point i in cluster j
Step 2: Iteration
2.1: Calculate the centroids G_k by considering the cluster membership
2.2: Update the membership matrix U based on the distances between data points and centroids
Step 3: Until
3.1: Convergence (the centroids don't change)

5.2. CLUSTER EVALUATION METHODS

As previously observed, clustering algorithms aim to classify data into clusters. All these algorithms, except for DBSCAN, require a predefined number of clusters before their execution. Although this number can be determined subjectively, there are specific techniques that help identify the optimal number of clusters more accurately. These techniques are known as cluster evaluation methods. This section presents the cluster evaluation methods applied to the previous cluster algorithms which are (i) the Elbow method, (ii) the Average silhouette method, (iii) the Davies-Bouldin index, and (iv) the Calinski-Harabasz index.

5.2.1. Elbow method

The *Elbow Method* is an approach used to determine the optimal number of centroids (k) for a clustering algorithm by iteratively evaluating the within-cluster-sum of squares (WCSS) value, also known as inertia (I), for different values of k ranging from 1 to n (where n is a hyperparameter chosen based on requirements).[36] The inertia, I, is calculated using the formula (8).

(8) $$I = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Here K is the number of clusters, C_i is the set of data points assigned to cluster i, μ_i is the centroid (mean) of cluster i, and $||x - \mu_i||^2$ is the squared Euclidean distance between a data point x and the centroid μ_i of its cluster.

Subsequently, we generate an elbow graph plotting the WCSS values (on the y-axis) against different values of k (on the x-axis). The optimal value for k is identified at the point where the graph exhibits an elbow, indicating a significant drop in the cost followed by a plateau as k increases further. With increasing k, the average distortion decreases, and the samples become closer to the centroids. The k value corresponding to the elbow signifies the point where the improvement in distortion diminishes the most, suggesting an optimal number of clusters.[37]

5.2.2. Average silhouette method

The Average silhouette method determines the optimal number of clusters (k) by identifying the k that maximizes the mean silhouette coefficient across all observations. This method evaluates how similar an object is to its own cluster compared to other clusters, providing an effective measure of clustering quality.

The silhouette coefficient (s_i) for each sample xi in the dataset X is calculated using two main metrics: the average distance within the cluster (a_i) and the average distance to the nearest neighboring cluster (b_i). The formula for the silhouette coefficient is as follows:

(9) $$\boldsymbol{s_i} = \frac{b_i - a_i}{\mathbf{max}(a_i, b_i)}$$

The silhouette normalized coefficient ranges from -1 to 1:

- A value close to 1 indicates that the sample is well matched to its own cluster and poorly matched to neighboring clusters.

- A value close to 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters.
- A negative value indicates that the sample might have been assigned to the wrong cluster.[38]

5.2.3. Davies Bouldin Index

Davies-Bouldin Index (DBI) is utilized as an internal cluster evaluation scheme, assessing the quality of cluster results based on both the quantity and proximity between clusters. The DBI measures cluster validity by considering cohesion, defined as the sum of the proximity of data points to the center point of their respective clusters, and separation, which is based on the distance between cluster center points. The optimal number of clusters is determined by identifying the clustering solution that yields the lowest DBI value.[39] Mathematically, the Davies-Bouldin Index is calculated as follows:

(10) $$\boldsymbol{DBI} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{max}_{i\neq j}\left(\frac{\sigma_i+\sigma_j}{d(C_i,C_j)}\right)$$

Here n is the number of clusters, σ_i is the average distance between each point in cluster i and the centroid Ci of that cluster, and d (C_i, C_j) is the distance between centroids C_i and C_j of clusters i and j, respectively.[38]

5.2.4. Calinski-Harabasz index

The Calinski-Harabasz Index (CHI), also known as the Variance Ratio Criterion, is an evaluation index used to measure the quality of a clustering result by assessing the degree of dispersion between clusters. The index is defined as follows:

(11) $$\boldsymbol{CHI(K)} = \frac{W(K)\cdot(N-K)}{B(K)\cdot(K-1)}$$

Here K is the corresponding number of clusters, $\mathrm{B}(K)$ is the inter-cluster divergence, also called the inter-cluster covariance, $\mathrm{W}(K)$ is the intra-cluster divergence, also called the intra-cluster covariance, and N is the number of samples.

The between-clusters scatter matrix $\mathrm{B}(K)$ (12) and the within-cluster scatter matrix $\mathrm{W}(K)$ (13) are defined as:

(12) $$\boldsymbol{B(K)} = \left(\sum_{k=1}^{K} a_k \|\overline{x_k} - \bar{x}\|^2\right)$$ **(13)** $$\boldsymbol{W(K)} = \left(\sum_{k=1}^{K}\sum_{c(j)=k}\left\|x_j - \overline{x_k}\right\|^2\right)$$

The larger the B(K) is, the higher the degree of dispersion between clusters is. The smaller the W(K) is, the closer the relationship is in the cluster. The higher the ratio is, the larger the value of the CH index is, that is, the better the clustering effect is.[40]

6. MACHINE LEARNING APPLIED TO STM-BJ DATA

As it has been introduced before, in the field of STM-BJ measurements, the classification of data has traditionally relied on subjective criteria, often leading to variability in results. To address this problem, unsupervised ML algorithms were used in this study due to its ability to find hidden patterns and details that humans might miss. By using these techniques, a Python tool was created to unbiasedly analyze the dataset and stablishing a standardized approach to classifying STM-BJ data. To provide the most reliable and consistent tool for data analysis a comprehensive comparative study of various clustering algorithms and representations was conducted.

This section describes the process of creating the Python tool for analyzing STM-BJ measurements. It will include dataset description, preprocessing steps, features selection, evaluation of clustering algorithms, and a final presentation of results.

6.1. ANALYZED DATA'S DESCRIPTION

Our Python tool focuses on analyzing data obtained from STM-BJ technique using both STM's electrodes made of Au. The measurements were acquired applying a 50 mV bias employing a 1000 nA/V amplification, and using mesitylene as liquid protective environment. As explained in section 3.1.2.1, when the tip, initially in contact with the surface, separates, a Au-Au atomic junction is formed by few metallic atoms shrinks up to a specific distance at which the last single-atom bond eventually breaks. This phenomenon becomes evident in the intensity vs. time plots of each individual junction (see Fig.12).

As the tip initiates separation, a decay in current due to the growing distance between the electrodes, is followed by distinctive steps consequence of the multiatomic junction. These steps, via on their distinctive current values during the breaking events (see events Fig. 12), enable us to discern the number of atomic units contacted in parallel between at the tip and surface. These

plots are part of individual '.acq' files which the program is capable of displaying, indicating the file selection number. It requires a previous enumeration and a preprocessing.

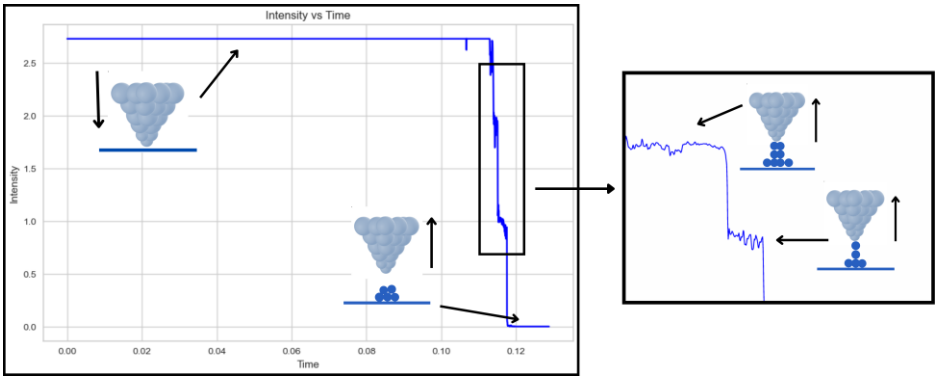

Figure 12. Plot Intensity vs. time of an illustrative sample. Note the variance in the current intensity due to the consecutive atomic disconnection during the junction evolution, evidenced as the distinct steps.

6.2. DATA PROCESSING

Preprocessing data contained in binary files ('.acq' files) is a crucial part of this task as Python only supports '.csv' files by default. As a solution, a binary unpacking process has been implemented to allow the program to process this type of data effectively. Next, the data is stored in a data frame (two-dimensional tabular data structure).

After conversing the files, the program aims to maximize efficiency and speed, needing data constraint. As illustrated in the Fig. 12, when intensity surpasses 2.5 nA, the current amplifier is saturated (upper detection limit exceeded), yielding data points irrelevant to analysis. Likewise, the current amplifier is below its lower detection limit as intensity approaches 0 when the tip is entirely separated from the surface electrode. For this reason, two functions were created that allows to the program to determine, based on upper and lower thresholds, when the current starts to decrease and when it becomes null, to remove the preceding and subsequent points.

The 'find_decay_start_point' function searches for the point where the current starts decreasing after a significant increase. It firstly finds the index where the current reaches its maximum using the 'np.argmax(current)' function. Next, it defines a threshold for the decrease, calculated as 33% of the maximum current. The function then looks for the index where the current

drops below this threshold from the index of the maximum current. The index found indicates the point where the current begins to decrease after the maximum step. On the other hand, the 'decay_stop_point' function searches for the point where the current becomes negligible, indicating complete electrode separation. Firstly, it finds the peaks of the current using the 'find_peaks(current)' function. Then, it calculates the maximum current of these peaks. The threshold for negligibility is calculated as 0.9% of the maximum current. The function then searches for the index where the current drops below this threshold and returns the found index, indicating the point where the current becomes negligible. Please note that the thresholds used in these functions may need adjustment depending on the specific employed current amplifiers or setup.

