# Clustering Algorithm for Reaction Identification with the Super-Enge Split-Pole Spectrograph at the John D. Fox Laboratory

*Dennis Houlihan*
Department of Physics, Florida State University

**Abstract**
We report the results of a clustering algorithm to identify and gate on reactions of interest measured with the Super-Enge Split-Pole Spectrograph (SE-SPS). The algorithm uses scikit's HDBSCAN clustering function to identify the number of clusters in particle-identification (PID) plots and to prescribe each data point to a cluster or as noise. Bayesian optimization of HDBSCAN parameters was performed using scikit's *gp_minimize* function, which uses Guassian Processes to increase evaluation efficiency. The algorithm was tested on experimental data with six SE-SPS magnetic field settings, with each test taking about 10 seconds to run and producing satisfactory identification of particle groups and gates around said groups. Limits to the algorithm arise if the dataset is too sparse or if there are less than three clusters, as seen in the misidentification of clusters in the 5.99 kG and 10 kG datasets.

## 1 Introduction

When performing a nuclear experiment, one does not have the power to control exactly what reaction should occur. In reality, several reactions may occur with varying probabilities during the duration of the experiment. It is imperative then to have some apparatus that can identify what reaction has occurred and provide the experimenter the ability to gate on that reaction so that proper analysis can proceed. One such nuclear detector system that requires reaction identification is the Super-Enge Split-Pole Spectrograph (SE-SPS) [1] located at the John D. Fox Laboratory at Florida State University. A schematic of the SE-SPS is shown in Figure 1.

To identify reactions with the SE-SPS, experimenters use particle-identification (PID) plots, as seen in Figure 2. These consist of the outgoing particle's energy loss, measured by the rear anode wire located inside the SE-SPS's light-ion focal-plane detector [2], plotted against the rest energy, measured by a plastic scintillator located behind the detector. Distinct groups or clusters can be seen which correspond to a specific particle type. By gating around the particle group of interest, one filters out any other reactions from the analysis. Manually gating around the reaction of interest is often time-consuming because the cut has to be saved to a separate file, and then the analysis pipeline has to be rerun with that cut file. Furthermore, the PID plot is not static, but rather the particle groups can shift positions and shape depending on the settings of the experiment (i.e. the SE-SPS's magnetic field and angle with respect to the beamline) as seen in Figure 2. Therefore, each setting needs a unique cut file.

The goal of this project is to implement a clustering algorithm that could identify each particle group, identify and omit outliers, and perform a cut around the desired group no matter what settings are applied in the experiment. The constructed algorithm was tested on experimental data taken with the SE-SPS in October 2022. This experiment consisted of a $\alpha$-particle beam impinged upon a $^{27}$Al target with the $^{27}$Al($\alpha$,d)$^{29}$Si as the reaction of interest. Several magnetic field settings were used in this experiment while the angle was kept constant at 35$^{\circ}$. This provided an excellent opportunity to test the algorithm on varying cluster shapes and positions, without introducing too many external variables. The following magnetic field settings were used: 5.99 kG, 6.80 kG, 7.30 kG, 8.00 kG, 9 kG, and 10 kG.

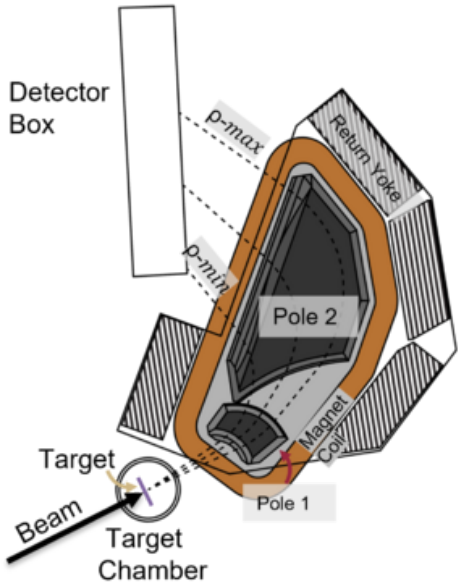


**Fig. 1:** A schematic of the FSU Super-Enge Split-Pole Spectrograph. The magnetic field deflects charged particles onto the charged-ion detector plane. The energy of each particle determines its magnetic rigidity and thus its position on the detector plane.

