A Machine Learning Based Approach to Magnetic Nanoparticle Classification

Daniel Tong

4th May 2023

Supervisor: Dr Ondrej Hovorka

Word Count: 10035

This report is submitted in partial fulfilment of the requirements for the BEng Mechanical Engineering, Faculty of Engineering and Physical Sciences, University of Southampton.

Declaration

I, Daniel Tong declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;
2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. None of this work has been published before submission.

Acknowledgements

I would like to thank Dr. Ondrej Hovorka for his support, enthusiasm, and guidance during this project. His expertise and genuine interest were invaluable, and his dedication to my success was greatly appreciated.

Abstract

Magnetic nanoparticles have emerged as a promising tool for various biomedical applications, including biosensing for cancer detection. The physical principles underlying biosensing applications of magnetic nanoparticles are reviewed. This study aimed to classify different magnetic nanoparticle clusters based on differences in particle sizes and shapes using a machine learning-based approach. A pipeline of the machine learning process is presented, from feature extraction to the evaluation of clustering and classification algorithms, to determine whether magnetic nanoparticle clusters can be accurately distinguished from data of varying degrees of data randomness. The study's findings suggest that machine learning models can accurately classify magnetic nanoparticle clusters based on differences in particle sizes and shapes, which could inform the design of chemically functionalised particles for cancer detection. Future prospects are also discussed.

Contents

List of Abbreviations

ANOVA	Analysis of Variance
FN	False Negatives
FP	False Positives
GMR	Giant Magnetoresistance
IQR	Interquartile Range
KNN	K-Nearest Neighbors
MNP	Magnetic Nanoparticle
MNT	Magnetic Nanotag
MRI	Magnetic Resonance Imaging
OVO	One-vs-One
OVR	One-vs-Rest
Q1	Lower Quartile
Q3	Upper Quartile
SPM	Superparamagnetic
SPMNP	Superparamagnetic Nanoparticles
SQUID	Superconducting Quantum Interference Device
SV	Spin Valve
SVM	Support Vector Machines
TP	True Positives

1 Introduction

1.1 Context and Motivation

In 2015, Ahmad et al. forecasted that over 50% of people under 65 in the UK will be diagnosed with cancer at some point in their lifetime [1]. Current cancer detection methods are invasive, and commonly used cancer treatments non-localized systemics treatment. Magnetic nanoparticles (MNPs) are a class of nanoparticle used for novel biomedical applications. MNPs have the ability to magnetize and be manipulated by magnetic fields; these properties make them useful for a variety of applications, including biosensing for cancer detection. Biosensing involves MNPs that can chemically be functionalised with specific biomolecules, such as antibodies, that recognise and bind to cancer-associated biomarkers. The functionalised MNPs are then introduced into the body, where they can bind to biomarkers and cancer cells and be detected. This allows doctors to non-invasively and rapidly detect and diagnose cancer, as well as monitor the effectiveness of cancer treatments.

1.2 Challenge

The challenge is to adopt a machine learning based approach to build a predictive model that will be able to classify different nanoparticle clusters based on differences in particle sizes and shapes. Different machine learning algorithms will be compared to find the best model to classify the data. The overall aim is to be able to create an accurate model, where inferences can be drawn to inform the scientists designing the structures, so that they can use the chemically functionalised MNPs to identify different types of cancer.

1.3 Objectives

The objectives outlined below must be fulfilled in order to achieve the desired aim:

1. To extract and engineer informative features that can produce a more interpretable model.
2. To cluster similar groups of particle clusters using unsupervised algorithms to discover underlying patterns and groupings in the data.
3. To accurately classify particle clusters using a range of supervised learning algorithms.
4. To evaluate the models and determine whether MNP clusters can be accurately distinguished with the data.

1.4 Outline

This study is organised into six chapters, including this introduction.

Chapter 2 discusses key works across the broader area of medical MNP applications, as well key works within the application of biosensing.

Chapter 3 gives a detailed background on the physics of MNPs and types of data that can be measured. Brief introductions to specific machine learning techniques used in this work are also given.

Chapter 4 describes the machine learning pipeline and how different techniques were used. The evaluation metrics are also defined and discussed in this chapter.

Chapter 5 reports the main research, including procedures, results and their discussion. More specifically the three sections in the chapter include feature selection, unsupervised learning and supervised learning.

Chapter 7 concludes this study with a summary of findings and explores directions for future work.

2 Literature Review

2.1 Nanoparticles in Medicine

Early cancer diagnosis, staging, and treatment can greatly improve human life. Chemotherapy is the second most common cancer treatment behind surgery [2]. Although chemotherapy is an effective treatment, therapeutic drugs are administered into the bloodstream leading to general systemic distribution [3]. This means that the drugs attack both healthy and cancerous fast-growing cells [3]. This results in many negative side effects, including a weakened immune system, sickness, hair loss etc. The ideal treatments are locally targeting treatments that only attack cancerous cells, thus reducing the associated side-effects.

For the ideal localised treatments, rapid and accurate identification and detection of cancer cells is imperative. Current cancer identification methods include physical examinations, clinical laboratory testing and imaging tests (CT scan, MRI, ultrasound etc.) [4]. However, if a suspicious area is identified, biopsy is currently the only detection method to confirm a cancer diagnosis [4]. These identification and detection methods all have limitations that depend on the type of cancer and its location, and can take weeks or months to confirm [5].

Around the late 1970s, research and case studies began to emerge with a proposition to use magnetic microspheres for drug delivery and targeting [6] [7]. Most recent studies in the medical field of magnetic nanoparticles reviewed the following applications: (i) biosensing, (ii) drug delivery, (iii) bioimaging; and (iv) hyperthermia cancer treatment [3] [8] [9] [10]. These are the four most widely used applications the field.

2.2 Biosensing

Biosensing is a method that is used for the detection of target molecules called biomarkers, and has the potential to diagnose cancer early. Early detection based on genetic and proteomic profiles of biomarkers are key to improving the survival rates of cancer patients [11]. Osterfeld et al. point out that although there are thousands of biomarkers available, there are few biomolecular techniques capable of detection of protein biomarkers that can be readily adopted in clinical settings for personalized diagnosis and treatment [11]. Previous research has established that ferrite nanoparticles doped with Mn, Co, or Ni, as well as iron core magnetic nanoparticles, exhibit desirable characteristics for use in biosensor applications [9] [12] [13].

Osterfeld et al.'s comprehensive study in 2008 of several research groups found that magnetic nanotag (MNT) based protein assay technology is a highly sensitive alternative to optical biosensing technology [11]. Osterfeld et al. conducted experiments which

demonstrated MNT-based biomolecular assays can quickly and sensitively detect multiplex proteins in real world serum samples [11]. Osterfeld et al. labelled the target analyte with MNTs, for analyte detection to occur using giant magnetoresistance (GMR) sensors [11]. Spin valve (SV) sensors are GMR based sensors which have been developed for use in hard drive disks and have proven to be reliable [14]. In the experiment, an antibody was attached to the sensor surface, and on a separate site, a biotinylated detection antibody binds to the antigen [11]. Finally, superparamagnetic nanoparticles (SPMNPs) were bound to the antibody to generate a change in magnetization of the top free-layer [11]. The SV sensors were then able to detect this.

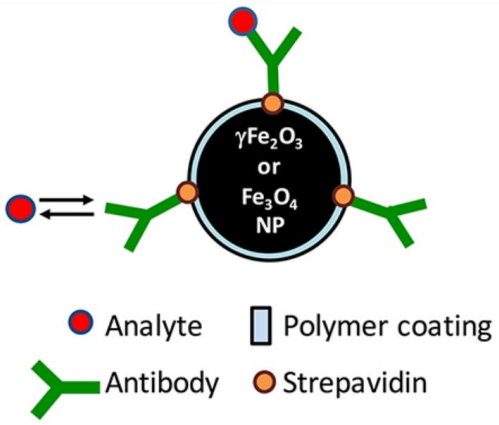

Figure 2.1: Modification of SPMNPs for analyte detection. SPMNPs can be covalently bonded with antibodies, in a process called chemical functionalisation, for a specific analyte. Reproduced from [15].

Osterfeld et al. detected multiple potential cancer markers with high sensitivity and fast detection in subpicomolar concentrations [11]. This result is supported by Ha et al., who explain that the particles have the potential to be detected in very dilute concentrations of analyte, due to their extremely large surface area per mass [15].

A more recent study in 2018 by Ha et al. built on the previous studies of MNT for the use of SPMNP for rapid diagnostic tests [15]. When SPMNPs are exposed to a controlled external magnetic field, they take on the role of "nano-magnets" and gain a number of useful characteristics, including: (i) the ability to move in a specific direction, (ii) the ability to be detected remotely, (iii) the ability to produce intense heat in a small area; and (iv) the ability to instantly and completely lose their magnetism when the external magnetic field is removed [15]. Additionally, Ha et al. note the SPMNP surface can be chemically functionalised without affecting the magnetic properties of the particles [15]. This allows for the manipulation of the particles for sensing, collection, and other purposes [15].

Almost every paper that has been written on MNPs includes a section relating to the technologies employed to detect the particles in a magnetic biosensor. A broad

perspective of sensing devices has been adopted by Ha et al. who note a wide range of devices to detect signal [15]. Reported devices include voltammetric, electro-chemiluminescent, superconducting quantum interference device (SQUID) sensors, nuclear magnetic resonance (NMR), surface plasmon resonance (SPR), tunneling magnetoresistance (TMR) and GMR [15]. Ha et al. have noted that recent studies have detected analytes such as cancer biomarkers and food allergens using GMR sensors developed by Osterfeld et al [15] [11].

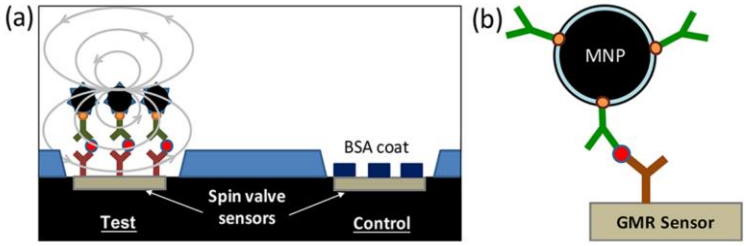

Figure 2.2: Spin-valve GMR sensor for analyte detection. (a) A capture antibody specific for the target analyte is initially functionalised to a test sensor. Next the antigen is captured and the magnetic signal from the immobilised SPMNPs enables quantification of the analyte concentration. In order to identify background signal brought on by nonspecific binding, bovine serum albumin (BSA) is connected with a separate control sensor. Reproduced from [15].

However, the main sensors in these studies are GMR devices and SQUIDs. GMR devices are also very sensitive and can be readily miniaturized to the nanoscale. SQUID is more sensitive alternative to GMR devices, where the best sensitivity is found at cryogenic temperatures, where magnetic fields as small as $10{-}15\ T{\cdot}Hz^{-1/2}$ can be detected [8]. These technologies can provide data about the MNP's, include relaxation decay signal discussed in Section 3.1. Overall, these processes can provide accurate detection of various biomarkers that relate to different cancers.

Pankhurst explains that the properties of magnetic nanoparticles, such as size, shape, and magnetic moment, are crucial for their performance in biomedical applications [3]. In fact, many study agree that size, shape and particle assembly configuration effect the magnetic signal, sensed by GMR and SQUID devices [8] [9] [11] [16].

Pankhurst et al.'s review of the biosensing concluded that the main challenges of MNP applications is within the issue of technology transfer [3]. The aim of technology transfer is to impart the knowledge of both product and process from development to manufacturing in order to achieve successful product realisation. However, Pankhurst et al. fail to fully define what steps need to be taken to ensure the laboratory experiments can be emulated in vivo. On the other hand, Farzin et al.'s analysis of the subject clearly explains the important challenges that prevent the proper treatment of cancer [9]. Farzin et al. have identified a gap in research of biosensing; they suggest that various MNPs could possibly

be differentiated within the body using imaging techniques to enable the rapid phenotyping of cancer cells [9].

3 Background

3.1 Nanoparticle Physics

This section covers types of data relating to the physics of MNPs that can be measured by technologies discussed in Section 2.2. The types of potential data that can be measured are: (i) M-H curves and (ii) magnetic relaxation.

3.1.1 M-H Curves

Firstly, it is important to note that in biomedical applications of MNPs, the magnetic signal from the injected particles, regardless of their size, is much stronger than that from the blood vessel, which provides a heightened selectivity advantage [3].

MNPs follow Coulomb's law and can be controlled by an external magnetic field gradient.

$$B = \mu(H + M) \quad (1)$$

Eq.(1) tells us that the total magnetic flux density (B) in a material is equal to the sum of the magnetic field strength (H) and the magnetization (M), multiplied by the permeability of the material (μ). Magnetization (M) is the magnetic moment per unit volume [3].

The magnetic response of MNPs magnetic depends on their atomic structure and temperature [3]. To determine the response of MNPs to external magnetic fields, MNPs can be engineered to have specific magnetic properties, such as superparamagnetism or ferromagnetism [3].

Ferromagnetic materials exhibit ordered magnetic states and retain magnetism in the absence of an external field [3]. The magnetic moments of the atoms or ions in ferromagnetic materials are aligned in a particular direction, resulting in a net magnetic field [3]. Ferromagnetic materials naturally exhibit this alignment of magnetic moments below a temperature known as the Curie temperature Tc, and an external magnetic field can also induce this alignment [3]. The magnetic susceptibility of ordered materials is influenced by both temperature and H resulting in the sigmoidal shape of the M-H curve [3]. The magnetization process in ferromagnetic materials often exhibits hysteresis, a phenomenon linked to intrinsic effects like magnetic anisotropy of the crystalline lattice that results in irreversibility [3].

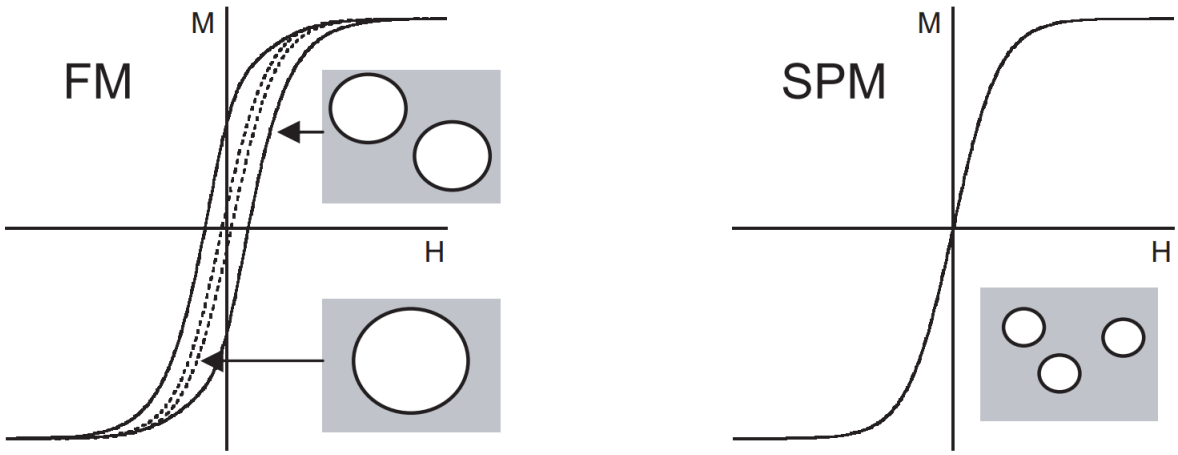

Figure 3.1: M–H curves called hysteresis loops are presented. The shape of these loops is determined in part by particle size. In superparamagnetism (SPM), small particles have an anhysteretic response. M approaches saturation at high H values for both cases. Reproduced from [3]

$$\Delta E = KV \qquad (2)$$

Eq. (2) relates the energy barrier (ΔE) required for magnetic moment reversal to the anisotropy density (K) and particle volume (V). This direct proportionality between ΔE and V is the reason that superparamagnetism is important for small particles. Since ΔE is comparable to the thermal energy at room temperature, this causes the thermally activated flipping of the net moment direction [3].

