Alexander David Martin Colville

Analysis of Time Series for the Intelligent Support to Early Mobilization with Machine Learning

MASTER'S DEGREE FINAL PROJECT

Directed by Dr. Agustí Solanas

Master Degree in Computer Security Engineering and Artificial Intelligence

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2014

This is dedicated to Maria and Jose, who showed me what really matters in life.

Table of Contents

1. Introduction

Machine Learning (ML) has gained huge strength in the Artificial Intelligence field as computational power has increased in off-the-shelf machines. Furthermore, algorithms can process large amounts of data that can be stored and accessed with ease. The widespread of ML techniques has increased its usage in many fields, such as finance, logistics or healthcare.

This document covers the study and implementation of ML models that are applied to healthcare, more specifically, to early mobilizations in Intensive Care Unit (ICU) patients. Early mobilization consists of applying passive and active movement dynamics to hospitalized patients earlier than usual (even when they still require mechanical ventilation) with the aim of reducing physical and cognitive sequelae caused by a prolonged lack of mobility. With the application of this technique, the aim is to reduce recovery time and, consequently, hospitalization and return to work (in the case of patients of working age). The incorporation of physical activity at the beginning of the course of the critical illness produces better clinical outcomes, including a lower incidence of acquired weakness in the Intensive Care Unit (ICU), a lower incidence of delirium and a shorter stay in the ICU.

The project strives to find the underlying causes produced by specific motions, specifically on early mobilization. Early mobilization consists of applying passive and active movement dynamics to hospitalized patients earlier than usual (even when they still require mechanical ventilation) with the aim of reducing physical and cognitive sequelae caused by a prolonged lack of mobility. With the application of this technique, the aim is to reduce recovery time and, consequently, hospitalization and return to work (in the case of patients of working age). The incorporation of physical activity at the beginning of the course of the critical illness produces better clinical outcomes, including a lower incidence of acquired weakness in the Intensive Care Unit (ICU), a lower incidence of delirium and a shorter stay in the ICU.

This master thesis will be carried out in the context of the SISMO Project. The project was created with the aim of helping to reduce the recovery time of patients with prolonged lack of by supporting analysis on early mobilization to try and reduce recovery time for patients in an ICU unit. Widespread of early mobilization is not yet achieved, although SISMO aims to implement IoT and ML technologies to monitor patient progress. This is done by tracking active and passive mobilizations. Thus, the main goal of the project is to support healthcare professionals by applying machine learning techniques on a knowledge-base, or dataset, to recognize movements made by patients. These movements can be active or passive and can be simpler or more complex, depending on the mobilization exercise, although ML models are determined to classify them in order to distinguish mobility patterns. The evolution over time of patients can, later on, be tracked in order to analyse recovery evolution.

Section 2 covers the state of the art that was consulted to plan and assist the project from a theoretical point of view, whilst section 3 is a study of the specific ML algorithms that are used in the implementation. Section 4 explains the dataset that is used for its further analysis in section 5, where the implementation process as a whole is also covered. The obtain performance and timing results are presented in section 6. Section 7 is a discussion of how the ML models perform against the problem and if they are valid enough for real applications. Section 8 proposes the next steps the project can take in order to develop even further the work. Section 9 lists all references used in this thesis.

2. State of the Art

Machine Learning (*ML*) techniques are increasing in popularity in many fields due to the increased amount of historical data and the available computational power. This has led to a higher demand of accurate forecasting and robust and efficient techniques that reveal the observations between past and future[1]. This is due to the fact that ML works by "learning" from data and can be used in many environments, working with examples with a known output, i.e. supervised learning, and the ones which are not classified or labelled, i.e. unsupervised learning. The usage of these techniques affect the accuracy of the final predictions, although the goal is to obtain insight of the features within the data that reveal information that cannot be seen with simple analytical models. Further, reinforcement learning tackles unknown input-output pairs by obtaining the most optimal solution with non-exact mathematical models, by obtaining sub-optimal solutions that correct within a time span. The increase in scale of data and the usage of data-driven approaches has affected notably the health sector.

Medical and health-related practices have generated large quantities of data in the past years and the ability to learn from the data developed into more innovative approaches to tackle caveats in the field. Studies regarding nativity and mortality, environmental health, nutrition, cardiovascular diseases and genetics, including many others, find better solutions in the form of predictions resulting from *ML* approaches[4]. The healthcare sector strives for effectiveness and the amount of data the sector has affects positively the decision-making process. Nonetheless, the full potential of data-driven techniques is yet to be exploited. Data can be rather sparse and its synchronicity with analytics limits the usage of *ML* techniques diagnosis and treatment adaptations. Furthermore, more and more applications arise as data becomes more complex, algorithms increase in robustness and predictions are more precise. The management of any disease can be done with *ML*, as insight into its evolution can be tracked by analysing the different variables and their interactions. Even further, the merge of medical data with other data collections can allow the exploration of relationships between datasets and the effects inferred from one to another[4][6]. Examples of datasets to relate to are social science studies, work- or profession-related data or even usage of social media. All the previous can boost disease diagnosis and treatment personalising, as well as drug manufacturing, prediction of epidemics and even health record management techniques.

A common methodology for the usage of *ML* is done with sequential steps that are the collection of data, the pre-processing of this data, selection of features, classification of predictions and the presentation of results[3]. Thus, the first steps are focused on collecting the data and tuning it to obtain a dataset that enables the user to work with it comfortably. There are many ways to pre-process data and it highly depends on the type of data at stake, i.e. blank numbers can be replaced with a default value or a mean value, whilst text data can be neglected or relegated to another class or grouping. The main concern of ML, though, is the choosing of specific algorithms that make the predictions. There are many algorithms and mentioning all of them would be tedious and cumbersome. The selection of a specific algorithm varies on the type of data we are dealing with and, more important, the output want to obtain. Furthermore, all algorithms work different and there is no best option at first glance, as different algorithms have different strengths and weaknesses. For supervised learning, we find algorithms such as *Support Vector Machines*, *Linear Regression*, *Logistic Regression*, *Naive Bayes*, *Decision Trees* and *k-Nearest Neighbour*. For unsupervised learning, we find clustering algorithms i.e. *k-means* and *hierarchical*; *Anomaly Detection* algorithms; *Neural Networks*; *Expectation-Maximization*; and

Principal Components Analysis, within others[1][3][4][5][6]. All the previous operate differently and, as aforementioned, they have their own pros and contras. *Support Vector Machines* (*SVM*), for instance, have proved to give successful results to classification tasks and regression. It classifies data points and uses loss functions to map categories and divide them as much as possible[7]. On the other hand, *Bayesian Networks* are graphical models that strive to show probabilistic dependencies between variables by the representation of directed acyclic graphs. This proves useful in terms of interpretability, although sparse data can be challenging when combining structure learning techniques[6]. *Random Forests* models are based on the construction of decision trees as a combination of tree predictors that depend on the values of random vectors samples independently[3]. Further information for *ML* algorithms can be consulted in the references section. *Local Learning*, *Recursive Strategy* and *Direct Strategy* approaches appear in [1]; [2] presents *Neural Network* approaches and *Gaussian Processes*; *Multilayer Perceptron* can be seen in [2][3]; simple *ML* models and *Neural Networks* are compared in [4]; and [5] presents a series of clustering algorithms that vary according to online/offline settings and known/unknown number of clusters.

The previous study of algorithms is to determine which algorithms are best for time series analysis and forecasting. Time series data is formed by sequences of historical measurements of an observable variable at equal time intervals. Forecasting of the future can be done by using knowledge of the past. The goal is to determine underlying features the measurements may have, as well as descriptors of the salient features within a time series, which can be determined[1]. The main challenge in time series forecasting is the determining of the random behaviour that provides non-linear interaction. Data points can have aperiodic behaviour and can be asymptotic, presenting different dimensions in terms of length[5]. On the other hand, time series analysis focuses on the direct understanding of the underlying causes in the data to develop mathematical models that provide plausible descriptors from sample data[9].

Time series are abundant although little is known about the nature of processes that generate them. This calls for specific strategies that have to be undertaken to enables successful analysis and forecasting. Strategies focus tackling the problem of making predictions based on the trends found in the data and its repeating patterns or cycles[9]. Predicting involves the determining of the future with only data subsets from the past. This can be done via local learning strategies. Local learning assumes no the data as plain - no previous knowledge applied, computational simplicity when new training samples are input - no re-training, and modelling is non-stationary, which enables the comparison of data points in a spatial and temporal way. The latter may provide further accuracy when making predictions. For one-step time series, we find *Nearest Neighbour* and *Lazy Learning*[1]. The *Nearest Neighbour* method bases its predictions on stating the output of a specific data point from the initial state to the final state with the evolution of the nearest neighbouring point with an already known output. *Lazy Learning* adapts the size of the neighbourhood by reducing a complex and non-linear problem into a sequence of problems and applies cross-validation criteria. Thus, we obtain manageable and linear problem sequences to solve. For multi-step data, we find recursive strategies, direct strategies and the combination of both that will apply to data horizons. Recursive strategies train a one-step model in order to use it recursively to return multi-step predictions. Direct strategies, on the other hand, learns models independently and concatenates all output predictions[1].

Dealing with time series can be complex. As aforementioned, time series can be dealt with depending on their periodicity and the step we are able to or willing to evaluate. Thus, datasets can be simplified and we can interpret problems sequentially or in a divide-and-conquer

manner. In time series analysis, in order to cluster or classify time series, subroutines can be used. This will enable the location of stable periods of time or to allocate changes in data points. A common subroutine is segmentation. Segmentation divides into discrete segment sequences the time series in search for inflection points[8]. The researched approach uses a criteria that is used as a threshold for stopping the iterative merge of the lowest cost pair of segments. The purpose is to find homogeneous segments, and its formalisation is based on the cost of the individual time intervals, by means of a cost function. The cost function evaluates the distance between a value and the data fitted by means of a simple function. The detection of changes in the correlation structure can be done in several ways. *Singular Value Decomposition*, for instance, is a model that projects the correlated high-dimensional data onto a hyperplane, where multivariate data can be analysed. The measured distance between the initial data and the hyperplane indicates the variations for the observed variables. This model strongly depends on the rank of the decomposition. A *Critical Point* approach, on the other hand, smoothes out a noisy signal with a filter - usually base-band-pass - in order to reduce its fluctuation and determine critical points[8]. Many other segmentation techniques can be applied, as well as the combination of them, in order to provide time series segmentation.

Furthermore, when a time series dataset has been segmented, it can thus be classified according to the found patterns. This technique is mainly applied in unsupervised learning and the main goal is to assign a set of observation to a specific subset, as the observations in a cluster must be similar in some sense[8]. Clustering or classification must always be done by evaluation expressive features that must be selected in order to fulfil the desired needs.

Overall, the state of the art points out that the increasing amount of data is enabling the widespread of *ML* techniques. Dealing with increasing amount of data is easier and more effective in different fields, such as finance, economics or healthcare. The latter can offer more accurate diagnosis and personalised treatment due to the performance of *ML* algorithms that make this possible. Furthermore, the collection of time series datasets tracks evolution of data over time and tackling this is becoming more feasible and of major interest to develop *ML* at a whole new level. Dealing with time series can be complex, although there are already consistent models that can provide insight of the features underlying in the variables. Also, several strategies strive to obtain the periodicity of data its cycles, in order to further cluster or classify the resulting predictions.

3. Machine Learning Algorithms

Theory about ML can be thorough and there are many algorithms with many variations and extensions. The reader must bear in mind that the implementation deals with a classification problem. Not to keep the document extremely tedious, the following points strive to collect the most important technical features to understand the ML algorithms used in the implementation and support the decisions agreed upon within the project[12].

3.1. Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure which goal is to measure data in terms of its principal components rather than on its normal axis. This means, the underlying structure of the data is analyzed to figure out where the highest variance of the components is found. PCA uses orthogonal transformation, that converts the correlated variables in the samples into linearly uncorrelated variables. The variables found are the principal components.

PCA applies to the data by transforming it to a new coordinate system for each component, that is projected sequentially. Variance is maximised by evaluating components one by one by means of weight vectors $w_k = (w_1, \ldots, w_p)_k$ which maps each sample x_i to a new vector, the principal components score vector, $t_i = (t_1, \ldots, t_l)_i$. Mapping is done:

$$t_{k_i} = x_i \cdot w_k \text{ for i=1, .., n and k=1, …, l.}$$

The first weight vector satisfies:

$$w_1 = arg \max_{||w||=1} \{\sum_i (x_i \cdot w)^2\}$$

The finding of the first component can give us the score of t_1 in the transformed coordinates. The rest of the components are calculated with the weight vector that extracts maximum variance:

$$w_k = arg \max_{||w||=1} \{\left||X_k w|\right|^2\}$$

Where the *k*-th component is the subtraction of the first k-1 components to X:

$$X_k = X - \sum_{s=1}^{k-1} X w_s w_s^T$$

Transformations are truncated by using eigenvectors. These relate directly to the desired amount of components that compound an uncorrelated dataset over the new space. PCA learns a linear transformation with an orthogonal basis for the number of specified components, maximising the variance found in the features.

As we are dealing with high-dimensional data, PCA can be useful to reduce these dimensions and make representations in a 2- or 3-dimensional space. Spreading out the principal components in a visualisation can help to observe at first glance the results that should be expected or to even see if any margin can be created to separate the dataset according to its labels.

3.2. Naive Bayes

Bayes' theorem is a theorem that determines an event probability regarding related events, which might be related to the event by means of their conditions. Based on the previous, Naïve Bayes has a set of methods that assume naïve independence between feature pairs. It is used for supervised learning purposes and classifiers are built by assigning labels to problem instances from an initial finite set. These instances are represented as feature value vectors. Thus, Bayes' theorem can be represented as:

$$P(B|A) = \frac{P(B|A)\ P(A)}{P(B)}$$

Naïve Bayes can be represented as:

$$P(B|a_1,..,a_n) = \frac{P(B)P(a_1,..,a_n|B)}{P(a_1,..,a_n)}$$

Naïve independence assumption gives us, for all *i*:

$$P(a_i|B,a_1,..,a_n) = P(a_i|B)$$

The relationship, thus:

$$P(B|a_1,..,a_n) = \frac{P(B)\Pi_{i=1}^{n}P(a_1|B)}{P(a_1,..,a_n)}$$

If the probabilities for all *a* values are constants, an estimation can be done to the output. Some estimators can be *Maximum A Posteriori* estimation, *Minimum Square Error* or *Alternative Risk Functions* of the output constants. Furthermore, naïve Bayes classifiers differ in results when it comes to assuming the relative frequencies of classes in the training set, considering the distribution of $P(a_i|B)$.

The previous demonstrates the simplicity of naïve Bayes methods, as the correlation between features are not considered and, therefore, its fast computation over large datasets. Its results in real-life application prove to be decent for classification, although it encounters flaws when it comes to estimation[14].

Classification of output results in a naïve Bayes method can be done in several ways. Therefore, we can find:

a. Gaussian Naïve Bayes: when dealing with continuous values, data associations that follow Gaussian distributions can be found. Probability distributions can be computed by means of a normal distributions equation.
b. Multinomial Naïve Bayes: the representation of event frequencies in the form of a multinomial is composed by the samples, or feature vectors. Thus, observations of an event is counted and classification occurs by the representation of occurrences. It is an interesting approach as it can be easily represented as a histogram, but it may encounter errors when it comes to estimation, as pseudo-counts should be considered to avoid probability wipe-outs, as values may be multiplied by zero, in the case that there is no occurrence of a specific class.
c. Bernouilli Naïve Bayes: features in this model are treated as binary variables, proving to be useful when it comes to expressing absence or occurrence of events based on the likelihood of a class expressed by the binary value.

d. Others: continuous values can also be dealt with by binning to discritize feature values and obtaining a Bernouilli distribution. Also, semi-supervised training algorithms can be trained to learn by running in a loop the supervised algorithm. These event models and other estimators can be applied to modify or extend a naïve Bayes model.

In the implementation, Gaussian Naïve Bayes (GNB) is going to be used. This decision was taken to see determine whether a specific feature from an (x, y, z) input is determining for specific motions without correlating the values.

3.3. Support Vector Machine

Support Vector Machines (SVM) are supervised learning models that represent features as points in space as a non-probabilistic binary linear classification by constructing one or more hyperplanes in a high-dimensional space for further supervision. It implies the mapping of features as points in space and outputs predictions based on the mapping of the feature that fall into a categorical space. The usage of hyperplanes enables classification or regression analysis of test data inputs by measuring the distances to the training data points within the hyperplane. Furthermore, high-dimensional space is used to deal with the fact that not all features sets can be separated linearly [13][16].

In terms of computation, SVM works with dot products of pairs of input data vectors within the original space and deals with the operation as a kernel function. The hyperplanes in high-dimensional space appear when the dot product of a vector with a set of points is constant within the dimensional space. Thus, vectors that define a hyperplane are represented as images of feature vectors within the dataset and the degree of closeness of a test point can be determined when the relation of parameters with the kernel becomes constant. It is to be noted that convolution appears when hyperplane mappings are done, providing major complexity to the discrimination offered at the time of classifying. SVM offers linear and non-linear classification, as well as several extensions.

3.3.1. Linear Support Vector Machines

Linear SVM strives to predict the classification of (x_i, y_i) points when the distance between the hyperplane and the nearest point, from both groups that divide the set x_i for which $y_i=1$ *or* $y_i=-1$, is maximized. The maximum-margin hyperplane is found and normal vectors per hyperplane can be found. Thus, linear separation is found and, furthermore, the determination of a margin is set. This margin can be hard or soft, determined by the linear separation of the hyperplanes surrounding it. If two hyperplanes are linearly separable, a hard margin is found and constraints set that data points must lie on the adequate margin side. The optimal weight vector for linearly separable data:

$$w_o = \sum_{i=1}^{N_s} \alpha_i y_i x_i$$

Where α is the Lagrange multiplier vector that finds the optimal solution and N_s the total amount of support vectors. Therefore, having w_o, the function for the optimal hyperplane:

$$y(x) = sgn(\sum_{i=1}^{N_i} y_i \alpha_i (x^T x_i) + b_o)$$

Where b_o is the optimal bias.

Data is classified, then, according to the sign of the function result. On the other hand, if linear separation cannot be accomplished, a loss function is introduced to determine whether if set of points lay on the correct side of the margin.

3.3.2. Non-Linear Support Vector Machines

As a variation of a linear SVM model, non-linear SVM replaces the dot product with a non-linear kernel function. Feature spaces is then transformed in order to fit the maximum-margin hyperplane. This is introduced by means of hard-margin constraints and penalty misclassifications. Multiple kernels can be examined, but the most common are the polynomial kernels, being these homogeneous and non-homogeneous. Homogeneous polynomial kernels are those in which all terms have the same degree, whilst, on the contrary, non-homogeneous kernels encounters polynomials with different degrees in some or all terms.

Classification in SVM is accomplished by the computation of the reduction of a problem to a quadratic space. Nevertheless, this is not always the case and extensions have found other ways to find the margins, such as sub-gradient descent and coordinate descent, although the classification problem must be solved by the minimization of the loss function.

In the implementation, SVM is chosen due to the fact that the data can be easily expressed as data points, with its pre-processed coordinate values or after analyzing its principal components. Striving to find the margins between the classification of the data points whilst observing the obtained loss can be interesting.

3.4. Decision Tree

Also used in supervised learning environments, Decision Trees (DTs) are non-parametric methods used for regression and classification goals. Observations or features are used to obtain predictions, being this a variable target value, by means of simple decision rules, representing conclusions as in leaves according to inference. When dealing with classification problems, the tree structures are represented so as the leaves represent class labels and the branches represent the conjunction of the features that lead to the leaves. As well as classification trees, regression trees can be found to solve ML problems where the predicted outcome should be a real number, and not a class, like in the latter problem type.

When it comes to the specific algorithm used in these model structures, we can find:

a. *Iterative Dichotomiser 3 (ID3)*: the algorithm finds categorical features for each node by yielding the largest information gain for categorical targets by means of a multiway tree. It works in a way that generalization to unseen data are avoided by applying a pruning to a previously maximized tree. First, it calculates the entropy for all features. Then, a splitting is done using the attribute for which the entropy is minimum. Next, a decision tree is modeled based on the previously found attribute and, lastly, recursion is applied to all subsets with the remaining attributes.
b. *C4.5*: In line with ID3, C4.5 uses the concept of information entropy whilst removing feature categorical restrictions. It thus defines a discrete feature that partitions the continuous attribute value into a discrete set of intervals[18]. The nodes are associated to attribute values and subsets are created, as in ID3, according to the normalized information gain. Further pruning is done by removing rule preconditions, depending on the accuracy of the rule itself.

c. *C5.0*: C5.0 is a further development of C4.5 that uses less memory and rulesets are decreased in size, although accuracy is increased.
d. *Classification And Regression Tree (CART)*: this algorithm is non-parametric technique that creates decision trees with rules based on values for selected values to split the most differential observations based on variable dependencies. On rule selection, it is applied to all child nodes recursively, stopping when set conditions are met or no further gain is possible. Pruning also applies to CART.

Decision trees are very easy to visualize and interpretation is very straightforward. The algorithms have logarithmic cost, proportional to the number of training nodes and statistical tests can be run to validate models. Nevertheless, stability can complicated to deal with as generalization can be hard to find depending on the dataset, requiring prior fitting and more pre-processing logic in a specific implementation.

A decision tree classifier that uses the CART algorithm is proposed within the implementation. This is due to the fact that the document strives to find whether time series datasets can be classified in a structural manner and easy visualizations can be obtained from data it is complex to interpret at first hand. Further, basic motions follow strict moving pattern, that can be reflected as a set of rules that could be applied by the algorithm.

3.5. Multilayer Perceptron

Multilayer Perceptron (MLP) is an artificial neural network that uses feedforward techniques to train datasets. The algorithm is represented by means of layers of nodes, where each node is a neuron. Neurons use non-linear activation functions, which can be arbitrary, to distinguish data that is not linearly separable. MLP, thus, learns an approximation function in order to classify a dataset. It can also be used for regression. It is formed by, at least, three layers, where perceptrons are organized. A perceptron is, by definition, a binary classifier – an artificial neuron.

MLP has a fully connected and is layer-structured. Layers are divided in three groups, being these the input layer, output layer and all in-between hidden layers – it needs a minimum of 1 hidden layer. Nodes are connected from one layer to another with a specific weight value.

Neurons encounter activation functions. An activation function is linear and takes action by mapping neuron weighted inputs to their outputs. The most common activation functions are sigmoidal:

a. Logistic function: this is a sigmoid curve that ranges results between 0 and 1. It is shown that initial growth is almost exponential until saturation strives to accomplish a stable, or mature, value that stops the growth.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

Where L is the maximum value of the curve; k is the curve steepness; x_0 is the midpoint of the sigmoid.

b. Hyperbolic tangent: this is defined as the ratio of the corresponding hyperbolic sine and hyperbolic cosine functions. This enables the evaluation of the logarithm of a rational number of the arguments.

$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$

We must bear in mind that in both previous formulas, the x_i values represents the value at the node – sometimes represented as v_i. Weights are typically represented as w_i.

When it comes to learning, MLP works by changing connection weights. Output neurons are those that are no source of any connection and depend on the targeted values, being these the ones in the test set. The connection scheme, organized in layers, is acyclic.

The algorithm works by enumerating all neurons and checking the existing connections, which are weighted. Bias weights are used in the hidden layers. Backpropagation is used and, thus, the algorithm executes as follows:

1. Input values are propagated into the network;
2. the cost is calculated and, therefore, the error term is determined;
3. the difference between the target value and actual value is calculated;
4. the weights are updated by multiplying the activation function result and the output delta;
5. a percentage of gradient weight is subtracted from each weight.

Iterations in the backpropagation process are controlled by number of epochs. This strives to minimize the error in the output nodes and the predicted output tends to approximate to the training example label.

MLP is used in this implementation because of its stochastic nature. This can provide useful to determine solutions for complex problems, such as time series predictions. Also, MLP is widely used to create models based on regression analysis and classification can then be done because of the categorical response that is found in the output layer of the network.

4. Early Mobilizations Data

Body motions are recorded from wearable sensors respectively placed on the chest, wrist and/or ankle of a subject. The usage of the sensors allows to measure the motion experience by one or multiple body parts, namely, the acceleration captured by the body dynamics. Accelerations are thus shown as G-forces and these compose the dataset of time series data that reflect motion primitives.

In order to provide a valid dataset for the main purpose of this project, the tacking and analysis of basic motions or movements, two different datasets have been analyses and pre-processed. These datasets are used for several applications, such as the generalization of common activities of the daily living or the adoption of common test benches for creation and validation of human motion primitives. The review of the different motions in both datasets helps the choosing of specific data basic motions that can be found in the ICU. Detailed documentation about the dataset can be found in [10][11].

Time series are mainly set of points measured over successive times. The elapsed time is represented, usually, by means of a vector. Data points are thus related to a specific time measurement. The most complicated part of assessing time series data is the determination of trends, cycles, seasons or irregularities. Data can repeat itself over periods of time, creating trends by the variations seen in increases, decreases or stagnations. Cycles can appear when repetitions are done over long periods of time – cycles tend to be generalizations of repetitions over time. Variations caused at random are referred to as irregularities and there is no way to define the root cause of irregular fluctuations. Irregular data variations can be complex to deal with, although models can approximate them to other observations.