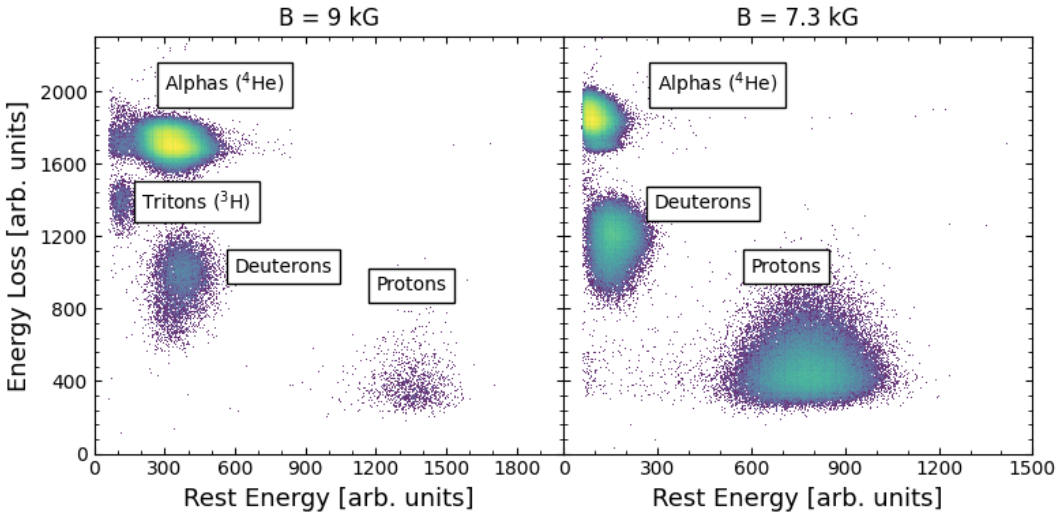


**Fig. 2:** Particle identification plots with the SE-SPS focal-plane detector. (Left) A PID plot with the SE-SPS magnetic field setting at 9 kG. Four particle groups corresponding to alpha particles, deuterons, protons, and tritons are distinguished. (Right) A PID plot with the SE-SPS magnetic field setting at 7.3 kG. Three groups corresponding to alpha particles, deuterons, and protons are distinguished. Due to the varying magnetic field settings, both the positions and shapes of the particle groups change and may even be omitted altogether.

## 2 Project Overview

In order to begin this project, it was essential to choose a good clustering algorithm that fits the project's needs. While there were several potential candidates, the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [3] was chosen to identify the number of clusters and to prescribe each data point to a cluster or as an outlier. More specifically, the HDBSCAN algorithm developed by Scikit-Learn, a public library with several Python functions and algorithms for data science and analysis purposes [4].

HDBSCAN is designed to uncover clusters in datasets based on the density distribution of data points. The algorithm performs DBSCAN [5] over varying epsilon values and integrates the result to find the clustering configuration that provides the best stability. In short, it groups together points that are closely packed, and marks as noise points that lie alone in low-density regions. More detailed description of the HDBSCAN algorithm can be found in Refs. [3, 5]. This algorithm is ideal since it does not assume that the density requirement is globally homogeneous like DBSCAN does. Instead, it explores all possible density scales which is important since clusters in the PID plot can, and often do, have different densities as reaction cross-sections (probabilities) vary.

While HDBSCAN is convenient for this purpose, its input parameters need to be well-tuned to provide a good output. The two main parameters that need optimizing are: *min_samples* and *min_cluster _size*. *Min_samples* is a parameter used to calculate the distance between a point and its k-th nearest neighbor. *Min_cluster _size* determines the minimum amount of data points in a group for that group to be considered a cluster; groupings smaller than this size will be labeled as noise.

Optimization was done using a Bayesian approach to maximize the mean probability, or strength, with which each data point is a member of its assigned cluster given by HDBSCAN. One of the challenges with a standard Bayesian optimization approach is that HDBSCAN can be computationally heavy, especially with large datasets, and thus very slow. To combat this, I used scikit-optimize's *gp_minimize* function [6]. This algorithm approximates the given function using a Guassian process where the function values are assumed to follow a multivariate guassian. The covariance of the function values are given by a GP kernel between the parameters. It then chooses the next parameter to evaluate based on the acquisition function over the Guassian prior which is much quicker to evaluate. To further increase efficiency, resampling of the data into smaller datasets was needed prior to HDBSCAN optimization and evaluation.