The observations of superparamagnetism are implicitly dependent on temperature and the measurement time τ_m of the experimental technique being used and can be seen below in Figure 3.2.

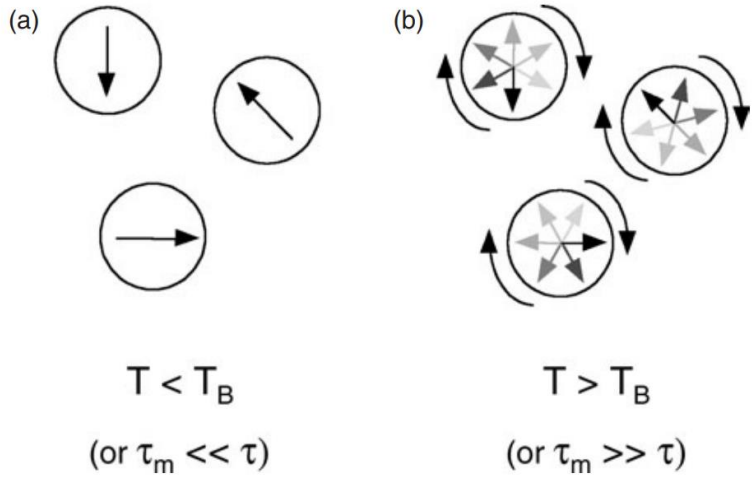

Figure 3.2: The concept of superparamagnetism is illustrated using three magnetic nanoparticles represented by circles, with the net magnetization direction indicated by arrows in those particles. In (a) the flipping of moments is slow and quasi-static properties are observed. In (b) the moment reversals are so rapid that in the absence of an external field, the average net moment of the particles over time is zero. Reproduced from [3].

Hysteresis in ferromagnets requires energy to overcome the barrier to domain wall motion caused by intrinsic anisotropy, impurities, and grain boundaries in the material [3]. A similar energy transfer argument applies to superparamagnetic (SPM) materials, where energy is required to align the particle moments coherently and reach the saturated state [3].

3.1.2 The Relaxation of Magnetic Nanoparticles

After a ferrofluid is removed from a magnetic field, its magnetization relaxes to zero due to ambient thermal energy. This relaxation can be attributed to either Brownian rotation which is the rotation of particles, or Neel relaxation which is the rotation of atomic magnetic moments. Each process has a relaxation time, τ_B for Brownian process and τ_N for Neel process, determined by hydrodynamic properties of the fluid and magnetic anisotropy energy of the SPM particles, respectively.

SQUID immunoassays use immobilized antibodies to capture and detect antigens via coated SPMNPs (see Section 2.2). When the field is turned off, free SPMNPs cannot be detected due to random orientation of dipole moments resulting from Brownian motion, while immobilized SPMNPs undergo slow relaxation via the Neel mechanism, producing a decaying magnetic signal [17] [18] [15].

Once the sample volume is magnetized, the measured magnetic field signal at a specified position is contributed by all magnetic nanoparticles within that volume [8]. Therefore,

magnetorelaxometry reflects all magnetic nanoparticles in the sample that undergo relaxation within the observed time window [8].

Data used for my project is data is computationally generated relaxation data, M(t), based on research of Laslett et al. [16]. Laslett et al suggest that the magnetization data is fitted by a stretched exponential profile [16] [19].

3.2 Machine Learning Methods

3.2.1 Machine Learning Algorithms

There are many types of machine learning algorithms. I used two main types of machine learning in this project: supervised and unsupervised learning. Supervised learning uses labelled datasets to train algorithms that classify data or predict outcomes accurately [20,pp.7]. I will be using multi-class classification methods that attempt to classify data based on chosen features. There are two main types of classification algorithms: inherently multi-class algorithms and binary algorithms adapted for multi-class classification [20]. Within binary algorithms adapted for the multi-class, there are two methods: One-vs-Rest (OVR) and One-vs-One (OVO) [20,pp.100] [20]. OVR classifiers split the dataset into multiple binary classification problems. For each class, a binary model that will categorise the data as either belonging or not belonging to that class. OVO classifiers split the dataset into many classifiers, with one classifier of every possible pair of classes. The class with the most votes is then chosen as the final prediction. The OVO approach can handle class imbalance and can also provide a more fine-grained understanding of the relationships between the classes. However, it can be computationally expensive, especially when the number of classes is large.

Unsupervised learning is a type of machine learning algorithm, often known as clustering, in which the model is not given any labelled or target data to train on. Instead, the model is left to discover patterns and relationships in the input data on its own. This can be useful for tasks such as clustering, where the goal is to group similar data points together. Unsupervised learning algorithms will be used in combination with supervised learning algorithms to improve the overall performance of the model [20,pp.236].

3.2.2 Logistic Regression

The first machine learning tool I will use for my project is logistic regression. Logistic regression is an inherently binary classification algorithm that can be adapted to the multiclass problem using OVR. It uses a sigmoid function to predict the probability that a given feature belongs to each of the possible classes. The class with the highest predicted probability is then chosen as the predicted class for that data point. [20,pp.142]

3.2.3 Support Vector Machines

Support vector machines (SVMs) are a type of machine learning algorithm that aim to find the optimal boundary that separates different classes of data points. SVMs employ an OVO strategy which leads to differences with Logistic Regression. One difference is that SVMs are less sensitive to outliers in the data, therefore SVMs can be more robust. Another difference is that SVMs can handle complex datasets better, however this comes at the expense of computational complexity [20,pp.153].

3.2.4 Decision Trees

Decision Trees are a machine learning algorithm that constructs a tree-like model of decisions and outcomes. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. The algorithm recursively partitions the data into subsets, based on the most informative features, until the tree is complete. An advantage of Decision Trees is that they are a non-parametric algorithm and do not make assumptions about the underlying distribution of the data. However, Decision Trees can be prone to overfitting, particularly when the tree is too deep, leading to poor generalization performance on unseen data [20,pp.175].

3.2.5 Random Forest

Random Forest is an algorithm that uses an ensemble of Decision Trees to make predictions. It can handle high-dimensional and non-linear datasets while reducing overfitting. It is a more powerful algorithm than Decision Trees and provides a higher level of accuracy, but it is much slower [20,pp.197].

3.2.6 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a simple but powerful, non-parametric algorithm used for classification tasks. The algorithm works by identifying the K nearest neighbors in the training dataset to a new data point based on a distance metric. The predicted value of the new data point is then determined by the most frequent class. It can also be sensitive to the choice of distance metric and the scaling of features. KNN is often used as a benchmark for more complex algorithms such as SVM and Neural Networks [21].

3.2.7 Neural Networks

Neural networks are a type of machine learning model that imitates the human brain. They consist of interconnected nodes called neurons that process and transmit information. The strength of the connections between neurons is determined by weights, which are optimized during training to improve accuracy. They are trained using large amounts of labelled data and have become popular for their ability to handle complex data and achieve state-of-the-art performance on many tasks. Neural networks can learn patterns and relationships in data that may not be easily detected by humans or traditional machine

learning algorithms. However, Neural networks can be computationally expensive and there is a risk of overfitting, where the model performs becomes too specialized to training data, resulting to poor performance on new unseen data [20,pp.279].

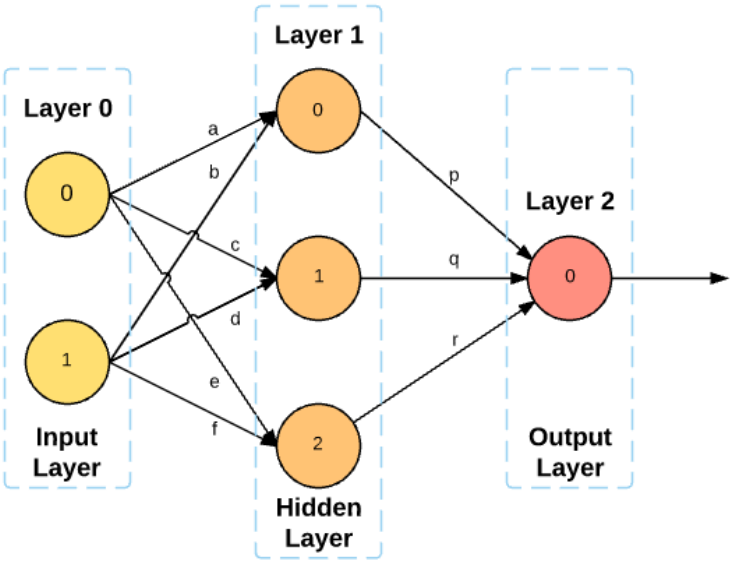

Figure 3.3: An artificial neural network consisting of 3 layers of interconnected group of nodes. Each circular node represents a neuron, and an arrow represents a connection from the output of one neuron to the input of another. Letters a-f and p-r represent layer 1 and 2 connection weights respectively.

3.2.8 K-means

K-means is an unsupervised clustering algorithm that is used to partition a given unlabelled dataset into K clusters, where K is a user-defined parameter. K-means iteratively assigns each data point to the closest centroid and recomputing the centroid based on the mean of the points assigned to it. One limitation of K-means is that it is sensitive to outliers, which can affect the position of the centroids and the quality of the clusters. Another limitation is the need to specify the number of clusters K [20,pp.238].

4 Methodology

4.1 Project Pipeline

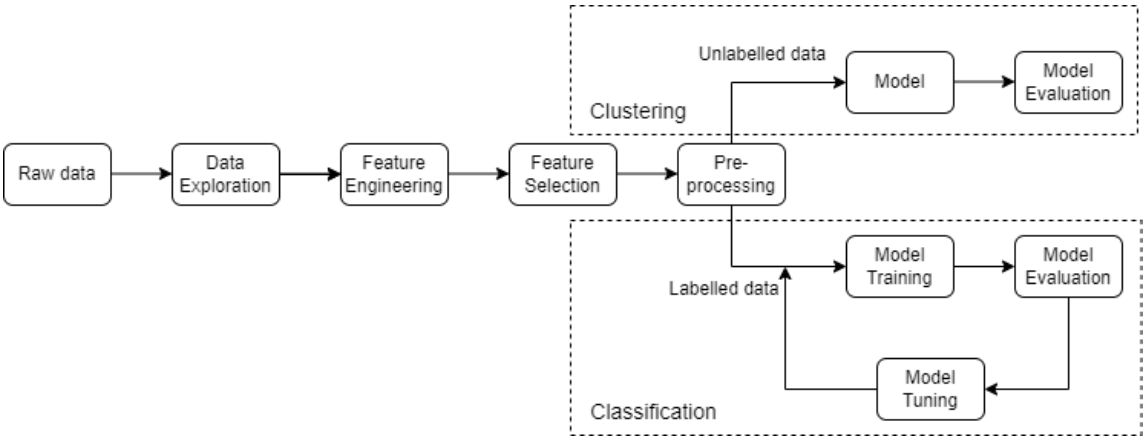

Figure 4.1: Machine learning flow diagram

4.2 Data Acquisition and Understanding

Data was computational since there wasn't sufficient experimental data to train the model. However experimental data could potentially be used to test the computational data. The data used was a dataset containing 42 H5 files; a format used to store large amounts of data in the form of multidimensional arrays. The H5py library was used for exploratory analysis of these h5 files [22].Each of the 42 files contained 1000 datasets; each dataset consists of a 100 x 2 array. The data in this array contains magnetisation values varying over time described in Section 3.1.2. There were 5 datasets with varying randomness in anisotropy axis direction and anisotropy values. The following table shows shared properties of clusters from the computationally generated data.

Table 4.1: Shared properties for all datasets

Property	Value
Grain size	100.0 [A]
Sigma size	0.0 [%/100]
aniso	1.0e6 [erg/cc]
Sigma aniso	var [%/100]
Mag satur	400.0 [emu/cc]
temp	300.0 [K]
Hext satur	8.0e3 [Oe]
Hext relax	0.0e3 [Oe]
Hext delta	5.0 [Oe]
Hext x	0.0 [unit_vector_comp]
Hext y	0.0 [set_0-0-0_for_rand]
Hext z	0.0

Aniso axis	0 or 1 [0-or/1-ra/2-cone]
Num rand rea	1000

Properties that are different between datasets:

Set 1 - clusterdata_1000rea_20200318_reduced, the primary dataset for this experiment:
- all particles in a cluster having anisotropy axis oriented in the z-direction
- constant anisotropy value
- the initialising external field oriented in (spherically) random directions

Set 2 - clusterdata_1000rea_20200109_reduced:
- with (spherically) random anisotropy axes of particles
- constant anisotropy value
- the initialising external field oriented in (spherically) random directions

Set 3 - clusterdata_1000rea_20210406_reduced:
- all particles in a cluster having anisotropy axis oriented in spherically random directions
- random anisotropy value with 1% variation
- the initialising external field oriented in (spherically) random directions

Set 4 - clusterdata_1000rea_20210409_reduced:
- all particles in a cluster having anisotropy axis oriented in the z-direction
- random anisotropy value with 1% variation
- the initialising external field oriented in (spherically) random directions

Set 5 - clusterdata_1000rea_20210416_reduced:
- all particles in a cluster having anisotropy axis oriented in the z-direction
- random anisotropy value with 10% variation
- the initialising external field oriented in (spherically) random directions

4.3 Data Preparation

4.3.1 Feature Engineering

Feature engineering is a method employed in the study, that involves using domain knowledge of the data to create features that can make machine learning algorithms work more effectively. Effective feature engineering can help improve the accuracy, speed and interpretability of machine learning models, and can help improve the performance of supervised and unsupervised learning algorithms [20,pp.28]. Features were the input to machine learning models, so the selection and quality of the features had a major impact

on the quality of insights gained from a model. In the context of my project, the ideal features should allow the model to accurately distinguish clusters in all the datasets.

The first feature I chose was the value of time (t), when the magnetisation (M) had dropped by half. The first step of calculating this value was finding the calculated this value by finding the mid-point of M. Then interpolated the data to find an estimate of the unknown points. I then inversed the interpolated function to find the value the t values at the mid-point of m. I repeated this process the time at points one third and two thirds m. I then iterated this calculation through every array, in every dataset, in every file. I saved these values to a csv file along with the label of the particle structure. This process can be seen in code below.

```
m1 = y[0]*0.5       # midpoint of magnetisation
m2 = y[0]*0.333       # 1/3 of magnetisation
m3 = y[0]*0.666       # 2/3 of magnetisation
y_values = [m1, m2, m3]     # list of y (magnetisation) values


interp_fn = interpolate.interp1d(x, y, 'quadratic')     # create the interpolated function
f2 = interpolate.interp1d(y, x, bounds_error=False)     # inverse interpolate function
x_values = f2(y_values)
```

The central difference approximation estimates the derivative of a function at a point by using neighbouring points. The following code shows how the derivate was calculated.

```
def fprime_central(f, x0, h=1e-1):
    """
    Returns derivative of function f(x) evaluated at point x0
    using the central difference approximation with step h.
    """
    return (f(x0+h) - f(x0-h)) / (2*h)

grad = fprime_central(interp_fn, m1)
```

The stretched exponential is a common model for describing relaxation data [16]. The following code shows the function implemented to fit the data.

```
def st_exp(x, c, tau, beta, y_offset):
        return c*(np.exp(-(x/tau)**beta))+y_offset

best_vals, covar = curve_fit(st_exp, x, y, maxfev=3000)
```

4.3.2 Feature Selection

Feature selection is a process in machine learning where the most relevant and informative features are chosen from a set of variables to improve the performance if a model and reduce computational complexity. The F value is often used as a metric for feature selection, where features with higher F values are deemed more important for predicting the target variable. The F value is used in the analysis of variance (ANOVA), where the F value is calculated by dividing two mean squares. It determines the ratio of explained variance to unexplained variance as shown in Eq. (3) [23].

$$F = \frac{Variance\ between\ groups}{Variance\ within\ groups} \quad (3)$$

By selecting features with high F values, we can reduce the number of input features while maintaining the classification accuracy, which can help to reduce model complexity. Additionally, F value-based feature selection methods are computationally efficient and can handle high-dimensional data. Therefore, I calculated the F value for all the features described in Section 4.3.1.