The above descrived approach enables the creation of a new Data Frame with the specific data required for analysis. Again, the program can display individual files, indicating the file selection number, as shown in Fig 13.

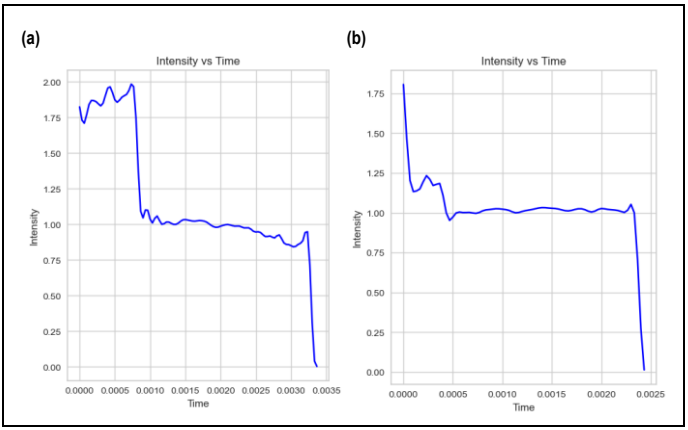

Figure 13. Plots Intensity vs. time of samples after preprocessing removing the points before the decay start point and after the decay stop point.

Visualizing the data individually facilitates thorough preprocessing, otherwise, a collective visualization is essential to capture the global trend of the measurements. Thus, in Fig. 14 (a) linear and (b) semi-logarithmic 1D histograms the whole dataset has been built. Initial observations from the plots indicate that the acquired data shows the expected behavior for Au-Au atomic contacts, mostly single-traces of 1 G_0 (one trapped Au atom) but occasionally they are preceded by 2 G_0 (two trapped Au atoms).

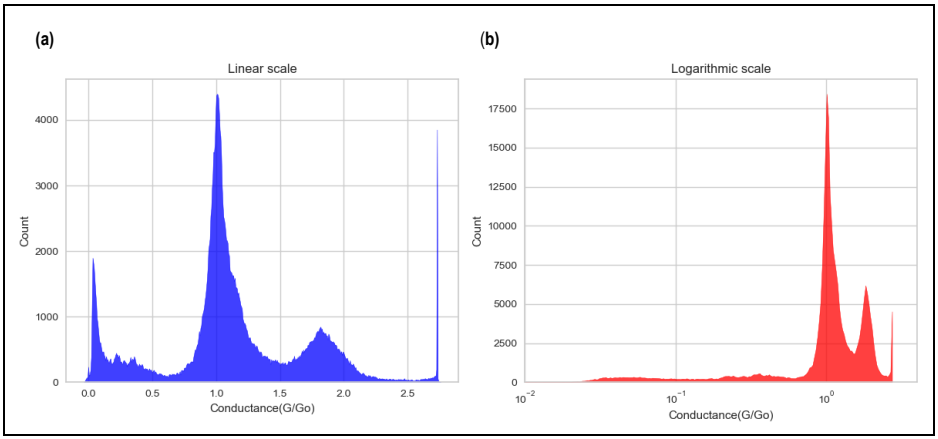

Figure 14. 1D lineal and semi-logarithmic conductance histograms of the whole analyzed dataset.

6.3. FEATURES EXTRACTION

Once the preprocessing is done, it is necessary to extract variables from the dataset to classify it into various groups using subsequent clustering algorithms. The aim of this process is to identify distinct features that make traffic patterns distinguishable from each other, aiming to create a new set of features F based on the original set. This is applied to the training dataset to produce a mapping function capable of transforming any future examples x into the same feature space.

Feature extraction is based on trial and error, where results and graphical representations play a key role. Initially, a series of variables were carefully selected based on their potential influence on data classification and their relevance to the experiment, which were **mean current, current kurtosis, noise, current skewness, slope, time, standard deviation, and coefficient of variation**. Take note that all data must be normalized to ensure consistency across the analysis.

Below, a description of each variable and its purpose in characterizing the current data are described:

- **Current Kurtosis**: Helps understand the shape of the current distribution, useful for identifying significant patterns.
- **Noise:** Represents external and instrumental fluctuations affecting current stability, ensuring data reliability.

- **Current Skewness:** Measures asymmetry in current distribution, revealing potential deviations with respect of the rest of dataset.
- **Slope**: Indicates the rate of change in the current, important for detecting abrupt changes and characterizing process kinetics.
- **Time**: Captures the temporal evolution of the current, essential for tracking changes over time.
- **Standard Deviation:** Quantifies data dispersion, complementing mean and range information.
- **Current Mean**: Provides the central value of the current distribution, offering a general understanding of the typical current magnitude observed.
- **Coefficient of Variation**: Measures data variability relative to the mean, useful for detecting anomalies.

Subsequently, clustering was performed which organized the data into groups according to criteria based on the implemented variables. Many variables have been selected, so it was necessary to observe which ones provide relevant information and which ones are redundant. To carry out a detailed study, two types of representations were made: plot histograms for each feature across each cluster, and plot correlation matrices for each cluster across each feature.

Histograms of each feature provide insights into the distribution of variables within each cluster. Fig. 15 shows that the histograms for "noise" and "coef_variation" evidence a clear correlation and thus they are redundant. It enables us to exclude one. In contrast, "current_skewness" shows a broad range, potentially affecting cluster dispersion.

The global correlation matrix (Fig. 16) confirms these observations, also revealing a correlation between "time_duration" and "current_std" with a correlation coefficient over 0.5.

Based on these observations and various trials, the features selected were **coefficient of variation, time, slope, current kurtosis, and current mean**. These selected features aim to enhance the accuracy and reliability of our clustering results, providing deeper insights into the STM-BJ experiments.

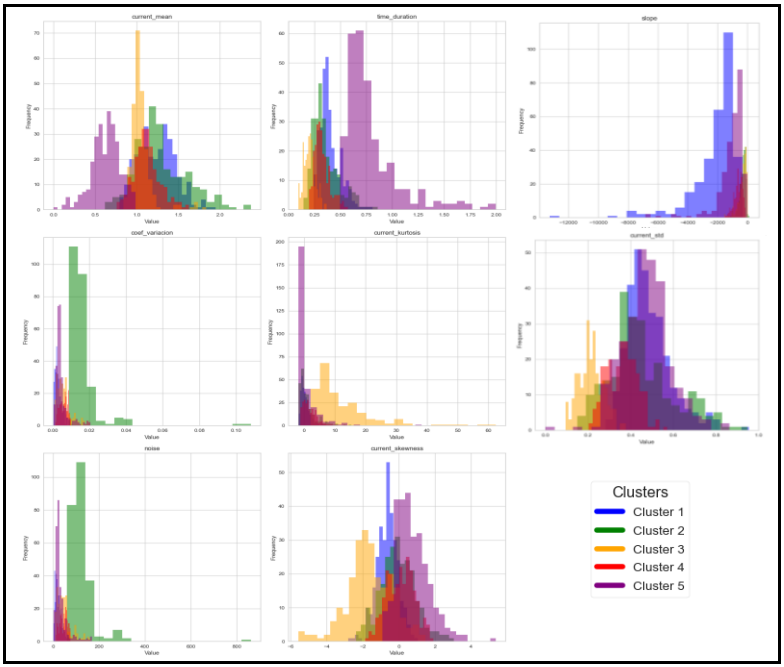

Figure 15. Histogram for each feature across each cluster.

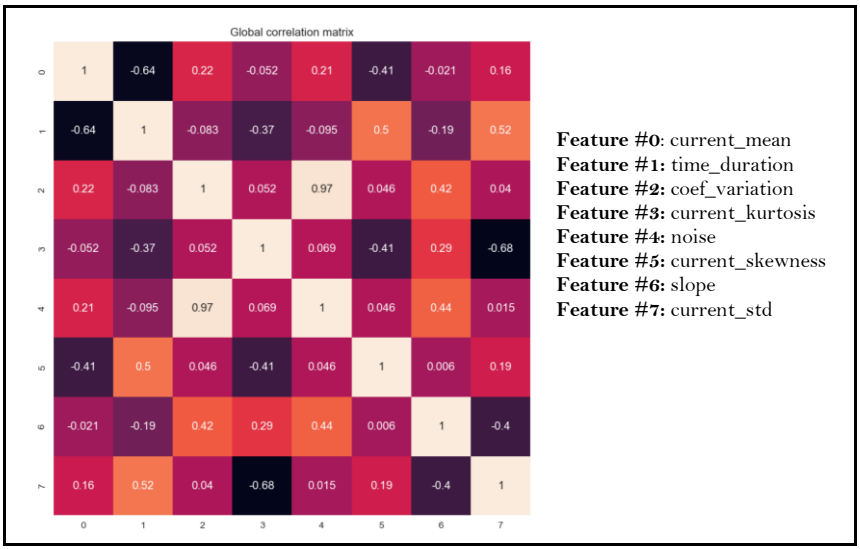

Figure 16. Global correlation matrix.

6.4. CLUSTERING EVALUATION

After selecting the variables that will represent the initial dataset, the data classification process, the clustering, begins. The clustering algorithms, described in Section 5, were implemented in the tool with the aim of determining the most appropriate criterion and the most reliable method for processing STM-BJ data. Except for the DBSCAN algorithm, which does not require an initial number of clusters, the procedure followed for the other algorithms is identical.

Below, the protocol followed for the selection of the optimal number of clusters is described using the K-means method as an example, employing different cluster evaluation methods.