After HDBSCAN evaluation, each cluster is prescribe a particle label based on several criteria, mainly focusing on the relative positions between each group. Once labels are prescribed, the vertices of each group's gate is computed. The full outline of the algorithm is explained in detail in the following section.

## 3 Outline of Algorithm

The algorithm consists of several stages: (1) Load and clean the data; (2) Resample the data; (3) Parameter Optimization for HDBSCAN; (4) Perform HDBSCAN Clustering Algorithm; (5) Assign Particle Classes to Clusters; (6) Create Gates Around Clusters. The entire algorithm is contained in the function *PIDClassifier* and more detail for each step is described in the following sections. The full project can be found in Ref. [7].

### 3.1 Load and Clean the Data

To begin, the algorithm requires a path to the directory where the data is located. This parameter is called *datapath* and the directory must contain text files of two columns: the scintillator data (rest energy) on the left column and the anode wire data (energy loss) on the right column. Currently, this algorithm only supports data in this form and future plans would be to generalize it to include not only text files with two columns but general text files, csv files, or ROOT tree files. The algorithm then limits the data to be

within a certain range in order to cut out unnecessary noise. This range should be monitored for other experiments with different magnetic field and angle settings.

### 3.2 Resample the Data

The algorithm uses the function *density_aware_resample* to resample the dataset if it is too large and would be computationally cumbersome. If the dataset has more than 3000 data points, the function resamples the data to the length specified by the user with the *n_subset_samples* parameter. It is recommended to use at least 2000 data points for accurate results. Figure 3 shows an example of an original and resampled dataset used for analysis.

*density_aware_resample* also weights the resampled data to make sure low-density clusters can still be identified by HDBSCAN. Two types of weights are calculated, combined, and then normalized: density-aware and uniform sampling weights. This procedure and approach was configured with the help of ChatGPT's AI interface [8]. The density-aware weights are calculated by taking the inverse of a data point's local density calculated using scikit-learn's *NearestNeighbors* function [9]. The uniform weights are calculated by taking the inverse of the total length of the dataset. The following equation is used to combine both weights

$$\omega = \alpha\omega_{density-aware} + (1-\alpha)\omega_{uniform}, \tag{1}$$

where $\alpha$ is a factor used to determine how much of the total weights are influenced by the density-aware sampling. A lower $\alpha$ value decreases the effect of the density-aware sampling. For this experiment, $\alpha = 0.5$ and should be reevaluated for other experiments.