4.3.3 Data Pre-Processing

Data cleaning is an important stage in data preparation, and involves identifying and correcting or removing inaccurate, incomplete, or irrelevant data in a dataset. Pandas, a popular Python library for data manipulation, was used throughout the machine learning process to pre-process and manipulate the input data [24]. The goal of pre-processing is to improve the quality and reliability of the data and ensure that it is suitable for analysis or modelling purposes. For that reason, I removed null values in the data. Furthermore, the data had to be split into features and labels, which the following code shows [20,pp.62].

```
X = df.drop(columns = ['Structure']) #features
y = df['Structure'] #labels
```

The supervised machine learning models were performed using sklearns train_test_split function [25]. This method involves randomly dividing the available data into two separate sets: the training set and the testing set. The training set was used to fit the machine learning model, to estimate the parameters of the model using the available data. The testing set was used to evaluate the performance of the model on new, unseen data. This was done to assess how well the model generalises to new data and to avoid overfitting, where the model learns the training data too well and performs poorly on new data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = (1))
```

I used sklearns train_test_split function which randomly selects a proportion of the available data to be used as the training set and the remaining proportion as the testing

set. A common split is to use 70-80% of the data for training and the remaining 20-30% for testing depending on the size of the dataset. The split must be at an optimal value in this trade-off, using a larger proportion for training provides the model with information and may lead to better performance on the training set. However, if too much data is used for training, the model may overfit to the training data, meaning it learns the specific characteristics of the training data and performs poorly on new, unseen data. In this case, the model has not learned the general patterns in the data but rather memorized the training set. On the other hand, using a larger proportion of the data for testing allows for a more reliable estimate of the model's performance on new, unseen data. However, if too little data is used for training, the model may not have enough data to learn the underlying patterns in the data, leading to poor performance on both the training and testing sets.

I chose a split of 80:20; when scaled to the dataset of 42,000 provided a split of 33,600:8,400. As shown here, 8,400 data samples appears to be a large enough value for the testing set.

Scaling is a crucial step in data pre-processing that involves transforming the numerical values of a dataset to a common scale. Scaling helps to prevent the model from overweighting some features over others, which can lead to better model performance and reduce the risk of overfitting. I used sklearn's StandardScalar to transform the data to have a mean of 0 and a standard deviation of 1. Feature Scaling is necessary for Gradient Descent Based algorithms (Logistic Regression, Neural Networks) and Distance-based algorithms (KNN, K-means, SVM) because these algorithms are highly affected by the range of the data points. Therefore, scaling the features helps to ensure that these algorithms can learn and generalise well [20,pp.69].

For supervised machine learning models, it was also important that the scaler was only applied to the training data. This was to reduce leaking of the testing data, since knowledge of the distribution of the testing data would be known, which is not representative of a real-world example. Therefore, the same transform applied to the training data was blindly applied to the test set. The following script shows this.

```
scaler  = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)
```

4.4 Clustering

4.4.1 Model Development

The following code describes the development of K-means model, with k=3.

```
kmeans = KMeans(n_clusters=3)      # Create a K-means model with 3 clusters
```

```
kmeans.fit(data_scaled)        # Fit the model to the scaled data
labels = kmeans.predict(data_scaled)        # Predict the cluster labels for each data point
X['cluster'] = labels     # Add the cluster labels to the DataFrame as a new column
```

4.4.2 Clustering Metrics

The elbow method and silhouette score are two metrics commonly used to evaluate the performance of unsupervised clustering algorithms such as K-means.

The elbow involves plotting the within-cluster inertia as a function of the number of clusters and looking for an "elbow" in the curve, which is a point where the rate of decrease in inertia slows down significantly. Inertia is the sum of squared distances between the points and their assigned centroid to determine how compact the clusters are. Higher inertia denotes looser, more dispersed clusters, and lower inertia denotes clusters with small variances. Typically, the k at which the plot starts to flatten out is used as the optimal number of clusters. However, inertia alone may not be sufficient to assess a clustering algorithm's performance because it might not accurately reflect the quality of the cluster separation [20,pp.246].

On the other hand, silhouette score quantifies how well-separated the clusters are. The values range from -1 to 1, with higher values suggesting greater clustering performance. A number close to 1 denotes the points are tightly concentrated and separated from nearby clusters, whereas a score close to 0 means the clusters are in close proximity. A negative score indicates that the points might have been assigned to the incorrect cluster [20,pp.247].

In summary, silhouette score and inertia provide complementary information to evaluate the quality of the clusters produced by a clustering algorithm. Silhouette score measures the separation between the clusters, while inertia measures the compactness of the clusters. Therefore, both methods were used.

4.5 Classification Model Development

First, I built models using sklearns library with default hyperparameters and fit with train data and iterated through all models for all sets. Parameters of all functions can be found in sklearns documentation [20].

```
lr  = LogisticRegression(multi_class='ovr', random_state=1)
svc = SVC(probability=True)
knn = KNeighborsClassifier()
rfc = RandomForestClassifier(random_state=1)
dtc = DecisionTreeClassifier(random_state=1)
mlp = MLPClassifier()
```

```
models_list = [lr, svc, knn, rfc, dtc, mlp]

for model in models_list:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)
```

4.6 Model Evaluation

4.6.1 Classification Metrics

There are many ways to measure the performance of a machine learning model. Accuracy tells us the number of correct predictions made out of the total number of predictions. Accuracy is a simple way to measure the baseline performance of models. However, accuracy doesn't provide any insights for how different classes are performing.

Table 4.2: Confusion Matrix.

Actual \ Predicted	0	1	2
0		FP	
1	FN	TP	FN
2		FP	

A confusion matrix is a table that is used to evaluate the performance of a classification algorithm. Table 4.2 shows the confusion matrix completed for Class 1. For a given class, e.g., class 1, let TP be the number of true positives, the number of structures that the classifier correctly predicts as class 1. Let FP be the number false positives, the number of structures that are incorrectly predicted as class 1. Let FN be the number of false negatives, the number of structures that are actually class 1, but the classifier predicted as another class. The entries in the confusion matrix can be used to calculate various performance metrics, such as precision, recall, and accuracy, for each class [20,pp.90].

$$Precision = \frac{TP}{(TP + FP)} \tag{4}$$

Precision is defined as the percentage of correct positive predictions out of all positive predictions. Precision ranges from 0 to 1 with 1 as the ideal value. Therefore, for class 1, the denominator is calculated by sum of values in the corresponding column, outlined in yellow, in Table 4.2.

$$Recall = \frac{TP}{(TP + FN)} \tag{5}$$

Recall is defined as the percentage of correct positive predictions out of all actual positives. Recall ranges from 0 to 1 with 1 as the ideal value. For class 1, the denominator is calculated by sum of values in rows, outlines in blue, in Table 4.2.

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (6)$$

F1 score is an error metric that varies between 0 and 1, where 1 is the best possible score. F1 score is the harmonic mean of precision and recall. The F1 score is a useful metric because it balances precision and recall, giving equal weight to both. A high F1 score indicates that the model is making a high number of true positive predictions, while also minimizing the number of false positive predictions. Recall and Precision are both as equally important, therefore F1 is the primary testing metric, accuracy was also used [20,pp.93].

4.7 Model Tuning

Hyperparameter tuning is the process of selecting the best hyperparameters for a machine learning model to optimize its performance on a specific task. Hyperparameters are parameters that are set before training a model and cannot be learned from the data. Examples of hyperparameters for neural networks include the number of neurons in each layer, the activation function for the hidden layer and the solver for weight optimaztion. These hyperparameters have a significant impact on the performance of the model, and selecting the optimal values for them is crucial for achieving the best performance. Grid search is a method that involves creating a grid of hyperparameters to search over, and training and evaluating the model for each combination of hyperparameters in the grid [20,pp.31]. A range of hyperparameters were defined for each model, and labelled params1-6, shown in the code below and in 8.7 Appendix G.

```
params = [params1, params2, params3, params4, params5, params6]