This project uses time series data points associated to periods of time when motions are done. The variables observed in the discrete time series are shown as continuous variables that are expressed by sets of real numbers. These numbers are the location in space of the accelerometer used on a certain subject. More specifically, recordings of 3-dimensional coordinates are done with the accelerometer and the obtained G-forces are collected. Thus, we obtain sets of three real numbers that are named as follows, to differentiate between the part of the body where the accelerometer is found:

1. “x_wrist”, which tracks the x coordinate from the wrist accelerometer;
2. “y_wrist”, which tracks the y coordinate from the wrist accelerometer;
3. “z_wrist”, which tracks the z coordinate from the wrist accelerometer;
4. “x_chest”, which tracks the x coordinate from the chest accelerometer;
5. “y_chest”, which tracks the y coordinate from the chest accelerometer;
6. “z_chest”, which tracks the z coordinate from the chest accelerometer;
7. “x_ankle”, which tracks the x coordinate from the ankle accelerometer;
8. “y_ankle”, which tracks the y coordinate from the ankle accelerometer;
9. “z_ankle”, which tracks the z coordinate from the ankle accelerometer.

We must bear in mind that not all data within the data has all the three sets of 3-dimensional coordinates, as some motions are only recorded by means of a wrist accelerometer. This can provide as an inconvenient in terms of analysis, but it can also provide advantageous in the case that wrist movements are more determining in terms of forecasting for the specific movements that are in the pre-processed dataset.

The chosen data refers to specific movement dynamics. These have different complexities and a different sensor setup has been used for each one of them. Six different movements are used, being the following, with their assigned labels:

a) Get up from bed (labeled as “0”)
b) Drink from a glass (labeled as “1”)
c) Waist bend forward (labeled as “2”)
d) Lying down (labeled as “3”)
e) Frontal elevation of arms (labeled as “4”)
f) Crouching i.e. knees bending (labeled as “5”)

The total amount of data regards 206406 samples, where the distribution of samples is not equal for each of the labels. Thus, we find 42792 samples for label 0, compounding the 20.73% of the data; 45801 samples for label 1, being this 22.19% of the dataset; 28315 samples for label 2, 13.72%; 30720 samples for label 3, thus being 14.88% of the data; 29441 samples for label 4, the 14.26% of the dataset; and 29337 samples for label 5, compounding the 14.21% of the whole data.

UCI patients can do the previous movements in their recovery phase. Thus, the selected motions arise when a patient signals something within the room, so he will elevate her arms to point a specific object; when bending her knees in bed or crouching, to change posture or try to flex her lower limbs; bending her waist forward to incorporate the upper body before eating; getting up from bed and brushing her teeth, being these motions in the latter part of the recovery phase and becoming more complex motions.

5. Implementation

In the previous chapter, several popular algorithms are studied and the basic technical features are outlined. Time series forecasting can be done by means of these algorithms and, therefore, models can be created to be trained on time series datasets and, when trained, fed with new data inputs to find predictions for the future.

This section covers the approach given to make predictions on time series data provided by early mobilization. The observations found in the datasets described in section 3 will be used to construct models that will give us training and test sets. Construction of the models will allow us to generate forecasts and, even further, if a validation set is used, observations can be verified. Time series forecasting with early mobilization patterns can be done with many different models for different reasons, depending on the way the data wants to be evaluated in terms of accuracy, performance and functionality. We must bear in mind that data must be pre-processed to deal with irregularities in the data and avoid empty fields when it comes to training and testing, as this may cause error in terms of accuracy or in the algorithms executions. Heuristics are applied to choose the training and testing datasets. Thus, 90% of the data will be dedicated to training and 10% of the data will be dedicated to testing. The number of samples selected is done at random and splitting is done according to the percentages to avoid knowledge transfers from the training set into the test set . Sections 4.4 and 4.5 cover each process, whilst section 4.6 covers a specific test set obtained from the SISMO project accelerometer.

5.1. Tools

The implementation is done on a machine with Windows 10 Enterprise 64-bit Operating System with an Intel® Core™ i5-7300U CPU @ 2.60GHz processor and 8.00GB of RAM.

The software is developed using an Anaconda 1.8.7 distribution, which is a very popular Python distribution for data science applications. The distribution has Python 3.6.5, which is programming language used for the implementation. Jupyter Notebook is the open-source web application that allows the creating of documents that contain Python code that can be compiled and executed in the web browser. From there, the Python code imports several modules or packages, being these:

a. *Scikit-learn*: package for machine learning that enables the usage of classification, regression and clustering functions by many different algorithms.
b. Numpy: package for scientific computing of N-dimensional array objects, broadcasting functions and useful linear algebra in data science applications.
c. Pandas: package for high-performance data analysis and large data structure preparation and management.
d. Matplotlib: package for 2D and 3D plotting with a variety of hardcopy formats and interactive environments across platforms.
e. Statsmodels: package with functionalities to deal with estimation of statistical models as well as exploration of statistical data and testing.

Therefore, the implementation contains a Jupyter Notebook document where Python code is executed. This code imports several packages in order to do create ML models and show results and visualisations for the predictions obtained.

5.2. Data Pre-Processing

The first step after having obtained all the datasets is to apply pre-processing techniques. Data gathered cannot be checked at a glance and assumptions that the data is fully controlled in the analysis process will arise code errors, data mismatches or other mistakes. Some issues that are to be dealt with are:

a. missing values,
b. out-of-range values,
c. wrong value types,
d. empty value fields.

Furthermore, irrelevancies, redundancies and other data that can produce misleading results has to be processed to obtain a stable and standardised dataset. In order to do so, this chapter cover the pre-processing steps done to assess the previous and obtain a competent dataset.

On the one hand, a series of functions are used to scale all the three dimensional data points. This means, normalising the dataset to input some boundaries, or limits, and obtain a more standardised dataset. This will produce better interpretations for data and will delete large spatial distances between data points. Furthermore, a more homogeneous dataset will derive in clearer visualisations, giving showing better separation between feature sets with different labels. As the first dataset used uses normalised values, the G-forces, between -1.5g and 1.5g, the whole dataset will be normalised to (-1.5g, 1.5g) values. This process can be done with a minimum and maximum scaler, provided within the *Scikit-learn* library, which can transform feature values individually into the range. The scaler only needs the dataset and the desired range it has to be normalised to as parameters[19].

On the other hand, the dataset is set to a standard set of columns. This means, that all data-points must have valid values. This is achieved by means of the *Pandas* library, allowing the creation of *DataFrames* with a set amount of columns, where each columns has a unique identifier. In addition, any columns with missing values are dropped to avoid later code failures. Having loaded the external datasets into *Pandas DataFrames*, these can be concatenated, bearing in mind that a sequential index, that can be easily reset to set a unique identifier to each row in the dataset. The dataset itself, with a standard amount of features, unique columns naming and normalised values is completed by a factorisation of its labels. Although feature values relate to a motion, the motion itself should not be a determining factor within the dataset and, thus, each motion is labelled to a value within the range (0, 1, …, 5). The label is the determining factor when it comes to predicting. The table below shows five random samples of the dataset.

	x_chest	y_chest	z_chest	x_wrist	y_wrist	z_wrist	x_ankle	y_ankle	z_ankle	label
82110	0.0000	0.00000	0.00000	-9.573158	4.903325	11.441092	0.00000	0.0000	0.00000	1
200193	4.4726	6.16720	-8.20290	0.895480	-12.922000	-1.208600	1.33190	-9.0918	3.69410	2
147820	-9.6590	-0.38581	1.64950	0.577270	7.791600	4.160400	0.14048	-9.4800	1.50660	4
188401	-6.2912	0.41903	0.29034	-0.861530	-2.720400	-0.042271	2.70170	-9.3503	0.61158	2
150616	-7.9949	0.24875	-2.56710	-1.686400	-8.519500	2.053200	0.38245	-9.8930	0.80084	5

Figure 1. Dataset sample rows

5.3. Feature Selection

Data analysis is done exhaustively with the usage of several algorithms that will interpret different features. Features are the individual and measurable properties or characteristics within the data, being these each G-force from each different sensor at a specific point in time. Each motions encounters different features and these can vary in importance. What is more, all data regarding the 0 and 1 motions is only defined by 3 features, being these (x_wrist, y_wrist, z_wrist) - the other features with the 0 and 1 labels are set to 0.

Thus, motions will be evaluated in several ways according the features:

a. With three features, being these: (x_wrist, y_wrist, z_wrist)
b. With a set of principal components (linearly uncorrelated set of values) found after applying Principal Component Analysis (PCA). These features will be named (a, b), as PCA will strive to find the two most deterministic features.
c. With all features, being these (x_chest, y_chest, z_chest, x_wrist, y_wrist, z_wrist, x_ankle, y_ankle, z_ankle)

5.4. Training

The previous section covers the pre-processing of the mobilization datasets into a standardised and homogeneous dataset. Thus, the next step for the implementation is the training of models with the dataset.

Training the dataset is quite straight-forward when using *Python* and *Scikit-learn*. Nevertheless, the selection of parameters has to be done appropriately to obtain a model that will give us the best results in terms of prediction. Machine learning models are trained on a number of samples and try to predict properties of unknown data. In this implementation, we are only covering supervised learning schemes[section 3], as we have a dataset that is labelled.Also, we are dealing with a classification problem, so we are going to use classifier models.

First, the training set has to be defined. The training set will depend the number of features or components we are looking into[section 4.3]. Training or learning in *Scikit-learn* is achieved by means of an estimator. An estimator is, basically, a rule applied for the estimation of a specific value. *Scikit-learn* has the `fit(X, y)` method to which the training set is passed on to, being X the features values in the training set and y the labels to which these values map to.

5.4.1. Support Vector Machine

Support Vector Machine models in *Scikit-learn* can be used as classifiers. Although scalability can be quite complex when it comes to large datasets with many sample – it is computationally expensive – it provides many parameters to tweak the algorithm to the needs of the dataset and strive to find better predictions. These parameters can be seen in [16] and, mathematical formulation proves that the parameters of most importance are the definition of the C, kernel, gamma, coef0 and degree parameters. This is because they allow the user to determine the error term – C – and the kernel type, its polynomial degree, coefficient and independent term. Tolerance, random state and the maximum number of iterations can also be set when instantiating the class.

The implementation will train the SVM classifier model a singleset of parameters, for each feature selection. This is because SVM can take long periods of time for large datasets and the main parameters we focus this models is the kernel function.

1. `SVC`(*C=1.0*, *kernel='rbf'*, *degree=3*, *gamma='auto'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *random_state=None*)

5.4.2. Gaussian Naïve Bayes

Gaussian Naïve Bayes is used as an algorithm for the classification model. It strives to find likelihood of features and it can be tweaked by adding the prior probabilities of the classes. This can provide useful when we know that a specific label appears with more probability than another, thus being prioritised when it comes to prediction[17].

The implementation will train the GNB classifier model the modification of priors, in order to vary the priorities due to motion complexities and the amount of data points, for each feature selection:

```
1. GaussianNB()
2. GaussianNB[0.2, 0.2, 0.15, 0.15, 0.15, 0.15])
```

5.4.3. Decision Tree

Decision Trees train a model with the goal of predicting values of a targeted samples by learning simple decision rules, provided by the sample features. Classification with DTs can be adjusted with parameters that allow the measurement of the quality of a split (*criterion*), the choosing of split at each node (*splitter*), the maximum depth of a tree (*max_depth*), the minimum number of samples required to split an internal node (*min_samples_split*) and minimum number of samples required to be at a leaf node (*min_samples_leaf*), the minimum weighted fraction of the sum total of weights (*min_weight_fraction_leaf*), the number of features considered when looking for a split (*max_features*) and the seed used by the random number generator (*random_state*). Also, the maximum leaf nodes (max_leaf_nodes), minimum impurity decrease and growth threshold (*min_impurity_decrease* and *min_impurity_split*), as well as the class weight (class_weight) and the option to whether the data should be presorted for fitting (presort) can be decided upon.

The implementation will train the Decision Treeclassifier model with several parameters, for each feature selection:

1. `DecisionTreeClassifier`(*criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *class_weight=None*, *presort=False*)
2. `DecisionTreeClassifier`(*criterion='entropy'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=1*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *class_weight=None*, *presort=True*)

5.4.4. Multilayer Perceptron

Neural networks can also be created and modelled in *Scikit-learn*. The MLP classifier implies a neural network with several parameters that strives to optimize the log-loss function using LBFGS or stochastic gradient descent.

The number of neurons within each hidden layer (*hidden_layer_sizes*), activation function (*activation*), penalty (*alpha*), size of the batch for the stochastic optimizers (*batch_size*), learning rate (*learning_rate*), exponent for inverse scaling learning rate (*power_t*), maximum number of iterations (*max_iter*), the ability to shuffle samples per iteration (*shuffle*), random state (*random_state*), tolerance (*tol*), momentum (momentum), proportion of training data to set aside (*validation_fraction*), exponential decays (*beta_1* and *beta_2*) and numerical stability (*epsilon*) can be defined.Furthemore, verbosity and re-usage parameters can be set to feed user purposes[20].

The implementation will train the Multilayer Perceptron classifier model with several parameters, for each feature selection:

1. `MLPClassifier`(*hidden_layer_sizes=(50,)*, *activation='relu'*, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*)
2. `MLPClassifier`(*hidden_layer_sizes=(100,)*, *activation='tanh'*, *solver='lbfgs'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*)

Training the models with the previous classifiers allows us to assist the data from early mobilizations considering different aspects related to time series. Therefore, SVM, GNB and DTs allow the training of models with imbalanced learning and MLP can provide large numbers of hidden layers in artificial neural networks. These ML techniques will allow the further distinguishing of mobilizations from one another by means of the predicted classifications.

5.5. Testing

Section 4.4 outlines the testing of the dataset that collect early mobilizations. Thus, a 90% of the data within the set is used by the algorithms to learn the underlying patterns. Belonging to the same distribution as the training data, test data has not yet been processed by the algorithm, and feeding the model with such data thrives to find predictions with this fresh data. The performance of such test will determine whether the model will generalise the dataset and works well on doing predictions of this type. Therefore, ML models with good performance will classify time series data and will be competent enough for forecasting.

Scikit-learn provides a very simple way to obtain predictions from a dataset. This step implies the fact that training must have already been done in order to output new values. These values are output as an array with the same size as the number of input samples that were given in the

to the fit function. In particular, all classifier models can be asked to output their predictions of a test set after learning. This is done by:

```
classifier_model_name.predict(test_X)
```

where test_X is the array-like structure with the same amount of features as the training set, but usually a reduced number of samples. The output array, or predictions, will be the labels that each sample has fallen into. Thus, the output and input array are the same length, being this the number of test samples.

5.6. Performance Measures

This section discusses the usage and properties of performance measure for time series forecasting.

1.6.1. Mean Square Error

Mean Square Error (MSE) measures the average square deviation of the predicted values, reflecting the large errors that occur in the prediction. It gives an overall idea of the error occurred during forecasting.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

The Sum of the Square Error (SSE) can also be measured as the total square deviation of the predictions from the sample values. Therefore:

$$SSE = \sum_{t=1}^{n} e_t^2$$

The Signed Mean Square Error (SMSE) keep the sign for all individual squared errors to panelise outstanding errors.

$$SMSE = \frac{1}{n}\sum_{t=1}^{n} \left(\frac{e_t}{|e_t|}\right) e_t^2$$

Last but not least, the Root Mean Squared Error (RMSE) is a commonly used performance measure that simply calculates the root square of MSE. Thus:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

1.6.2. Mean Forecast Error

Mean Forecast Error (MFE) measures the average deviation obtained from the predictions regarding the actual labels. Direction of error is shown, implying that a result that is close to zero does not particularly imply a low amount of errors, but a low bias of errors on both directions.

$$MFE = \frac{1}{n}\sum_{t=1}^{n} e_t$$

1.6.3. Mean Absolute Error

The Mean Absolute Error (MAE) measures the absolute deviation of predictions from the original label values. This results in the determination of an overall magnitude of the error, although direction is not specified.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|e_t|$$

After the training of the various models and having the predictions from different sets of features, the following section collects a table where all the mean squared error, root mean squared error and mean absolute error of the predictions are covered, in order to evaluate performance with the most relevant error rates and observe which ML model gives best results.

5.7. Validation: SISMO Project Data

Early mobilization consists of applying passive and active movement dynamics to hospitalized patients earlier than with the aim of reducing physical and cognitive sequelae caused by a prolonged lack of mobility. With the application of this technique, the aim is to reduce recovery time and, consequently, hospitalization and return to work, given the case. The incorporation of physical activity at the beginning of the course of the critical illness produces better clinical outcomes, including a lower incidence of acquired weakness in the ICU, a lower incidence of delirium and a shorter stay in the ICU.

With the aim of supporting healthcare professionals in the process of implementing and generalizing early mobilization in practice, the SISMO project develops technology based on the Internet of Things (*IoT*) and machine learning. The *IoT* technology makes the SISMO project robust and guarantees its viability as its development goes beyond its scope. The SISMO project uses wearable devices, specially designed and adapted for the task of monitoring the mobilization, interconnected through efficient communication protocols such as Bluetooth Low Energy (LE), with its own software developed in house. This software is used within this implementation to obtain a dataset that will be used to validate the ML models.

The software, that has been slightly modified to obtain small datasets in text files which can be easily imported into the project notebook, is used for validation. Therefore, we do not only test the models with the test set from the given datasets, but we can also test with new data obtained from the replication of motions. These motions are obtained by allocation an accelerometer on one part of the body, this mean the wrist on the right arm of a subject - thus creating a validation set of three features. The output text file is named according to the motion label (recall section 3) e.g. the motions recorded when drinking from a glass with be under the text file named "1.txt". This naming will be using for the further comparison of data, as all the sample predictions will be compared to the label. Following the example, the output array values will be compared to an array of the same size where all values will be 1. Within the file, we find a set of three G-forces for each row separated by a space. This will be imported into an array in the Python code, leaving it ready to apply to predict(X) method for each model that has been previously fitted with the training set.

The training set used will be the one used for the models that evaluate three-feature, to follow conventions that only (x, y, z) coordinates for the wrist will be evaluated. Nevertheless, the same procedure can be applied and measurements can be taken from an accelerometer at the chest and ankle. Having only one accelerometer, it is not possible to merge the data from three different measurements on different body parts even though the motion is the same one. This can induce larger errors when it comes to predictions, and this implementation thrives to validate as accurately as possible the models.

Validation of the ML models is covered in the notebook and the following section collects the obtained results. We must bear in mind that, in order to avoid unnecessary calculations, only the best models with best predictions rates, and lower error rates, are used for the validation set. In section 7, discussion about the best models is explained based on the observations of section 6.

6. Results

The implementation covered in this document attempts to train and test ML models with a specific dataset created from various mobilizations from patients in the ICU. Therefore, the results obtained regarding the testing and validation phases are collected below. The results show the time undertaking for training the model, the precision of the model when predicting test samples for each label, the mean square error, root mean square error and mean absolute error values, as well as 2D (in the cases for PCA) and 3D visualisations to enhance visualisation of areas where data points are predictions correctly and mistakenly. For all 3D visualisations, green data points show the correct predictions and red data points the mistaken predictions. The blue area is the area determined by the training dataset. Thus, models will prove to be good at predicting when points that fall out of the training set area are predicted correctly. All parameter sets can be consulted in section 5.3.

6.1. Three features

6.1.1. SVM

Results for SVM with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.82				
1	0.89				
2	0.72				
3	0.97				
4	0.84				
5	0.64				
AVERAGE/TOTAL	0.82	1.13	1.07	0.42	18min 51s ± 1min 43s per loop

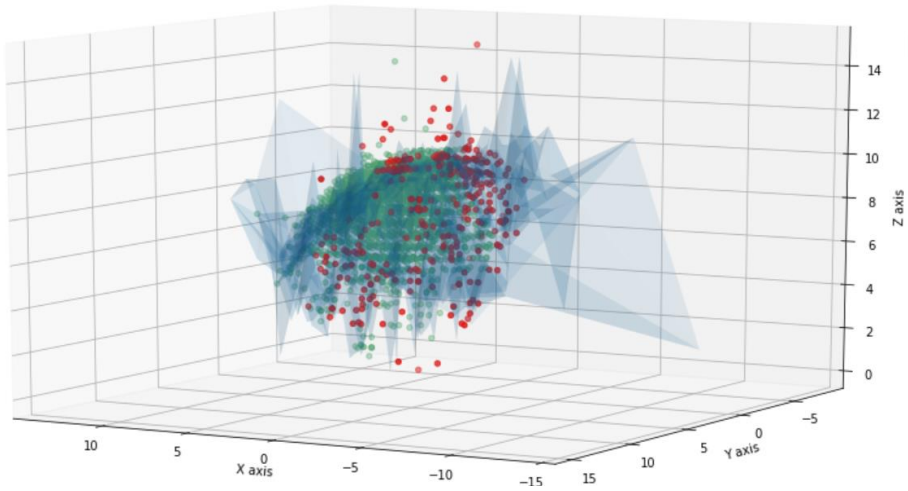

Figure 2. SVM 3D visualisation of test predictions for label 0.

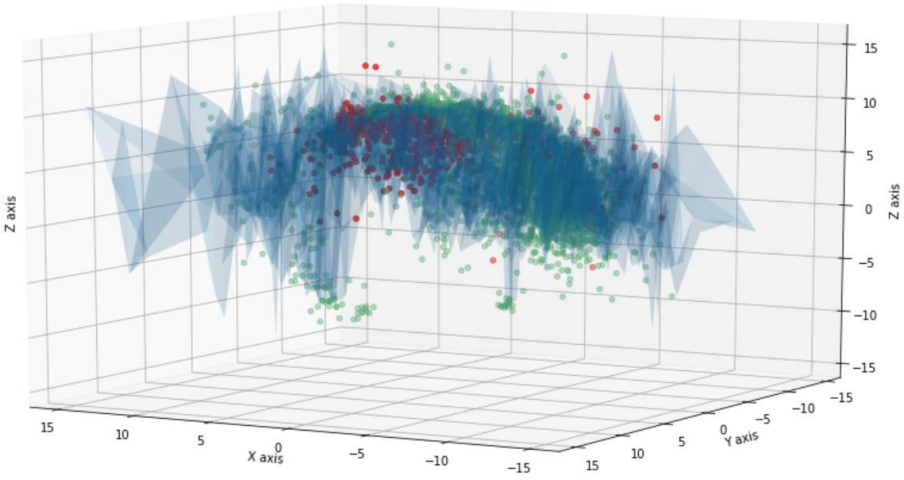

Figure 3. SVM 3D visualisation of test predictions for label 1.

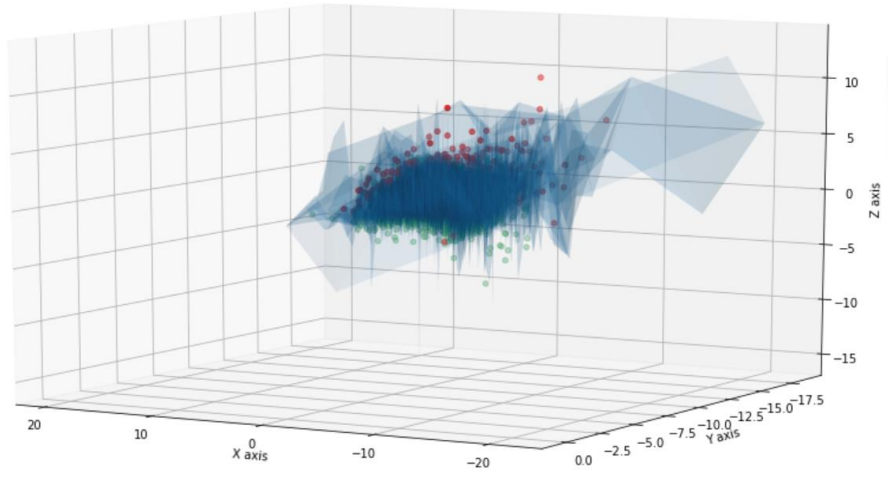

Figure 4. SVM 3D visualisation of test predictions for label 2.

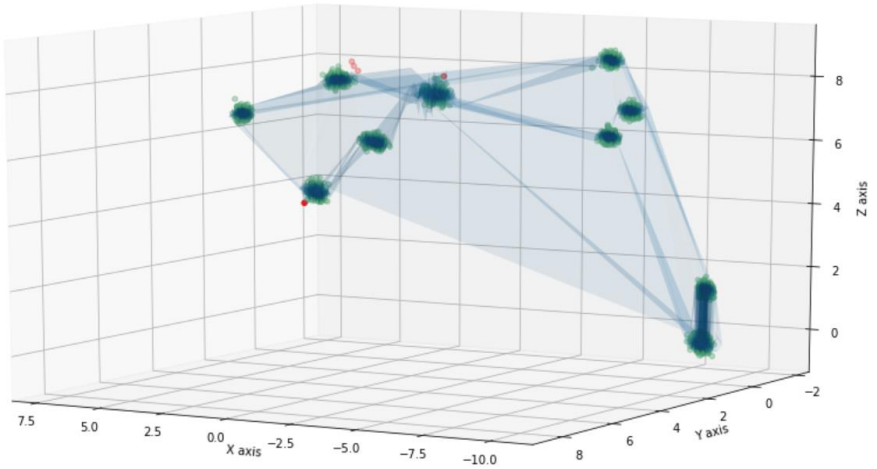

Figure 5. SVM 3D visualisation of test predictions for label 3.