Each cluster evaluation method is implemented by executing the K-means algorithm with an initial number of clusters k, iterating up to a maximum number of clusters. When applying the Elbow method, an optimal result of k=5 is obtained, indicating that this is the most appropriate number of clusters to execute the algorithm. This is determined by representing the WCSS values as a function of the number of clusters. As observed in Fig. 17, when k=1, the WCSS value is high. As k increases to 2, the WCSS value decreases. However, when choosing k=5, the reduction in WCSS stabilizes, forming a plateau. Finally, the WCSS equals zero when each point has its own cluster, as the centroid is exactly at the point, making the distance between them zero. Identifying the point where the elbow starts to stabilize is a complex process, but the program can determine it automatically.

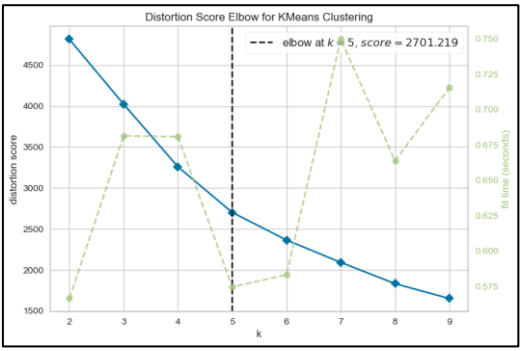

Figure 17. Representation of Elbow method applied to k-means algorithm.

Subsequently, the Average silhouette method was implemented in a similar manner, obtaining an optimal value of k=4, but that is not sufficient to select the optimal k. The following conditions should be checked to pick the right 'k' using the Silhouette plots.

Initially, the average silhouette score for each value of k is calculated, representing the mean silhouette coefficient across all points for each cluster (Fig. 18). The highest average silhouette score indicates the best clustering configuration, where points are well matched within their clusters and poorly matched with neighboring clusters.

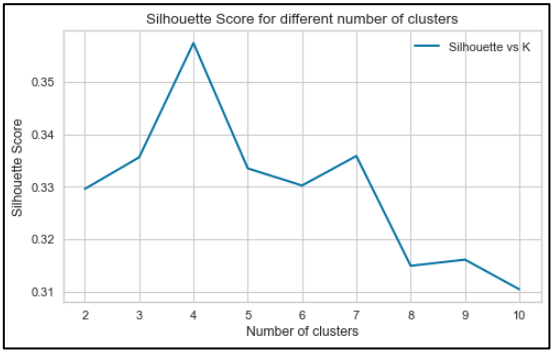

Figure 18. Representation of Average silhouette method applied to k-means algorithm.

Additionally, visualizing the silhouette plots for different numbers of clusters helps to understand the quality of the clustering. Each plot displays the silhouette coefficient of each point, sorted within their respective clusters. This visualization aids in confirming the number of clusters chosen by the program or exploring other potential configurations. As shown in Fig. 19, when k=4, a very large group of points is formed, and the remaining clusters are relatively small. In contrast, when k=5, the largest cluster remains dominant, but a fifth cluster emerges, which can provide additional insights into the data. Since the elbow method had determined 5 clusters, this configuration is the most reliable.

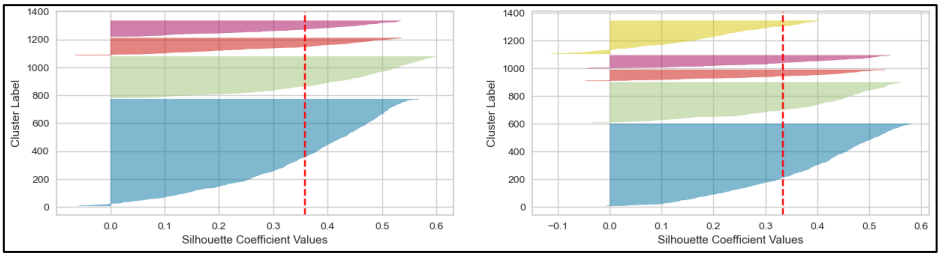

Figure 19. Additional representation of Average silhouette method.

In addition to the Elbow method and Average silhouette method, other clustering evaluation techniques were employed to further assess the quality of the clustering results. These methods are the Davies-Bouldin index and the Calinski-Harabasz index.

The DBI measures the average similarity between each cluster and its most similar cluster, considering both the scatter within clusters and the separation between clusters. After applying this method, a value k=4 was obtained for the clusters (Fig 20a).

On the other hand, the CHI, also known as the variance ratio criterion, evaluates clustering quality based on the ratio of the between-cluster dispersion to the within-cluster dispersion. Upon calculation, a value k=5 was obtained for the clusters (Fig 20b).

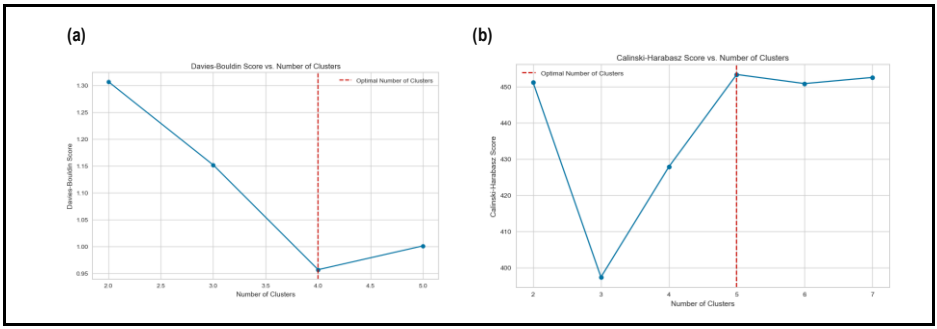

Figure 20. Representation of DBI and CHI Index applied to k-means algorithm.

Due to two of the proposed methods yielding a value of k=4 and two others yielding a value of k=5, it was decided to try the algorithm with both configurations to determine which one produces better results.

As has been mentioned earlier, the procedure followed for each clustering algorithm is like the one described above. Table 1 compiles the information from each clustering algorithm with each cluster evaluation method and the final chosen number of clusters. It is important to emphasize that the selection of the optimal number of clusters involves a subjective assessment, which relies primarily on the coherence of the results obtained from various cluster evaluation methodologies and their subsequent analysis. Additionally, it is worth noting that the effectiveness of cluster evaluation techniques can vary significantly depending on the clustering algorithm employed. Some methodologies may demonstrate superior performance in assessing the quality of clustering, while others may not offer as reliable results.

Cluster algorithm		Elbow method	Average silhouette method	DBI	CHI	Optimal k
Hard Clustering	K-means	5	4	4	5	4-5
	Single linkage	2	2	3	2	2
	Complete linkage	4	2	2	4	4
	Average linkage	2	2	4	2	2
	Ward's method	5	5	5	6	5
	DBSCAN	----	----	----	----	4
Soft clustering	FCM	6	4	6	5	5-6

Table 1. Comparison of the optimal number of clusters for different clustering algorithms using various cluster evaluation methods.

Observing Table 1, we can discard two methods of agglomerative clustering for data classification: the Average linkage method and the Single linkage method. Both the Single method and the Average method have yielded an optimal number of clusters of 2. However, these methods are not suitable for our STM-BJ data classification task due to their tendency to create clusters based solely on proximity or distance metrics, which may not adequately capture the underlying patterns and complexities present in our data.

6.4.1. Dimensionality reduction

After determining the optimal number of clusters for each clustering algorithm, we define the value of k for each algorithm and execute them. To facilitate the understanding of data classification, we employ dimensionality reduction techniques to visualize the clusters. Below, the following visualizations are presented: (a) UMAP, (b) t-SNE, and (c) PCA, for each method— K-means (Fig. 21), Complete linkage (Fig. 22), Ward's method (Fig. 23), DBSCAN (Fig. 24), and FCM (Fig. 25).

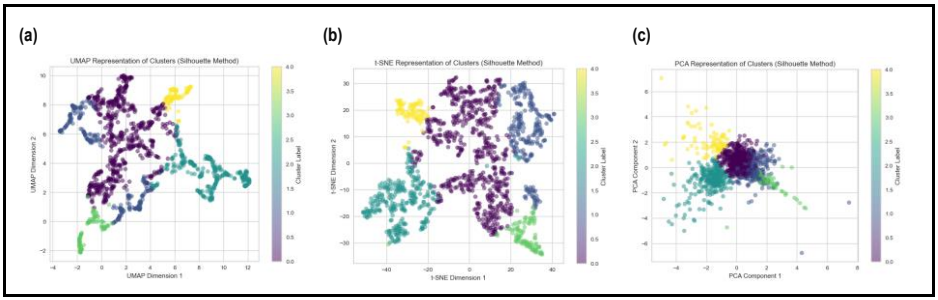

Figure 21. Dimensionality reduction representations for K-means.

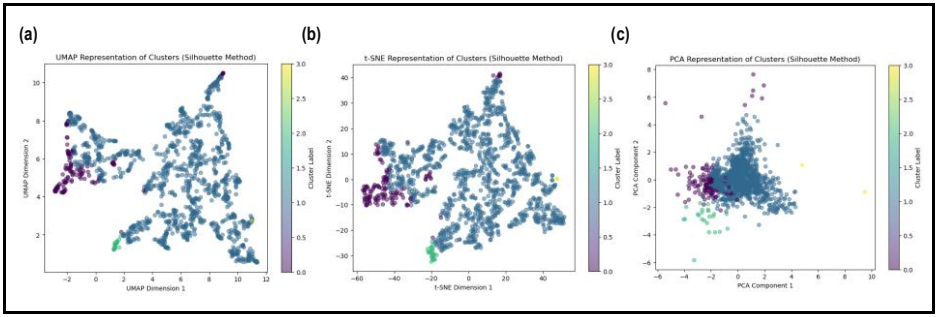

Figure 22. Dimensionality reduction representations for Complete linkage.