for param in params:
    grid = GridSearchCV(pipe, param, cv=5)
    grid.fit(X_train, y_train)
    print('Score:', grid.best_score_)
    print('Score:', grid.best_params_)
```

This code returns the best parameters and allows for the model to be re-evaluated.

5 Results

5.1 Feature Selection

Data exploration is important for feature selection because it helps identify relationships between variables and potential predictors, leading to an informed and effective feature selection process (see Section 4.3.2). Exploratory data analysis on the h5 files showed that the x-values (time) had a range over several orders of magnitude of time. Consequently, I logarithmically transformed time. Figure 5.1 below shows the magnetisation plotted against log time.

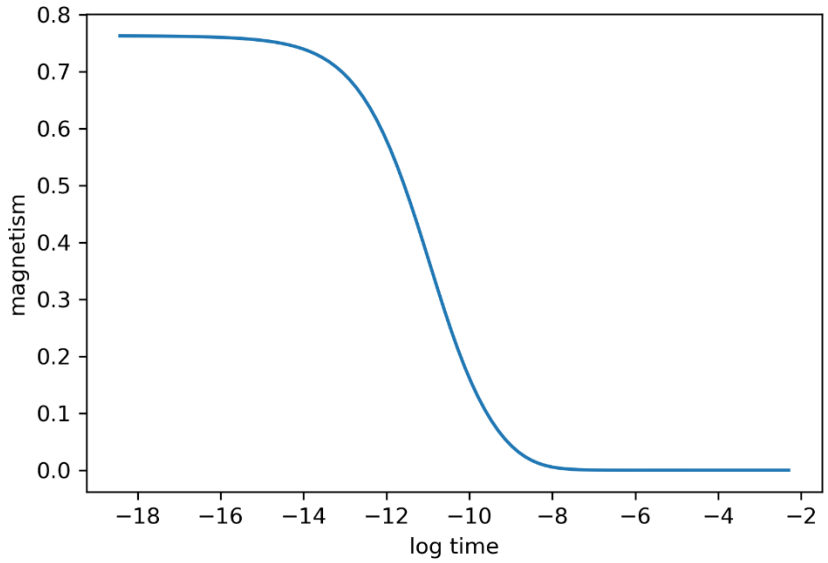

Figure 5.1: Magnetism vs Log Time for a randomly chosen dataset, 7p5 arkus. Curve represents relaxation of particle, called relaxation curve as discussed in Section 4.1.

Figure 5.1 presents magnetism plotted against log time for a randomly chosen dataset. It is known as the relaxation curve and represents the time taken for the magnetisation to drop to zero. The relaxation curve in Figure 5.1 is as expected, (more detail in Section 3.1.2).

Table 5.1: ANOVA F-values for features.

Feature	**F Score**
t-half	24096.87
beta	68.25
gradient	67.12
tau	29.60
c	2.21
y-offset	1.60

Table 5.1 illustrates a comparison of different features using the ANOVA F values described in Section 4.3.2. It can be seen from the data in Table 5.1 that t-half has the highest score, so is the best feature. Eq. (3) explains that for t-half, there is high variance between the different structures and low variance within the structure groups. Since the F score for t-half was significantly higher than the other features, the other features were dropped because they could lead to poor accuracy and overfitting.

As shown in Table 5.1, gradient has a low score and therefore didn't work well as a feature because the values were inconsistent. On the other hand, the stretched exponential fit well on some of the data, so some of the output data was partially consistent, however there were a lot of outliers. Another feature that was engineered was t1/3, t2/3, and visualised in the figure below.

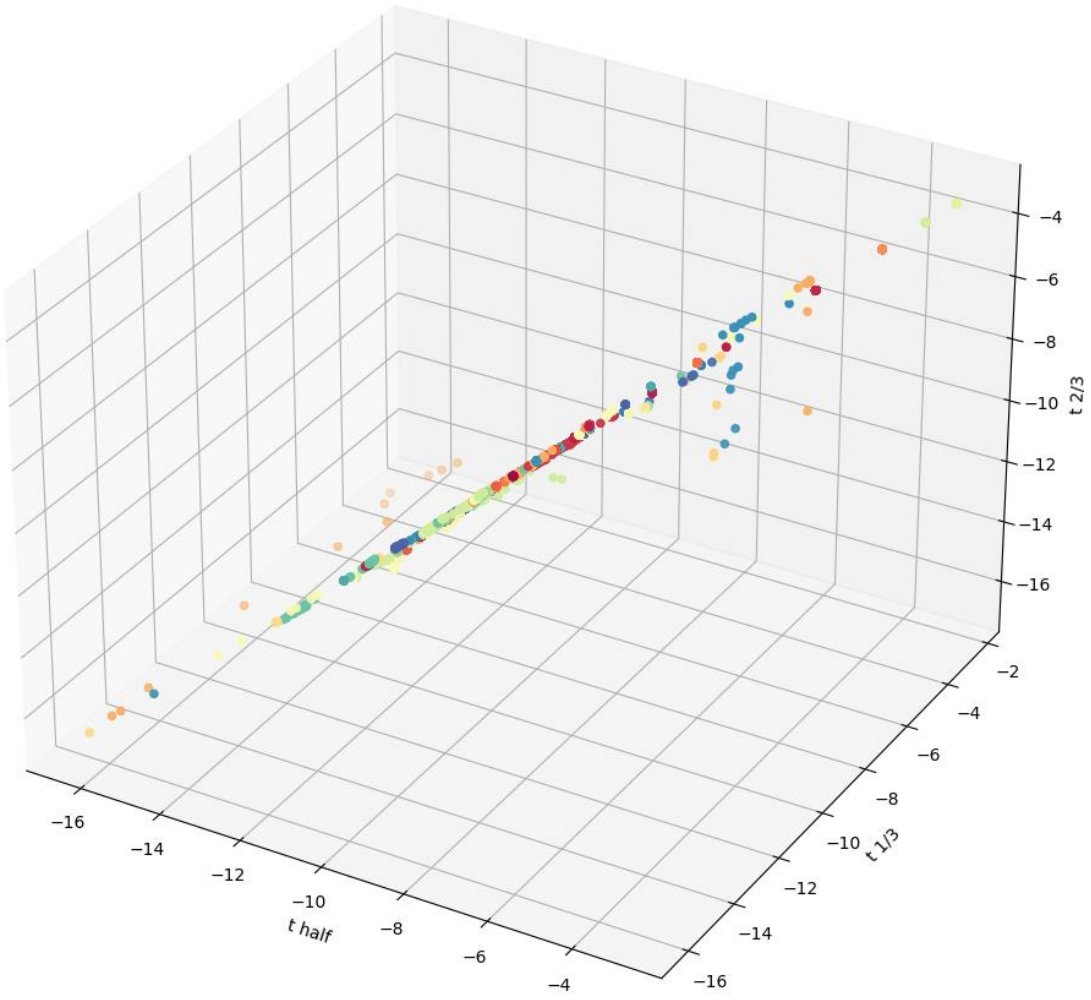

Figure 5.2: 3D plot of features t half, t1/3, and t2/3 for Set 1. Different colours represent different particle clusters. There is a strong correlation between these three features. Units of the axes is the log of time (s).

Figure 5.2 shows that these features, when plotted with t-half, were highly correlated which means that they could introduce bias if used together. Using all three features together causes no new insights to be gained from this data. In an ideal world, the feature plot would contain 42 clear clusters that can be distinguished by eye.

Firstly, I noticed that during feature engineering, t 2/3 had a large number of null values. After further investigation, it was discovered that the relaxation curve had not fully relaxed and reached zero.

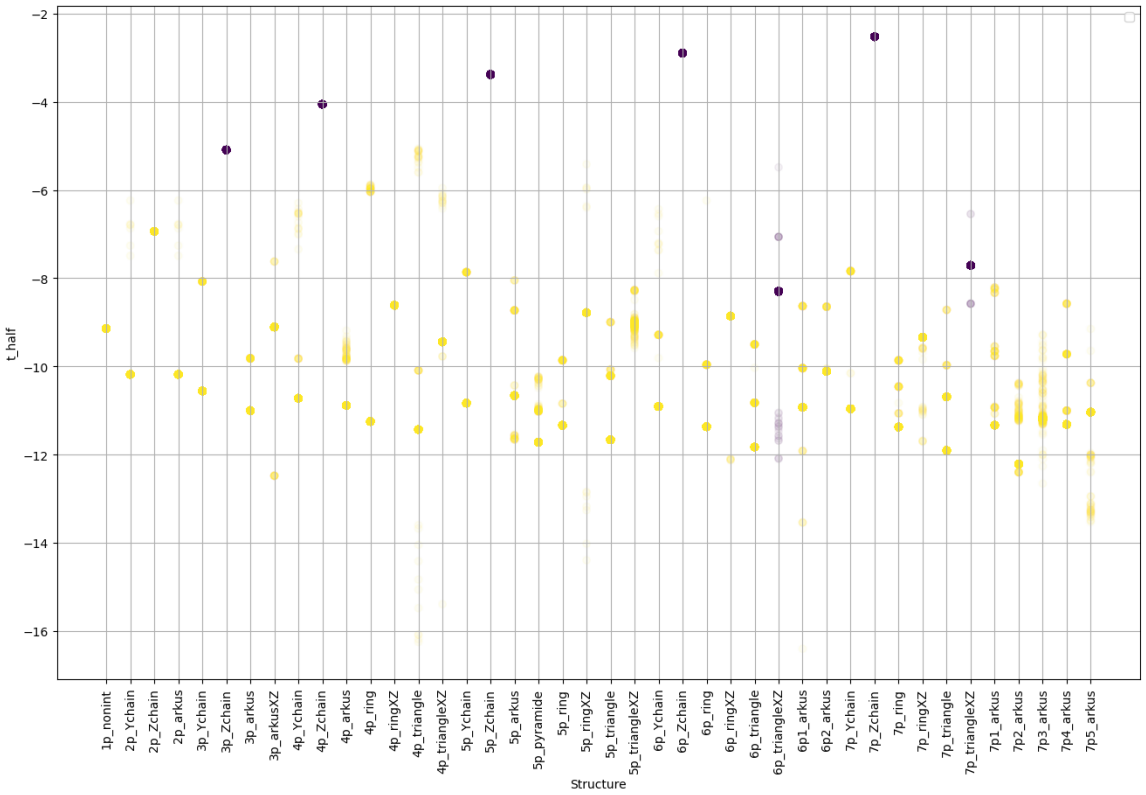

Figure 5.3: t-half plotted for each structure, representing whether the relaxation curve for each dataset had reached zero. Yellow represents the complete relaxation, while purple represents the incomplete relaxation. Alpha was set as 0.05, which specifies the opacity level of the data points to be 5%, as a visual aid to show density. Units of the y-axis is the log of time (s).

What is interesting about the data in Figure 5.3 is that the structures that have not fully relaxed are majority z-chain clusters. This is because the energy barrier of cluster is very high (more on energy barriers in Section 3.1).

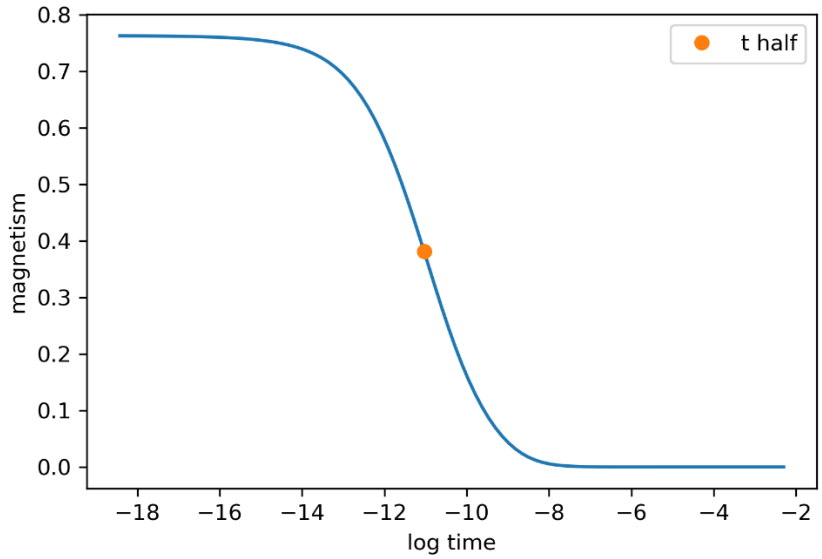

Figure 5.4: t-half plotted on Magnetism vs Log Time graph for the randomly chosen dataset, 7p5 arkus. The orange point represents the t-half value calculated. t-half is the time it takes for magnetisation to half.

Table 5.2: Statistical Description of Set 1 Data

description	t-half
count	41991
mean	-9.52
std	2.51
min	-16.4
25%	-11.2
50%	-10.7
75%	-8.86
max	-2.52

Table 5.2 provides a statistical description of all of the datapoints for t-half. 9 datapoints have been removed from the original 42000 for returning null values. One major limiation of this statistical description is the inability to compare different groups of structures.

Another way to explore the data is through a box plot visualization of t-half. Box plots graphically depict groups of numerical data based on their quartiles. The box shows the interquartile range (IQR), where 50% of the data is found. The median marks the mid-point of the data and is shown by the line that divides the box into two parts. Additionally, box plots may feature whiskers that extend vertically from the boxes, representing variability outside the upper and lower quartiles. More specifically, the lower whisker is the smallest value within 1.5 times the interquartile range below Q1 (25^{th} percentile) and the upper

whisker is the largest value within 1.5 times the interquartile range above Q3 (75th percentile). Any outliers are depicted as individual points and are located further away from the median than the whiskers.

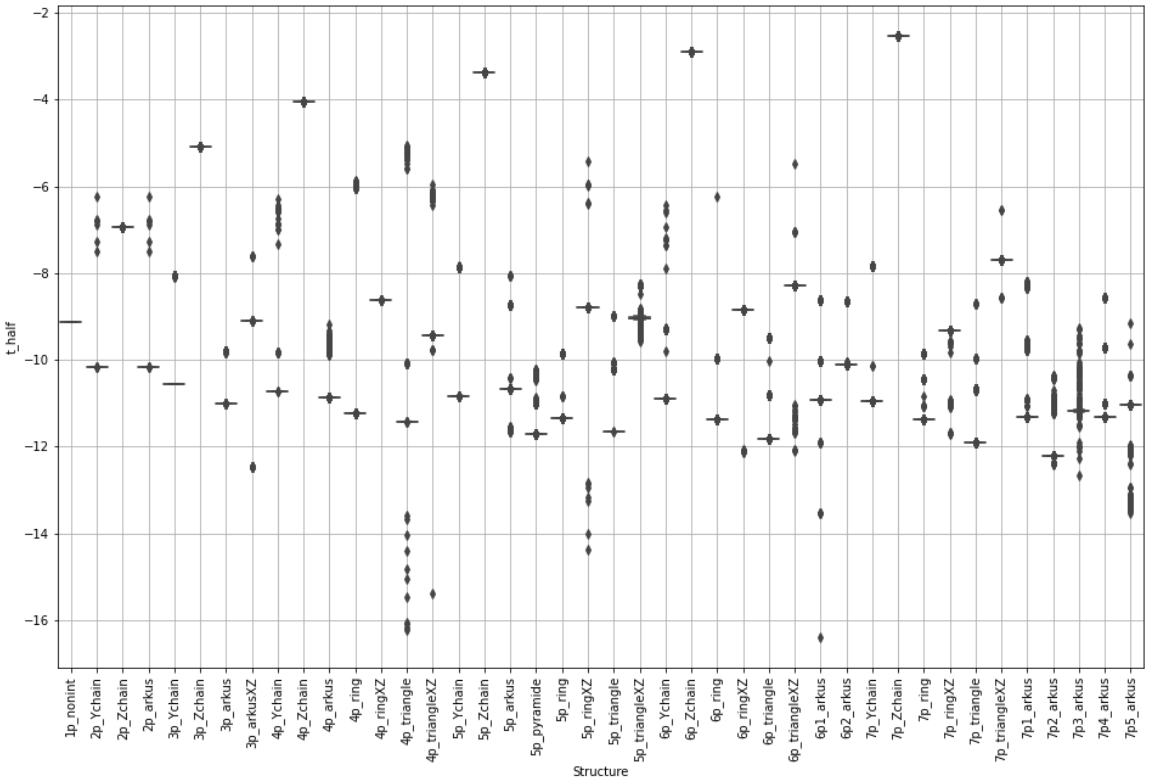

Figure 5.5: Box plot of t-half for different structures for Set 1. Horizontal lines represent the box, no whiskers are present due to the condensed data. All particles in a cluster have anisotropy axis oriented in the z-direction and a constant anisotropy value. Units of the y-axis is the log of time (s).

Figure 5.5 is a boxplot showing the all the t-half values for Set 1. In the figure, there are no whiskers, and the box is showing as a horizontal line because the majority or the data is condensed to a single point. Furthermore, this line can be interpreted at the median value. In the graph, the data points are tightly clustered together with minimal spread, indicating low variability within cluster groups. There is variation between the large majority of cluster groups. This is consistent with the findings from F value calculations in Table 5.1 and is further proof that t-half is a quality feature. Additionally, there are only a few data points that deviate significantly from the rest of the data, which indicates a low number of outliers.

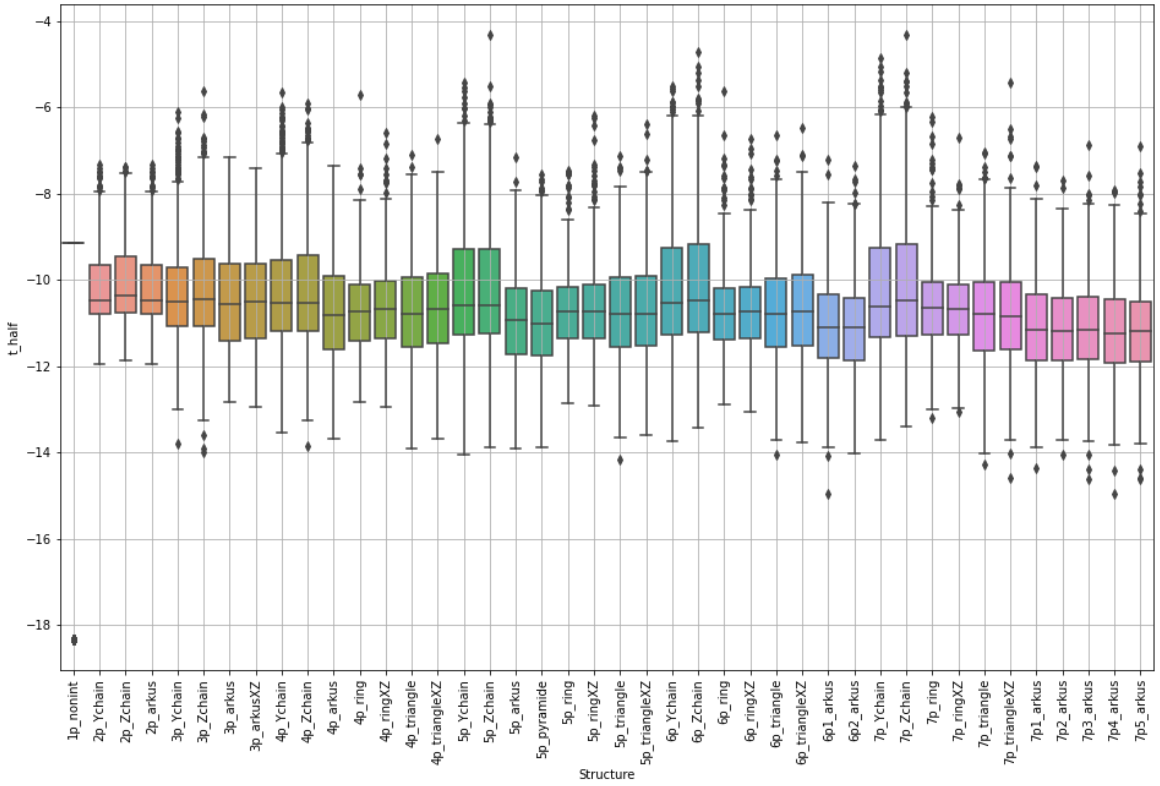

Figure 5.6: Box plot of t-half for different structures for Set 2. Different colours represent different particle clusters. Cluster contains particles with (spherically) random anisotropy axes and a constant anisotropy value. Units of the y-axis is the log of time (s).

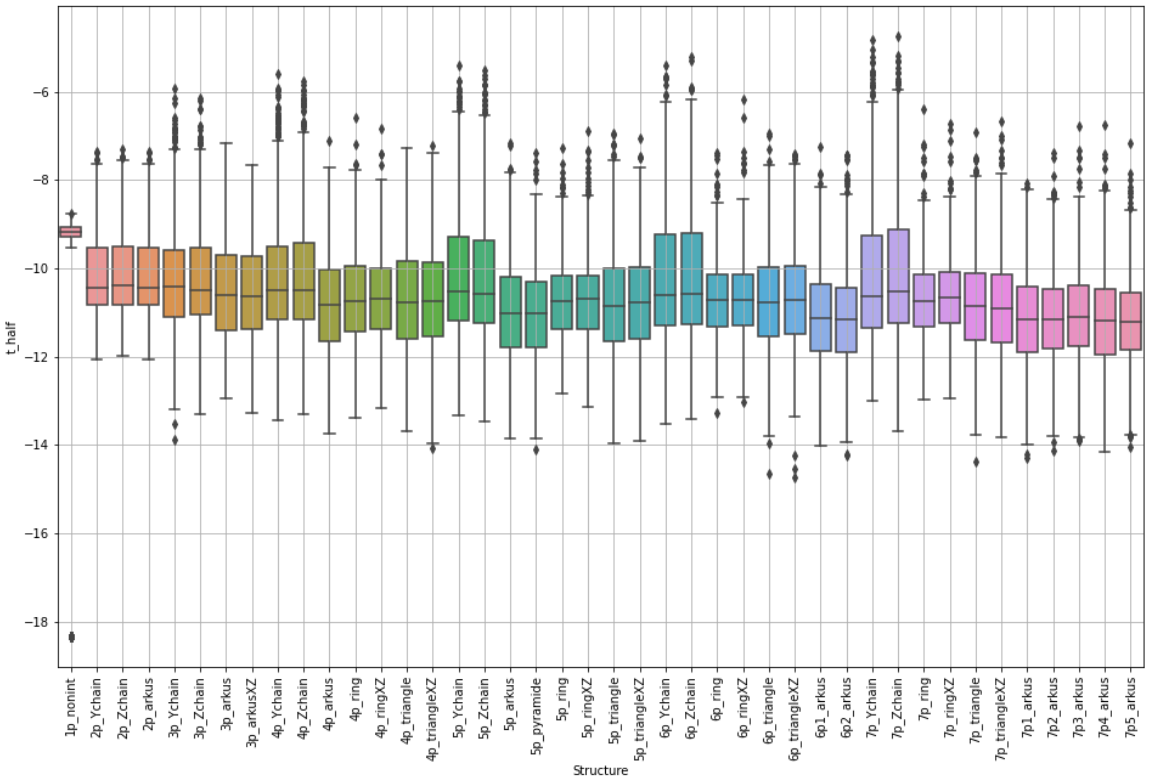

Figure 5.7: Box plot of t-half for different structures for Set 3. Different colours represent different particle clusters. All particles in a cluster having anisotropy axis oriented in spherically random directions and a random anisotropy value with 1% variation. Units of the y-axis is the log of time (s).

What stands out in Figure 5.6 and Figure 5.7 is that there are no distinct groups of particle clusters. Additionally, there is a large spread of data with all the median values of t-half situated at approximately -11. At first glance, it appears that both of these datasets are of low quality.

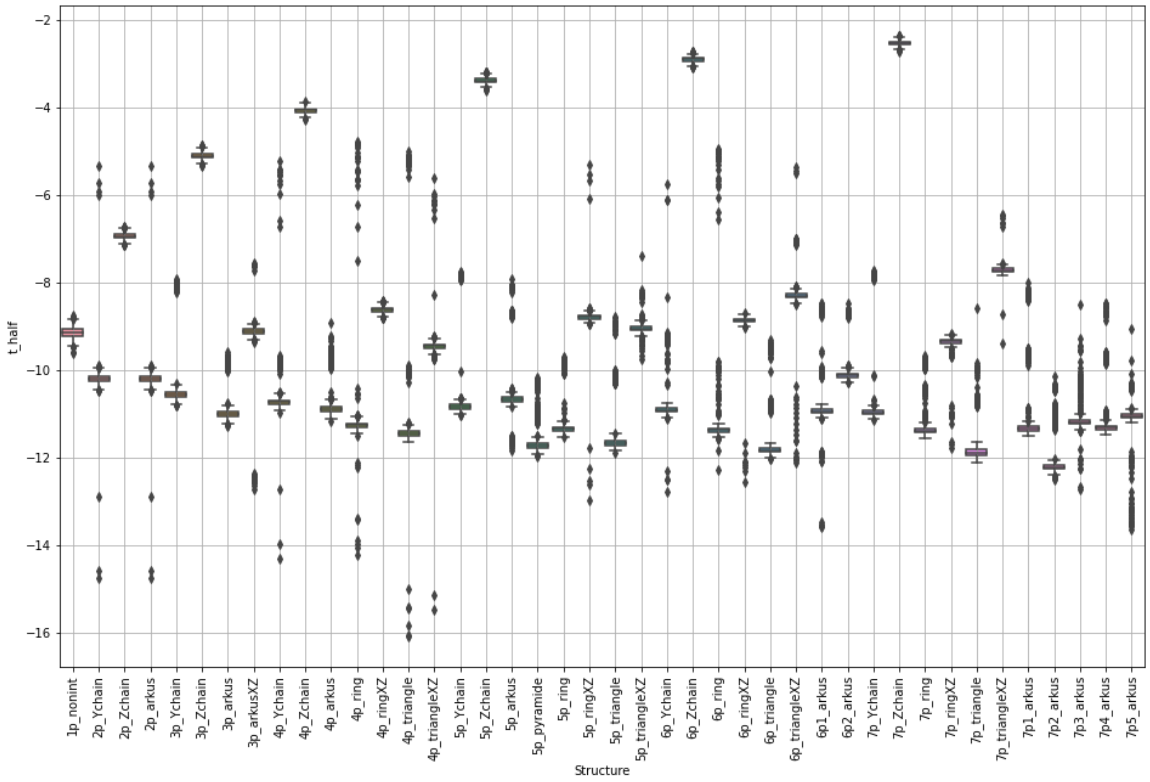

Figure 5.8: Box plot of t-half for different structures for Set 4. Thick horizontal lines represent the box, small whiskers are present due to low spread in the data. All particles in a cluster have anisotropy axis oriented in the z-direction and a random anisotropy value with 1% variation. Units of the y-axis is the log of time (s).

In Figure 5.8, the interquartile range is represented as a thick horizontal line because the majority of the data is tightly clustered around a single point. Additionally, this line can also be interpreted at the median value. However, there is slight variation within cluster groups, as whiskers and outliers are present. The key takeaway from this graph is that generally there is variation between cluster groups, although some groups' whiskers and IQRs overlap.

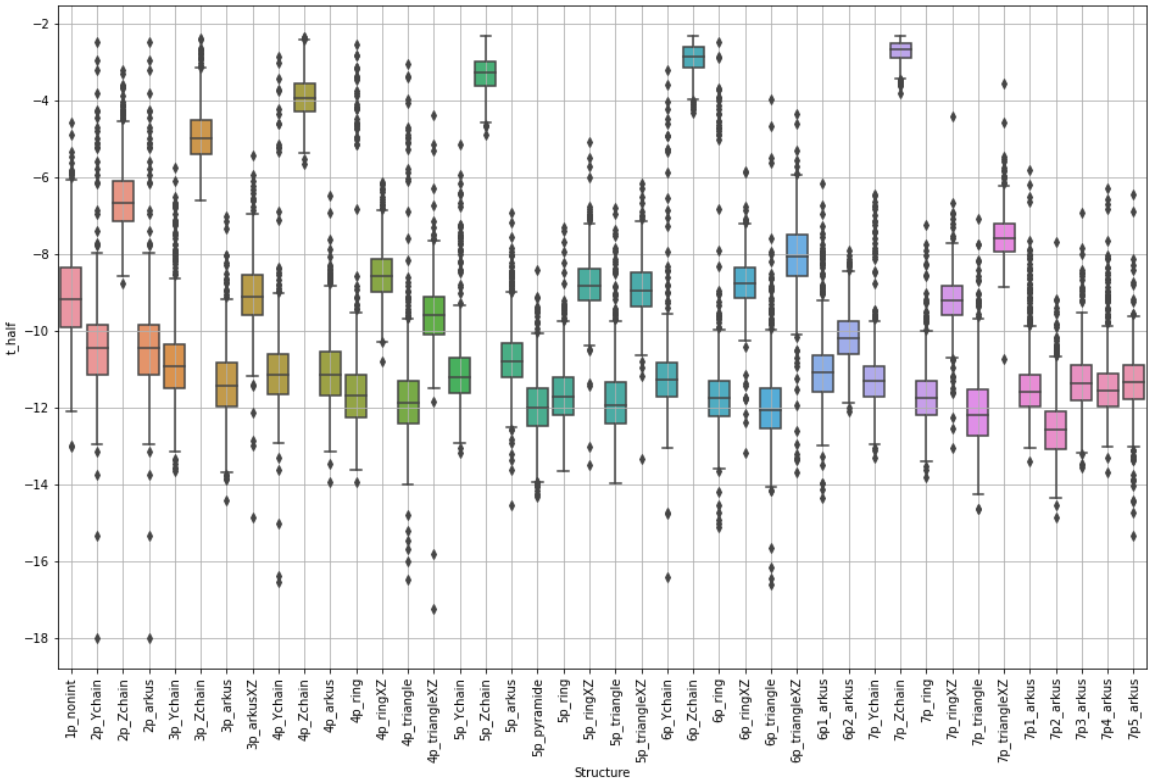

Figure 5.9: Box plot of t-half for different structures for Set 5. Different colours represent different particle clusters. All particles in a cluster having anisotropy axis oriented in in the z-direction and a random anisotropy value with 10% variation. Units of the y-axis is the log of time (s).

From looking at Figure 5.9, the most interesting aspect of this graph shows three clear groups of structures. The first are the structures with a t-half value of above -7, interestingly these are all Z-chain clusters. Generally, the Z-chain clusters have a low number of outliers. The next set of structures are the XZ clusters, situated in the range of -10 to -7. These data points are more spread than the first group since there are outliers and larger whiskers. The third group contains all of the other structures not mentioned already. For this figure, the selected feature t-half provides three specified cluster groups for this set.

Key findings from the box plots in Figure 5.5-5.9 are that Set 1,4 and 5 appear to be similar in the fact that there is variability between cluster groups, but varying degrees of variability within groups. The reason all three appear to be similar is that all three sets have all particles in a cluster, where the anisotropy axis is oriented in the z-direction.

Set 1 is of the highest quality of the three, where there is no variability within groups. Given that Sets 1,4 and 5's descriptions only differ in anisotropy value, the high quality of Set 1 is caused by clusters have a constant anisotropy value. Set 4 has slightly more variation within groups cluster than Set 1, this is due to 1% randomness introduced of particles anisotropy. Finally Set 5 has a greater variation within groups, which is an effect of 10% randomness of anisotropy value.