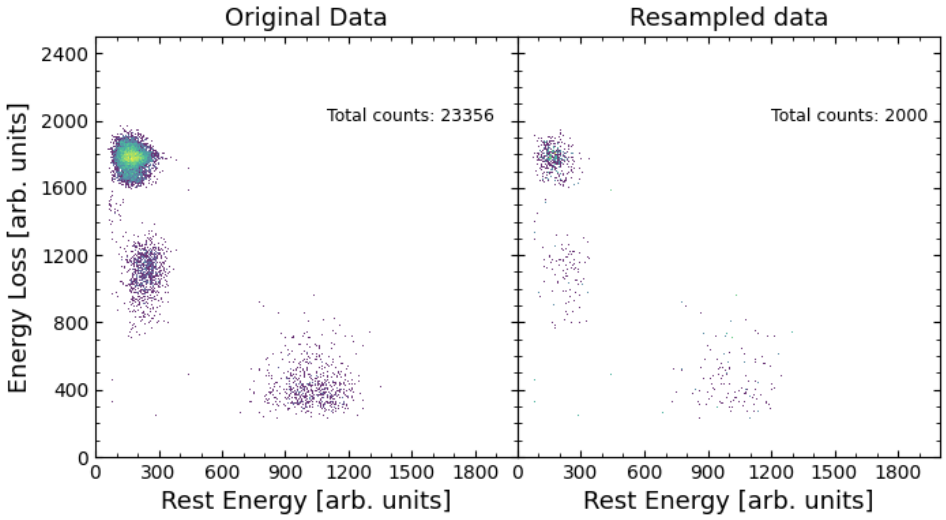


**Fig. 3:** A comparison of an original and resampled dataset with *n_subset_samples* = 2000 and $\alpha$ = 0.5.

### 3.3 Parameter Optimization for HDBSCAN

The *gp_minimize* function takes several parameters, but four were used in this project: *func*, *dimensions*, *n_calls*, and *random_state*. *func* is the function to be minimized. I created a function, with the help of chatGPT's AI interface, called *clustering_score* that runs the HDBSCAN function and returns $1 - \bar{P}$ where $\bar{P}$ is the mean strength of the data points given by HDBSCAN. By minimizing this function, the mean probabilities are maximized. It also filters out HDBSCAN fits that identify too little (< 1) or too many (> 4) clusters. *dimensions* is a list of search space dimensions that contains the lower and upper bounds of each parameter that *func* takes. In this case, the ranges are upper and lower bounds for

*min_samples* and *min_cluster_size*. *n_calls* is the number of calls to *func*, which was set to 50 for this project, but could be varied for other projects. *random_state* is a parameter that allows for reproducible results.

### 3.4 Perform HDBSCAN Clustering Algorithm

After the *min_samples* and *min_cluster_size* values are optimized, they are passed into the HDBSCAN algorithm. The cluster labels are then stored and the centroid positions of each cluster are calculated as the mean position of all the data points within each respective cluster. This helps identify what cluster corresponds to what particle, as described in the following section.

### 3.5 Assign Particle Classes to Clusters

To assign particle classes to clusters, I created the function called *get_particle_class*. This function takes the centroid positions and calculates the relative positions of the clusters. It also assigns conditions that need to be met in order for a cluster to be assigned to a particle. For example, for a cluster to be identified as a proton cluster, it needs to be the lowest and right-most one. Other conditions for varying numbers of identified clusters are also put in. If no conditions are satisfied, there is no assignment made. As the algorithm is implemented for different projects, there will most likely need to be additions and/or alterations to these conditions. Table 3.5 shows an example output of centroid positions and their assigned particle label. As you can see, the Proton cluster has the greatest centroid x-position and lowest centroid y-position.

| Centroid X Position | Centroid Y Position | Particle Label |
|---|---|---|
| 1309.90 | 405.99 | Protons |
| 372.40 | 961.76 | Deuterons |
| 323.43 | 1712.45 | Alphas |
| 122.76 | 1427.09 | Tritons |

### 3.6 Create Gates Around Clusters

Gates are constructed using the *ParticleGate* function. This function takes data points that are associated to a single cluster and uses scipy's *ConvexHull* function [10] to compute the smallest convex shape that encloses all the points in the set, i.e. the convex hull. The function then outputs the vertices of the convex hull which can be plotted and used as a gate for further analysis.

## 4 Results

Figure 4 shows the results of the algorithm after running it for each magnetic field setting. Each setting was resampled to 2000 data points except for the 10 KG setting, which had less than 1000 data points in its original set. Overall, the algorithm works very well as it identifies the correct amount of clusters and also prescribes the correct particle label in most field settings. However, proton gates often have a 'tail' that includes data points to the far left of the dense centroid. This tail may or may not be including noise or particles that are not protons. This could arise from the resampling phase of the algorithm and future analysis will be need to fine tune this and to make sure it does not include incorrect data points.

There are misidentifications in the 10 kG and 5.99 kG setting. In the 10 kg setting, the algorithm does not recognize the proton group as it is too sparse. As a result, it incorrectly labels the deuteron group as the proton group and the triton group as the alpha group. In the 5.99 kG setting, the proton group was labeled as the deuteron group and the deuteron group was labeled as the alpha group. It seems the conditions currently set for the algorithm have major limitations when there are only two groups present.