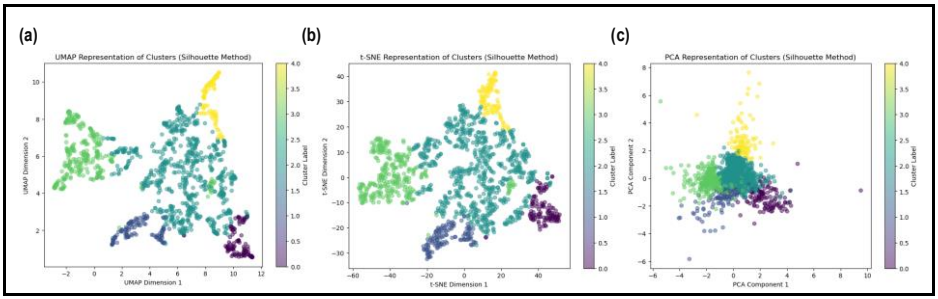

Figure 23. Dimensionality reduction representations for Ward's method.

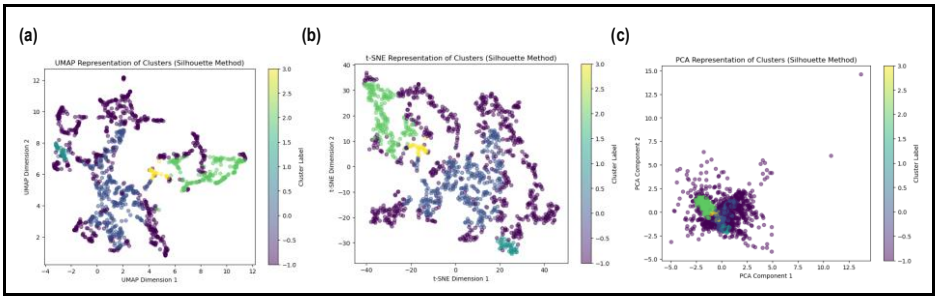

Figure 24. Dimensionality reduction representations for DBSCAN.

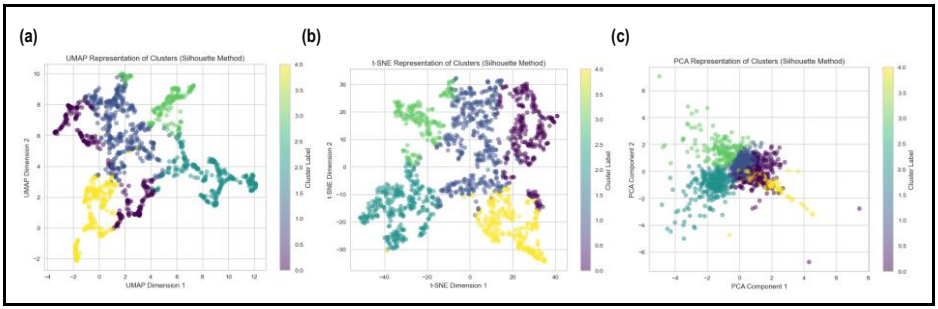

Figure 25. Dimensionality reduction representations for Fuzzy-c means.

After evaluating the results obtained through PCA, UMAP, and t-SNE, it is observed that each method offers a unique perspective on the underlying structure of the data. While PCA provides a quick dimensionality reduction and preserves the global structure of the data, both UMAP and t-SNE stand out for their ability to capture both global and local structures, which is especially relevant in nonlinear datasets. Therefore, based on the complexity of the data, both UMAP and t-SNE are considered to play a fundamental role in understanding the data structure, providing detailed and insightful perspectives.

Observing the set of visualizations, we can see those three methods (K-means, Ward's method, and FCM) reflect a good distribution of the data. In contrast, DBSCAN and Complete linkage exhibit a disordered distribution. In the later, smaller populations are encompassed within larger ones, indicating a lack of clear differentiation between them.

The DBSCAN algorithm does not perform well with our STM-BJ data. This method primarily focuses on detecting anomalies and forming clusters based on density. As observed in our visualizations, DBSCAN creates an external cluster that surrounds the data and several internal clusters. This indicates that DBSCAN struggles with the varying densities present in STM-BJ measurements, which can lead to an overemphasis on detecting noise rather than forming coherent clusters. The nature of STM-BJ data, which includes intricate patterns and variations, does not align well with DBSCAN's density-based clustering approach.

Similarly, the Complete linkage method also fails to provide a meaningful clustering solution. It detects one very large cluster and three much smaller groups with very few points, leading to an uneven and uninformative clustering structure. This method tends to create clusters by maximizing the distance between the most distant points within a cluster. For STM-BJ data, this

results in merging dissimilar clusters and forming disproportionately large clusters. This approach is not well-suited for our data, which contains subtle but important variations that are crucial for accurate clustering.

Therefore, due to these observations and the specific characteristics of STM-BJ data, we can anticipate that both, DBSCAN and Complete linkage, are not suitable for our clustering needs.

6.5. VALIDATION

To validate of each clustering method, we generated final 2D conductance histograms. Below, the clusters formed by each algorithm are described and compared to select the optimal clustering method according to the phenomenology and characteristics of atomic junctions' evolution.

K-means achieved its best classification with five clusters, as shown in Fig. 26. Cluster 1, with a 45.7% of selected traces, reflects the most common scenario, junctions of 2 G_o and 1 G_o conductance values[3], due to the sequential disconnection from two to single atoms during junction evolution. Cluster 2 also shows a trend of 1 G_o and 2 G_o signatures but with step-lengths 10 times longer than Cluster 1, highlighting stability in the junctions, and thus less common. Cluster 3 involves junctions due to common electrode-solvent interactions, forming Au-C-Au bonds, with short steps at ca. 0.3 G_o. Cluster 4 represents the simplest and most unlikely condition with a single Au atom junction at 1 G_o. Finally, Cluster 5 comprises residual data with shorter lifetimes and thus datapoints.

FCM also classified the data into five clusters, as shown in Fig. 27, following the same criteria as the K-means algorithm but with notable differences in the populations. The most remarkable difference is in Cluster 4, comprising 17.8% of the data with only 1 G_o compared to 7.5% in K-means. This indicates that K-means combined some single-step data with multiple-step data, hindering the analysis, and FCM successfully distinguished them. Also, the percentage of residual data in Cluster 5 increased significantly, indicating FCM's higher capacity to detect poorly defined traces, helping discard data that do not accurately reflect the experiment's behavior.

Ward's method also resulted in five clusters, as shown in Fig. 28. The organization of most clusters differed from the previous algorithms except for Cluster 3. Cluster 2, which contains long steps, had a very low percentage, while Cluster 1 presented more than 50% of the data. This suggests that the method failed to distinguish between step durations, combining long and short steps. On the other hand, Cluster 4 was very similar to the Cluster 4 in the K-means algorithm,

again not effectively separating samples with a single step as FCM did. Lastly, if we compare Cluster 4 and Cluster 5, which in the previous algorithms was the residual cluster, Cluster 5 surpass in population Cluster 4 with a similar length step. We can conclude that the algorithm created two minority groups very similar without obtaining a discrimination between the clusters. Ward's method achieved a proper discrimination between clusters but not a good assignment.

This analysis confirms that FCM clustering is the most effective method for analyzing STM-BJ data, providing clear and organized groupings. The success of FCM can be attributed to its approach of testing each data point against multiple cluster centers, ensuring the most accurate and nuanced classification. This ability to discern subtle differences in data patterns and to detect poorly defined traces underlines FCM's superior performance in handling complex datasets.

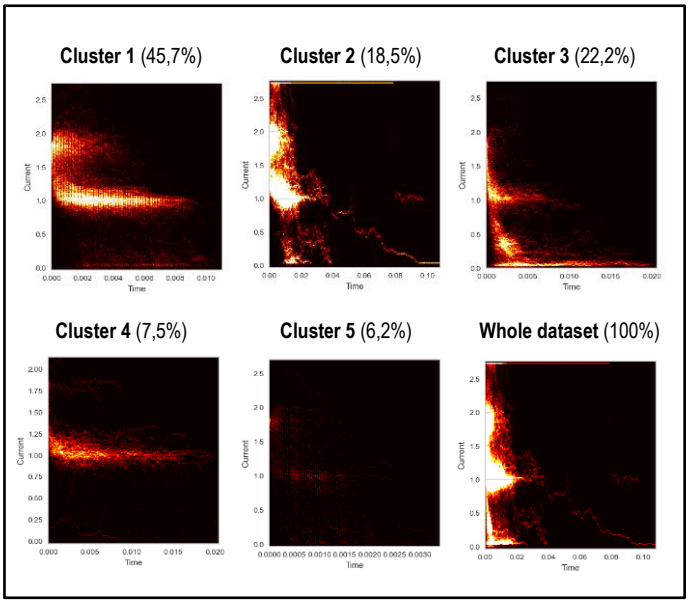

Figure 26. Lineal 2D histograms representations for each cluster applying K-means.

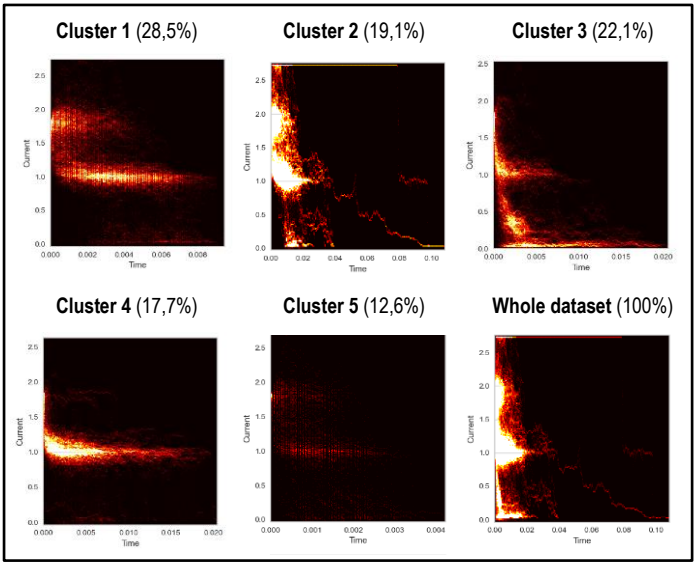

Figure 27. Lineal 2D histograms representations for each cluster applying FCM.