Sets 2 and 3 differ from the rest in that all particles in a cluster have anisotropy axis oriented in spherically random directions. This seems to be the key reason of the poor-quality distribution of t-half. Looking at Figure 5.6 and Figure 5.7, on further inspection there appears to be slightly more variation within cluster groups for Set 3. This is caused by Set 2 having constant anisotropy value and Set 3 having 1% randomness of anisotropy value. However, anisotropy value does not appear to be the key driver in the poor quality of data, where there is little variation between cluster groups.

To conclude t-half is a high-quality feature, however it is limited to quality of data. T-half as a feature is only meaningful for datasets where randomness is limited, more specifically randomness of anisotropy axis, followed by randomness in anisotropy value. Based on Figure 5.5-5,9, supervised learning models should be able to distinguish between cluster groups very well for Set 1 leading to high accuracy. Accuracy will be lower for Sets 4 and 5, with Sets 2 and 3 having the lowest accuracy.

5.2 Unsupervised Learning

K-means is a clustering algorithm that was used in identifying natural grouping in the data and to find an optimal number of clusters, using unlabelled data, t-half (see Section 3.2.8). The algorithm was used on the median points of the cluster groups of Set 1, since Figure 5.5 shows that most points are situated at the median values.

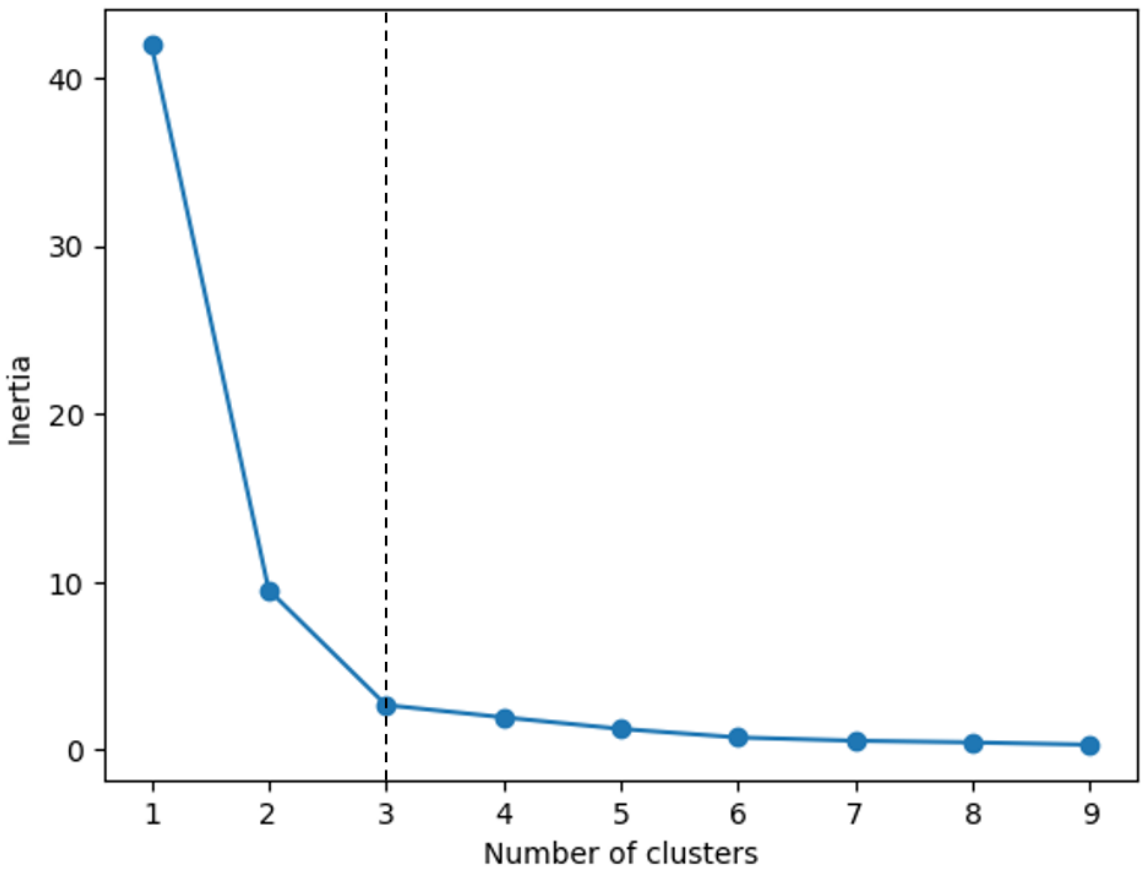

Figure 5.10: Inertia plot for K-means clustering. Dotted black line indicating an elbow at k=3. Inertia does not significantly improve after k=3. Increasing the number of clusters further would result in diminishing returns.

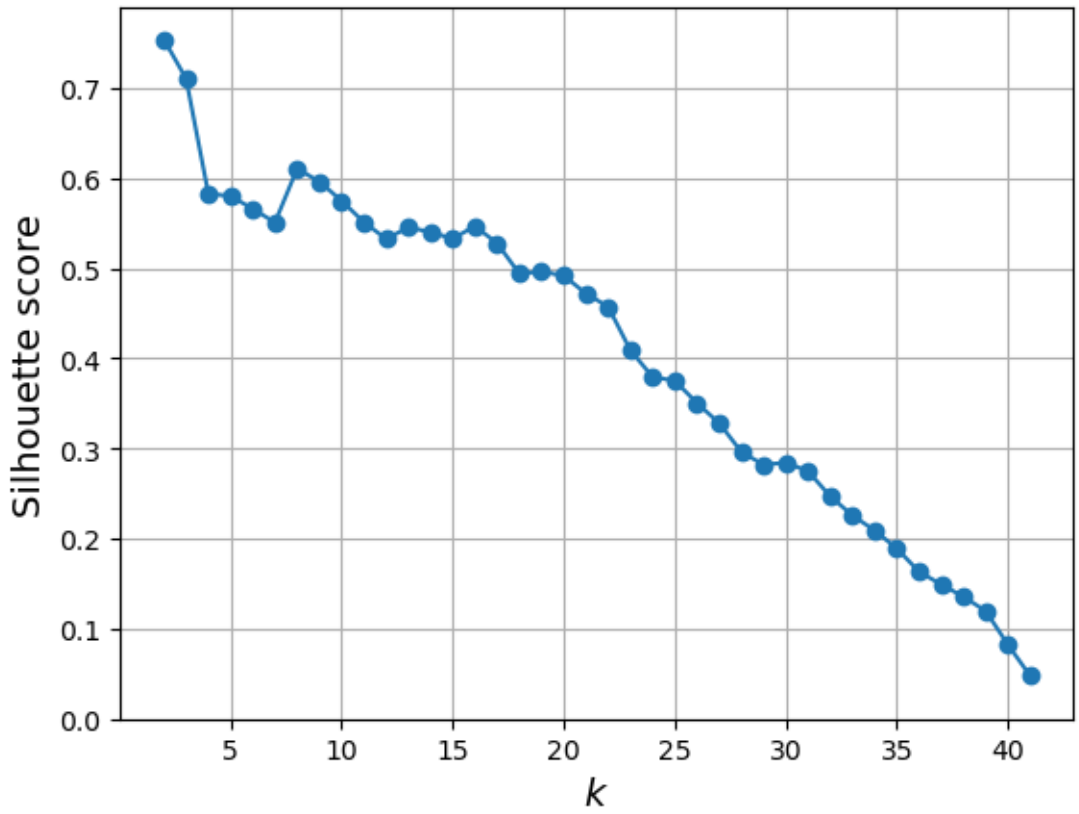

Figure 5.11: Silhouette score plot for K-means clustering. The silhouette scores are highest for k=2 and k=3 at 0.75 and 0.71 respectively. There is a noticeable drop in silhouette score to 0.58 at k=4, then there is a steady decline to 0.05 at k=41.

Figure 5.11 can be interpreted that there is an optimal number of clusters at k=3, where there is a sudden drop to k=4. A silhouette score of 0.71 at k=3 indicates there are compact clusters that are separated from nearby clusters. A silhouette score of 0.05 at k=41 means that on average, points are slightly more similar to points in their own cluster than other clusters, but there is overlapping present.

Figure 5.10 and Figure 5.11 together show good clustering performance at an optimal number of clusters of k=3. What also stands out is that in both figures, K-means struggles to cluster well into all the different cluster groups. This is likely due to cluster overlapping; however, it could be a limitation of the K-means algorithm, in which it struggles to behave well in clusters of different densities [26].

Each cluster group was then assigned a group based on the K-means model, where k=3.

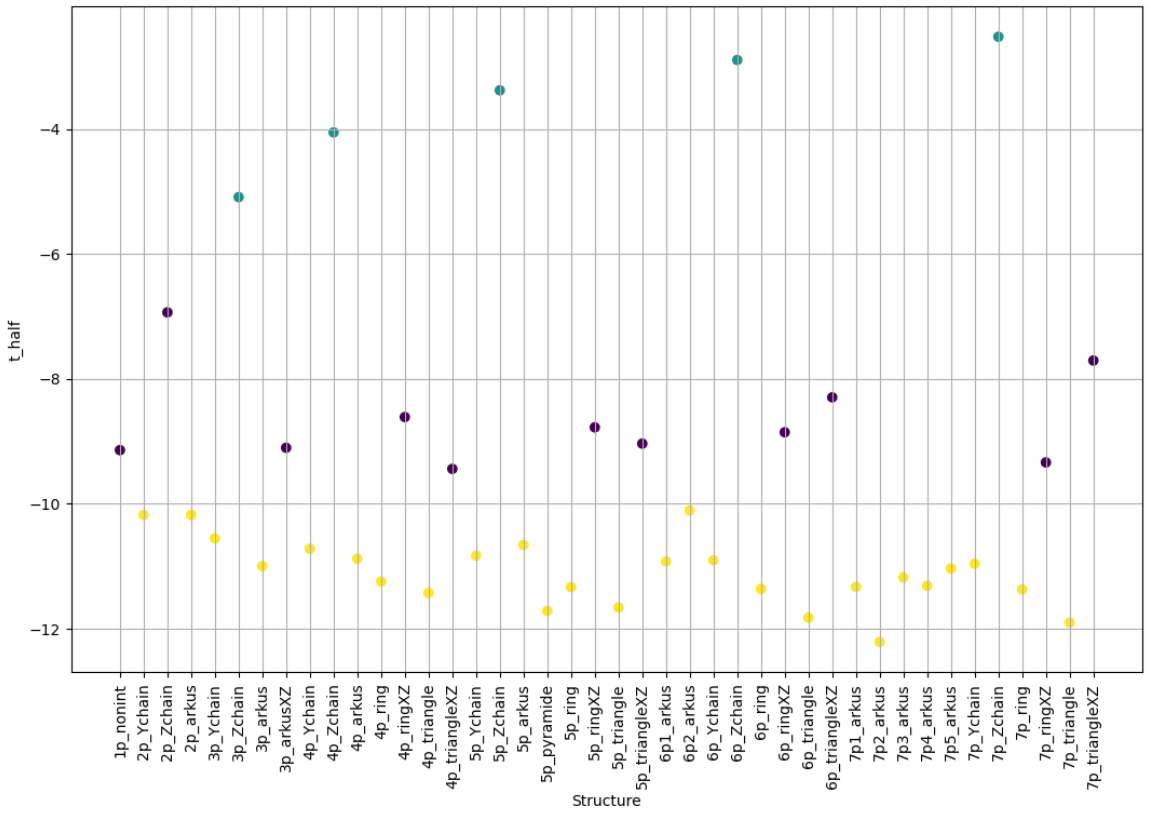

Figure 5.12: t-half median plot for Set 1. Colours are the cluster assigned by K-means algorithm with k=3.

What stands out in Figure 5.12 is groups of clusters with similar particle configurations. We see that K-means three major groups: (1) Z-chain, (2) XZ; and (3) other. This confirms the observation from Section 5.1. It is apparent from this figure that similar particle configurations behave similarly, in terms of magnetic signal, when magnetic relaxation occurs.

The cluster assigned to the median was then applied to all points with the same particle cluster. A violin plot in the figure below shows a visualisation of the densities of different clusters assigned by K-means.

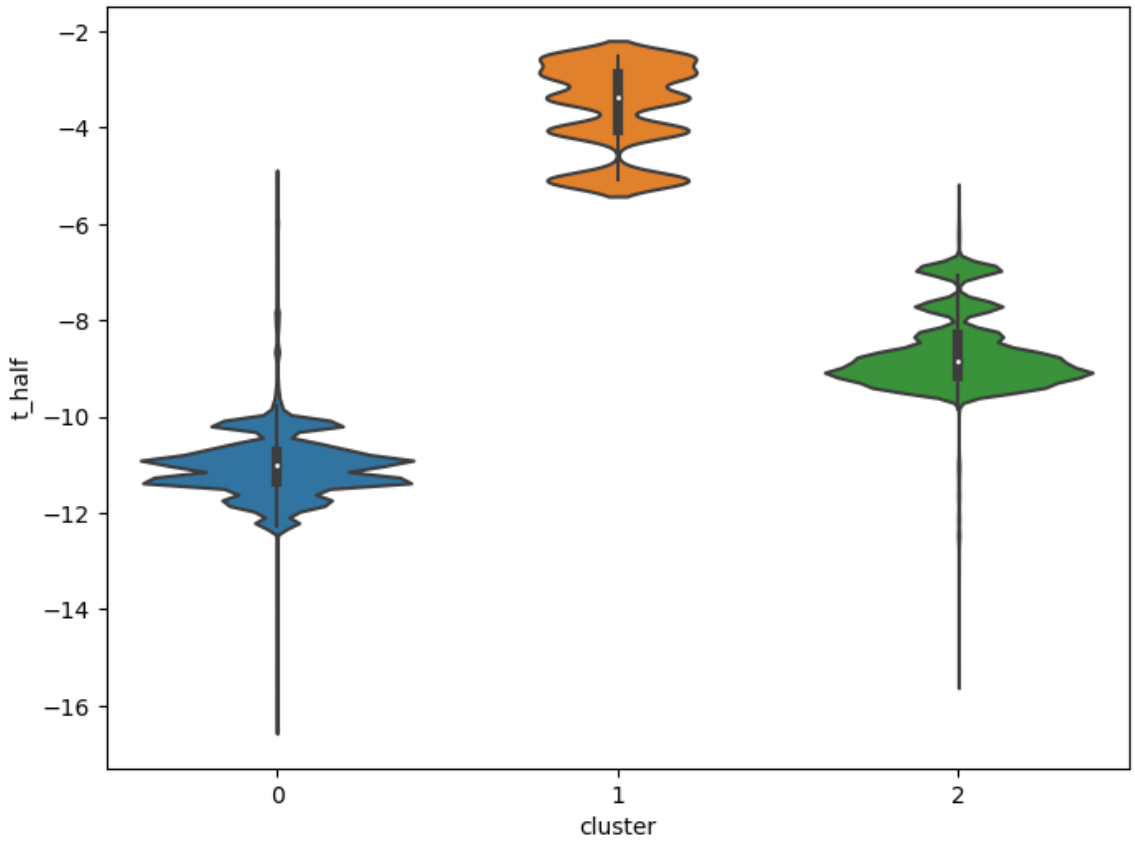

Figure 5.13: Violin plot of the 3 cluster groups assigned to the median, then applied to all the points in that particular structure. Violin plot shows kernel density estimation. The white dots show the median value of each violin cluster, the thick black line in the middle of the violin shows the interquartile range, where 50% of the data is distributed. The black line extending from the ends of the violin represent the few outliers.

Figure 5.13 shows that the performance of K-means algorithm was accurately modelled using only median values and was representative of all the data. Another observation is that large densities of data are at t-half of -9.2, -10.9 and -11.4. These are areas where multiple groups have similar responses.

5.3 Supervised Learning

In this study, the performance of logistic regression, SVM, KNN, decision trees, random forest and neural networks were evaluated for their performance in predicting the cluster group of MNPs. These approaches were chosen because they were techniques used in the Géron's book and described in Section 3.2 *[26]*. It is noted that following tables contain the final testing metrics, after model optimisation through hyperparameter tuning described in Section 4.7.

Table 5.3: Model evaluation of Set 1 using full data set

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.90	0.89	0.90	0.90
SVM	0.95	0.95	0.96	0.96
KNN	0.92	0.92	0.92	0.92
Decision Tree	0.95	0.95	0.95	0.95
Random Forest	0.95	0.95	0.95	0.95
Neural Network	0.92	0.92	0.93	0.92

Table 5.4: Model evaluation of Set 1 using t-half feature

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.32	0.26	0.24	0.32
SVM	0.89	0.87	0.88	0.89
KNN	0.95	0.95	0.95	0.95
Decision Tree	0.96	0.96	0.96	0.96
Random Forest	0.96	0.96	0.96	0.96
Neural Network	0.88	0.86	0.85	0.88

Table 5.5: Model evaluation of Set 2 using t-half feature

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.03	0.01	0.01	0.03
SVM	0.05	0.02	0.01	0.05
KNN	0.05	0.04	0.05	0.05
Decision Tree	0.05	0.05	0.05	0.05
Random Forest	0.05	0.05	0.05	0.05
Neural Network	0.06	0.04	0.02	0.06

Table 5.6: Model evaluation of Set 3 using t-half feature

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.03	0.01	0.01	0.03
SVM	0.06	0.02	0.01	0.06
KNN	0.04	0.03	0.03	0.04
Decision Tree	0.04	0.04	0.04	0.04
Random Forest	0.04	0.04	0.04	0.04
Neural Network	0.05	0.03	0.01	0.05

Table 5.7: Model evaluation of Set 4 using t-half feature

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.31	0.24	0.19	0.31
SVM	0.52	0.47	0.42	0.52
KNN	0.49	0.49	0.49	0.49
Decision Tree	0.45	0.45	0.45	0.45
Random Forest	0.45	0.45	0.45	0.45
Neural Network	0.53	0.49	0.47	0.53

Table 5.8: Model evaluation of Set 5 using t-half feature

Model	Accuracy	F1	Precision	Recall
Logistic Regression	0.15	0.10	0.07	0.15
SVM	0.16	0.11	0.07	0.16
KNN	0.12	0.11	0.12	0.12
Decision Tree	0.10	0.10	0.10	0.10
Random Forest	0.10	0.10	0.10	0.10
Neural Network	0.16	0.11	0.07	0.16

Set 1 was evaluated in two different ways, the first was using all 100 data points from the relaxation curve, seen in Figure 5.1. The second method was using only one engineered feature, t-half. The results of these models can be seen in Table 5.3 and Table 5.4. Maximum F1 score of 0.96 was obtained using t-half feature, slightly higher than 0.95 from the full data set. These scores were obtained by the decision tree and random forest classifier models in both cases. Looking at the distribution of the input feature in Figure 5.5, it is unlikely to obtain a score much higher than this, due to the number of outliers. Model performance for each model using the full data set was high across the board, with

a minimum F1 score of 0.90. Whereas logistic regression failed to classify clusters correctly given only t-half as a feature. This is likely due to increased computational complexity, where the model training, testing and tuning took significantly longer than for the case with t-half. However, looking at the maximum scores, t-half enabled the highest F1 score with significantly reduced time complexity. Therefore, models for the remaining Sets were evaluated using the t-half feature.

A confusion matrix was plotted for Set 1 for in-depth analysis of where the model was misclassifying training data (see Section 4.6.1).

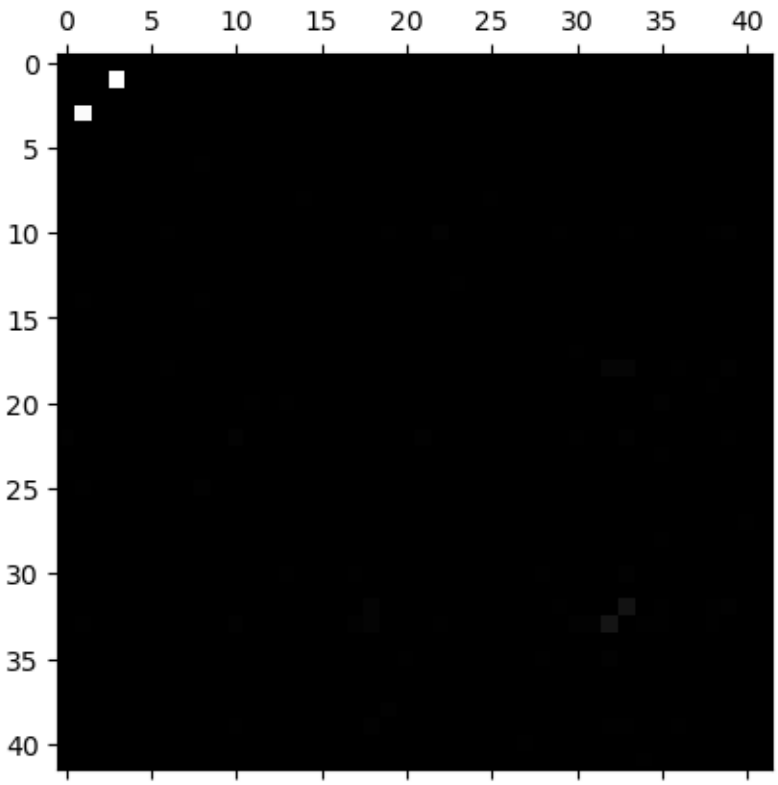

Figure 5.14: Confusion Matrix for Set 1 representing the random forest classifier. Diagonal has been filled in black to only show errors. Groups 2 and 4, are the brightest and have been misclassified the most. Groups 32 and 33 also show light grey which shows that a proportion of them have been confused with each other in the model.

Table 5.9: Set 1 Classification report with for individual cluster groups

Group index	Structure	Precision	Recall	F1-score	Support
0	1p_nonint	0.99	1.00	1.00	793
1	2p_Ychain	0.39	0.38	0.39	803
2	2p_Zchain	1.00	1.00	1.00	790
3	2p_arkus	0.39	0.40	0.40	794
4	3p_Ychain	1.00	1.00	1.00	803

5	3p_Zchain	1.00	1.00	1.00	787
6	3p_arkus	0.99	1.00	1.00	787
7	3p_arkusXZ	1.00	1.00	1.00	815
8	4p_Ychain	0.99	0.99	0.99	771
9	4p_Zchain	1.00	1.00	1.00	797
10	4p_arkus	0.98	0.97	0.97	788
11	4p_ring	0.99	1.00	1.00	796
12	4p_ringXZ	1.00	1.00	1.00	817
13	4p_triangle	0.99	0.99	0.99	786
14	4p_triangleXZ	0.99	0.99	0.99	820
15	5p_Ychain	1.00	1.00	1.00	801
16	5p_Zchain	1.00	1.00	1.00	797
17	5p_arkus	0.99	1.00	0.99	797
18	5p_pyramide	0.97	0.96	0.97	772
19	5p_ring	0.99	1.00	0.99	817
20	5p_ringXZ	1.00	0.99	0.99	812
21	5p_triangle	0.99	1.00	0.99	786
22	5p_triangleXZ	0.99	0.97	0.98	791
23	6p1_arkus	0.99	0.99	0.99	795
24	6p2_arkus	1.00	1.00	1.00	812
25	6p_Ychain	0.99	0.99	0.99	818
26	6p_Zchain	1.00	1.00	1.00	806
27	6p_ring	1.00	1.00	1.00	797
28	6p_ringXZ	0.99	1.00	1.00	815
29	6p_triangle	1.00	1.00	1.00	812
30	6p_triangleXZ	0.99	0.98	0.98	792
31	7p1_arkus	0.99	1.00	1.00	782
32	7p2_arkus	0.93	0.93	0.93	814
33	7p3_arkus	0.92	0.89	0.90	789
34	7p4_arkus	0.99	1.00	0.99	787
35	7p5_arkus	0.98	0.99	0.99	812
36	7p_Ychain	0.99	1.00	1.00	800
37	7p_Zchain	1.00	1.00	1.00	823
38	7p_ring	0.99	0.99	0.99	789
39	7p_ringXZ	0.98	0.98	0.98	825
40	7p_triangle	1.00	1.00	1.00	808
41	7p_triangleXZ	1.00	1.00	1.00	796
	weighted avg	0.96	0.96	0.96	33592