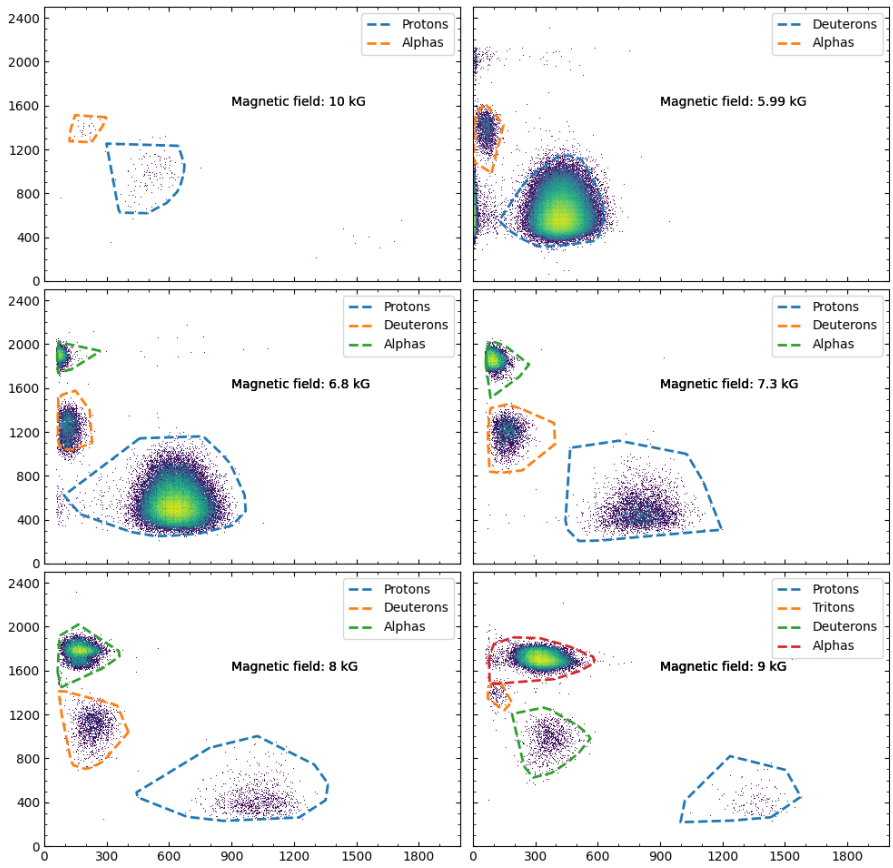


**Fig. 4:** The particle gates created for each magnetic field setting in the experiment. Correct identification of clusters is shown for every magnetic field setting except for the 10 kG dataset, which has unusually sparse data. Another common feature is a 'tail' in the proton cluster. This tail may include data points that are not protons.

Future work is needed to improve the accuracy of the algorithm. This should involve testing the algorithm on datasets from different experiments to help fine-tune the parameters of the algorithm and the functions that it uses as well as the conditions needed for assigning particle labels to clusters. One good test, for example, would be on experimental data where both the SE-SPS magnetic field and angle was varied. For especially sparse datasets, adding data to, or resampling, may be needed so that appreciable data points are in each cluster.

## Acknowledgements

I would like to thank Dr. Prosper for his help and guidance as instructor for this course. I would also like to thank my advisor, Mark Spieker, and my research groupmates for teaching me how to operate the lab and perform data analysis on nuclear experiments. Lastly, I'd like to thank all of the students and faculty in the Department of Physics at FSU for all their help and support.

## References


[1] H. A. Enge, "Magnetic Spectrographs for Nuclear Reaction Studies," Nuclear Instruments and Methods **162**, 1-3 (1979)

[2] C. Good, "A Study of $^{26}$Al(p,$\gamma$)$^{27}$Si with the Silicon Array for Branching Ratio Experiments (SABRE)," PhD Dissertation, Louisiana State University, 2020

[3] L. McInnes, J, Healy, "hdbscan: Hierarchical density based clustering," Journal of Open Source Software **2**, 11 (2017).

[4] SCIKIT-LEARN HDBSCAN https://scikit-learn.org/

[5] M. Esterm, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, 2nd International Conference on on Knowledge Discovery and Data Mining, Portland, Oregon, August 1996

[6] SCIKIT-OPTIMIZE GP_MINIMIZE https://scikit-optimize.github.io/stable/index.html

[7] PROJECT REPOSITORY https://github.com/DJHoulihan/MLinPhysicsFinalProject

[8] CHATGPT https://chatgpt.com/

[9] SCIKIT-LEARN NEARESTNEIGHBOR https://scikitlearn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html

[10] SCIPY.SPATIAL CONVEXHULL https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html