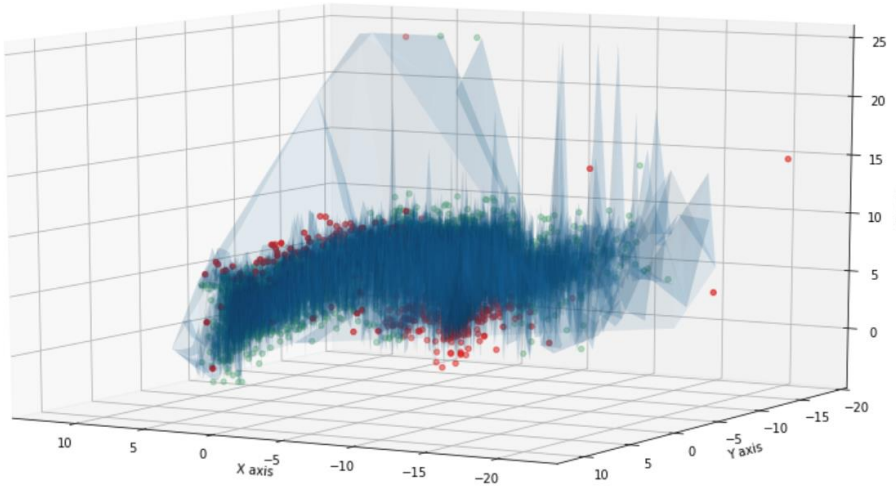

Figure 6. SVM 3D visualisation of test predictions for label 4.

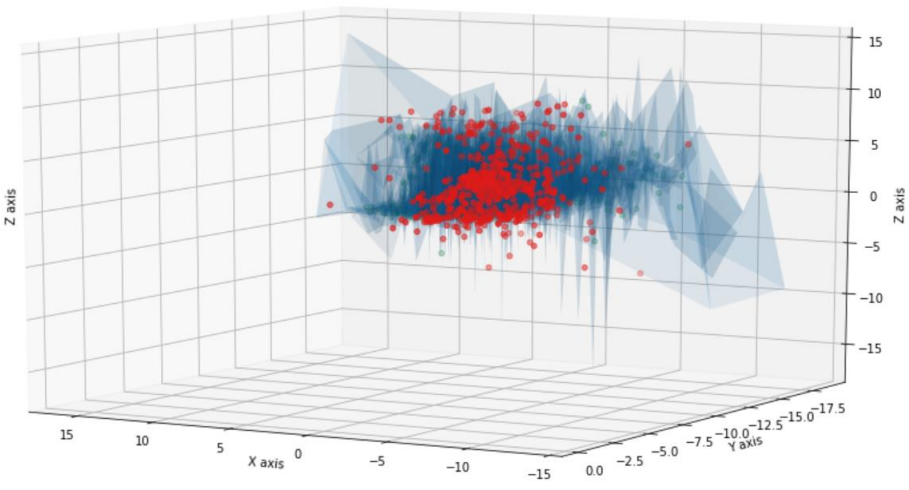

Figure 7. SVM 3D visualisation of test predictions for label 5.

6.1.2. GNB

Results for GNB with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.59				
1	0.57				
2	0.59				
3	0.47				
4	0.62				
5	0.55				
AVERAGE/TOTAL	0.57	2.73	1.65	1.00	43.4 ms ± 5 ms per loop

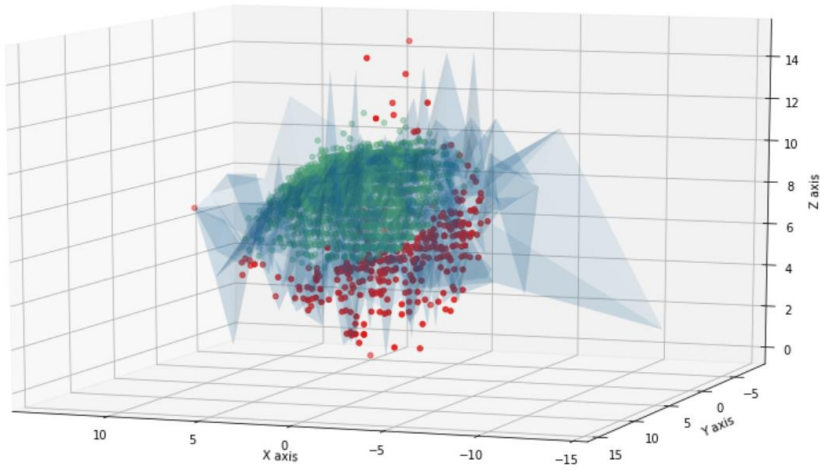

Figure 8. GNB 3D visualisation of test predictions for label 0.

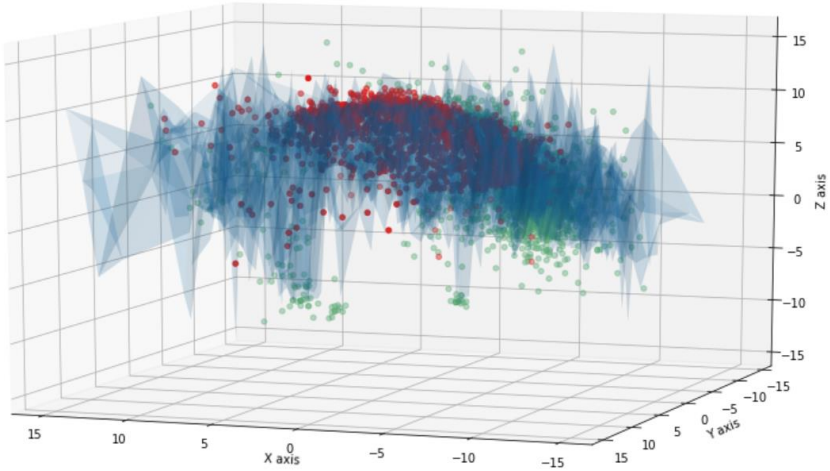

Figure 9. GNB 3D visualisation of test predictions for label 1.

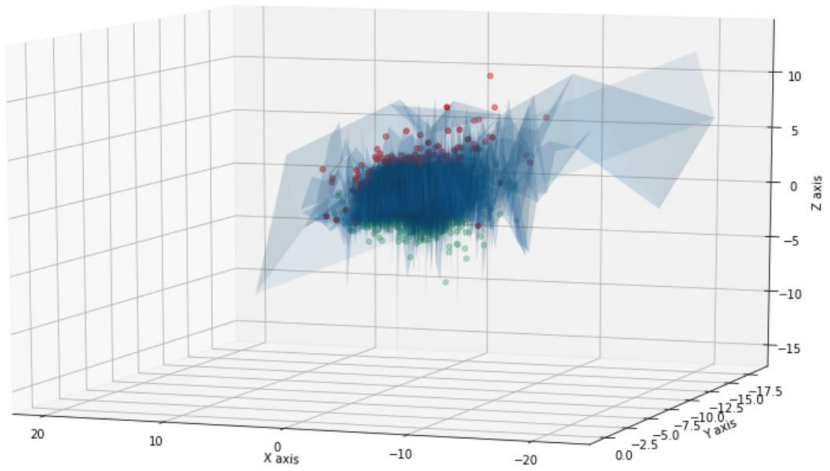

Figure 10. GNB 3D visualisation of test predictions for label 2.

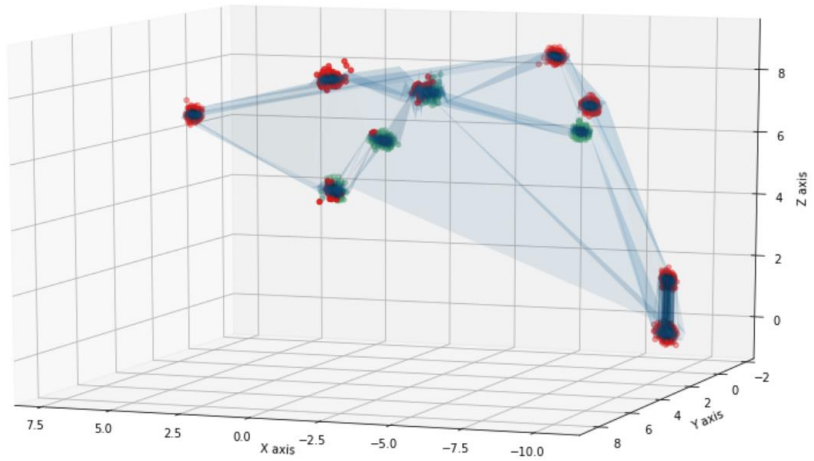

Figure 11. GNB 3D visualisation of test predictions for label 3.

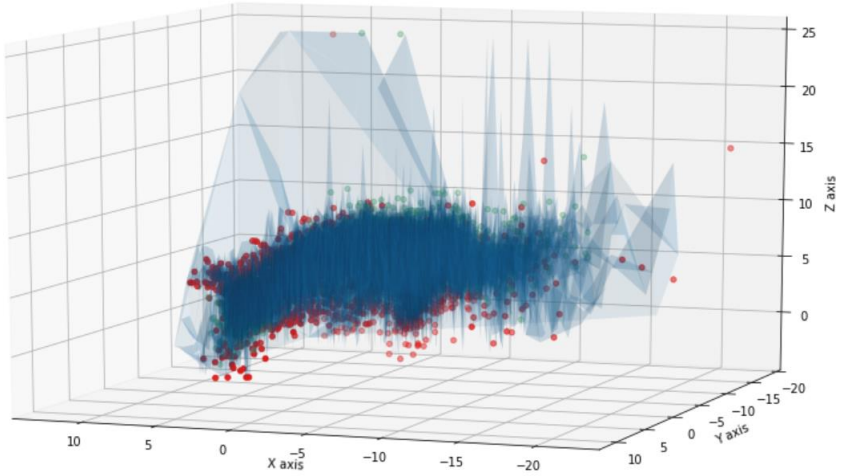

Figure 12. GNB 3D visualisation of test predictions for label 4.

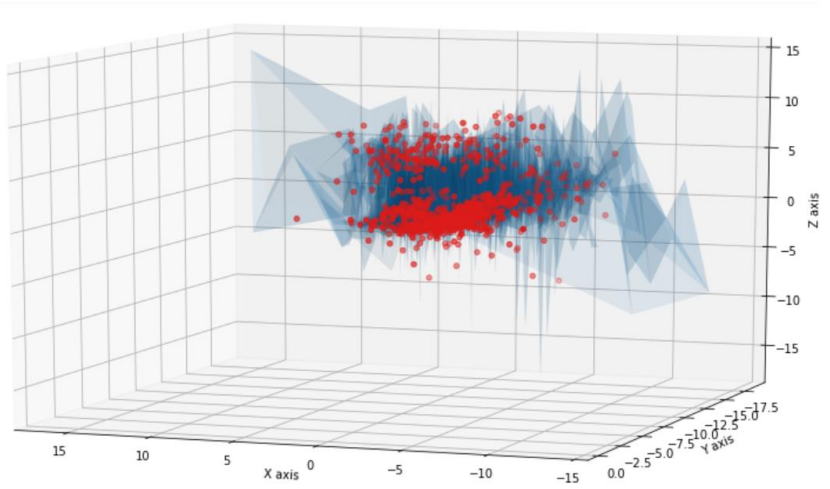

Figure 13. GNB 3D visualisation of test predictions for label 5.

Results for GNB with priors set in the parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.59				
1	0.61				
2	0.54				
3	0.48				
4	0.59				
5	0.55				
AVERAGE/TOTAL	0.57	2.72	1.65	1.01	41.7 ms ± 6 ms per loop

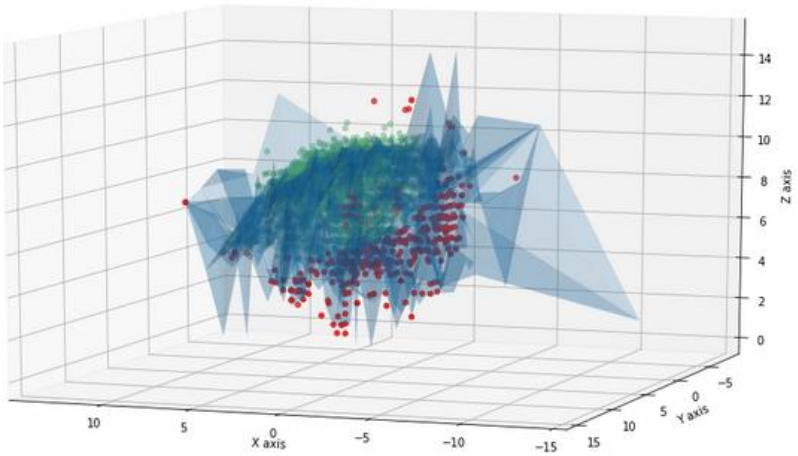

Figure 14. GNB 3D visualisation of test predictions for label 0.

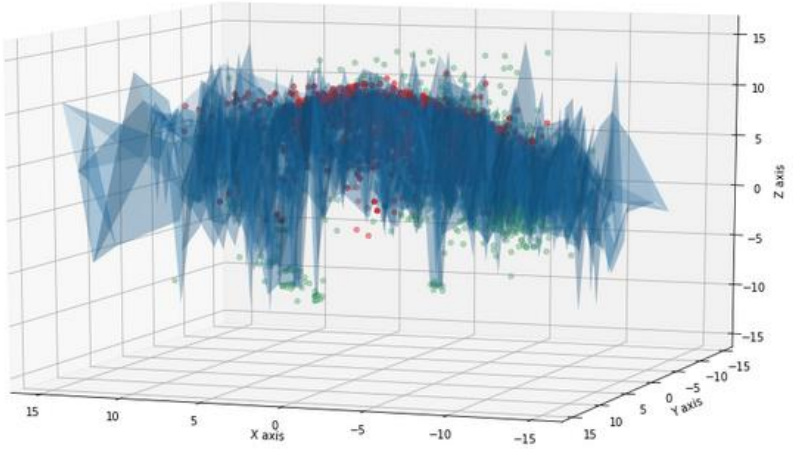

Figure 15. GNB 3D visualisation of test predictions for label 1.

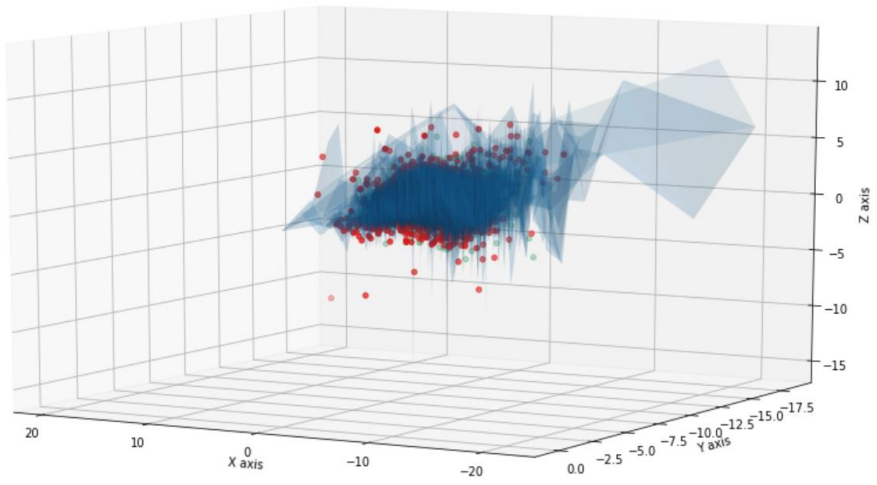

Figure 16. GNB 3D visualisation of test predictions for label 2.

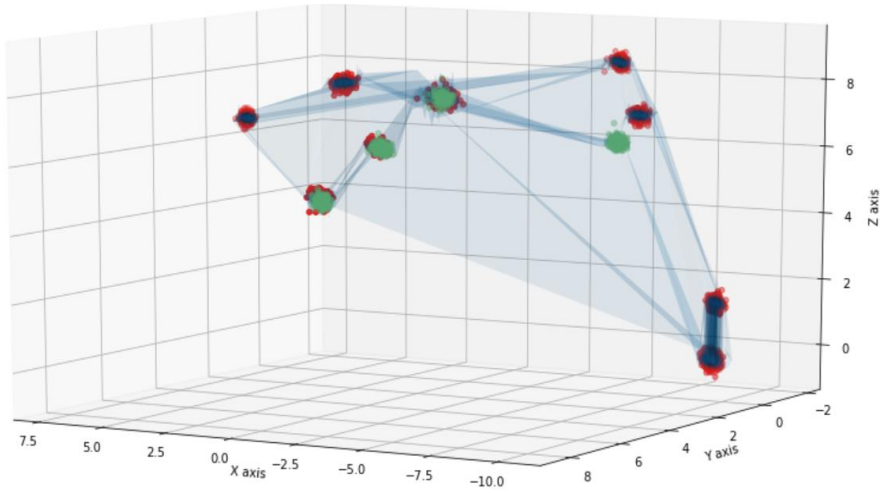

Figure 17. GNB 3D visualisation of test predictions for label 3.

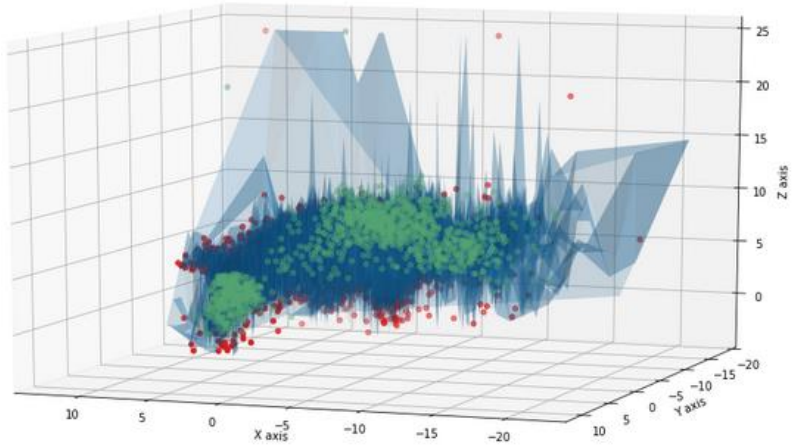

Figure 18. GNB 3D visualisation of test predictions for label 4.

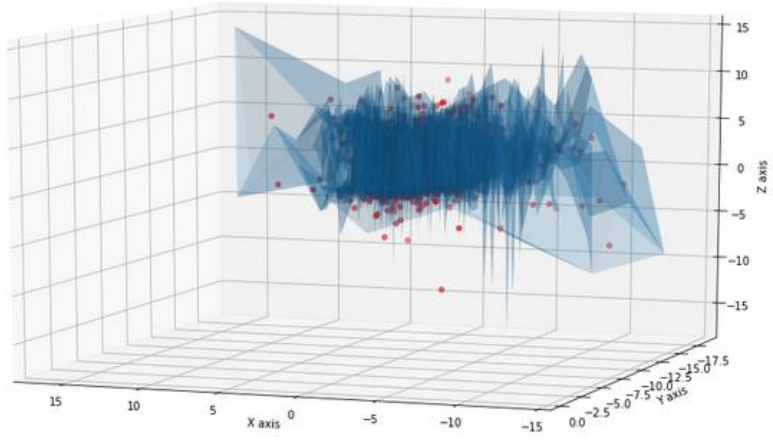

Figure 19. GNB 3D visualisation of test predictions for label 5.

6.1.3. DT

Results for DT with the first parameter set.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.83				
1	0.90				
2	0.60				
3	1.00				
4	0.84				
5	0.57				
AVERAGE/TOTAL	0.80	1.04	1.02	0.42	2.04 s ± 248 ms per loop

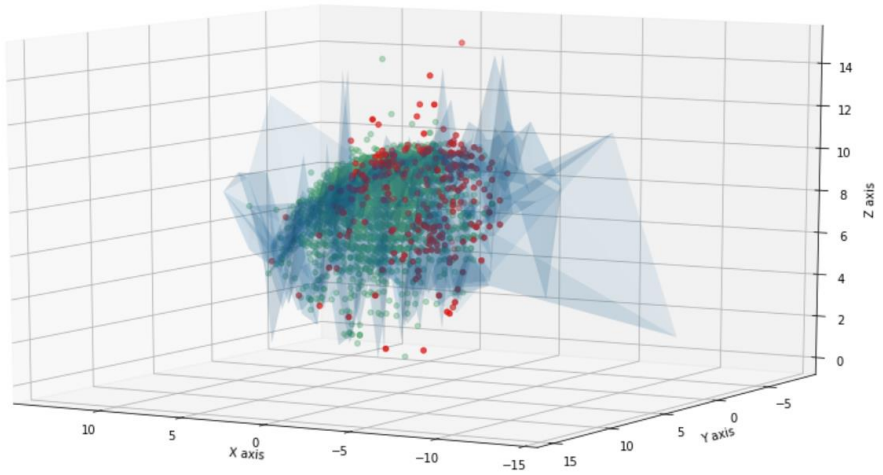

Figure 20. DT 3D visualisation of test predictions for label 0.

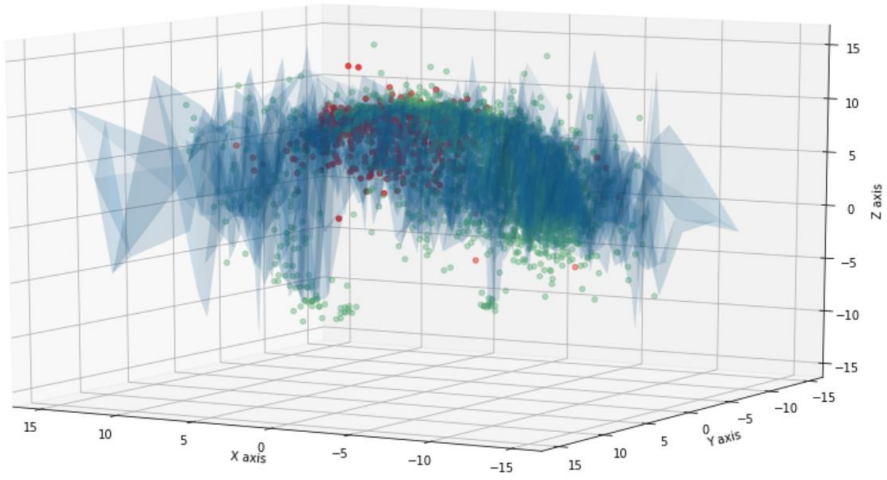

Figure 21. DT 3D visualisation of test predictions for label 1.

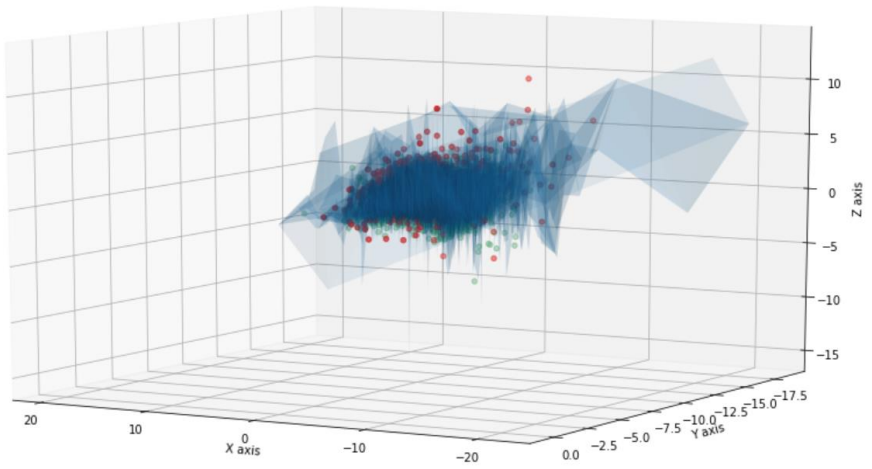

Figure 22. DT 3D visualisation of test predictions for label 2.

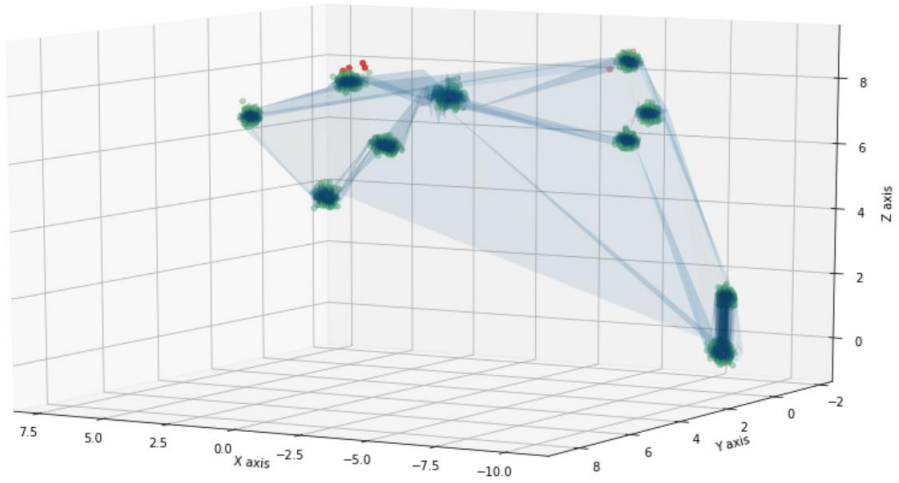

Figure 23. DT 3D visualisation of test predictions for label 3.