Table 5.9 contains precisions, recall and F1 scores for individual cluster groups. It is helpful in discovering problematic cluster groups. Upon examination of Table 5.9, groups 1 and 3 correspond to 2p Y-chain and 2p arkus, and have an F1 score of 0.39 and 0.40 respectively. Looking back at Figure 5.5, we can see that 2p Y-chain and 2p arkus have identical t-half values, which is the cause of the misclassification. The majority of the remaining groups have F1 scores of 0.99 to 1.00 indicating the model can classify most of the groups correctly, with outliers accounting for up to 0.01 accuracy. The confusion of groups 1 and 3 decreases the overall weighted average F1 score of the model.

Table 5.10: Dataset comparison of highest F1 score obtained

Dataset	Highest F1 score	Model
Set 1	0.96	Decision Tree, Random Forest
Set 2	0.05	Decision Tree, Random Forest
Set 3	0.04	Decision Tree, Random Forest
Set 4	0.49	KNN, Neural Network
Set 5	0.11	SVM KNN, Neural Network

A comparison of the datasets in Table 5.10 shows that the model for Set 1 could classify almost all clusters correctly with an F1 score of 0.96. The next highest F1 score was for Set 4 at 0.49, which shows that the model was able to classify the cluster half of the time. Set 5 had the third highest F1 score at only 0.11. This means that clusters were only correctly classified 1 in 10 times. Finally, the worst two performing groups were Sets 2 with an F1 score of 0.05 and Set 3 at 0.04. These results are in-line with predictions made based on the feature distribution in Section 6.1. Interestingly, not 1 classifier worked best for all, indicating that they all had strengths depending on the type of data.

Overall, the evaluation of multiple supervised learning models confirms conclusions drawn in Section 5.1. This is that although t-half is a high-quality feature, it is limited to quality of data. The usefulness of t-half as a feature is constrained to datasets with limited randomness, particularly in terms of anisotropy axis. Another conclusion is that having 1% randomness in in anisotropy value directly causes a decrease in categorisation accuracy by half, from comparing Set 1 and 4. Sets 2 and 3 represent an ill-posed problem, in which there is no way to categorize using current data, due to randomness in the axes of anisotropy.

Overall, supervised learning has shown to be more effective than unsupervised learning when labelled data is available and the goal is to predict outcomes.

6 Conclusions and Future Outlook

The study set out to categorize MNPs through machine learning models based on data of varying randomness. The engineered feature, t-half, enabled random forest and decision tree algorithms to correctly categorize MNP clusters with controlled direction of anisotropy axes and constant anisotropy values, with 96% accuracy. The second major finding was that controlling the direction of the anisotropy axes was the key driver in enabling models to distinguish different clusters. The study also found that introducing randomness in anisotropy value greatly decreases the precision of the models. For clusters with even a degree of randomness in anisotropy values, t-half alone is not sufficient for machine learning models to classify clusters correctly. The K-means classifier showed that groupings of similar cluster groups, had similar magnetisation signals. A different feature such as temperature or hysteresis could be measured to distinguish clusters within similar groupings (see Section 3.1). In terms of machine learning, bagging, boosting and voting classifiers could improve model accuracy as well as dimensionality reduction techniques, such as principal component analysis that could be used on the full dataset. However, my research shows this can only increase accuracy by less than 1% with current data, due to my current modelling almost maximising accuracy. Manufacturing particles with minimised randomness could improve the accuracy of these models. Measuring the magnetic signal better could also produce more precise measurements and limit the number of outliers, which in turn will improve the accuracy of the models. Future projects could include cloud applications, where data can be input on a website, and the model machine learning model returns an output prediction.

7 Bibliography

[1] A. S. Ahmad, N. Ormiston-Smith, and P. D. Sasieni, 'Trends in the lifetime risk of developing cancer in Great Britain: comparison of risk for those born from 1930 to 1960', *British Journal of Cancer*, vol. 112, no. 5. Springer Science and Business Media LLC, pp. 943–947, Feb. 03, 2015. doi: 10.1038/bjc.2014.606.

[2] "Cancer treatment statistics," Cancer Research UK, [Online]. Available: https://www.cancerresearchuk.org/health-professional/cancer-statistics/treatment#heading-Four. [Accessed 23 April 2023].

[3] Q. A. Pankhurst, J. Connolly, S. K. Jones, and J. Dobson, 'Applications of magnetic nanoparticles in biomedicine', *Journal of Physics D: Applied Physics*, vol. 36, no. 13. IOP Publishing, pp. R167–R181, Jun. 19, 2003. doi: 10.1088/0022-3727/36/13/201.

[4] "Diagnosing Cancer," Cancer Center, April 29 2022. [Online]. Available: https://www.cancercenter.com/diagnosing-cancer. [Accessed 23 April 2023].

[5] "Cancer," NHS, 13 October 2022. [Online]. Available: https://www.nhs.uk/conditions/cancer/. [Accessed 23 April 2023].

[6] K. J. Widder, A. E. Senyei, and D. G. Scarpelli, "Magnetic microspheres: a model system of site specific drug delivery in vivo," *Experimental Biology and Medicine,* vol. 158, no. 2, pp. 141-146, 1978.

[7] K. Mosbach and U. Schröder, "Preparation and application of magnetic polymers for targeting of drugs," *FEBS Letters,* vol. 102, no. 1, pp. 112-116, 1979.

[8] F. Wiekhorst, U. Steinhoff, D. Eberbeck, and L. Trahms, 'Magnetorelaxometry Assisting Biomedical Applications of Magnetic Nanoparticles', *Pharmaceutical Research*, vol. 29, no. 5. Springer Science and Business Media LLC, pp. 1189–1202, Dec. 08, 2011. doi: 10.1007/s11095-011-0630-3.

[9] A. Farzin, S. A. Etesami, J. Quint, A. Memic, and A. Tamayol, 'Magnetic Nanoparticles in Cancer Therapy and Diagnosis', *Advanced Healthcare Materials*, vol. 9, no. 9. Wiley, p. 1901058, Mar. 20, 2020. doi: 10.1002/adhm.201901058.

[10] O. Hosu, M. Tertis, and C. Cristea, 'Implication of Magnetic Nanoparticles in Cancer Detection, Screening and Treatment', *Magnetochemistry*, vol. 5, no. 4. MDPI AG, p. 55, Oct. 01, 2019. doi: 10.3390/magnetochemistry5040055.

[11] S. J. Osterfeld et al., 'Multiplex protein assays based on real-time magnetic nanotag sensing', *Proceedings of the National Academy of Sciences*, vol. 105, no. 52. Proceedings of the National Academy of Sciences, pp. 20637–20640, Dec. 30, 2008. doi: 10.1073/pnas.0810822105.

[12] J.-H. Lee et al., 'Artificially engineered magnetic nanoparticles for ultra-sensitive molecular imaging', *Nature Medicine*, vol. 13, no. 1. Springer Science and Business Media LLC, pp. 95–99, Dec. 24, 2006. doi: 10.1038/nm1467.

[13] Y. Jun, J. Lee, and J. Cheon, 'Chemical Design of Nanoparticle Probes for High-Performance Magnetic Resonance Imaging', *Angewandte Chemie International Edition*, vol. 47, no. 28. Wiley, pp. 5122–5135, Jun. 27, 2008. doi: 10.1002/anie.200701674.

[14] G. Li, S. Sun, R. J. Wilson, R. L. White, N. Pourmand, and S. X. Wang, 'Spin valve sensors for ultrasensitive detection of superparamagnetic nanoparticles for biological applications',

Sensors and Actuators A: Physical, vol. 126, no. 1. Elsevier BV, pp. 98–106, Jan. 2006. doi: 10.1016/j.sna.2005.10.001.

[15] Y. Ha et al., 'Recent Advances Incorporating Superparamagnetic Nanoparticles into Immunoassays', *ACS Applied Nano Materials*, vol. 1, no. 2. American Chemical Society (ACS), pp. 512–521, Feb. 14, 2018. doi: 10.1021/acsanm.7b00025.

[16] O. Laslett, S. Ruta, J. Barker, R. W. Chantrell, G. Friedman, and O. Hovorka, 'Interaction effects enhancing magnetic particle detection based on magneto-relaxometry', *Applied Physics Letters*, vol. 106, no. 1. AIP Publishing, p. 012407, Jan. 05, 2015. doi: 10.1063/1.4905339.

[17] K. Enpuku et al., 'Biological Immunoassays Without Bound/Free Separation Utilizing Magnetic Marker and HTS SQUID', *IEEE Transactions on Applied Superconductivity*, vol. 17, no. 2. Institute of Electrical and Electronics Engineers (IEEE), pp. 816–819, Jun. 2007. doi: 10.1109/tasc.2007.897699.

[18] J. Li, Q. Xu, X. Wei, and Z. Hao, 'Electrogenerated Chemiluminescence Immunosensor for Bacillus thuringiensis Cry1Ac Based on Fe3O4@Au Nanoparticles', *Journal of Agricultural and Food Chemistry*, vol. 61, no. 7. American Chemical Society (ACS), pp. 1435–1440, Feb. 08, 2013. doi: 10.1021/jf303774x..

[19] R. G. Palmer, D. L. Stein, E. Abrahams, and P. W. Anderson, 'Models of Hierarchically Constrained Dynamics for Glassy Relaxation', *Physical Review Letters*, vol. 53, no. 10. American Physical Society (APS), pp. 958–961, Sep. 03, 1984. doi: 10.1103/physrevlett.53.958..

[20] "Multiclass and multioutput algorithms," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/multiclass.html. [Accessed 04 05 2023].

[21] "K-Nearest Neighbors Algorithm," IBM, [Online]. Available: https://www.ibm.com/topics/knn#:~:text=Next%20steps-,K%2DNearest%20Neighbors%20Algorithm,of%20an%20individual%20data%20point.. [Accessed 04 05 2023].

[22] A. Collette, "HDF5 for Python," h5py, [Online]. Available: https://www.h5py.org/. [Accessed 04 05 2023].

[23] "F value," IBM, 03 01 2023. [Online]. Available: https://www.ibm.com/docs/no/cognos-analytics/11.1.0?topic=terms-f-value. [Accessed 19 04 2023].

[24] pandas, [Online]. Available: https://pandas.pydata.org/. [Accessed 04 05 2023].

[25] "train test split," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed 04 05 2023].

[26] G. A, Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow, CA 95472: O'Reilly, 2019.

8 Appendices

8.1 Appendix A: Feature Extraction