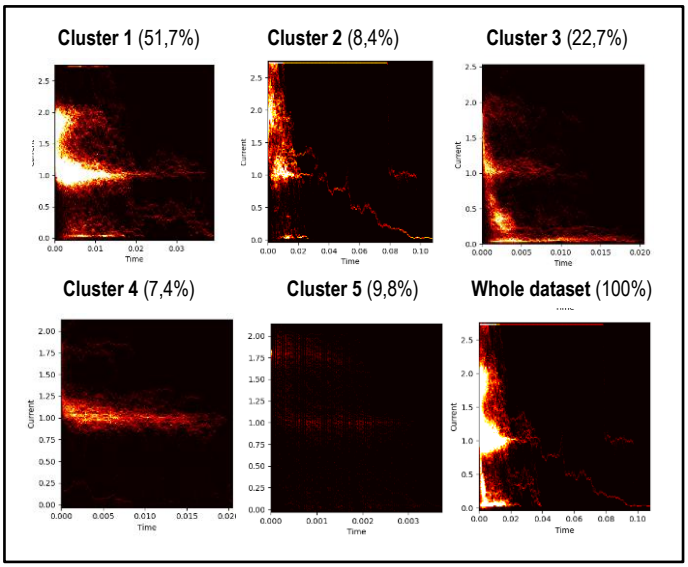

Figure 28. Lineal 2D histograms representations for each cluster applying Ward's method.

7. CONCLUSIONS

The development of a Python tool based on ML for analysing data from STM-BJ measurements has proven to be highly effective and objective in understanding the diverse patterns and trends in electron transport through Au atomic junctions. Unsupervised learning offered great potential for classifying data, adapting to various datasets, and producing consistent results that surpass the limitations of traditional human classification, which often leads to scattered and variable outcomes.

The extensive range of ML algorithms for clustering allowed for a detailed and thorough study to achieve the most accurate comparison between them. By testing our tool on a acquired STM-BJ dataset in the host lab, DBSCAN, Complete linkage, Single linkage, and Average linkage, were found to be ineffective in capturing the complex patterns in our data. Conversely, algorithms like K-means, FCM, and Ward's method proved to be highly effective, adapting to the complexity of the dataset and providing valuable insights into the mathematical data's parametrisation.

Among all the algorithms, FCM significantly stood out for its efficiency, producing clear and organized clusters that reflected the behaviours of atomic junctions. The identified clusters revealed relevant information about the conductance of electrons at the atomic level, distinguishing the number of junctions, the duration of conductance steps, and even possible interactions with the solvent. We speculate that fuzzy algorithms represent the most suitable framework since the subtle of atomic junctions' phenomenology. Our work opens new avenues for exploring complex patterns and subtle details for atomic junctions in ways that were not previously achievable.

11. REFERENCES AND NOTES

1. Kaliginedi, V. *et al.* Promising anchoring groups for single-molecule conductance measurements. Physical Chemistry Chemical Physics **16**, 23529–23539 (2014).
2. Kuznetsov, A. & Ulstrup, J. Electron Transfer in Chemistry and Biology: An Introduction to the Theory. (1999).
3. Chen, F., Hihath, J., Huang, Z., Li, X. & Tao, N. J. Measurement of Single-Molecule Conductance. *Annual Review of Physical Chemistry* **58**, 535–564 (2007).
4. Starr, R. Reactivity in the Single Molecule Junction. (2021).
5. Janssens, T. V. W. *et al.* Insights into the reactivity of supported Au nanoparticles: combining theory and experiments. *Topics in Catalysis* **44**, 15–26 (2007).
6. Zeng, B.-F. *et al.* Quantitative studies of single-molecule chemistry using conductance measurement. *Nano Today* **47**, 101660 (2022).
7. Dief, E. M., Low, P. J., Díez-Pérez, I. & Darwish, N. Advances in single-molecule junctions as tools for chemical and biochemical analysis. *Nature Chemistry* **15**, 600–614 (2023).
8. Nichols, R. J. *et al.* The experimental determination of the conductance of single molecules. Physical Chemistry Chemical Physics **12**, 2801 (2010).
9. Wang, L., Wang, L., Zhang, L. & Xiang, D. Advance of Mechanically Controllable Break Junction for Molecular Electronics. *Topics in Current Chemistry* **375**, 61 (2017).
10. Chen, J. Introduction to Scanning Tunneling Microscopy: Second Edition. *American Journal of Physics* **62**, 23-25 (1994).
11. Lv, S.-L. *et al.* Recent Advances in Single-Molecule Sensors Based on STM Break Junction Measurements. *Biosensors* 12, 565 (2022).
12. Gorenskaia, E. & Low, P. Methods for the analysis, interpretation, and prediction of single-molecule junction conductance behaviour. *Chemical Science* (2024)
13. Komoto, Y., Fujii, S., Iwane, M. & Kiguchi, M. Single-molecule junctions for molecular electronics. *Journal of Materials Chemistry C* **4**, 8842–8858 (2016).
14. Wold, D. J. & Frisbie, C. D. Fabrication and Characterization of Metal−Molecule−Metal Junctions by Conducting Probe Atomic Force Microscopy. *Journal American Chemistry Society* **123**, 5549–5556 (2001).
15. Komoto, Y., Ryu, J. & Taniguchi, M. Machine learning and analytical methods for single-molecule conductance measurements. *Chemical Communications* **59**, 6796–6810 (2023).
16. Mahesh, B. *Machine Learning Algorithms -A Review*. (2019).
17. Russell, S. J. & Norvig, P. *Artificial Intelligence: A Modern Approach*. (2016).
18. Bi, Q., Goodman, K. E., Kaminsky, J. & Lessler, J. What is Machine Learning? A Primer for the Epidemiologist. *American Journal of Epidemiology* **188**, 2222–2239 (2019).
19. Y C a, P., Pulabaigari, V. & B, E. Semi-supervised learning: a brief review. *International Journal of Engineering & Technology* **7**, 81 (2018).
20. James, G., Witten, D., Hastie, T. & Tibshirani, R. *An Introduction to Statistical Learning: With Applications in R*. (2021)
21. Shugara, R., Ernawati, E. & Andreswari, D. Implementation of the Fuzzy C-Means Clustering Algorithm and Simple Additive Weighting in the Provision of Assistance for the Residential Area Quality Improvement Program. *Pseudocode* **3**, 91–97 (2016).

22. Usama, M. *et al.* Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. *IEEE Access* **7**, 65579–65615 (2019).
23. Petrus,J., Ermatita and Sukemi, Soft and Hard Clustering for Abstract Scientific Paper. in *2019 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)* 131-136 (2019).
24. Van der Maaten, L. & Hinton, G. Viualizing data using t-SNE. *Journal of Machine Learning Research* **9**, 2579–2605 (2008).
25. Wang, Y., Chen, L., Jo, J. & Wang, Y. Joint *t* -SNE for Comparable Projections of Multiple High-Dimensional Datasets. *IEEE Transactions on Visualization and Computer Graphics* **28**, 623–632 (2022).
26. McInnes, L., Healy, J., Saul, N. & Großberger, L. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software* **3**, 861 (2018).
27. Kodinariya, T. & Makwana, P. Review on Determining of Cluster in K-means Clustering. *International Journal of Advance Research in Computer Science and Management Studies* **1**, 90–95 (2013).
28. Morissette, L. & Chartier, S. The k-means clustering technique: General considerations and implementation in Mathematica. *Tutorials in Quantitative Methods for Psychology* **9**, 15–24 (2013).
29. Murtagh, F. & Contreras, P. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery* **2**, 86–97 (2012).
30. Müllner, D. Modern hierarchical, agglomerative clustering algorithms. *arXiv* **1**, 1-12 (2011).
31. Yim, O. & Ramdeen, K. T. Hierarchical Cluster Analysis: Comparison of Three Linkage Measures and Application to Psychological Data. *The Quantitative Methods for Psychology* **11**, 8–21 (2015).
32. Deng, D. DBSCAN Clustering Algorithm Based on Density. in *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)* 949–953 (2020).
33. Cretulescu, R., Morariu, D., Breazu, M. & Volovici, D. DBSCAN Algorithm for Document Clustering. *International Journal of Advanced Statistics and IT&C for Economics and Life Sciences* **9**, 58–66 (2019).
34. Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* 226–231 (1996).
35. Suganya, R. & Shanthi, R. Fuzzy C- Means Algorithm- A Review. International Journal of Scientific and Research Publications **2**, 440-442 (2012).
36. Milligan, G. W. & Cooper, M. C. An examination of procedures for determining the number of clusters in a data set. *Psychometrika* **50**, 159–179 (1985).
37. Ketchen, D. J. & Shook, C. L. The Application of Cluster Analysis in Strategic Management Research: An Analysis and Critique. *Strategic Management Journal* **17**, 441–458 (1996).
38. Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M. & Perona, I. An extensive comparative study of cluster validity indices. *Pattern Recognition* **46**, 243–256 (2013).
39. Mughnyanti, M., Efendi, S. & Zarlis, M. Analysis of determining centroid clustering x-means algorithm with davies-bouldin index evaluation. *Materials Science and Engineering* **725**, 012128 (2020).
40. Wang, X. & Xu, Y. An improved index for clustering validation based on Silhouette index and Calinski-Harabasz index. *Materials Science and Engineering* 569, 052024 (2019).