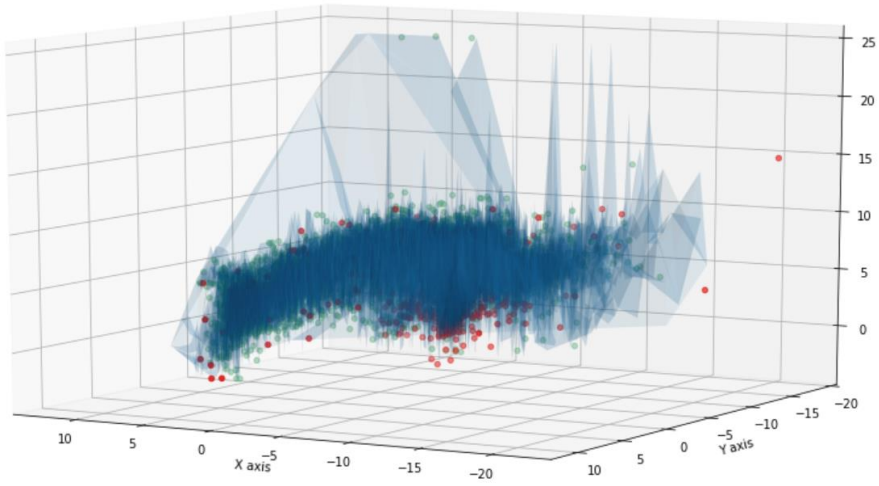

Figure 24. DT 3D visualisation of test predictions for label 4.

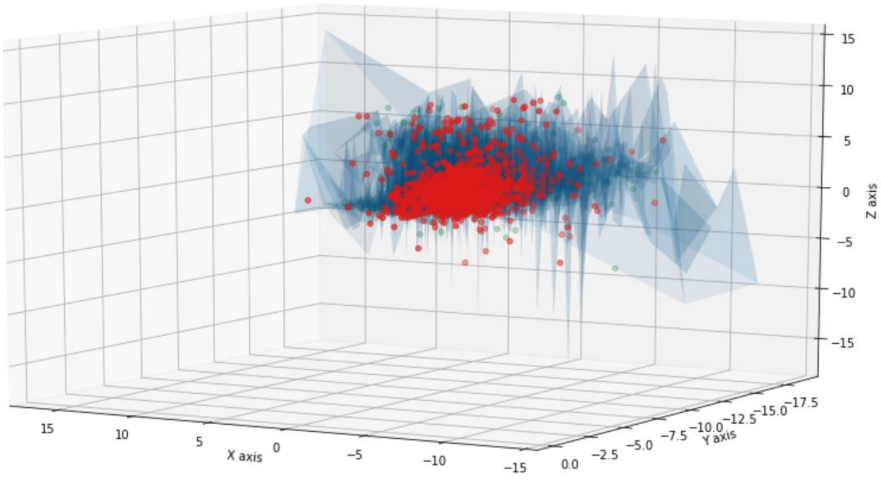

Figure 25. DT 3D visualisation of test predictions for label 5.

Results for DT with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.82				
1	0.90				
2	0.60				
3	1.00				
4	0.83				
5	0.59				
AVERAGE/TOTAL	0.81	1.03	1.01	0.41	1.71 s ± 254 ms per loop

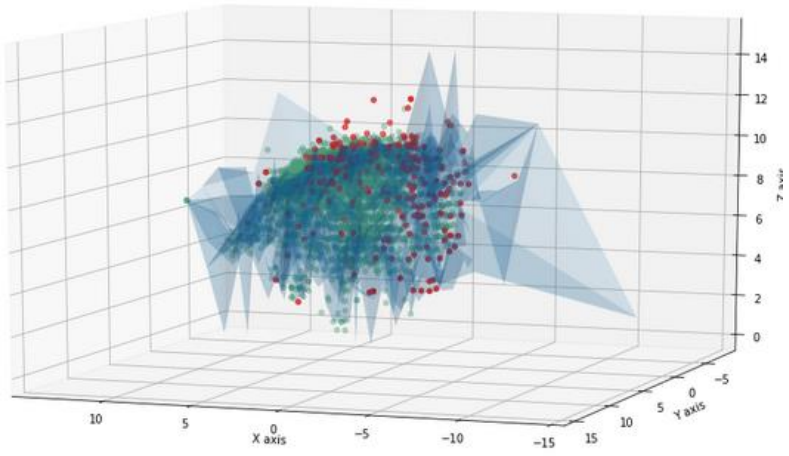

Figure 26. DT 3D visualisation of test predictions for label 0.

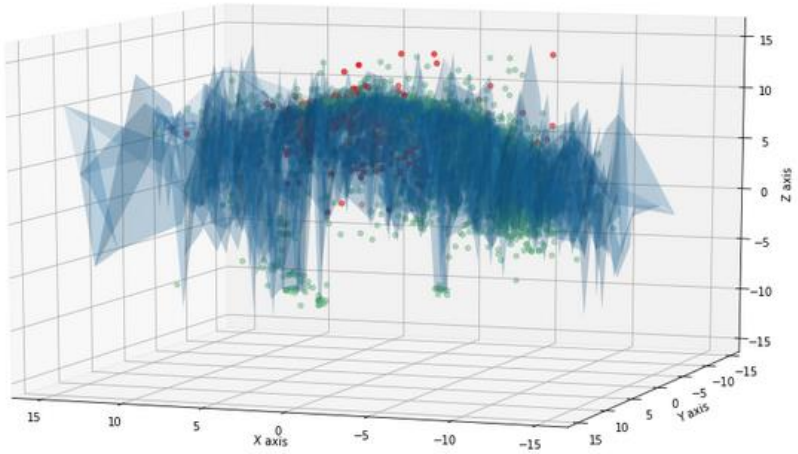

Figure 27. DT 3D visualisation of test predictions for label 1.

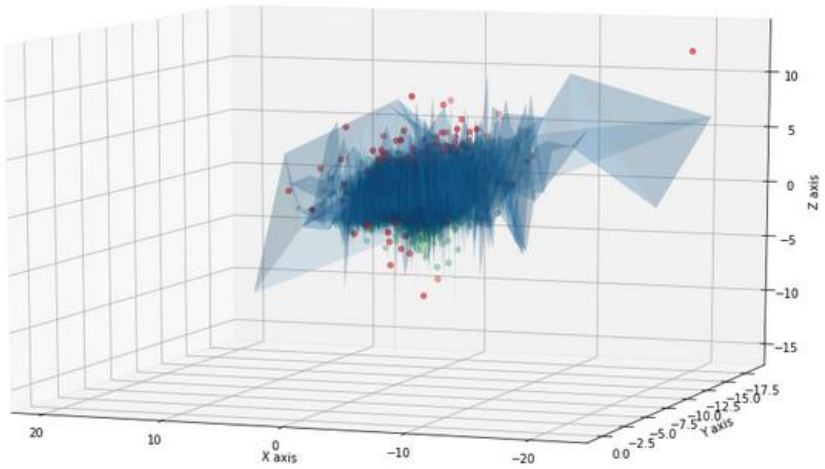

Figure 28. DT 3D visualisation of test predictions for label 2.

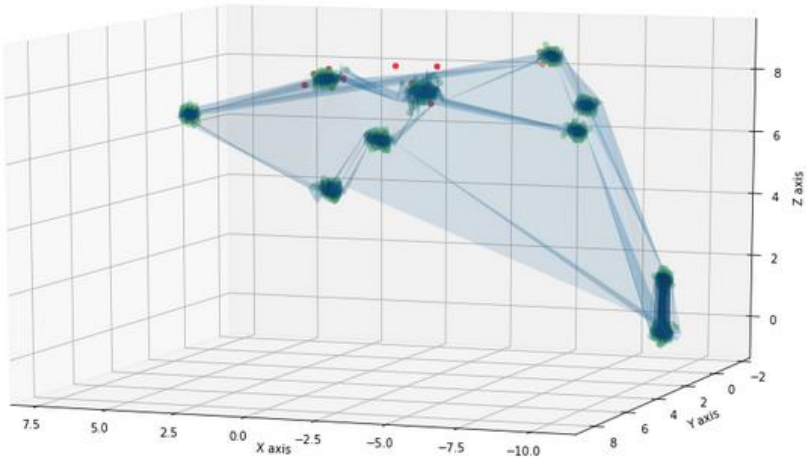

Figure 29. DT 3D visualisation of test predictions for label 3.

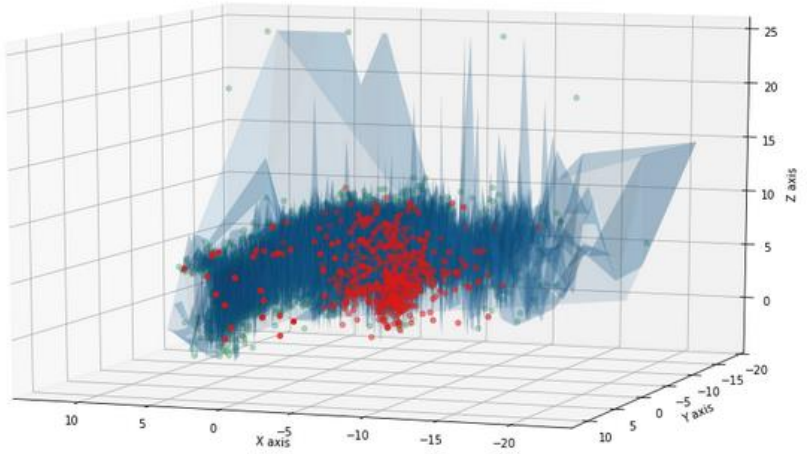

Figure 30. DT 3D visualisation of test predictions for label 4.

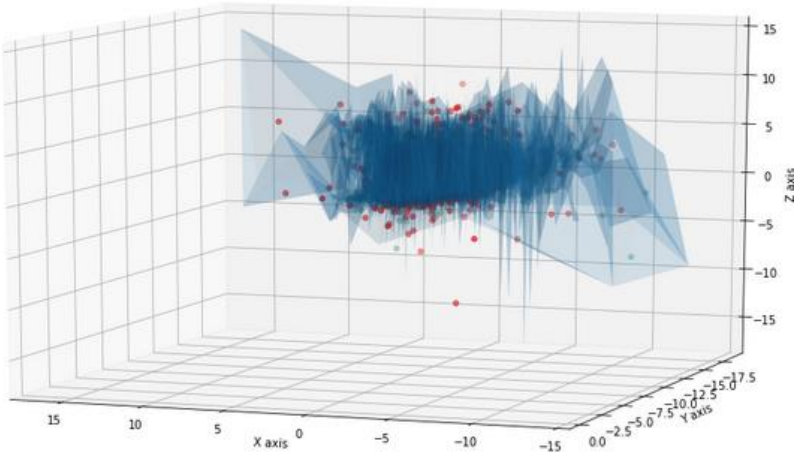

Figure 31. DT 3D visualisation of test predictions for label 5.

6.1.4. MLP

Results for MLP with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.80				
1	0.88				
2	0.60				
3	0.92				
4	0.80				
5	0.65				
AVERAGE/TOTAL	0.79	1.31	1.14	0.48	1min 23s ± 4.48 s per loop

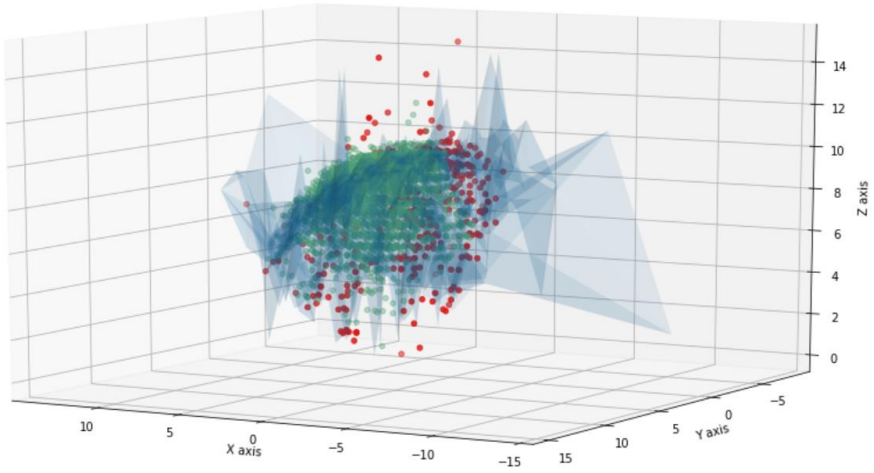

Figure 32. MLP 3D visualisation of test predictions for label 0.

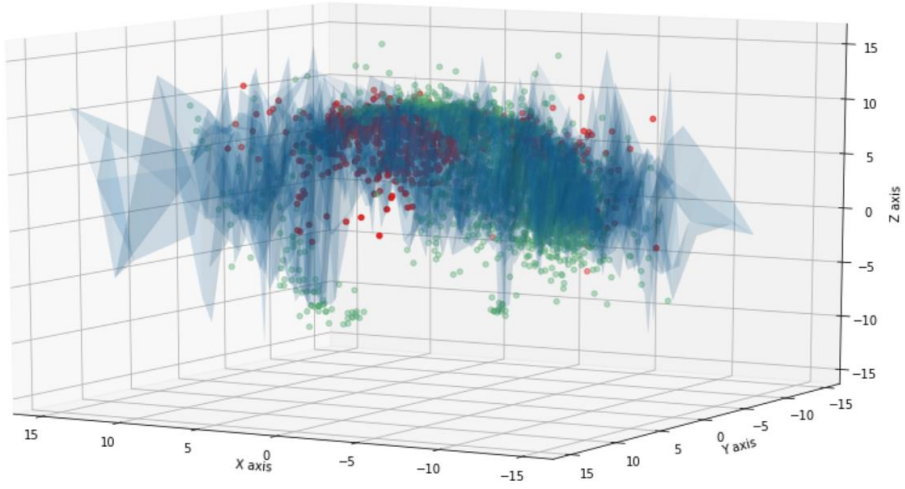

Figure 33. MLP 3D visualisation of test predictions for label 1.

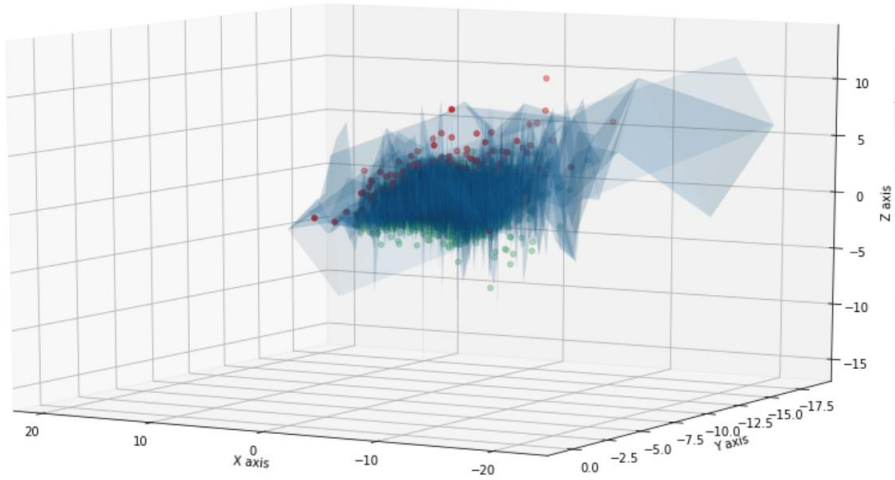

Figure 34. MLP 3D visualisation of test predictions for label 2.

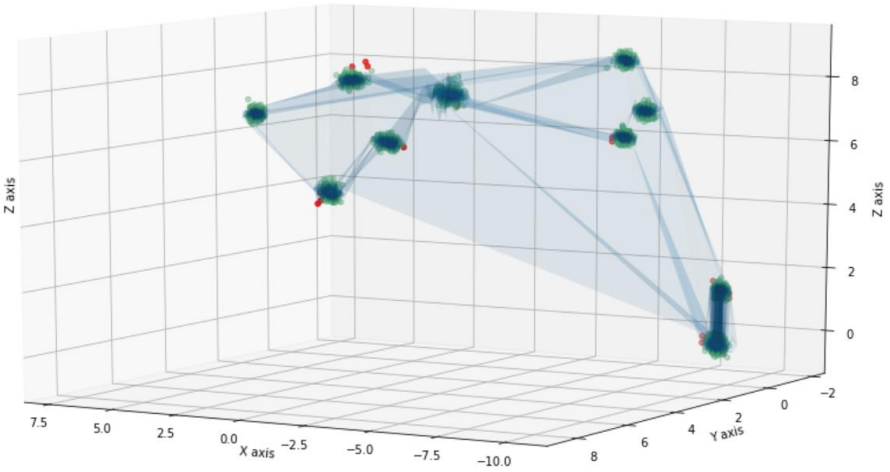

Figure 35. MLP 3D visualisation of test predictions for label 3.

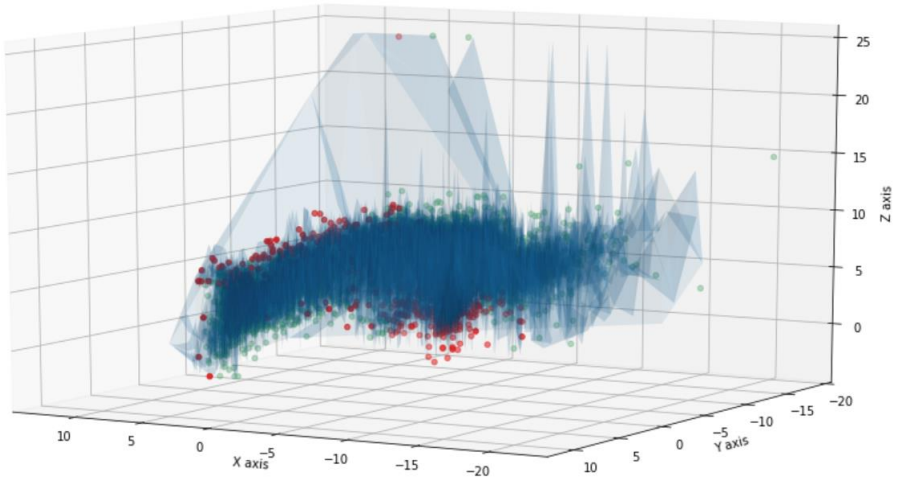

Figure 36. MLP 3D visualisation of test predictions for label 4.

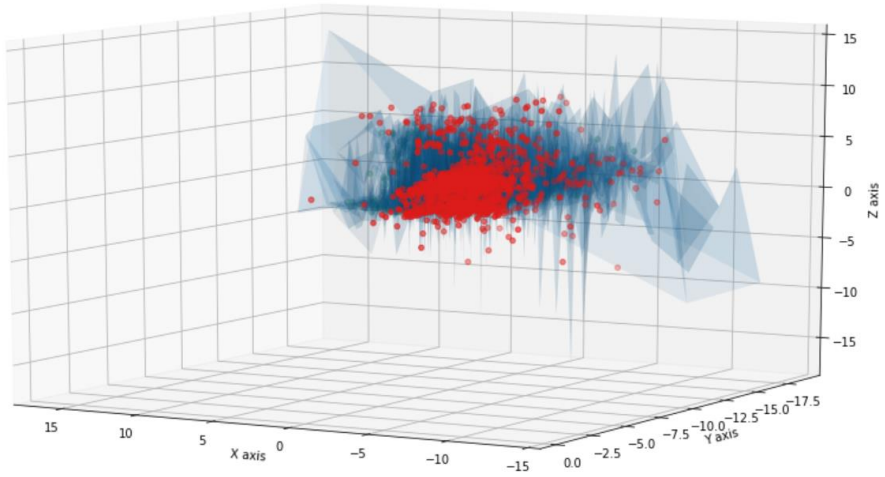

Figure 37. MLP 3D visualisation of test predictions for label 5.

Results for MLP with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.79				
1	0.87				
2	0.65				
3	0.94				
4	0.81				
5	0.62				
AVERAGE/TOTAL	0.79	1.32	1.15	0.49	2min 56s ± 13.6 s per loop

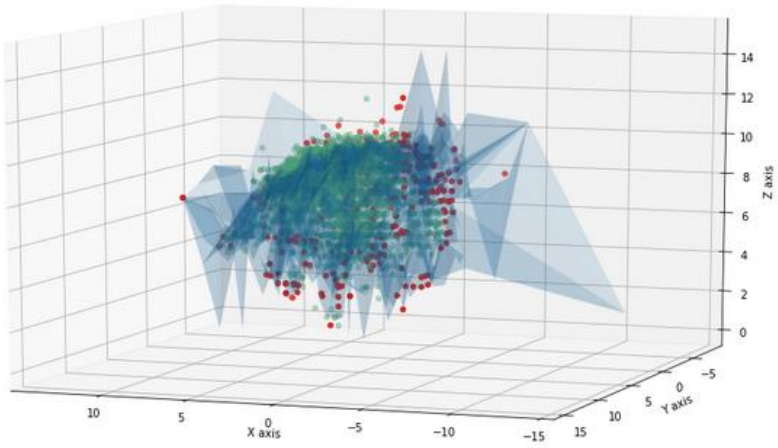

Figure 38. MLP 3D visualisation of test predictions for label 0.

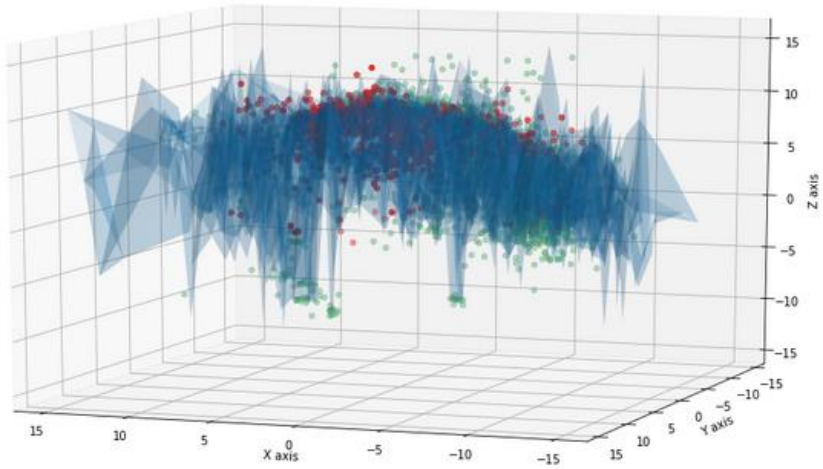

Figure 39. MLP 3D visualisation of test predictions for label 1.

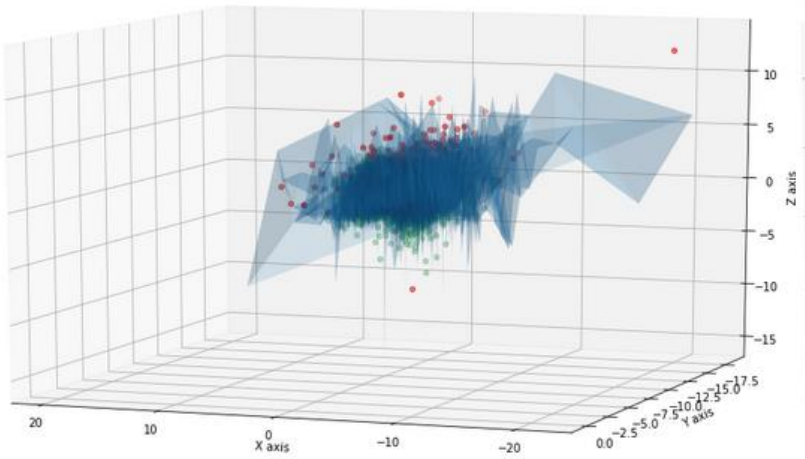

Figure 40. MLP 3D visualisation of test predictions for label 2.

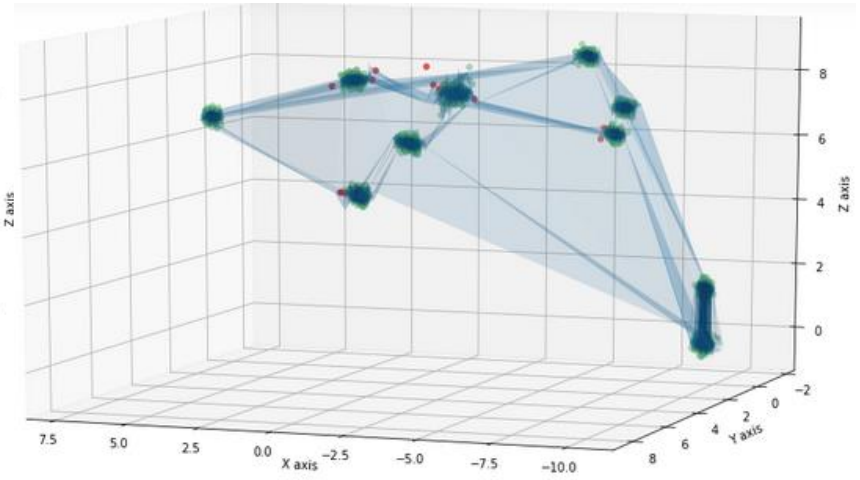

Figure 41. MLP 3D visualisation of test predictions for label 3.