```
import h5py
import numpy as np
from scipy import interpolate

def fprime_central(f, x0, h=1e-1):
    return (f(x0+h) - f(x0-h)) / (2*h)

t_halves = []   # initialise array for t halves

path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20200318_reduced'

structures = ['npStruct_1p_nonint',
              'npStruct_2p_Ychain',
              'npStruct_2p_Zchain',
              'npStruct_2p_arkus',
              'npStruct_3p_Ychain',
              'npStruct_3p_Zchain',
              'npStruct_3p_arkus',
              'npStruct_3p_arkusXZ',
              'npStruct_4p_Ychain',
              'npStruct_4p_Zchain',
              'npStruct_4p_arkus',
              'npStruct_4p_ring',
              'npStruct_4p_ringXZ',
              'npStruct_4p_triangle',
              'npStruct_4p_triangleXZ',
              'npStruct_5p_Ychain',
              'npStruct_5p_Zchain',
              'npStruct_5p_arkus',
              'npStruct_5p_pyramide',
              'npStruct_5p_ring',
              'npStruct_5p_ringXZ',
              'npStruct_5p_triangle',
              'npStruct_5p_triangleXZ',
              'npStruct_6p_Ychain',
              'npStruct_6p_Zchain',
              'npStruct_6p_ring',
              'npStruct_6p_ringXZ',
              'npStruct_6p_triangle',
              'npStruct_6p_triangleXZ',
              'npStruct_6p1_arkus',
              'npStruct_6p2_arkus',
              'npStruct_7p_Ychain',
              'npStruct_7p_Zchain',
              'npStruct_7p_ring',
              'npStruct_7p_ringXZ',
              'npStruct_7p_triangle',
              'npStruct_7p_triangleXZ',
              'npStruct_7p1_arkus',
              'npStruct_7p2_arkus',
              'npStruct_7p3_arkus',
              'npStruct_7p4_arkus',
              'npStruct_7p5_arkus']

for i in structures:

```

```
        clust = i.split('_')

        fp = path + '/' + i + '/spin_mc_red.h5'
        f = h5py.File(fp, 'r')

        for j in f:
            if any(char.isdigit() for char in j):

                dset = f[j]

                a = np.array(dset['mt'])    # convert to np array
                x = np.log(a[:,0])          # column 1, log time
                y = a[:,1]                  # column 2, magnetism

                m1 = y[0]*0.5       # midpoint of magnetisation
                m2 = y[0]*0.333       # 1/3 of magnetisation
                m3 = y[0]*0.666       # 2/3 of magnetisation
                y_values = [m1, m2, m3]     # list of y (magnetisation) values

                interp_fn = interpolate.interp1d(x, y, 'quadratic')     # create the interpolated function
                f2 = interpolate.interp1d(y, x, bounds_error=False)     # inverse interpolate function
                x_values = f2(y_values)

                grad = fprime_central(f2, m1)