12. ACRONYMS

BJ: Break Junction

STM-BJ: Scanning Tunneling Microscope Break-Junction

PZT: piezoelectric transducer

MCBJ: Mechanically Controllable Break Junctions

CP-AMF-BJ: Conducting Probe Atomic-Force-Microscope Break-Junction

ML: Machine Learning

AI: Artificial Intelligence

PCA: Principal Component Analysis

UMAP: Uniform Manifold Approximation and Projection

t-SNE: t-distributed Stochastic Neighbor Embedding

KL: Kullback-Leibler

DBSCAN: Density-Based Spatial Clustering of Applications with noise

FCM: Fuzzy C-Means

WCSS: within-cluster-sum of squares

DBI: Davies-Bouldin Index

CHI: Calinski-Harabasz index

APPENDICES

APPENDIX 1: STM-BJ TOOL (FUZZY-C MEANS)

IMPORT MODULES

```
#import modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.signal import find_peaks
import glob
import os
import math
from tqdm.notebook import tqdm, trange
from tqdm import tqdm
import struct
from scipy.signal import butter, filtfilt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
import sklearn.metrics as metrics
from sklearn.preprocessing import StandardScaler
from joblib import Parallel, delayed
from scipy.stats import kurtosis, skew, linregress
!pip install kneed
from kneed import KneeLocator
import warnings
warnings.filterwarnings("ignore")
!pip install umap-learn
from sklearn.manifold import TSNE
import umap
from sklearn.decomposition import PCA
from yellowbrick.cluster import SilhouetteVisualizer
from yellowbrick.cluster import KElbowVisualizer
from sklearn.metrics import davies_bouldin_score, calinski_harabasz_score
!pip install scikit-fuzzy
import skfuzzy as fuzz
from sklearn.datasets import make_blobs
```

EXPERIMENTAL PARAMETERS

```
#Experimental Parameters Introduction
Vbias = float(input("Enter the bias value in mV: ")) # potential applied between the two electrodes
Vbias = Vbias/1000
bins_1D = int(input("Enter the number of bins for 1D: "))
Amplification = float(input("Enter the amplification value in nA/V: "))
VtoG = (((Amplification)*1E-9) / 77.4E-6) * (1/Vbias) # output voltage in V
```

LOADING DATA

```
# Get the path of the "results" folder on the desktop
folder_path = os.path.join(os.path.expanduser("~"), "Desktop", "results")

# Get all .acq files in the "results" folder
acq_files = sorted(glob.glob(os.path.join(folder_path, '*.acq')))

# Display the number of files found
print(f"Number of .acq files found in the 'results' folder on the desktop: {len(acq_files)}")

# Check if .acq files were found
if not acq_files:
    print("No .acq files found in the 'results' folder on the desktop.")
```

READ BINARY FILES

```
# Read .acq binary files
def getDataframe(filename):
    binary = open(filename, "rb").read()
    number_of_points = struct.unpack("@I", binary[0:4])[0]
    datafile = struct.unpack("@" + "d" * number_of_points, binary[4:4 + number_of_points * 8])
    time_interval = struct.unpack("@d", binary[4 + number_of_points * 8 + 16:])[0]
    return pd.DataFrame({"time": [time_interval * i for i in range(number_of_points)], "current": datafile})
```

FILE SELECTION

```
data = []
for filename in acq_files:
    df = getDataframe(filename)
    df["current"] = df["current"].abs() # Take absolute values (log...)
    df['current'] = df['current'] * VtoG
    data.append(df)

# Select a file from the list
file_selected = int(input("Select the file number to visualize: ")) - 1
df_acq = data[file_selected]

# Plot intensity vs time
plt.figure(figsize=(10, 6))
plt.plot(df_acq['time'], df_acq['current'], color='blue')
plt.title('Intensity vs Time')
plt.xlabel('Time')
plt.ylabel('Intensity')
plt.grid(True)
plt.show()
```

FILTER DATA

```
#Signal Filtering Parameters
N = 2 # Butterworth filter order
Wn = 7000 # Cutoff frequency
fs = 30000 # Sampling rate

#Function to find the decay start point
def find_decay_start_point(time, current):
    # Find the index where the current starts decreasing after a significant increase
    start_index = np.argmax(current)  # Index of the maximum current
    max_current = current[start_index]
    # Define a threshold for the decrease
    threshold = 0.33  # You may need to adjust this threshold based on your data
    # Find the index where the current drops below the threshold
    decay_start_index = start_index + np.argmax(current[start_index:] < max_current * (1 - threshold))
    return time[decay_start_index], current[decay_start_index]

# Function to find the decay stop point
def decay_stop_point(time,current):
    peaks, _ = find_peaks(current)
    peak_max= max(current[peaks])
    threshold= 0.009 * peak_max
    for i in range(len(current)):
        if current[i]<threshold:
            decay_stop_point_index = i
            break
    return time[decay_stop_point_index], current[decay_stop_point_index]

# Functions to "coerce" or adjust the data to the decay start point and decay stop point
def coerce1(time, current):
    decay_start_time, decay_start_current = find_decay_start_point(time, current) # Get the coordinates of
the decay start point
    return decay_start_time, decay_start_current # Return the coordinates of the decay start point

def coerce2(time, current):
    decay_stop_time, decay_stop_current = decay_stop_point(time, current) # Get the coordinates of the
decay stop point
    return decay_stop_time, decay_stop_current # Return the coordinates of the decay stop point

# Read data from all capture files
data = [] #List to store valid files
```

```
invalid_files = []  # List to store invalid files
for filename in acq_files:
    df = getDataframe(filename)
    df["current"] = df["current"].abs()  # Take absolute values (log...)

    # Filtering
    B, A = butter(N=N, Wn=Wn, btype='lowpass', output='ba', fs=fs)
    df["current"] = filtfilt(B, A, df["current"])

    # Find the saturation point of the current and the inflection point
    saturation_current = df['current'].max()
    inflection_point_1 = coerce1(df['time'], df['current'])
    inflection_point_2 = coerce2(df['time'], df['current'])

    # Remove the saturation region
    df = df.loc[df['time'] >= inflection_point_1[0]]
    df = df.loc[df['time'] <= inflection_point_2[0]]
    df['time'] = df['time'] - df['time'].min()
    df['current'] = df['current'] * VtoG #Convert current to the Go scale

    # Check if the dataframe is empty after processing
    if df.empty:
        invalid_files.append(filename)  # Add filename to the list of invalid files
        continue  # Skip further processing for this file

    data.append(df)

# Print invalid files
if invalid_files:
    print("\033[4m\033[1mInvalid files (empty dataframes):\033[0m\n")
    for file in invalid_files:
        print(file)

# Concatenate all DataFrames
allinone = pd.concat(data, axis=0, ignore_index=True)

#Select a file from the list
file_selected = int(input("Select the file number to visualize: ")) - 1
df_acq = data[file_selected]

#Current vs. time plot for preprocesing files
plt.figure(figsize=(5, 6))
plt.plot(df_acq['time'], df_acq['current'], color='blue')
plt.title('Intensity vs Time')
plt.xlabel('Time')
plt.ylabel('Intensity')
plt.grid(True)
plt.show()
```

PLOT WHOLE DATASET

```
print('The plots of the data represented as a function of current are as follows:')

# Histograms linear and logarihtmic
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
sns.histplot(data=allinone["current"],  x=None,  log_scale=False,  bins=500,  ax=axes[0],  element="poly",
color="blue")
axes[0].set_title('Linear scale')
axes[0].set_xlabel('Conductance(G/Go)')

sns.histplot(data=allinone["current"],  x=None,  log_scale=True,  bins=500,  ax=axes[1],  element="poly",
color="red")

axes[1].set_title('Logarithmic scale')
axes[1].set_xlabel('Conductance(G/Go)')
axes[1].set_xlim(0.01, 4)
axes[1].set_xscale('log')
plt.tight_layout()

plt.show()
```

FEATURES EXTRACTION

```
# Function to extract features from all DataFrames using parallel processing (WITH progress bar)
def extract_features_parallel(data):
    features = Parallel(n_jobs=-1)(delayed(extract_features)(df) for df in tqdm(data, desc='Feature
Extraction'))
    return np.array(features)

# Function to extract features from a single DataFrame
def extract_features(df):
    current_mean = df['current'].mean()  # Current mean
    time_duration = df['time'].max() - df['time'].min()  # Time duration of capture
    current_kurtosis = kurtosis(df['current'])  # Kurtosis of current
    current_std = df['current'].std()  # Standard deviation of current
    slope, _, _, _, _ = linregress(df['time'], np.exp(df['current']))  # Calculate the slope of the linear
regression line of current vs. time
    coef_variacion = current_std / current_mean
    return [  coef_variacion,time_duration ,current_kurtosis,  slope, current_mean]

# List of feature names
feature_names = [  'time_duration','coef_variacion', 'current_kurtosis',  'slope', 'current_mean']

# Extract features from the data
features = extract_features_parallel(data)

# Normalize the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

CLUSTERING

ELBOW METHOD

```
# Elbow method
print("\033[4m\033[1mElbow Method:\033[0m\n")
wcss = []
for i in range(2, 11):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        data=features_scaled.T, c=i, m=2, error=0.005, maxiter=1000, init=None, seed=42)
    distances = np.min(d, axis=0)
    wcss.append(np.sum(distances ** 2))

# Determine the optimal number of clusters
kl = KneeLocator(range(2, 11), wcss, curve='convex', direction='decreasing')
elbow_point = kl.elbow