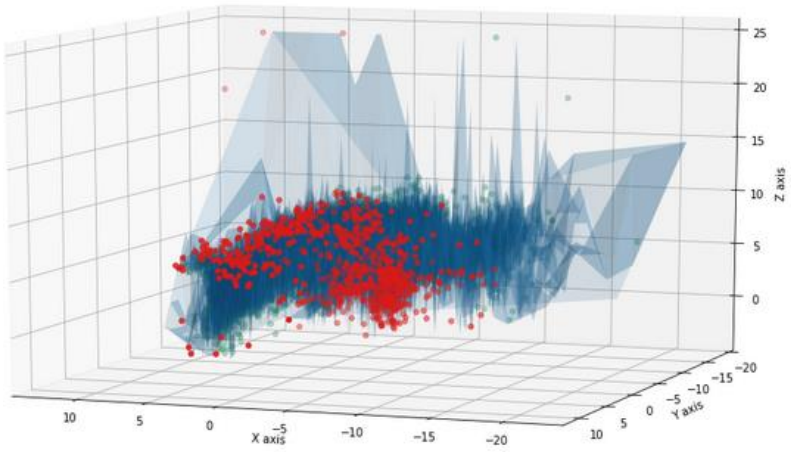

Figure 42. MLP 3D visualisation of test predictions for label 4.

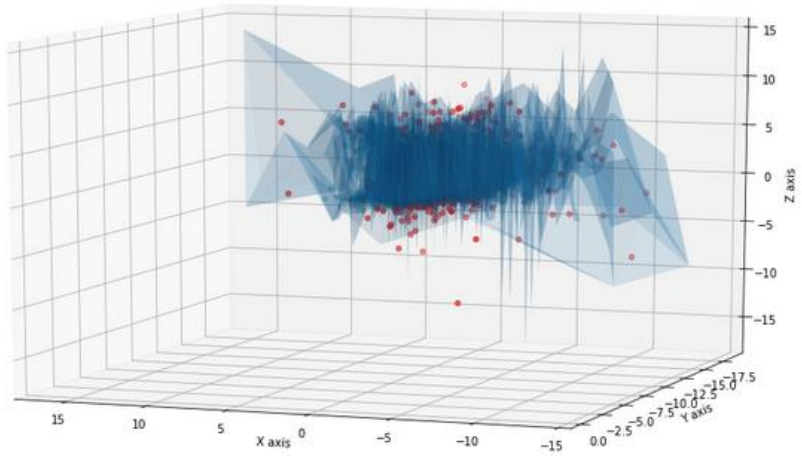

Figure 43. MLP 3D visualisation of test predictions for label 5.

6.2. Principal Components

6.2.1. SVM

Results for SVM with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.79				
1	0.89				
2	0.56				
3	0.99				
4	0.77				
5	0.64				
AVERAGE/TOTAL	0.79	1.22	1.10	0.48	10min 55s ± 2min 38s per loop

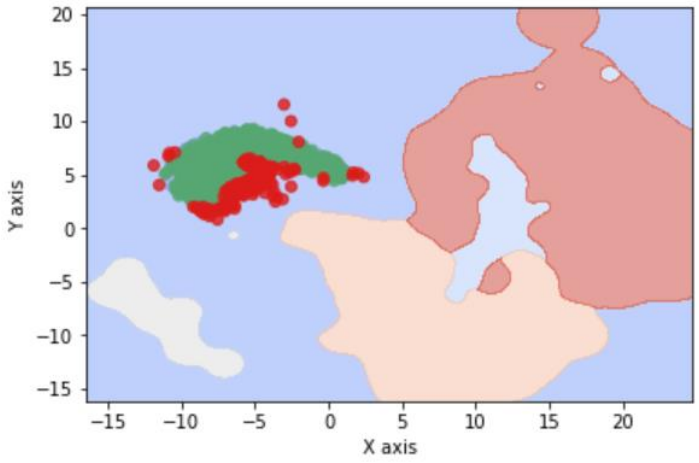

Figure 44. SVM 2D visualisation of test predictions for label 0.

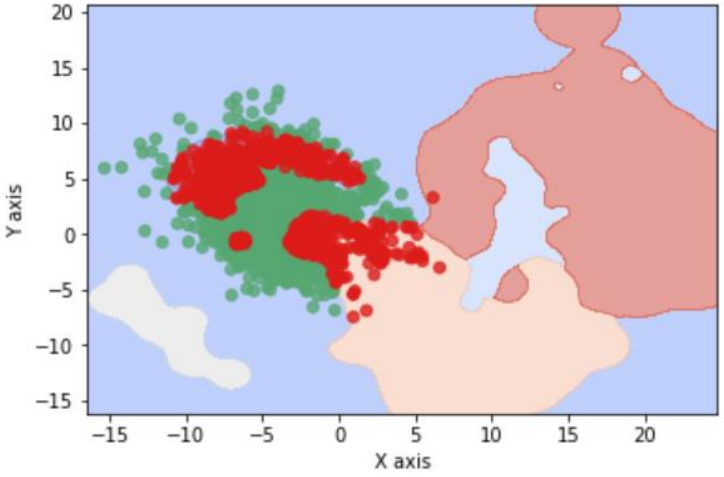

Figure 45. SVM 2D visualisation of test predictions for label 1.

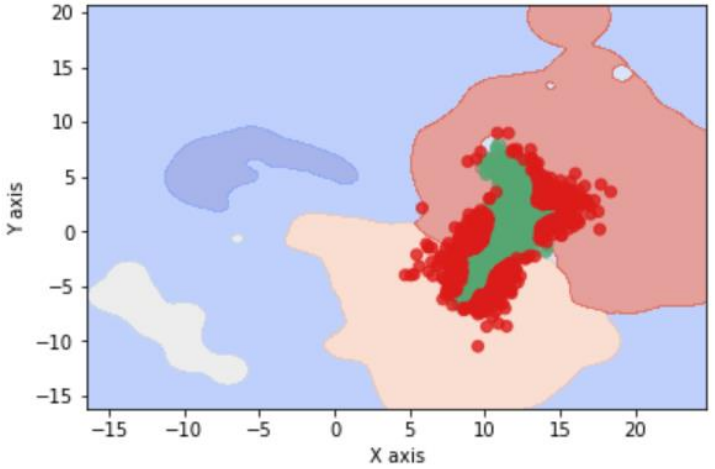

Figure 46. SVM 2D visualisation of test predictions for label 2.

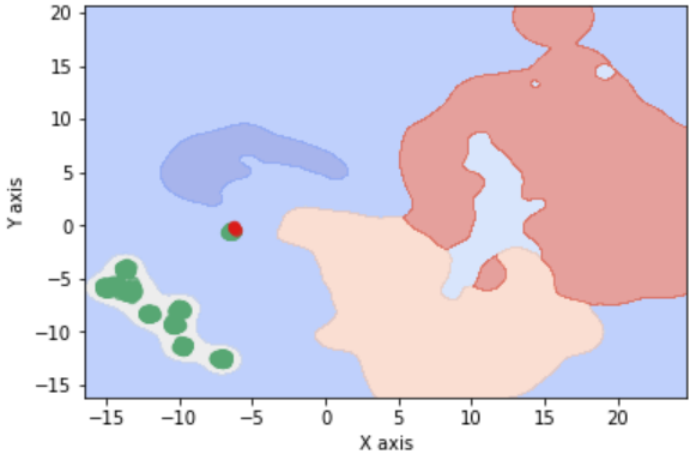

Figure 47. SVM 2D visualisation of test predictions for label 3.

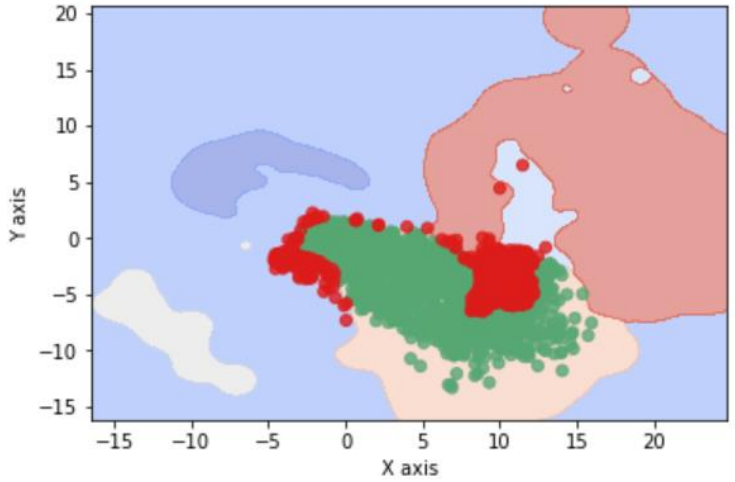

Figure 48. SVM 2D visualisation of test predictions for label 4.

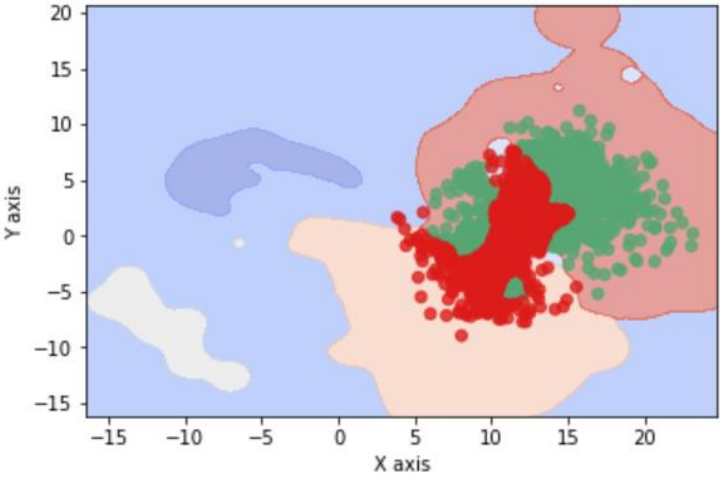

Figure 49. SVM 2D visualisation of test predictions for label 5.

6.2.2. GNB

Results for GNB with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.75				
1	0.69				
2	0.44				
3	1.00				
4	0.67				
5	0.59				
AVERAGE/TOTAL	0.70	1.80	1.34	0.70	29.2 ms ± 656 µs per loop

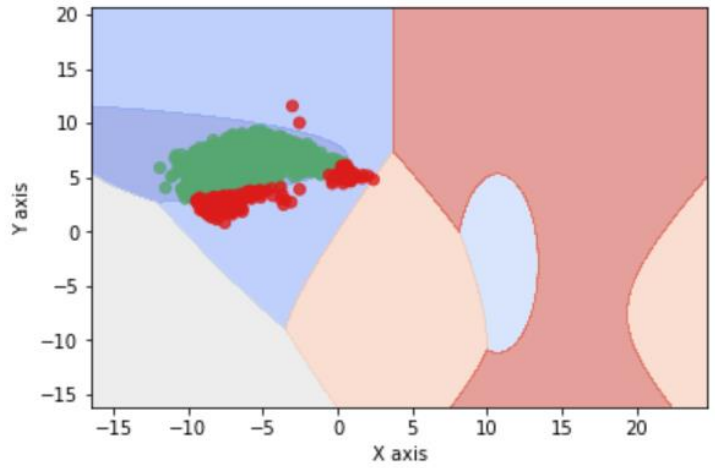

Figure 50. GNB 2D visualisation of test predictions for label 0.

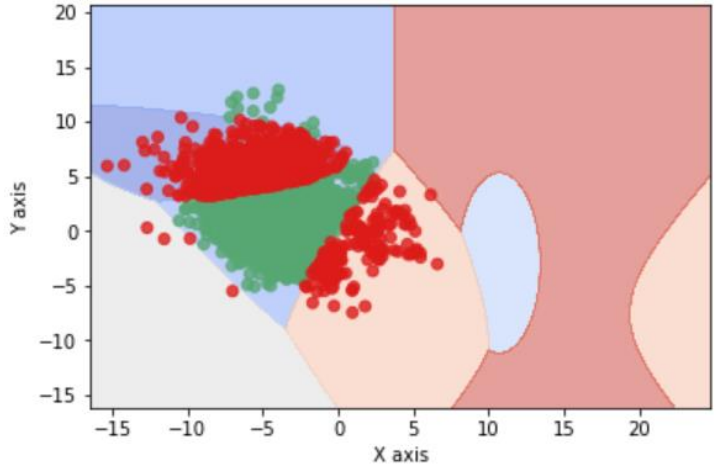

Figure 51. GNB 2D visualisation of test predictions for label 1.

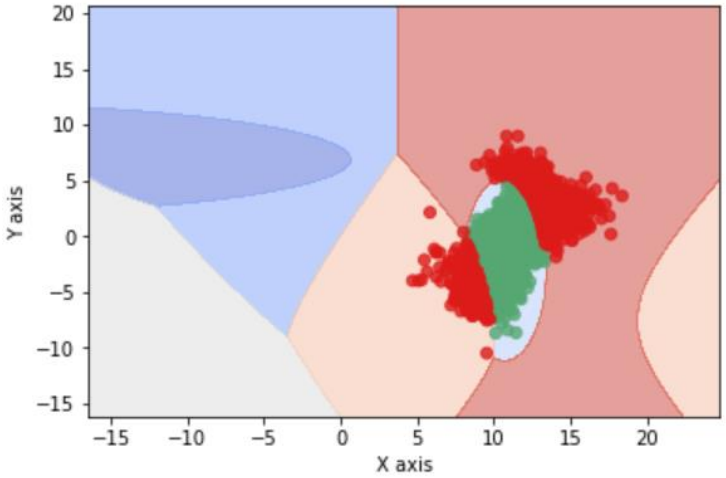

Figure 52. GNB 2D visualisation of test predictions for label 2.

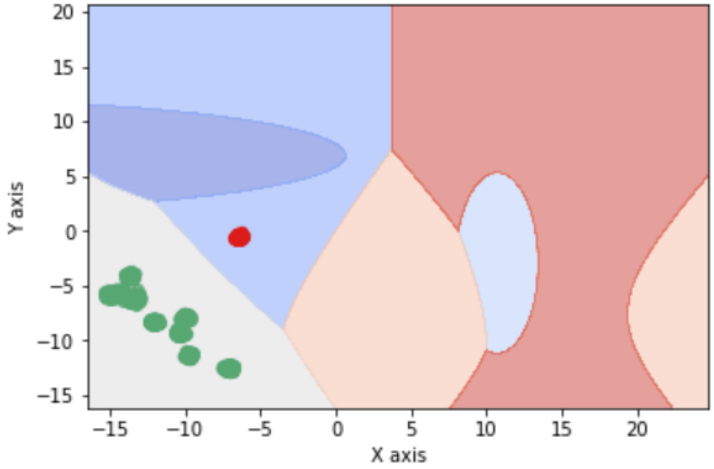

Figure 53. GNB 2D visualisation of test predictions for label 3.

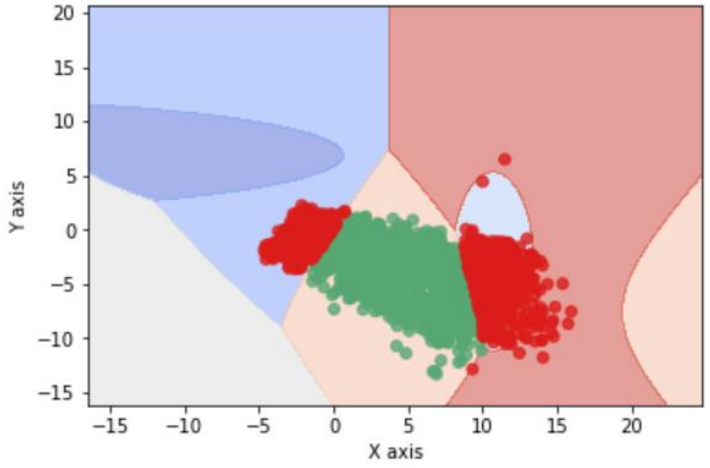

Figure 54. GNB 2D visualisation of test predictions for label 4.

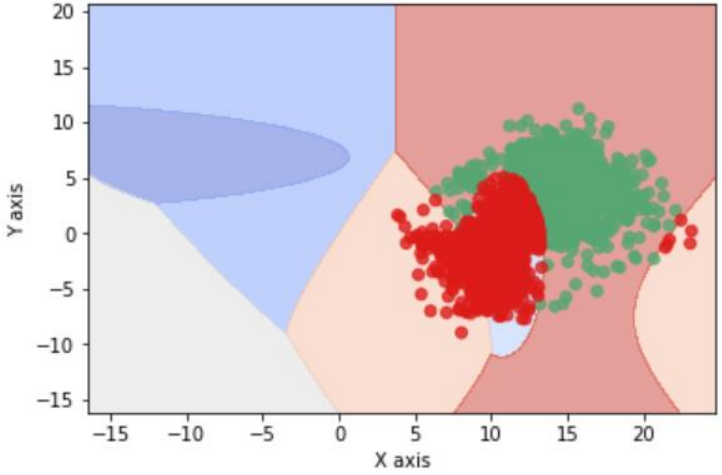

Figure 55. GNB 2D visualisation of test predictions for label 5.

Results for GNB with priors set in the parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.75				
1	0.70				
2	0.45				
3	1.00				
4	0.67				
5	0.61				
AVERAGE/TOTAL	0.70	1.80	1.34	0.71	27.8 ms ± 743 µs per loop

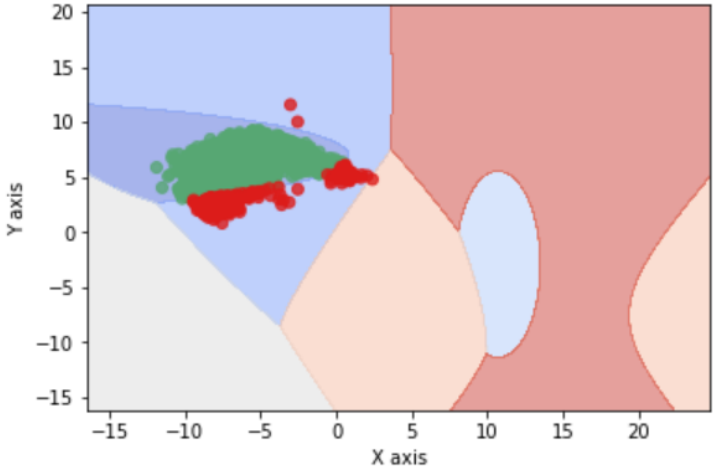

Figure 56. GNB 2D visualisation of test predictions for label 0.

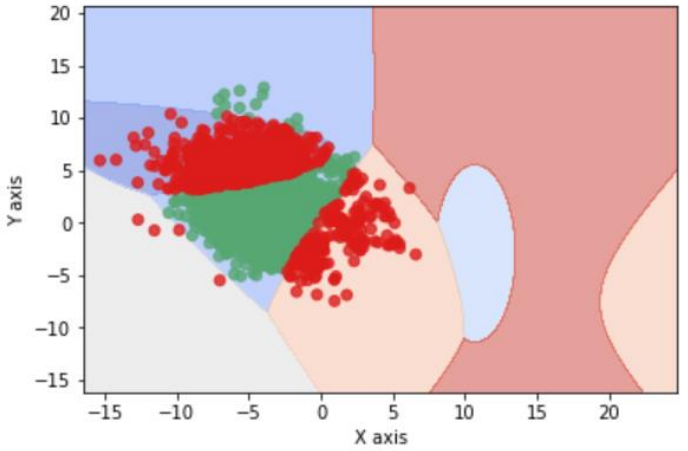

Figure 57. GNB 2D visualisation of test predictions for label 1.

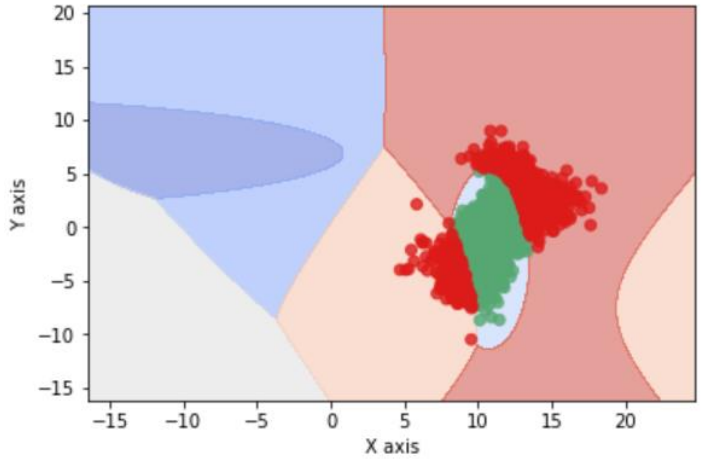

Figure 58. GNB 2D visualisation of test predictions for label 2.

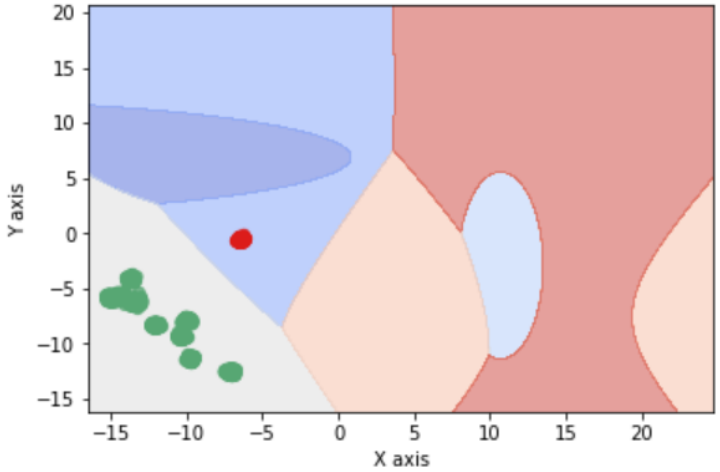

Figure 59. GNB 2D visualisation of test predictions for label 3.

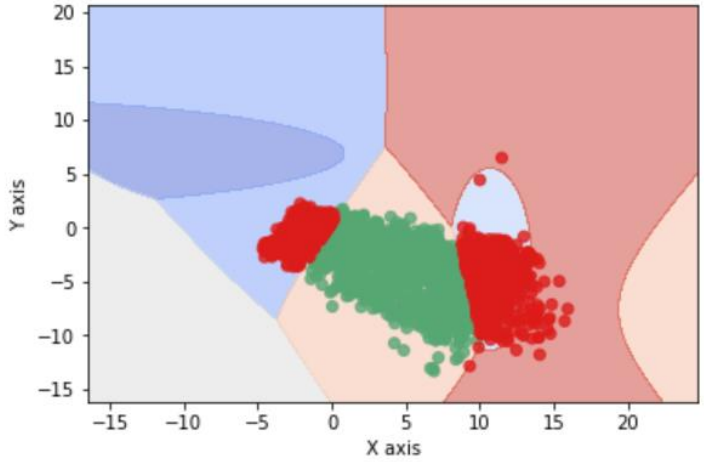

Figure 60. GNB 2D visualisation of test predictions for label 4.

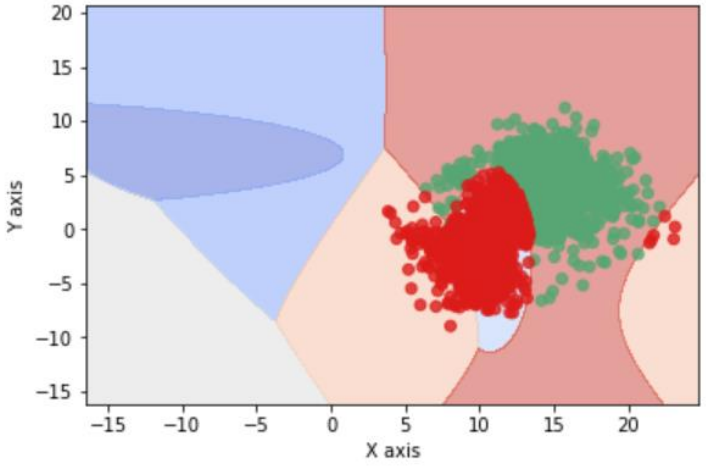

Figure 61. GNB 2D visualisation of test predictions for label 5.