                #assign a value to each curve to determine if it has fully relaxed
                if y[-1] == y[-2]:
                    relaxed = 1
                else:
                    relaxed = 0

                t_half = [i, relaxed, grad]
                t_half.extend(x_values)
                t_halves.append(t_half)

    t_halves_np = np.array(t_halves)
    print(len(t_halves_np))
    np.savetxt('fulldata_new11.csv', t_halves_np, fmt= '%s', delimiter= ',')
```

8.2 Appendix B: Stretched Exponential Feature

```python
import h5py
import numpy as np
from scipy.optimize import curve_fit
from scipy import interpolate

def st_exp(x, c, tau, beta, y_offset):
        return c*(np.exp(-(x/tau)**beta))+y_offset

t_halves = []   # initialise array for t halves

path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20200318_reduced'

structures = ['npStruct_1p_nonint',

for i in structures:

    clust = i.split('_')

    fp = path + '/' + i + '/spin_mc_red.h5'
    f = h5py.File(fp, 'r')

    for j in f:
        if any(char.isdigit() for char in j):
            dset = f[j]

            a = np.array(dset['mt'])    # convert to np array
            x = -np.log(a[:,0])          # column 1, log time
            y = a[:,1]                   # column 2, magnetism

            m1 = y[0]*0.5       # calculate midpoint of y using first and last
            y_values = [m1]     # list of y values to find x

            interp_fn = interpolate.interp1d(x, y, 'quadratic')     # create the interpolated function
            f2 = interpolate.interp1d(y, x, bounds_error=False)     # inverse interpolate function
            x_values = f2(y_values)

            try:
                best_vals = curve_fit(st_exp, x, y, maxfev=3000)
            except:
                pass

            t_half = [i]
            t_half.extend(x_values)

            try:
                t_half.extend(best_vals)
            except:
                pass
            t_halves.append(t_half)

            t_half = [i]
            t_half.extend(best_vals)
            t_halves.append(t_half)

t_halves_np = np.array(t_halves)
print(len(t_halves_np))
np.savetxt('stretchedexp.csv', t_halves_np, fmt='%s', delimiter=',')
```

8.3 Appendix C: T-half Feature Dataset Iteration

```python
import h5py
import numpy as np
from scipy import interpolate

t_halves = []   # initialise array for t halves

path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20200318_reduced' #set 1
#path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20200109_reduced' #set 2
#path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20210406_reduced' #set 3
#path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20210409_reduced' #set 4
#path = 'C:/Users/44798/OneDrive - University of Southampton/Individual Project Daniel Tong/clusterdata_1000rea_20210416_reduced' #set 5

structures = ['npStruct_1p_nonint',

for i in structures:

    clust = i.split('_')

    fp = path + '/' + i + '/spin_mc_red.h5'
    f = h5py.File(fp, 'r')

    for j in f:
        if any(char.isdigit() for char in j):

            dset = f[j]

            a = np.array(dset['mt'])    # convert to np array
            x = np.log(a[:,0])          # column 1, log time
            y = a[:,1]                  # column 2, magnetism

            m1 = y[0]*0.5       # calculate midpoint of y using first and last
            y_values = [m1]     # list of y values to find x

            interp_fn = interpolate.interp1d(x, y, 'quadratic')     # create the interpolated function
            f2 = interpolate.interp1d(y, x, bounds_error=False)     # inverse interpolate function
            x_values = f2(y_values)


            t_half = [i]
            t_half.extend(x_values)
            t_halves.append(t_half)

t_halves_np = np.array(t_halves)
print(len(t_halves_np))
np.savetxt('set_1.csv', t_halves_np, fmt= '%s', delimiter= ',')
```

8.4 Appendix D: Models Comparison

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import warnings
warnings.filterwarnings("ignore")

files = ['set_1', 'set_2', 'set_3', 'set_4', 'set_5']

for dset in files:
    file = dset +'.csv'
    df = pd.read_csv(file)

    df.columns = ['Structure', 't_half']
    df = df[df['t_half'].notna()]

    X = df.drop(columns = ['Structure'])
    #labels
    y = df['Structure']

    #create random train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = (1))

    #scale data
    scaler = StandardScaler().fit(X_train)
    X_train= scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    #define models
    lr  = LogisticRegression(multi_class='ovr', random_state=1)
    svc = SVC(probability=True)
    knn = KNeighborsClassifier()
    rfc = RandomForestClassifier(random_state=1)
    dtc = DecisionTreeClassifier(random_state=1)
    mlp = MLPClassifier()

    models_list = [lr, svc, knn, rfc, dtc, mlp]
    metrics_ls = ['accuracy score', 'f1 score', 'precision score', 'recall score']

    ar = []

    for model in models_list:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)
```

```python
        scores = [
            accuracy_score(y_test, y_pred),
            f1_score(y_test, y_pred, average= 'weighted'),
            precision_score(y_test, y_pred, average= 'weighted', zero_division=0),
            recall_score(y_test, y_pred, average= 'weighted'),
            ]
        ar.append(scores)

    df = pd.DataFrame(ar, index =['Logistic Regression',
                                  'SVM',
                                  'K Nearest Neighbors',
                                  'Random Forest',
                                  'Decision Tree',
                                  'Neural Network'
                                    ],
                                  columns=metrics_ls)

    print(dset, df)
```

8.5 Appendix E: Models Comparison Full Data

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import warnings
warnings.filterwarnings("ignore")

df= pd.read_csv('fulldata.csv', header=None)
X = df.iloc[:,3:]
y = df.iloc[:,0]

#create random train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = (1))

#scale data
scaler  = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)

lr  = LogisticRegression(C=10, multi_class='ovr', random_state=1, solver='newton-cg')
svc = SVC(C=1000, gamma=1, probability=True)
knn = KNeighborsClassifier(metric='euclidean', n_neighbors=1)
rfc = RandomForestClassifier(random_state=1)
dtc = DecisionTreeClassifier(criterion='entropy', max_depth=20, min_samples_leaf=5,random_state=1)
mlp = MLPClassifier(hidden_layer_sizes=(10, 30, 10))

models_list = [lr, svc, knn, rfc, dtc, mlp]
metrics_ls = ['accuracy score', 'f1 score', 'precision score', 'recall score']

ar = []

for model in models_list:
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)

    scores = [
        accuracy_score(y_test, y_pred),
        f1_score(y_test, y_pred, average= 'weighted'),
        precision_score(y_test, y_pred, average= 'weighted', zero_division=0),
        recall_score(y_test, y_pred, average= 'weighted'),
        ]
    ar.append(scores)
```

```python
df = pd.DataFrame(ar, index =['Logistic Regression',
                              'SVM',
                              'K Nearest Neighbors',
                              'Random Forest',
                              'Decision Tree',
                              'Neural Network'
                                ],
                              columns=metrics_ls)

print(df)
```

8.6 Appendix F: Kmeans

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import matplotlib.cm as cm
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns
```

In [ ]:
```
#read csv
df = pd.read_csv('fulldata_new1.csv', header=None)
df
```

In [ ]:
```
#name columns
df.columns = ['Structure', 'index', 'relaxed', 't_half', 't2', 't3']
df["Structure"] = df["Structure"].str.replace("npStruct_","")
df
```

In [ ]:
```
#null values in t half
df['t_half'].isnull().values.sum()
```

In [ ]:
```
df = df[df['t_half'].notna()]
df['t_half'].isnull().values.sum()
```

In [ ]:
```
plt.figure(figsize=(16,10))
plt.scatter(df['Structure'], df['t_half'], c=df['relaxed'], cmap='viridis', alpha=0.05)
plt.xticks(rotation='vertical')
plt.grid()
plt.legend()
plt.ylabel('t_half')
plt.xlabel('Structure')
```

In [ ]:
```
median = df.groupby('Structure').median()
median.reset_index(inplace = True)
median
```

In [ ]:
```
#assign colour to structure
cmap = cm.get_cmap('Spectral')
colour_dict = pd.Series({k:cmap(np.random.rand()) for k in df['Structure'].unique()})
colour_dict.name = 'colour_dict'
df = pd.merge(df, colour_dict, how='left', left_on='Structure', right_index=True)
df
```

In [ ]:
```
# Create a figure and an Axes3D object
fig = plt.figure(figsize=(16,12))
ax = fig.add_subplot(111, projection='3d')

# Use the scatter function to plot the data points
ax.scatter(df['t_half'], df['t2'], df['t3'], c=df['colour_dict'])
ax.set_xlabel('t half')
ax.set_ylabel('t 1/3')
ax.set_zlabel('t 2/3')
plt.show()
```

In [ ]:
```
X = median.drop(columns=['Structure', 'index', 'relaxed', 't2', 't3'])
X.head()
```

In [ ]:
```
# Extract the data as a NumPy array
data = X.values
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
data_scaled = scaler.transform(X)
```

In [ ]:
```
inertias = []

for i in range(1,10):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data_scaled)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,10), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

In [ ]:
```
kmeans_per_k = [KMeans(n_clusters=k).fit(data_scaled) for k in range(1,42)]

silhouette_scores = [silhouette_score(data_scaled, model.labels_)
                     for model in kmeans_per_k[1:]]
```

In [ ]:
```
plt.plot(range(2,42), silhouette_scores, marker='o', label = 'silhouette curve')
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.ylim(ymin=0)
plt.grid()
```

In [ ]:
```
# Create a KMeans model with 3 clusters
kmeans = KMeans(n_clusters=3)

# Fit the model to the data
kmeans.fit(data_scaled)

# Predict the cluster labels for each data point
labels = kmeans.predict(data_scaled)

# Add the cluster labels to the DataFrame as a new column
X['cluster'] = labels
X.head()
```

In [ ]:
```
plt.figure(figsize=(12.8,8))
plt.scatter(median['Structure'], median['t_half'], c=X['cluster'], cmap='viridis')
plt.xticks(rotation='vertical')
plt.grid()
plt.ylabel('t_half')
plt.xlabel('Structure')
plt.title('median t-half cluster plot')
```

In [ ]:
```
df1 = median.join(X['cluster'])
df1.head()
```

In [ ]:
```
df1 = df1.drop(columns=['index', 'relaxed', 't_half', 't2', 't3'])
df1.head()
```

In [ ]:
```
inner = pd.merge(df, df1)
inner
```

Classification

In [ ]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, ShuffleSplit, RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import metrics
import seaborn as sns
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, precision_score, recall_score, make_scorer
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
```

In [ ]:
```
inner
```

In [ ]:
```
X = inner.drop(columns = ['Structure','index', 'relaxed', 't2', 't3', 'colour_dict', 'cluster'])
y = inner['Structure']
```

In [ ]:
```
df= pd.read_csv('fulldata.csv', header=None)
X = df.iloc[:,3:]
y = df.iloc[:,0]
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = (1)) #create random train test split
```

In [ ]:
```
X.describe()
```

In [ ]:
```
scaler = StandardScaler().fit(X_train)
X_train_scaled= scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled= pd.DataFrame(X_train_scaled)
X_test_scaled = pd.DataFrame(X_test_scaled)
np.round(X_train_scaled.describe(), 2)
```

In [ ]:
```
from sklearn.multiclass import OneVsRestClassifier
logreg = RandomForestClassifier(random_state=1)
logreg.fit(X_train_scaled, y_train) #define logistic regression model
print('train accuracy =', logreg.score(X_train_scaled, y_train))
print('test accuracy =', logreg.score(X_test_scaled, y_test))
```

In [ ]:
```
cross_val_score(logreg, X_train_scaled, y_train, cv=3, scoring='accuracy')
```

In [ ]:
```
from sklearn.model_selection import cross_val_predict
y_train_pred = cross_val_predict(logreg, X_train_scaled, y_train, cv=3)
conf_mx = metrics.confusion_matrix(y_train, y_train_pred)
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.show
```

In [ ]:
```
plt.figure(figsize=(42,42))
row_sums = conf_mx.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```

In [ ]:
```
print(metrics.classification_report(y_train, y_train_pred, zero_division=0))
```

In [ ]:
```
plt.figure(figsize=(8,6))
sns.violinplot(x='cluster', y='t_half', data = inner)
```

8.7 Appendix G: Hyperparameter Tuning Pipeline

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV, ShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.exceptions import ConvergenceWarning
from sklearn import set_config
from warnings import simplefilter
simplefilter("ignore", category=ConvergenceWarning)
```

In [ ]:
```
df= pd.read_csv('fulldata.csv', header=None)
X = df.iloc[:,3:]
y = df.iloc[:,0]
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,random_state = (1)) #create random train test split
```

In [ ]:
```
X_train = X_train.to_numpy()
y_train = y_train.to_numpy()
X_test = X_test.to_numpy()
y_test = y_test.to_numpy()
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:
```
scaler  = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test  = scaler.transform(X_test)
```

```
X.describe()
```

In [ ]:
```
ct = ColumnTransformer(
     [('scaler', StandardScaler(),['t_half'])])
```

In [ ]:
```
clf1 = LogisticRegression(random_state=1, multi_class='ovr')
clf2 = SVC()
clf3 = KNeighborsClassifier()
clf4 = RandomForestClassifier(random_state= 1)
clf5 = DecisionTreeClassifier(random_state= 1)
clf6 = MLPClassifier()
```

In [ ]:
```
pipe = Pipeline([('preprocessor', ct), ('classifier', clf1)])
```

In [ ]:
```
set_config(display='diagram')
```

In [ ]:
```
pipe
```

In [ ]:
```
params1 = {}
params1["classifier__penalty"] = ['l2']
params1["classifier__C"] = [0, 1, 3, 5, 10]
params1["classifier__solver"] = ['lbfgs','newton-cg','saga','sag','liblinear']
params1["classifier"] = [clf1]
```

In [ ]:
```
params2 = {}
params2["classifier__C"] = [0.1, 1, 10, 100, 1000]
params2["classifier__gamma"] = [1, 0.1, 0.01, 0.001, 0.0001]
params2["classifier__kernel"] = ['rbf']
params2["classifier"] = [clf2]
```

In [ ]:
```
params3 = {}
params3["classifier__n_neighbors"] = [1, 3, 5, 10 ,15, 20]
params3["classifier__weights"] = ['uniform', 'distance']
params3["classifier__metric"] = ['euclidean', 'manhattan', 'minkowski']
params3["classifier"] = [clf3]
```

In [ ]:
```
params4 = {}
params4["classifier__n_estimators"] = [100, 200]
params4["classifier__min_samples_leaf"] = [1,2]
params4["classifier"] = [clf4]
```

In [ ]:
```
params5 = {}
params5["classifier__max_depth"] = [2, 3, 5, 10, 20]
params5["classifier__min_samples_leaf"] = [5, 10, 20, 50, 100]
params5["classifier__criterion"] = ["gini", "entropy"]
params5["classifier"] = [clf5]
```

In [ ]:
```
params6 = {}
params6["classifier__hidden_layer_sizes"] = [(10,30,10),(20,)]
params6["classifier__activation"] = ['tanh', 'relu']
params6["classifier__solver"] = ['sgd', 'adam']
params6["classifier__alpha"] =  [0.0001, 0.05]
params6["classifier__learning_rate"] = ['constant','adaptive']
params6["classifier"] = [clf6]
```

In [ ]:
```
params = [params1, params2, params3, params4, params5, params6]
```

In [ ]:
```
for param in params:
    grid = GridSearchCV(pipe, param, cv=5)
    grid.fit(X_train, y_train)
    print('Score:', grid.best_score_)
    print('Score:', grid.best_params_)
```