#Plot elbow
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), wcss, marker='o', linestyle='-')
plt.axvline(x=elbow_point, color='r', linestyle='--', label=f'Optimal Clusters: {elbow_point}')
plt.title('Elbow Method for Optimal Clusters (FCM)')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.legend()
plt.show()
print('Optimal number of clusters (Elbow Method):', elbow_point)
```

AVERAGE SILHOUETTE METHOD

```
#Average Silhouette method
print("\033[4m\033[1mSilhouette Method:\033[0m")

# Calculate the Silhouette Score for different numbers of clusters
min_clusters = 2
max_clusters = 10
silhouette = []

for n_clusters in range(min_clusters, max_clusters + 1):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        data=features_scaled.T, c=n_clusters, m=2, error=0.005, maxiter=1000, init=None, seed=42)
    cluster_labels = np.argmax(u, axis=0)
    silhouette_avg = metrics.silhouette_score(features_scaled, cluster_labels, metric='euclidean')
    silhouette.append(silhouette_avg)
# Determine the optimal number of clusters
```

```
optimal_num_clusters = np.argmax(silhouette) + min_clusters

# Plot Silhouette scores
plt.figure(figsize=(7, 4), dpi=80)
plt.plot(range(min_clusters, max_clusters + 1), silhouette, label='Silhouette vs K')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.legend(loc='upper right')
plt.title('Silhouette Score for different number of clusters')
plt.show()
print("Optimal number of clusters by the silhouette method:", optimal_num_clusters)

silhouette = []
fig, axs = plt.subplots(6, sharex=True, figsize=(20, 12))

for i in range(2, 8):  # Modify according to your results
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        data=features_scaled.T, c=i, m=2, error=0.005, maxiter=1000, init=None, seed=42)
    cluster_labels = np.argmax(u, axis=0)
    silhouette_samples = metrics.silhouette_samples(features_scaled, cluster_labels, metric='euclidean')
    silhouette.append(silhouette_samples)
    axs[i - min_clusters].plot(silhouette_samples, label=f'K={i}')
    axs[i - min_clusters].set_title("Silhouette average score for K={}: {:.3f}".format(i,
np.mean(silhouette_samples)))
    axs[i - min_clusters].legend()
plt.show()
```

DAVIES BOULDIN SCORE

```
# Davies-Bouldin Score
print("\n\033[4m\033[1mDavies-Bouldin Score:\033[0m\n")
davies_bouldin_scores = []
for i in range(2, 11):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        data=features_scaled.T, c=i, m=2, error=0.005, maxiter=1000, init=None, seed=42)
    cluster_labels = np.argmax(u, axis=0)
    db_score = davies_bouldin_score(features_scaled, cluster_labels)
    davies_bouldin_scores.append(db_score)

optimal_db_index = np.argmin(davies_bouldin_scores) + 2
print('Optimal number of clusters (Davies-Bouldin):', optimal_db_index)

# Plot Davies-Bouldin Score
plt.figure(figsize=(10, 6))
plt.plot(range(2, 11), davies_bouldin_scores, marker='o', linestyle='-')
plt.axvline(x=optimal_db_index, color='r', linestyle='--', label=f'Optimal Clusters: {optimal_db_index}')
plt.title('Davies-Bouldin Score vs. Number of Clusters (FCM)')
plt.xlabel('Number of Clusters')
plt.ylabel('Davies-Bouldin Score')
plt.legend()
plt.show()
```

CALINSKI-HARABASZ SCORE

```
# Calinski-Harabasz Score
print("\n\033[4m\033[1mCalinski-Harabasz Score:\033[0m\n")

# Calculate Calinski-Harabasz Score for different numbers of clusters
ch_scores = []
for i in range(3, 11):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        data=features_scaled.T, c=i, m=2, error=0.005, maxiter=1000, init=None, seed=42)
    cluster_labels = np.argmax(u, axis=0)  # Get cluster labels
    ch_score = calinski_harabasz_score(features_scaled, cluster_labels)
    ch_scores.append(ch_score)

optimal_ch_index = np.argmax(ch_scores) + 3
print('Optimal number of clusters (Calinski-Harabasz):', optimal_ch_index)

# Plot Calinski-Harabasz Score
plt.figure(figsize=(10, 6))
plt.plot(range(3, 11), ch_scores, marker='o', linestyle='-')
plt.title('Calinski-Harabasz Score vs. Number of Clusters')
plt.xlabel('Number of Clusters')
```

```
plt.ylabel('Calinski-Harabasz Score')
plt.axvline(x=optimal_ch_index, color='r', linestyle='--', label='Optimal Number of Clusters')
plt.legend()
plt.show()
```

CLUSTERING LOOP

```
def fuzzy_cmeans(features_scaled, n_clusters):
    cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
        features_scaled.T, n_clusters, 2, error=0.005, maxiter=1000, init=None)

    #Assigning samples to a single cluster based on maximum membership
    labels_soft = np.argmax(u, axis=0)

   # Visualize fuzzy membership of samples to clusters
    fig, ax = plt.subplots(figsize=(35, 6))
    for i in range(n_clusters):
        ax.plot(u[i], label=f'Cluster {i+1}')
    ax.set_xlabel('Sample')
    ax.set_ylabel('Fuzzy Membership')
    ax.set_title('Fuzzy Membership of Samples to Clusters')
    ax.legend()
    plt.show()
    return labels_soft

num_clusters= 5 #Adjust to your result
# Perform soft clustering using c-fuzzy means
print("\n\033[4m\033[1mSoft Clustering (Fuzzy C-Means):\033[0m\n")
labels = fuzzy_cmeans(features_scaled, num_clusters)

# Create list of dataframes for each cluster
cluster_data = [[] for _ in range(num_clusters)]
for i, df in enumerate(data):
    cluster_data[labels[i]].append(df)
```

DIMENSIONALITY REDUCTION REPRESENTATIONS

UMAP

```
# Apply UMAP for dimensionality reduction
umap_model = umap.UMAP(n_components=2, random_state=0)
umap_result = umap_model.fit_transform(features_scaled)

# Plot UMAP representation colored by cluster labels
plt.figure(figsize=(8, 6))
plt.scatter(umap_result[:, 0], umap_result[:, 1], c=labels, cmap='viridis', marker='o', alpha=0.5)
plt.title('UMAP Representation of Clusters')
plt.xlabel('UMAP Dimension 1')
plt.ylabel('UMAP Dimension 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

t-SNE

```
# Apply t-SNE for dimensionality reduction
tsne = TSNE(n_components=2, random_state=0)
features_tsne = tsne.fit_transform(features_scaled)

# Plot t-SNE representation colored by cluster labels
plt.figure(figsize=(8, 6))
plt.scatter(features_tsne[:, 0], features_tsne[:, 1], c=labels, cmap='viridis', marker='o', alpha=0.5)
plt.title('t-SNE Representation of Clusters')
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('t-SNE Dimension 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

PCA

```
# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
pca_result = pca.fit_transform(features_scaled)

# Plot PCA representation colored by cluster labels
```

```
plt.figure(figsize=(8, 6))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=labels, cmap='viridis', marker='o', alpha=0.5)
plt.title('PCA Representation of Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster Label')
plt.show()
```

FILES PER CLUSTER

```
# Create a dictionary to store the file numbers per cluster
file_numbers_by_cluster = {i: [] for i in range(num_clusters)}

# Associate each file number with its respective cluster
for i, label in enumerate(labels):
    # Ensure label is within the valid range
    if label < num_clusters:
        file_numbers_by_cluster[label].append(i + 1)  # Add 1 to start file numbers from 1

# Print the file numbers per cluster
print("\033[4m\033[1mFiles per Cluster:\033[0m \n")
for cluster, file_numbers in file_numbers_by_cluster.items():
    print(f"\033[1mCluster {cluster + 1}:\033[0m {file_numbers}\n")
```

CLUSTERING REPORT

```
# Print information for each cluster with data
print("\033[4m\033[1mCluster Information:\033[0m \n")
for cluster, file_numbers in file_numbers_by_cluster.items():
    if len(file_numbers) > 0:
        print(f"\033[1mCluster {cluster + 1}:\033[0m")
        print(f"Number of captures: {len(file_numbers)}\n")
        print("\033[1mFeature Information (Min-Max):\033[0m")
        for feature_name, feature_values in zip(feature_names, np.vstack((np.min(features_scaled[labels
== cluster], axis=0), np.max(features_scaled[labels == cluster], axis=0))).T):
            print(f"{feature_name}: ({feature_values[0]:.2f} , {feature_values[1]:.2f})")
        print("\n")

# Plot histograms for each cluster with data
num_features = len(feature_names)
colors = ['blue', 'green', 'orange', 'red', 'purple', 'brown', 'pink', 'gray', 'olive', 'cyan',
'magenta']
fig, axes = plt.subplots(len(file_numbers_by_cluster), 1, figsize=(6, 4 * len(file_numbers_by_cluster)))

for cluster, file_numbers in file_numbers_by_cluster.items():
    if len(file_numbers) > 0:
        ax_index = list(file_numbers_by_cluster.keys()).index(cluster)
        for j, feature_name in enumerate(feature_names):
            cluster_features = features_scaled[labels == cluster, j]
            axes[ax_index].hist(cluster_features, bins=20, color=colors[j], alpha=0.5,
label=feature_name)

        axes[ax_index].set_title(f'Cluster {cluster+1}')
        axes[ax_index].set_xlabel('Value')
        axes[ax_index].set_ylabel('Frequency')
        axes[ax_index].grid(True)
        axes[ax_index].legend()
plt.tight_layout()
plt.show()
```

FEATURES REPORT