6.2.3. DT

Results for DT with the first parameter set.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.82				
1	0.86				
2	0.50				
3	1.00				
4	0.72				
5	0.53				
AVERAGE/TOTAL	0.75	1.31	1.14	0.52	1.33 s ± 96.2 ms per loop

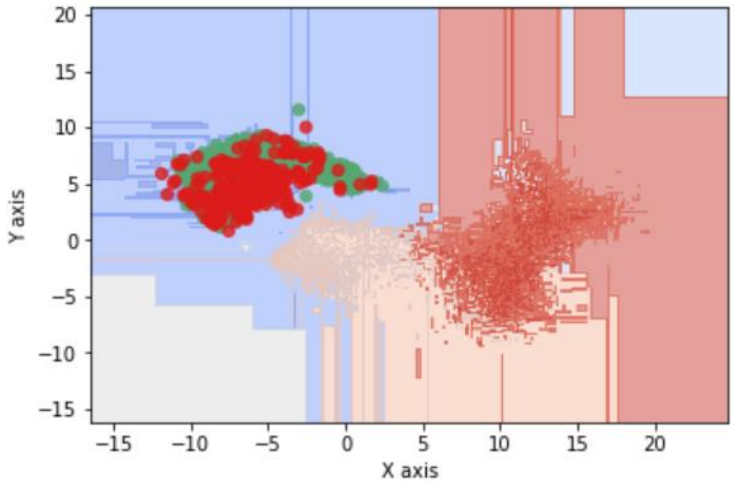

Figure 62. DT 2D visualisation of test predictions for label 0.

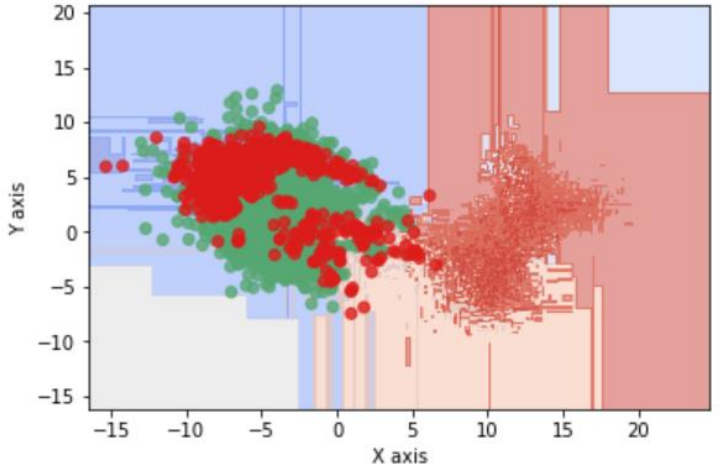

Figure 5763 DT 2D visualisation of test predictions for label 1.

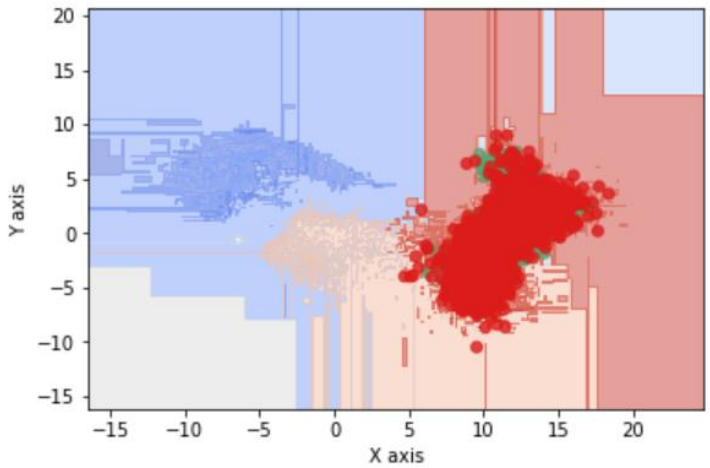

Figure 64. DT 2D visualisation of test predictions for label 2.

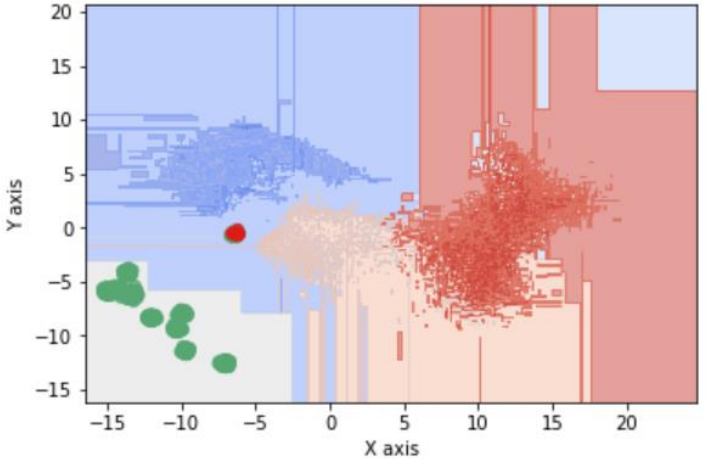

Figure 65. DT 2D visualisation of test predictions for label 3.

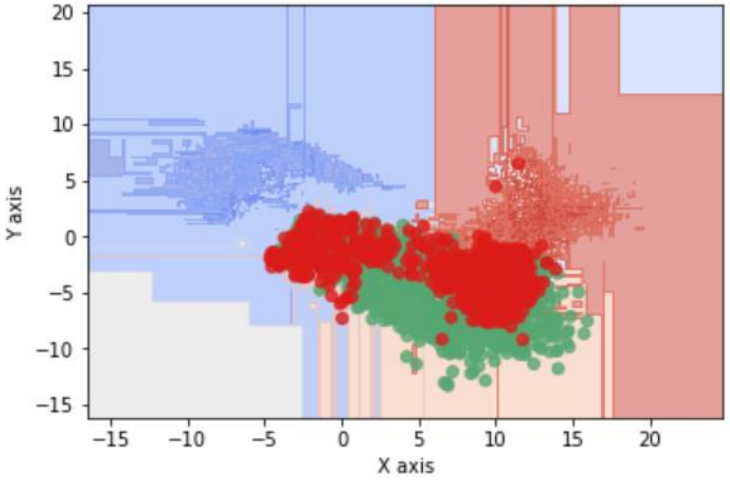

Figure 66. DT 2D visualisation of test predictions for label 4.

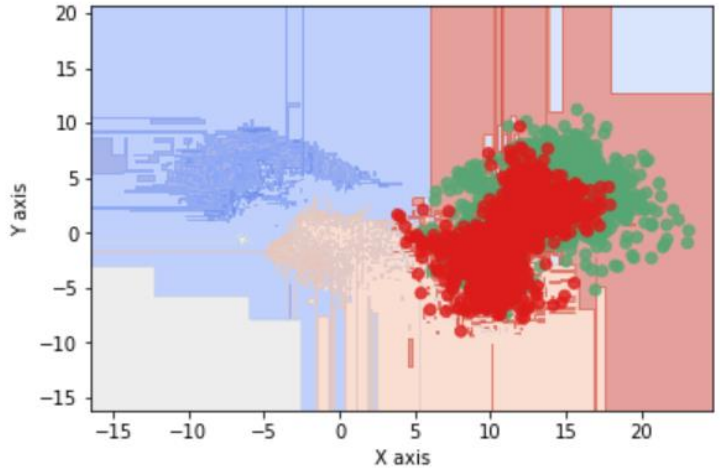

Figure 67. DT 2D visualisation of test predictions for label 5.

Results for DT with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.82				
1	0.86				
2	0.51				
3	1.00				
4	0.73				
5	0.52				
AVERAGE/TOTAL	0.75	1.33	1.15	0.52	1.51 s ± 92.5 ms per loop

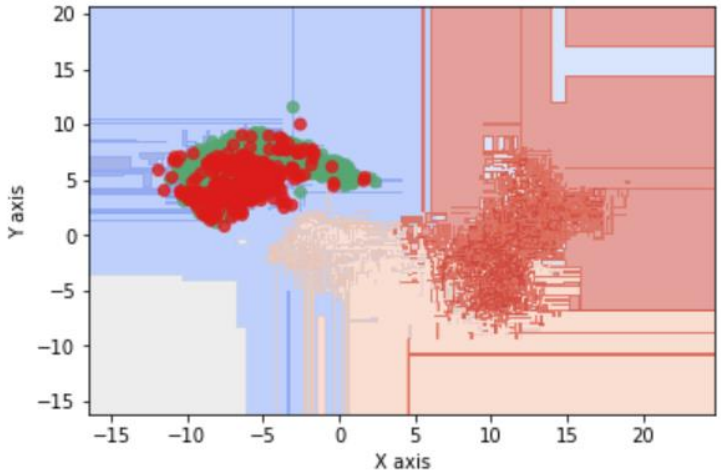

Figure 68. DT 2D visualisation of test predictions for label 0.

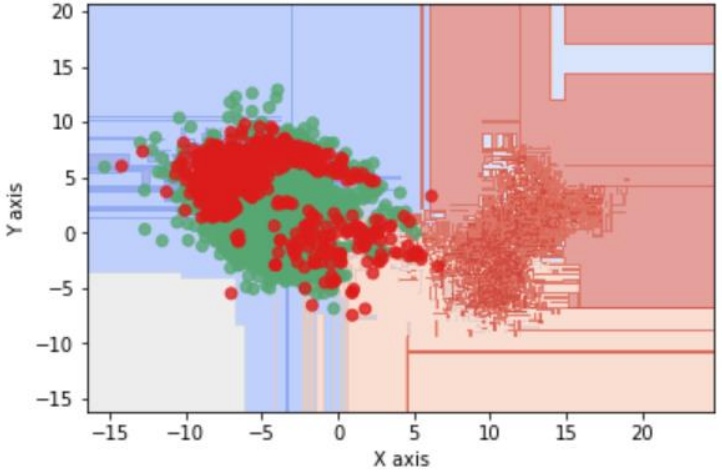

Figure 69. DT 2D visualisation of test predictions for label 1.

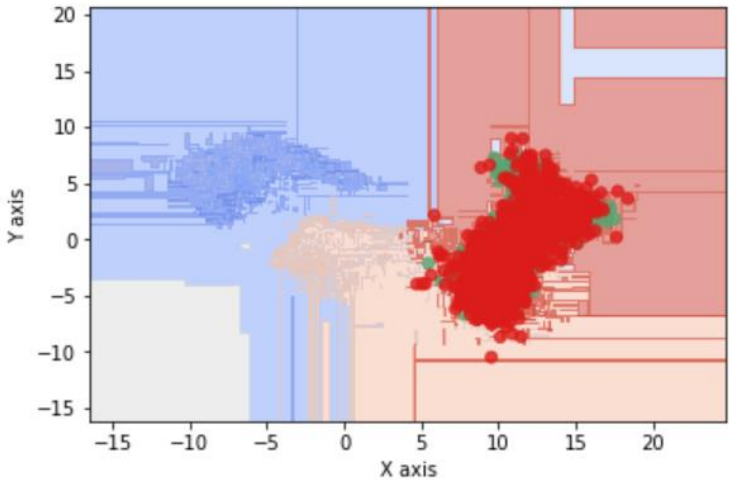

Figure 70. DT 2D visualisation of test predictions for label 2.

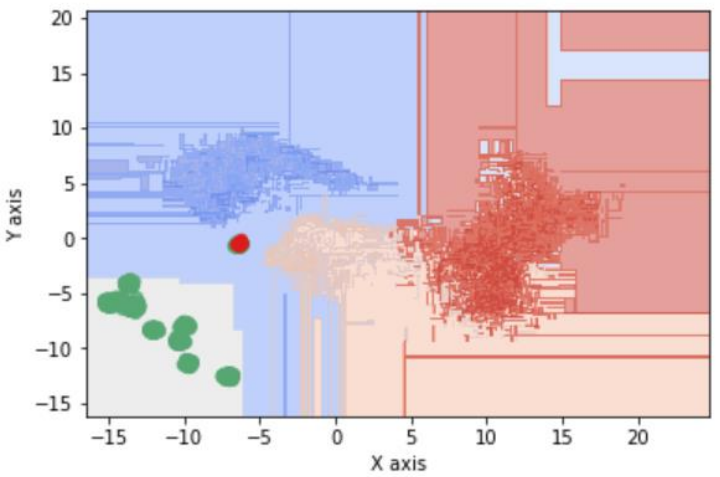

Figure 71. DT 2D visualisation of test predictions for label 3.

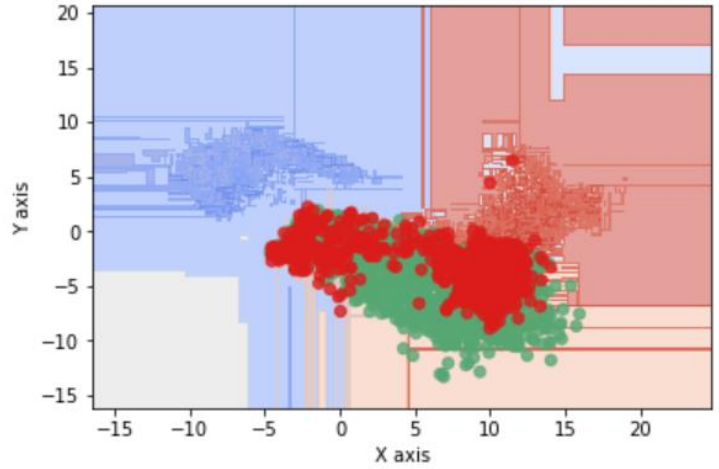

Figure 72. DT 2D visualisation of test predictions for label 4.

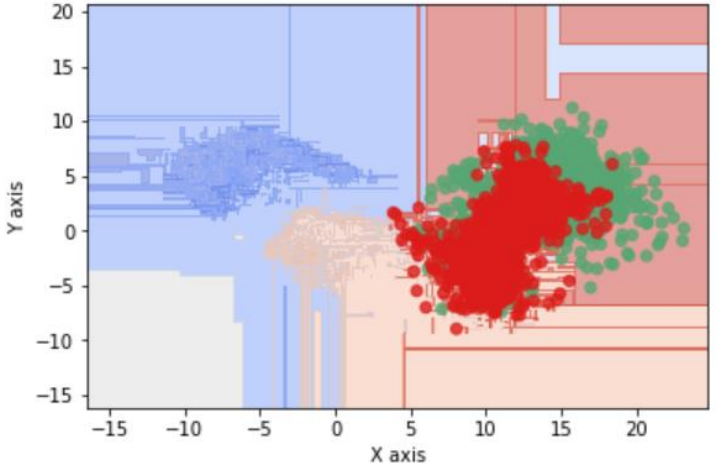

Figure 73. DT 2D visualisation of test predictions for label 5.

6.2.4. MLP

Results for MLP with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.78				
1	0.87				
2	0.57				
3	0.99				
4	0.77				
5	0.57				
AVERAGE/TOTAL	0.77	1.25	1.12	0.50	32.3 s ± 11.4 s per loop

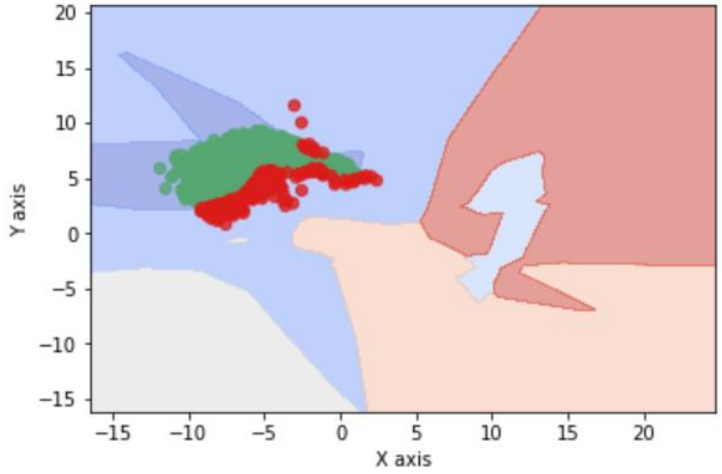

Figure 74. MLP 2D visualisation of test predictions for label 0.

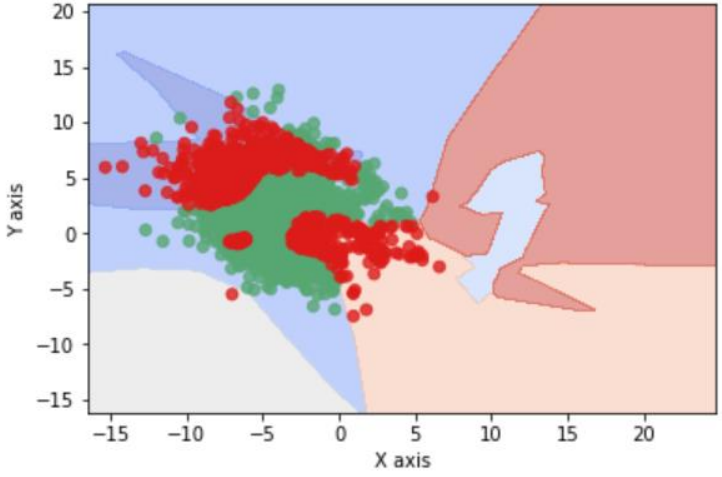

Figure 75. MLP 2D visualisation of test predictions for label 1.

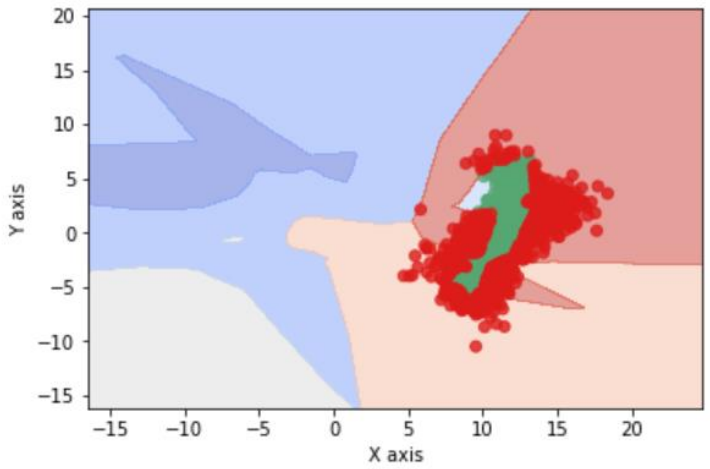

Figure 76. MLP 2D visualisation of test predictions for label 2.

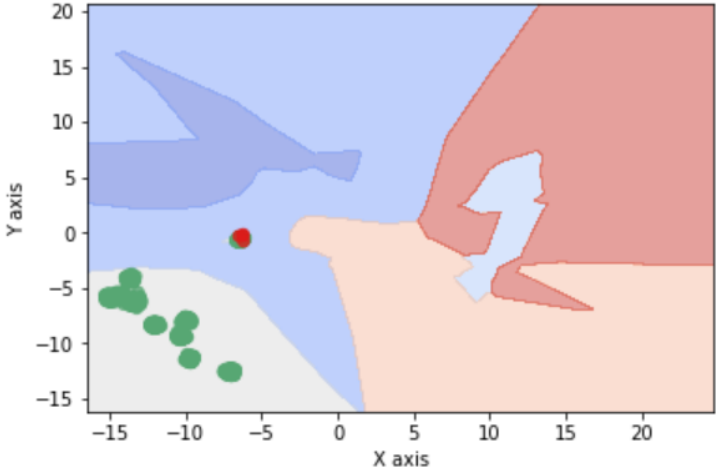

Figure 77. MLP 2D visualisation of test predictions for label 3.

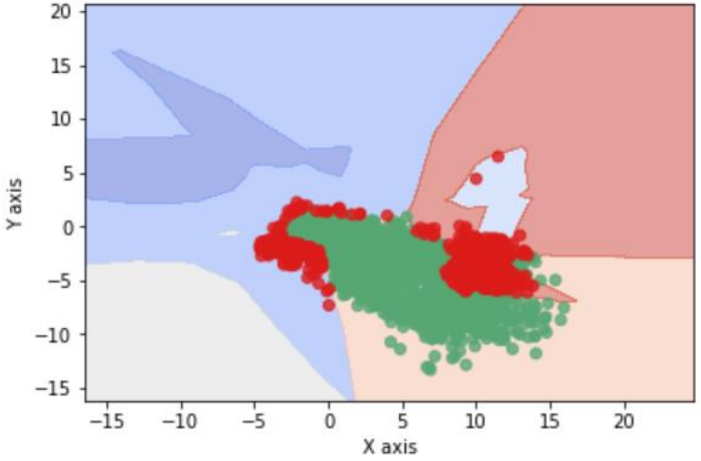

Figure 78. MLP 2D visualisation of test predictions for label 4.

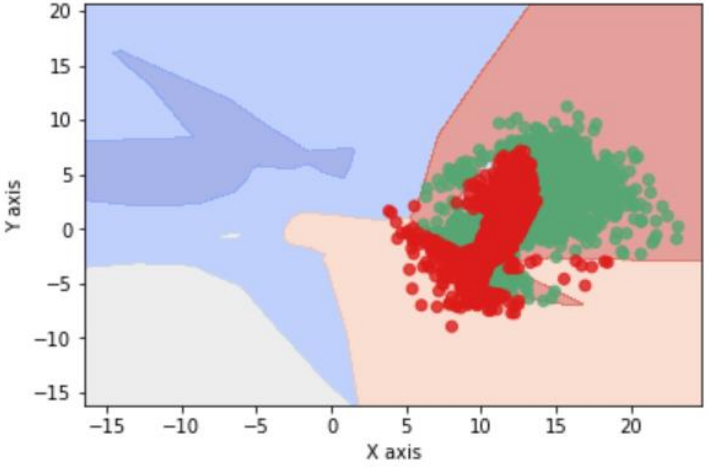

Figure 79. MLP 2D visualisation of test predictions for label 5.

Results for MLP with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.78				
1	0.87				
2	0.57				
3	0.99				
4	0.77				
5	0.57				
AVERAGE/TOTAL	0.77	1.25	1.12	0.50	29.7 s ± 12.2 s per loop

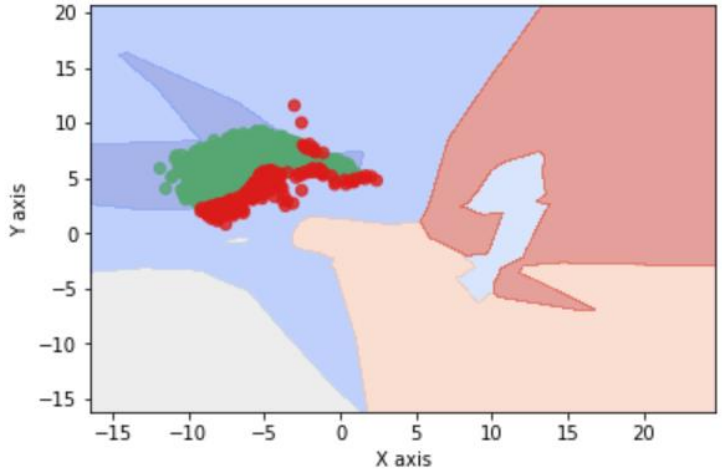

Figure 80. MLP 2D visualisation of test predictions for label 0.

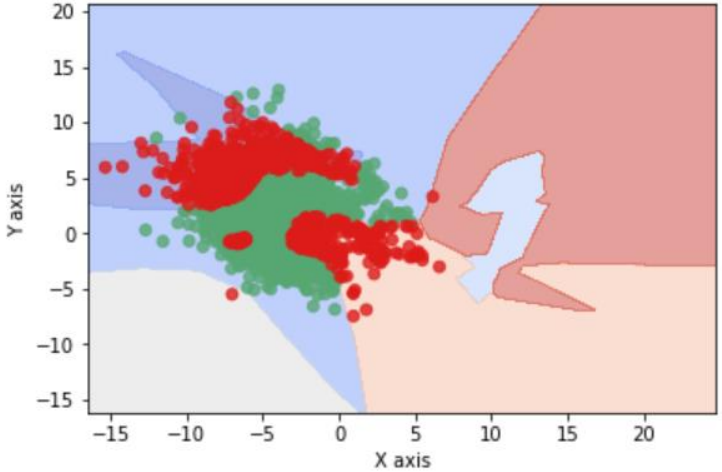

Figure 81. MLP 2D visualisation of test predictions for label 1.

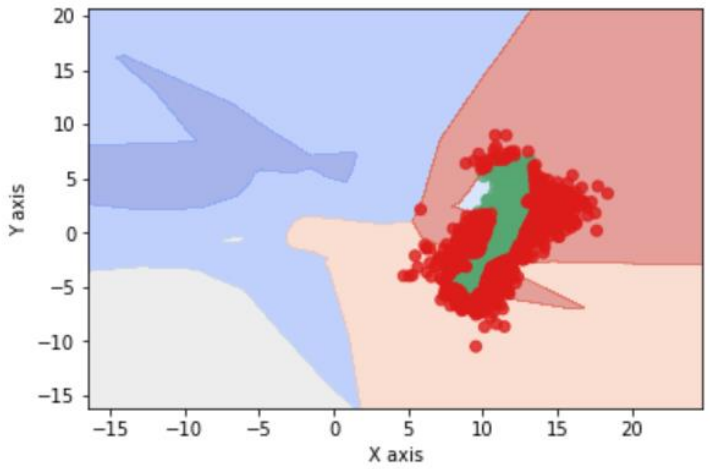

Figure 82. MLP 2D visualisation of test predictions for label 2.

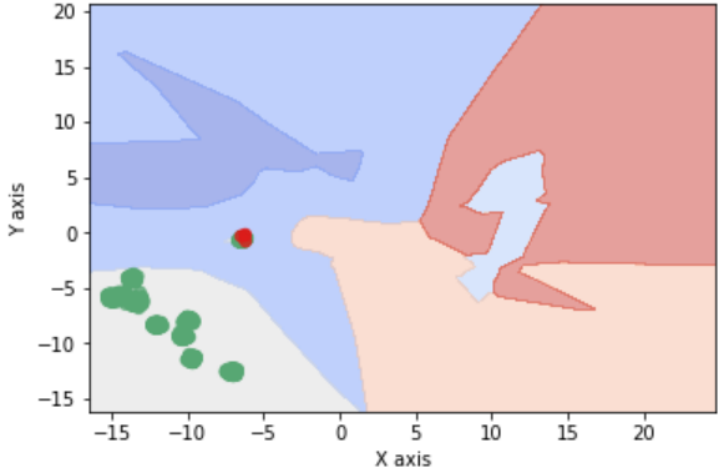

Figure 83. MLP 2D visualisation of test predictions for label 3.

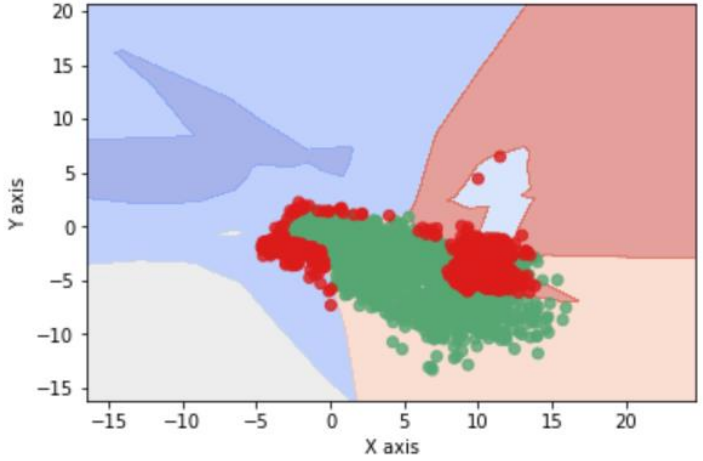

Figure 84. MLP 2D visualisation of test predictions for label 4.

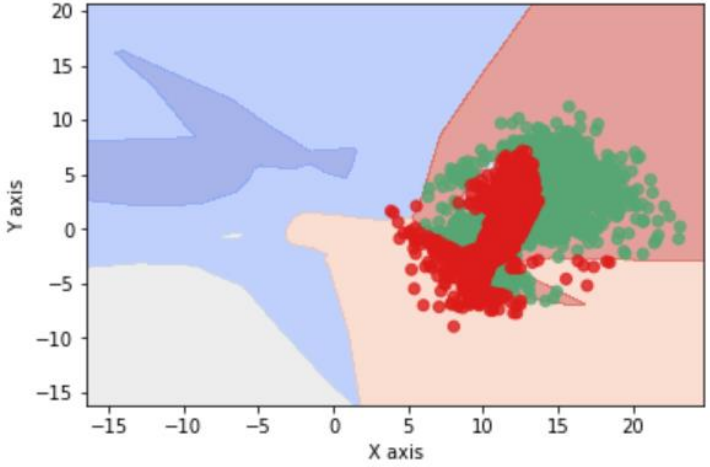

Figure 85. MLP 2D visualisation of test predictions for label 5.

6.3. All Features

6.3.1. SVM

Results for SVM with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.75				
1	0.93				
2	0.84				
3	1.00				
4	0.95				
5	0.88				
AVERAGE/TOTAL	0.89	1.81	1.32	0.71	14min 48s ± 4min 22s per loop

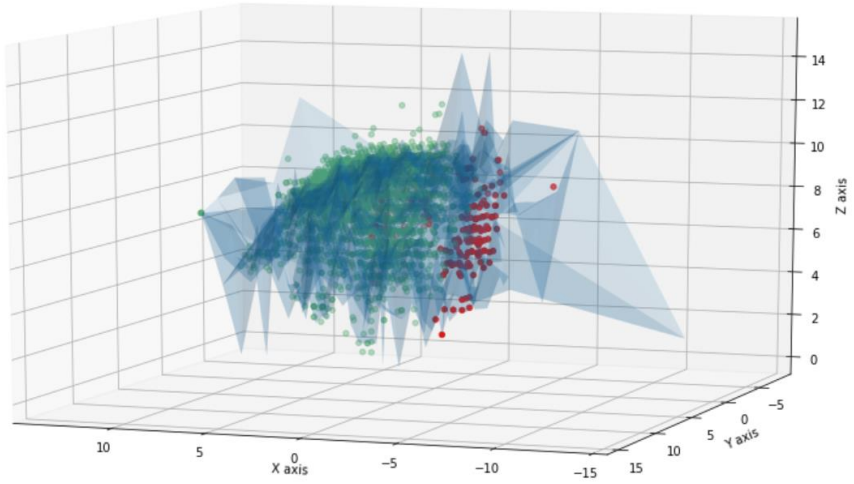

Figure 86. SVM 3D visualisation of test predictions for label 0.

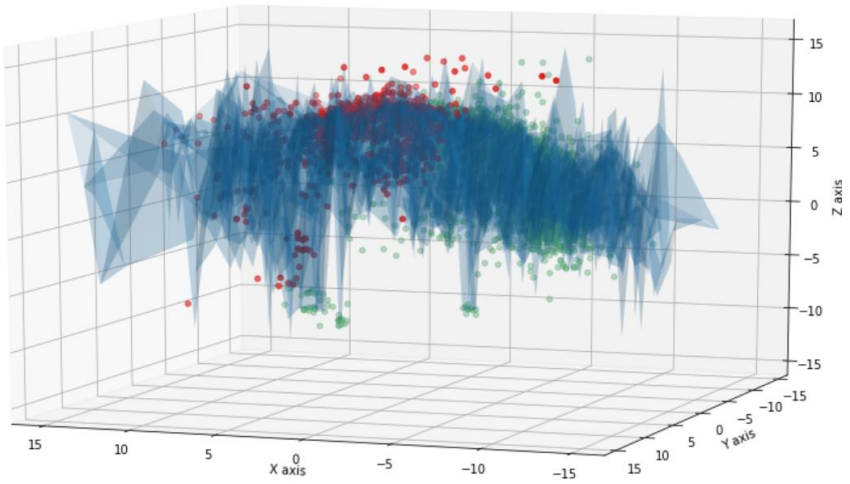

Figure 87. SVM 3D visualisation of test predictions for label 1.

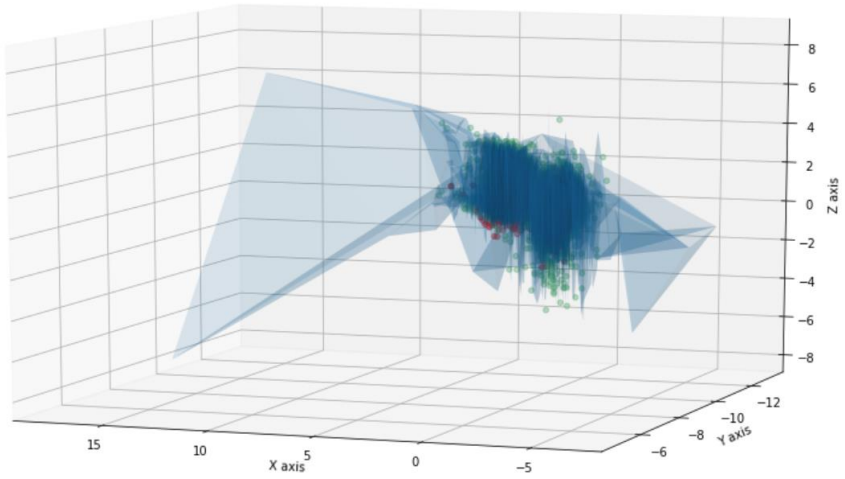

Figure 88. SVM 3D visualisation of test predictions for label 2 on ankle.

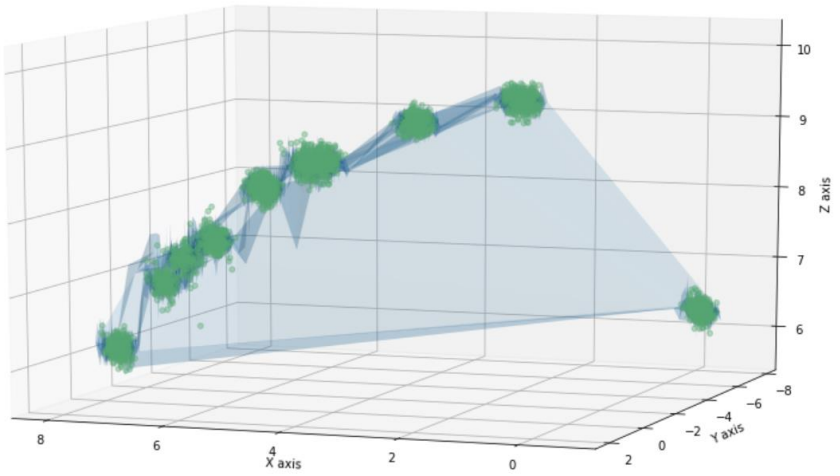

Figure 89. SVM 3D visualisation of test predictions for label 3 on ankle.

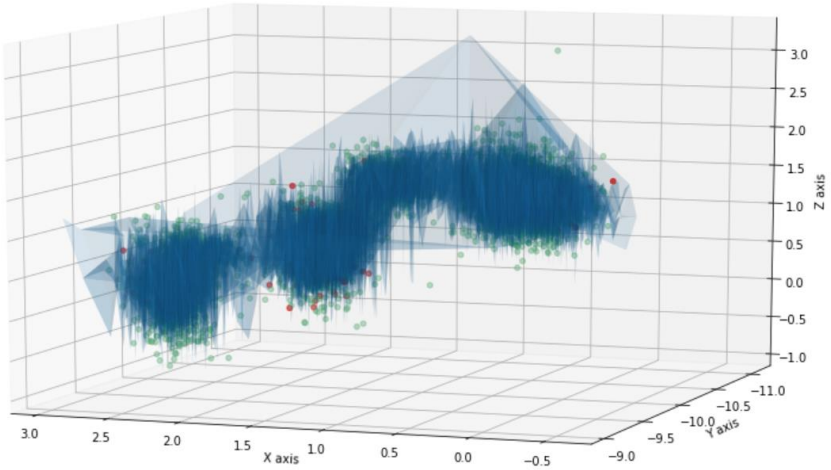

Figure 90. SVM 3D visualisation of test predictions for label 4 on ankle.

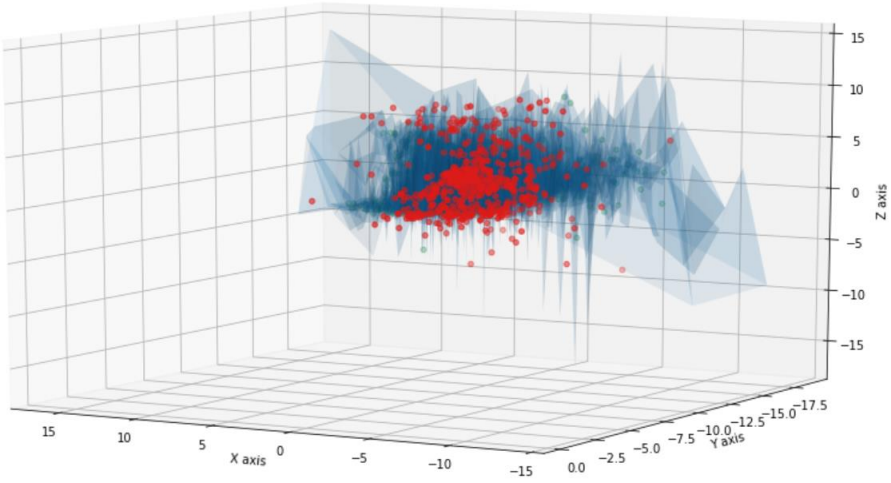

Figure 91. SVM 3D visualisation of test predictions for label 5 on chest.

6.3.2. GNB

Results for GNB with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.72				
1	0.91				
2	0.84				
3	1.00				
4	0.77				
5	0.97				
AVERAGE/TOTAL	0.86	1.84	1.36	0.75	76.8 ms ± 6.2 ms per loop

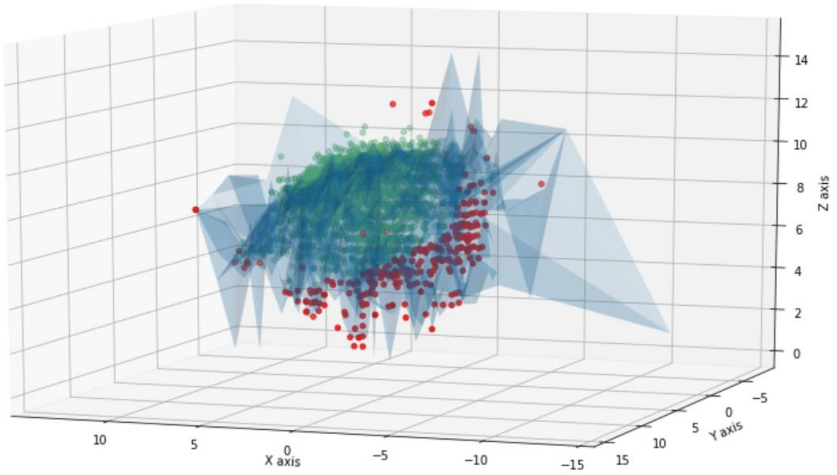

Figure 92. GNB 3D visualisation of test predictions for label 0.

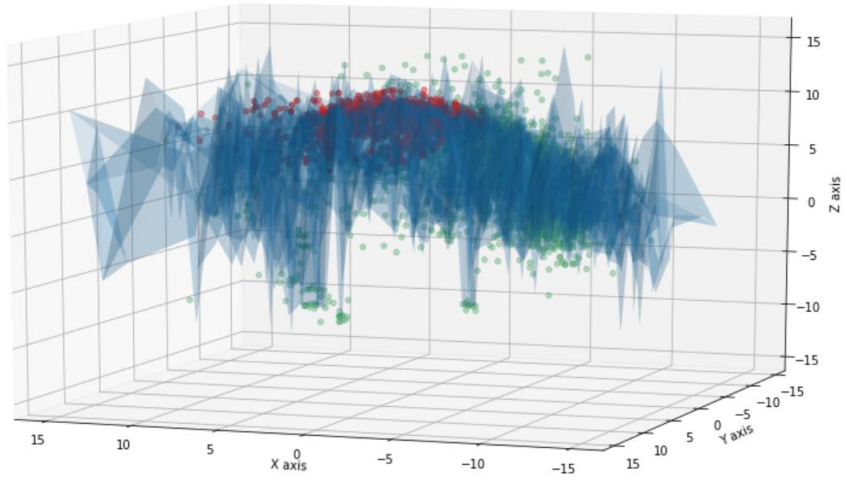

Figure 93. GNB 3D visualisation of test predictions for label 1.

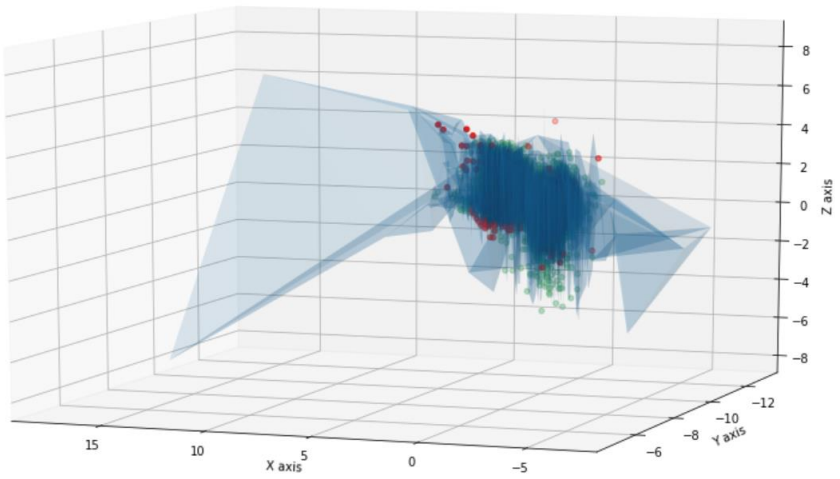

Figure 94. GNB 3D visualisation of test predictions for label 2 on ankle

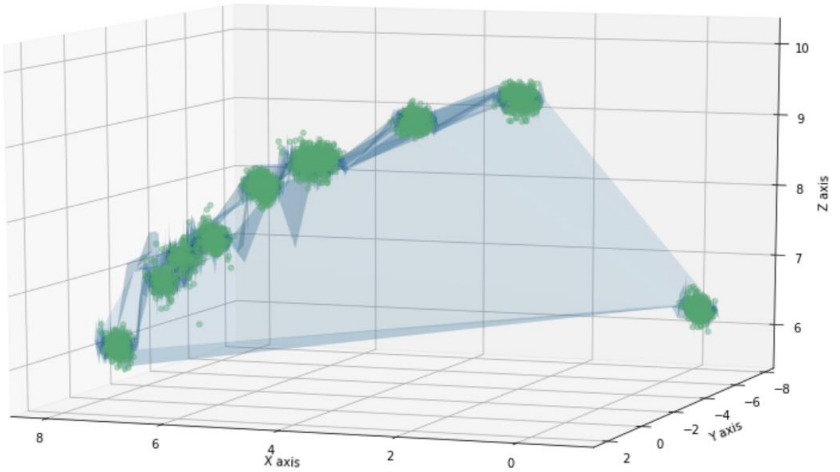

Figure 95. GNB 3D visualisation of test predictions for label 3 on ankle

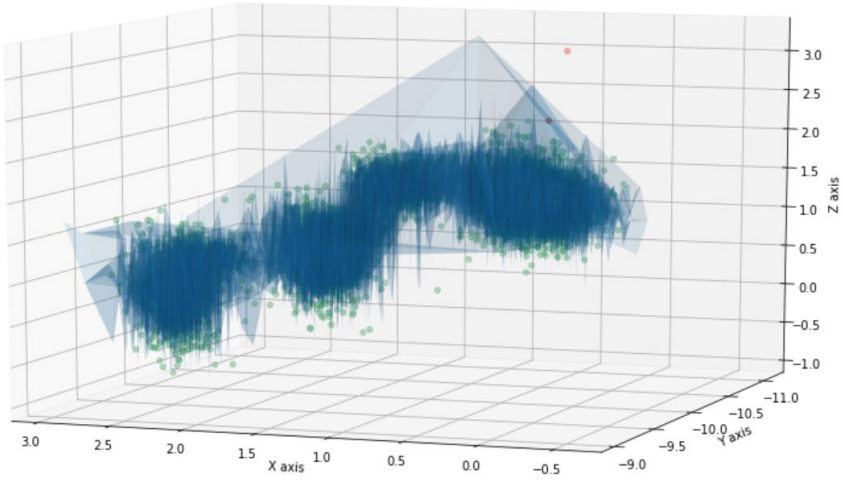

Figure 96. GNB 3D visualisation of test predictions for label 4 on ankle.

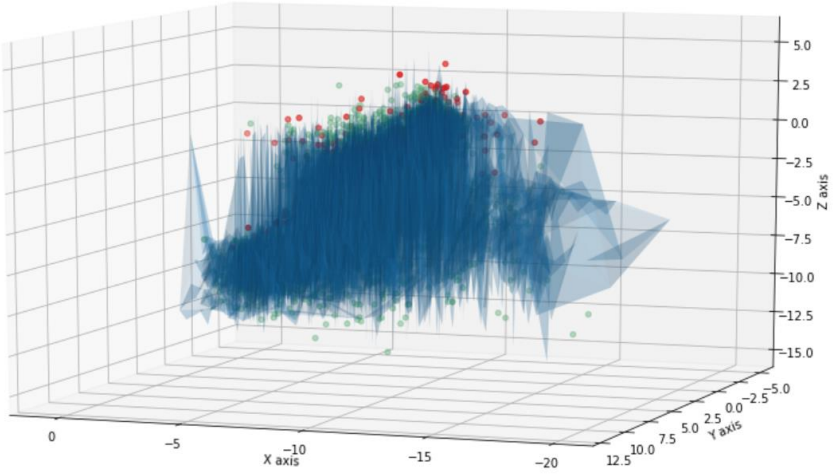

Figure 97. GNB 3D visualisation of test predictions for label 5 on chest.

Results for GNB with priors set in the parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.72				
1	0.91				
2	0.83				
3	1.00				
4	0.77				
5	0.97				
AVERAGE/TOTAL	0.86	1.83	1.35	0.74	82.2 ms ± 3.91 ms per loop

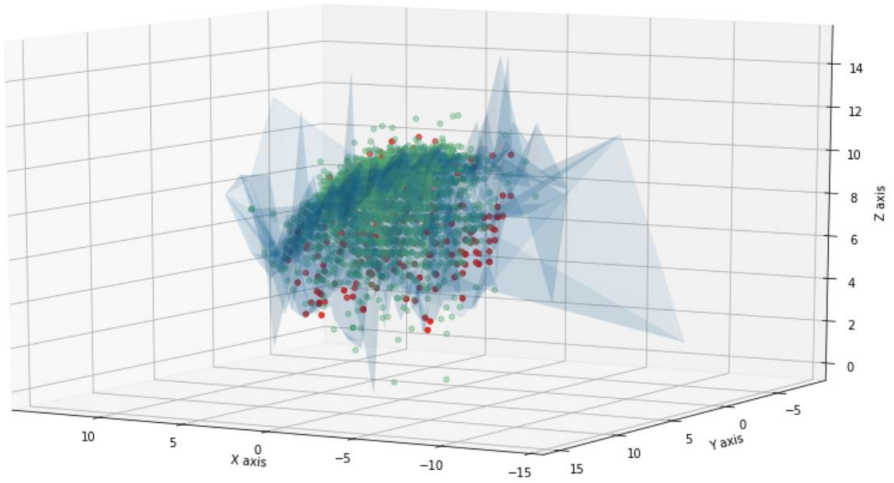

Figure 98. GNB 3D visualisation of test predictions for label 0.

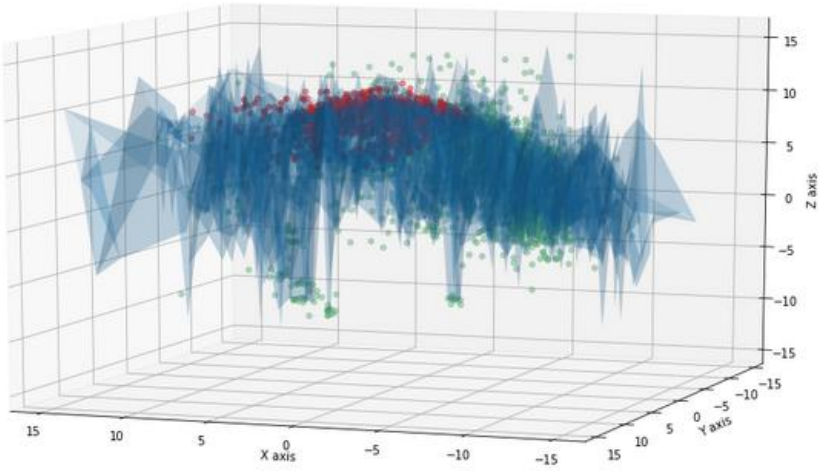

Figure 99. GNB 3D visualisation of test predictions for label 1.

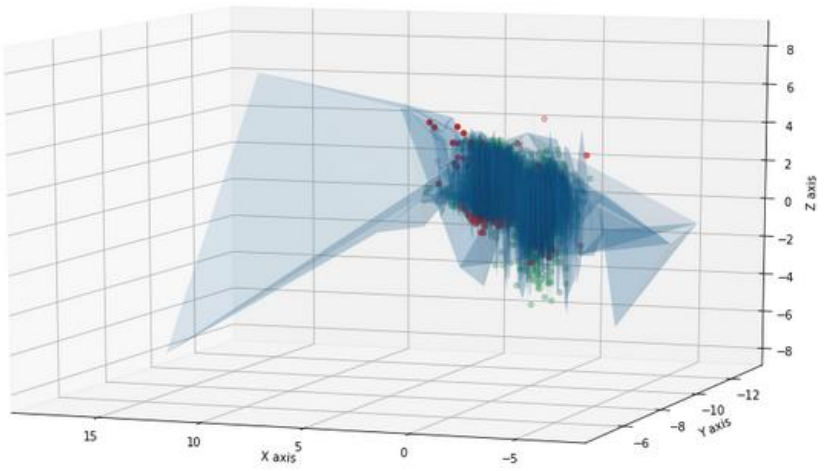

Figure 100. GNB 3D visualisation of test predictions for label 2 on ankle.

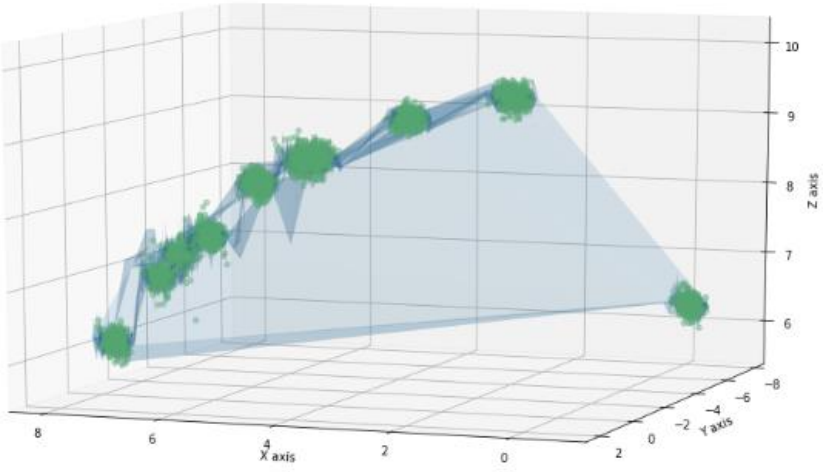

Figure 101. GNB 3D visualisation of test predictions for label 3 on ankle.