```
# Plot histograms for each feature across all clusters
num_clusters = len(cluster_data)
num_features = len(feature_names)
colors = ['blue', 'green', 'orange', 'red', 'purple', 'brown', 'pink', 'gray', 'olive', 'cyan',
'magenta']

fig, axes = plt.subplots((num_features + 1) // 2, 2, figsize=(15, 25))

for j, feature in enumerate(feature_names):
    row = j // 2
    col = j % 2
    for i, cluster_df in enumerate(cluster_data):
```

```
        cluster_features = features[labels == i]
        axes[row, col].hist(cluster_features[:, j], bins=20, color=colors[i], alpha=0.5)
    axes[row, col].set_title(f'{feature}')
    axes[row, col].set_xlabel('Value')
    axes[row, col].set_ylabel('Frequency')
    axes[row, col].grid(True)

# Remove the last axis if the number of features is odd
if num_features % 2 != 0:
    fig.delaxes(axes[(num_features) // 2, 1])

# Add a global legend in a box
handles = [plt.Line2D([0], [0], color=colors[i], lw=4) for i in range(num_clusters)]
labels = [f'Cluster {i+1}' for i in range(num_clusters)]
fig.legend(handles, labels, loc='upper right', bbox_to_anchor=(1.15, 0.5), ncol=1, frameon=True,
title="Clusters")

plt.tight_layout(rect=[0, 0, 0.85, 1])   # Adjust layout to prevent the legend from overlapping
plt.show()
```

CORRELATION MATRIX

```
# Plot correlation matrices for each cluster in a row
fig, axs = plt.subplots(1, num_clusters+1, figsize=((num_clusters+2)*9, 9))
for i in range(num_clusters):
    # Extract features for current cluster
    cluster_features = [features[j] for j in range(len(features)) if labels[j] == i]
    cluster_data_scaled = scaler.fit_transform(cluster_features)

    # Plot correlation matrix for current cluster
    corr_matrix = pd.DataFrame(cluster_data_scaled).corr()
    sns.heatmap(corr_matrix, annot=True, cbar=False, ax=axs[i])
    axs[i].set_title(f"Correlation matrix for cluster #{i+1}")

    # Compute global correlation matrix
    if i == num_clusters - 1:
        global_data_scaled = scaler.fit_transform(features)
        global_corr_matrix = pd.DataFrame(global_data_scaled).corr()
        sns.heatmap(global_corr_matrix, annot=True, cbar=False, ax=axs[num_clusters], center=None)
        axs[num_clusters].set_title("Global correlation matrix")

plt.show()
print()

for i in range(len(feature_names)):
    print("Feature #{}: {}".format(i, feature_names[i]))
```

CURRENT LINEAL HISTOGRAMS (2D)

```
# Define x-axis limits for histogram scaling
x_limits1D = [0, 3]
bins1D = 200
bins2D = 200
saturation2D = 25

# Create subplots for each cluster
fig, axs = plt.subplots(num_clusters+1, 3, figsize=(4*3, 4*(num_clusters+1)))

# Calculate percentages for each cluster
percentages = [(len(cluster_data[i]) / len(acq_files)) * 100 for i in range(num_clusters)]
rounded_percentages = [round(p, 1) for p in percentages]

# Adjust the last percentage to ensure the sum is exactly 100%
percentage_sum = sum(rounded_percentages)
if percentage_sum != 100:
    adjustment = 100 - percentage_sum
    rounded_percentages[-1] += adjustment

# Plot histograms for each cluster
for i in range(num_clusters):
    # Plot 1D histogram of current
    all_currents = np.concatenate([df['current'] for df in cluster_data[i]])
    axs[i, 0].hist(all_currents, alpha=0.5, label=f"Cluster {i+1}: {rounded_percentages[i]} % samples",
range=x_limits1D, bins=bins1D)
    axs[i, 0].set_xlim(x_limits1D)
```

```
    axs[i, 0].set_xlabel("Current")
    axs[i, 0].set_ylabel("Frequency")
    axs[i, 0].legend()

    # Plot 2D histogram of all captures in each cluster
    all_currents = np.concatenate([df['current'] for df in cluster_data[i]])
    all_times = np.concatenate([df['time'] for df in cluster_data[i]])
    axs[i, 1].hist2d(all_times, all_currents, cmap=plt.cm.hot, density=False, bins=bins2D,
cmax=saturation2D)
    axs[i, 1].set_xlabel("Time")
    axs[i, 1].set_ylabel("Current")

    # Plot all captures in each cluster
    for df in cluster_data[i]:
        axs[i, 2].plot(df['time'], df['current'], alpha=0.5)
    axs[i, 2].set_xlabel("Time")
    axs[i, 2].set_ylabel("Current")

# Plot histograms for the entire dataset
# Plot 1D histogram of current
all_currents = np.concatenate([df['current'] for df in data])
axs[num_clusters, 0].hist(all_currents, alpha=0.5, range=x_limits1D, label="Whole dataset",
density=False, bins=bins1D)
axs[num_clusters, 0].set_xlim(x_limits1D)
axs[num_clusters, 0].set_xlabel("Current")
axs[num_clusters, 0].set_ylabel("Frequency")
axs[num_clusters, 0].legend()

# Plot 2D histogram of all captures in the entire dataset
all_currents = np.concatenate([df['current'] for df in data])
all_times = np.concatenate([df['time'] for df in data])
axs[num_clusters, 1].hist2d(all_times, all_currents, cmap=plt.cm.hot, density=True, bins=bins2D,
cmax=saturation2D)
axs[num_clusters, 1].set_xlabel("Time")
axs[num_clusters, 1].set_ylabel("Current")

# Plot all captures in the entire dataset
for df in data:
    axs[num_clusters, 2].plot(df['time'], df['current'], alpha=0.5)
axs[num_clusters, 2].set_xlabel("Time")
axs[num_clusters, 2].set_ylabel("Current")

plt.tight_layout()
plt.show()
```

CURRENT LOGARITHMIC HISTOGRAMS (2D)

```
# Define x-axis limits for histogram scaling
G_limit = [-1, 0.5]
bins1D = 100
bins2D = 200

df['log_current'] = np.log10(df['current'])

# Ensure 'log_current' column is present in all dataframes
filtered_cluster_data = []

for i in range(num_clusters):
    filtered_cluster_data.append([])
    for df in cluster_data[i]:
        if 'log_current' not in df.columns:
            df['log_current'] = np.log10(df['current'])
        filtered_cluster_data[i].append(df[(df['log_current'] >= G_limit[0]) & (df['log_current'] <=
G_limit[1])])

# Concatenate all clusters
all_cluster_data = pd.concat([pd.concat(cluster) for cluster in filtered_cluster_data])

# Calculate percentages for each cluster
percentages = [(len(cluster) / len(data)) * 100 for cluster in filtered_cluster_data]
rounded_percentages = [round(p, 1) for p in percentages]

# Adjust the last percentage to ensure the sum is exactly 100%
percentage_sum = sum(rounded_percentages)
if percentage_sum != 100:
```

```
    adjustment = 100 - percentage_sum
    rounded_percentages[-1] += adjustment


# Create subplots for each cluster
fig, axs = plt.subplots(num_clusters + 1, 3, figsize=(4 * 3, 4 * (num_clusters + 1)))

# Plot histograms for each cluster
for i in range(num_clusters):
    # Plot 1D histogram of current
    all_currents = np.concatenate([df['log_current'] for df in filtered_cluster_data[i]])
    axs[i, 0].hist(all_currents, alpha=0.5, label=f"Cluster {i+1}: {rounded_percentages[i]} % samples",
bins=bins1D)
    axs[i, 0].set_xlabel("Current")
    axs[i, 0].set_ylabel("Frequency")
    axs[i, 0].legend()

    # Plot 2D histogram of all captures in each cluster
    all_times = np.concatenate([df['time'] for df in filtered_cluster_data[i]])
    valid_indices = ~np.isnan(all_times) & ~np.isnan(all_currents)
    axs[i, 1].hist2d(all_times[valid_indices], all_currents[valid_indices], cmap=plt.cm.hot,
density=False, bins=bins2D)
    axs[i, 1].set_xlabel("Time")
    axs[i, 1].set_ylabel("Current")
    axs[i, 1].set_xlim([min(all_times), max(all_times)])

    # Plot all captures in each cluster
    for df in filtered_cluster_data[i]:
        axs[i, 2].plot(df['time'], df['log_current'], alpha=0.5)
    axs[i, 2].set_xlabel("Time")
    axs[i, 2].set_ylabel("Current")

# Plot histograms for the entire dataset
# Plot 1D histogram of current
all_currents = all_cluster_data['log_current']
axs[num_clusters, 0].hist(all_currents, alpha=0.5, label="Whole dataset", density=False, bins=bins1D)
axs[num_clusters, 0].set_xlabel("Current")
axs[num_clusters, 0].set_ylabel("Frequency")
axs[num_clusters, 0].legend()

# Plot 2D histogram of all captures in entire dataset
all_times = all_cluster_data['time']
valid_indices = ~np.isnan(all_times) & ~np.isnan(all_currents)
axs[num_clusters, 1].hist2d(all_times[valid_indices], all_currents[valid_indices], cmap=plt.cm.hot,
density=True, bins=bins2D)
axs[num_clusters, 1].set_xlabel("Time")
axs[num_clusters, 1].set_ylabel("Current")
axs[num_clusters, 1].set_xlim([min(all_times), max(all_times)])

# Plot all captures in entire dataset
for df in data:
    axs[num_clusters, 2].plot(df['time'], df['log_current'], alpha=0.5)
axs[num_clusters, 2].set_xlabel("Time")
axs[num_clusters, 2].set_ylabel("Current")

plt.tight_layout()
plt.show()
```