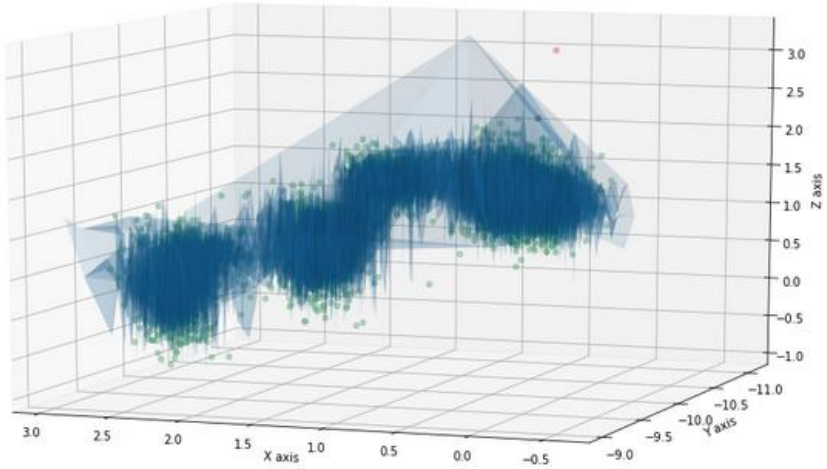

Figure 102. GNB 3D visualisation of test predictions for label 4 on ankle.

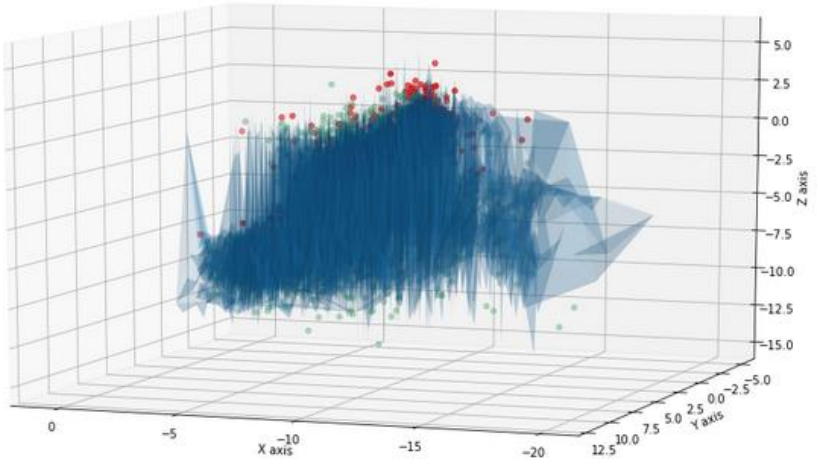

Figure 103. GNB 3D visualisation of test predictions for label 5 on chest.

6.3.3. DT

Results for DT with the first parameter set.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.83				
1	0.90				
2	0.60				
3	1.00				
4	0.84				
5	0.57				
AVERAGE/TOTAL	0.80	1.33	1.16	0.57	1.86 s ± 219 ms per loop

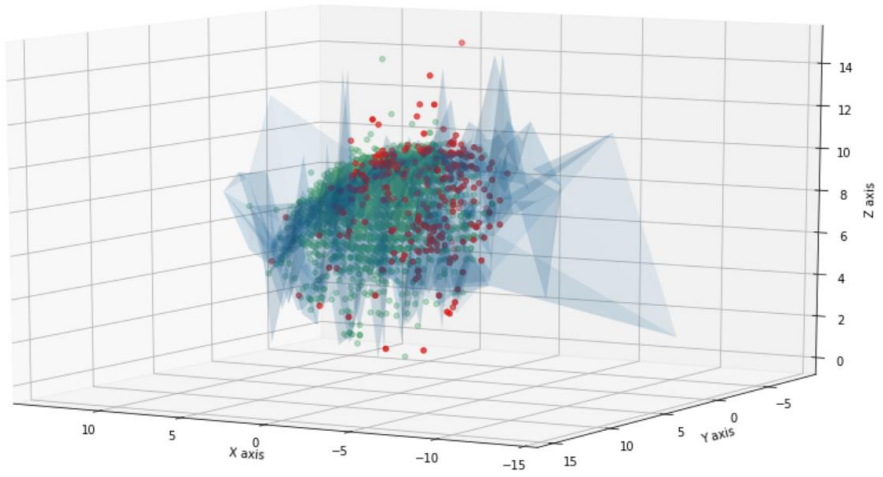

Figure 104. DT 3D visualisation of test predictions for label 0.

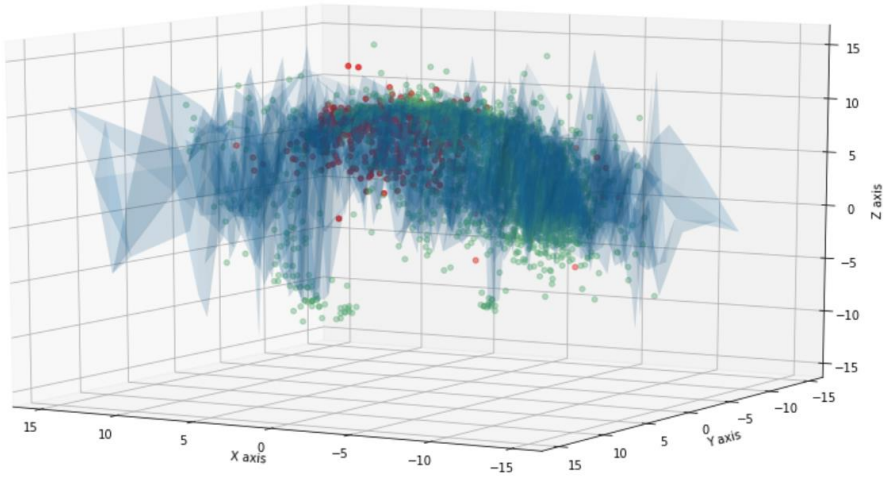

Figure 105. DT 3D visualisation of test predictions for label 1.

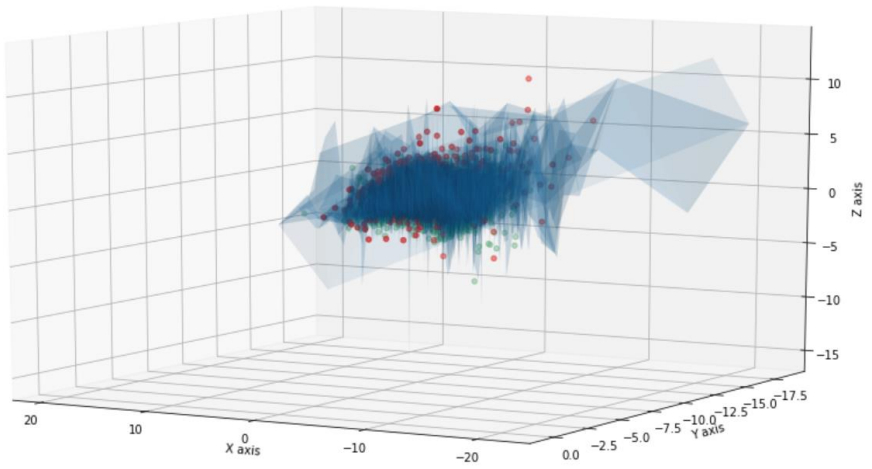

Figure 106. DT 3D visualisation of test predictions for label 2 on ankle.

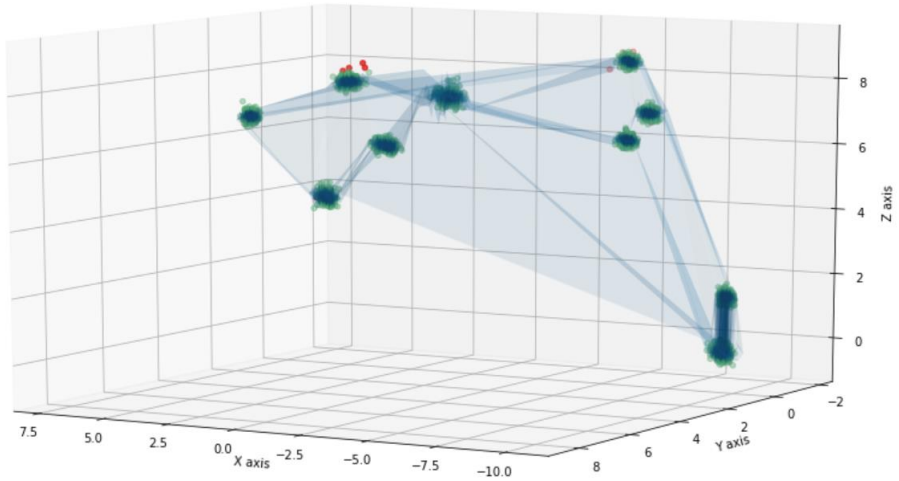

Figure 107. DT 3D visualisation of test predictions for label 3 on ankle.

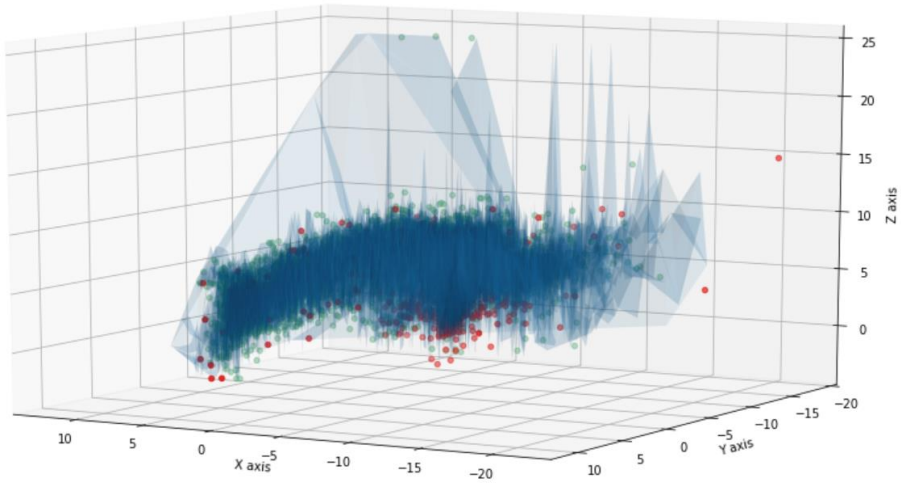

Figure 108. DT 3D visualisation of test predictions for label 4 on ankle.

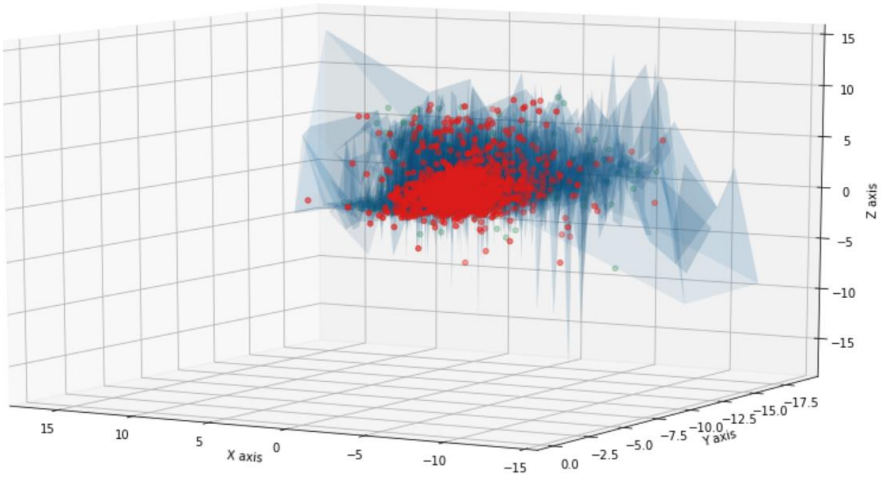

Figure 109. DT 3D visualisation of test predictions for label 5 on chest.

Results for DT with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.83				
1	0.92				
2	0.95				
3	1.00				
4	0.97				
5	0.95				
AVERAGE/TOTAL	0.92	0.18	0.42	0.10	2.47 s ± 187 ms per loop

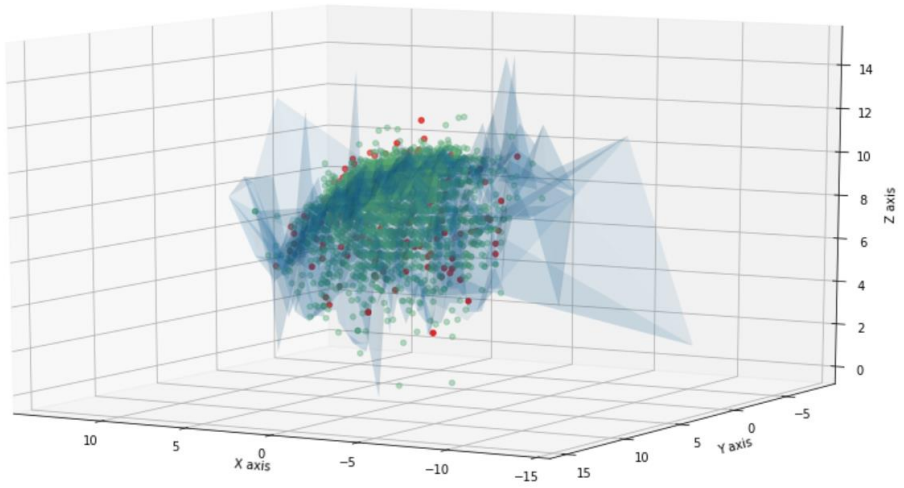

Figure 110. DT 3D visualisation of test predictions for label 0.

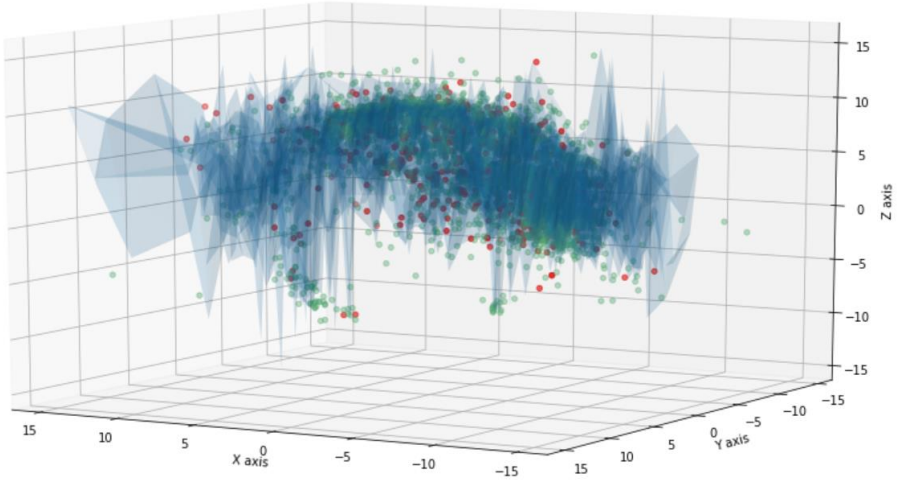

Figure 111. DT 3D visualisation of test predictions for label 1.

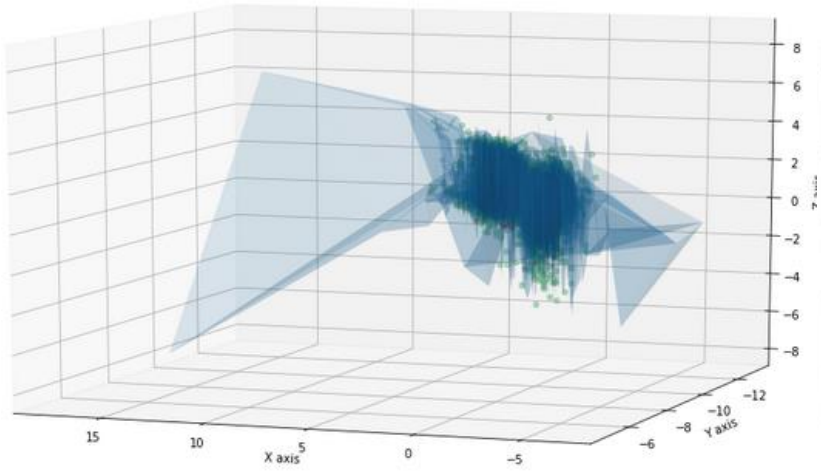

Figure 112. DT 3D visualisation of test predictions for label 2 for ankle.

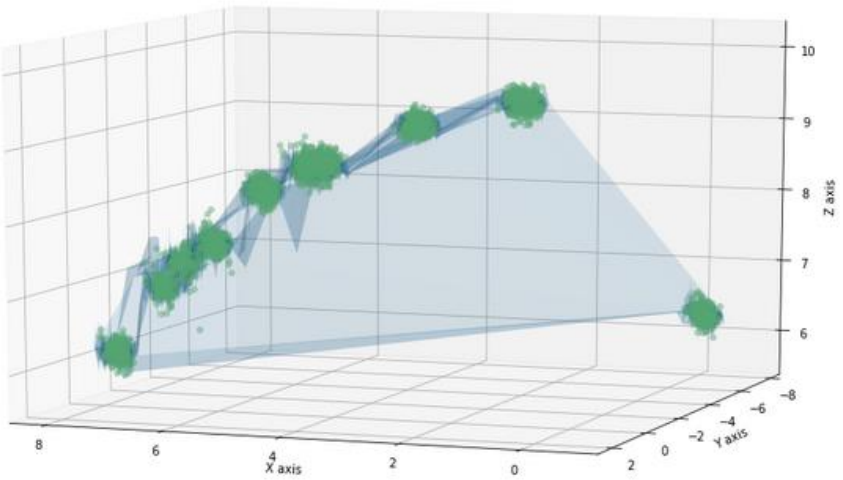

Figure 113. DT 3D visualisation of test predictions for label 3 for ankle.

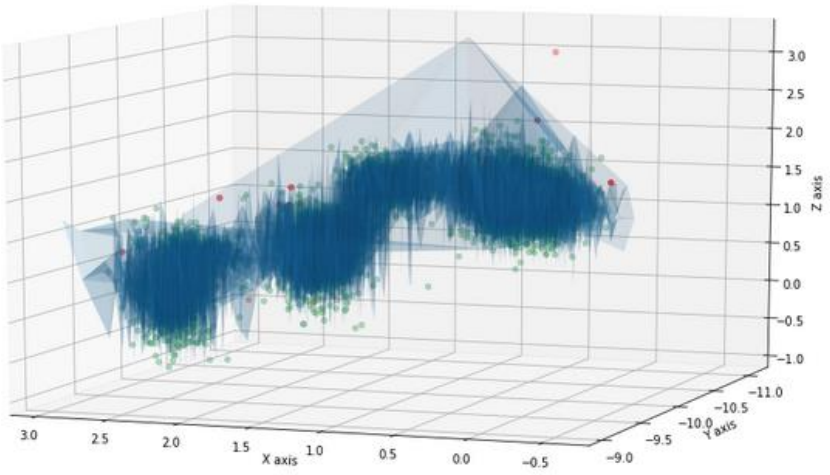

Figure 114. DT 3D visualisation of test predictions for label 4 for ankle..

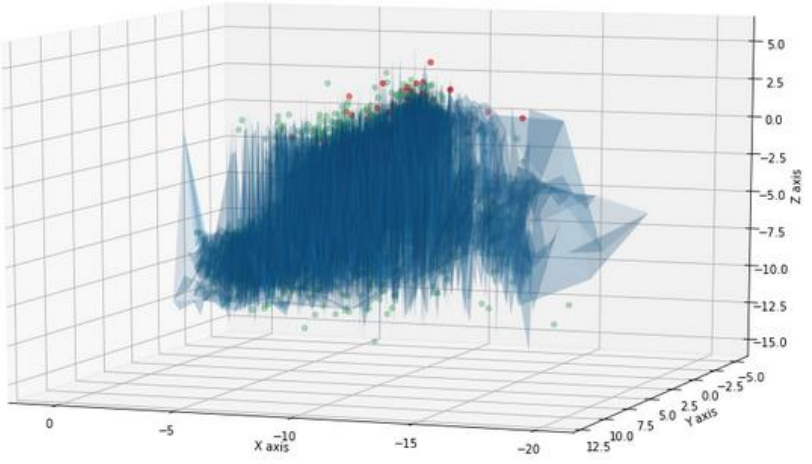

Figure 115 DT 3D visualisation of test predictions for label 5 for chest.

6.3.4. MLP

Results for MLP with the first set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.82				
1	0.92				
2	0.98				
3	1.00				
4	0.98				
5	0.95				
AVERAGE/TOTAL	0.93	0.17	0.41	0.11	1min 5s ± 9.77 s per loop

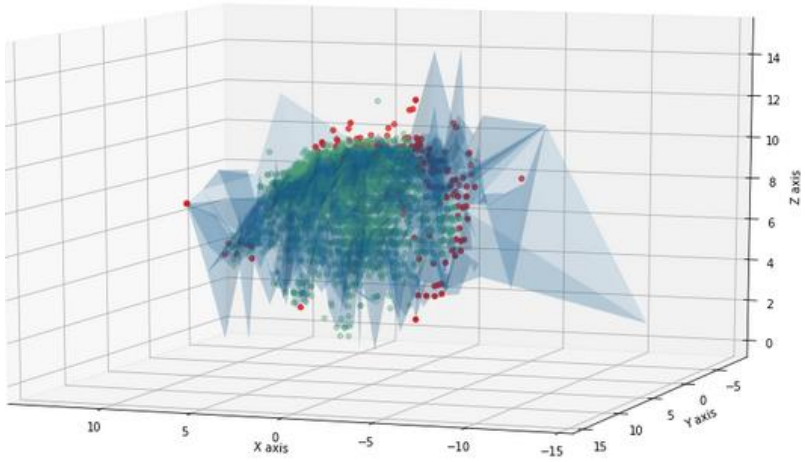

Figure 116. MLP 3D visualisation of test predictions for label 0.

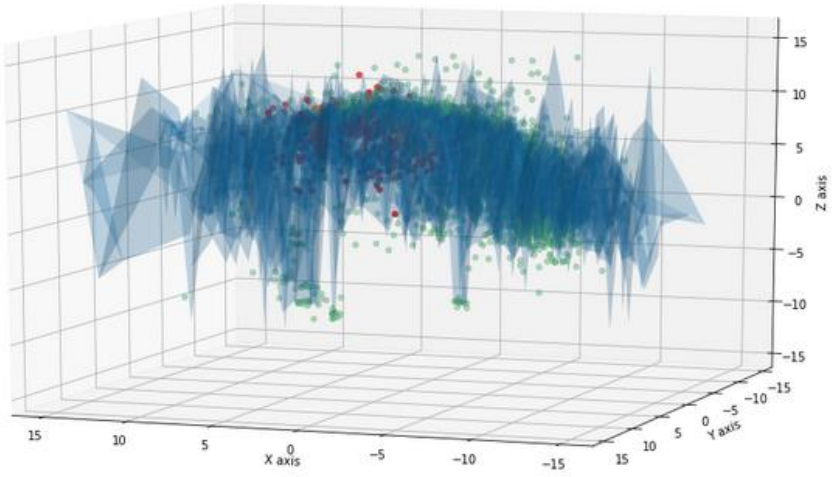

Figure 117. MLP 3D visualisation of test predictions for label 1.

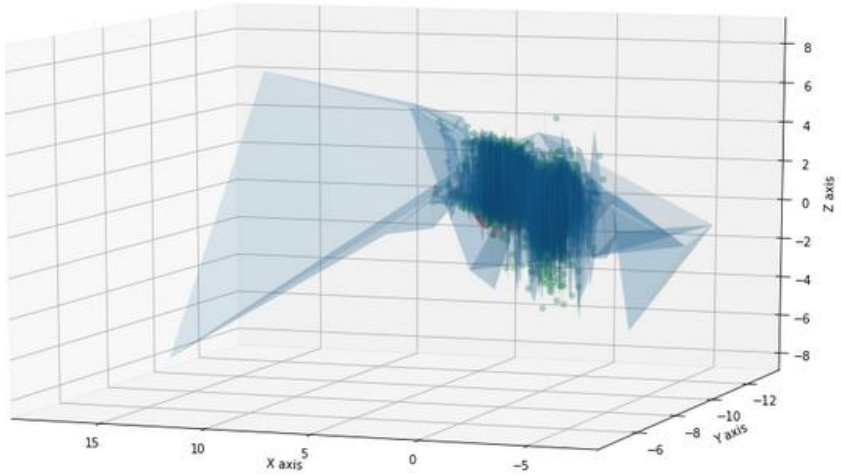

Figure 118. MLP 3D visualisation of test predictions for label 2 on ankle.

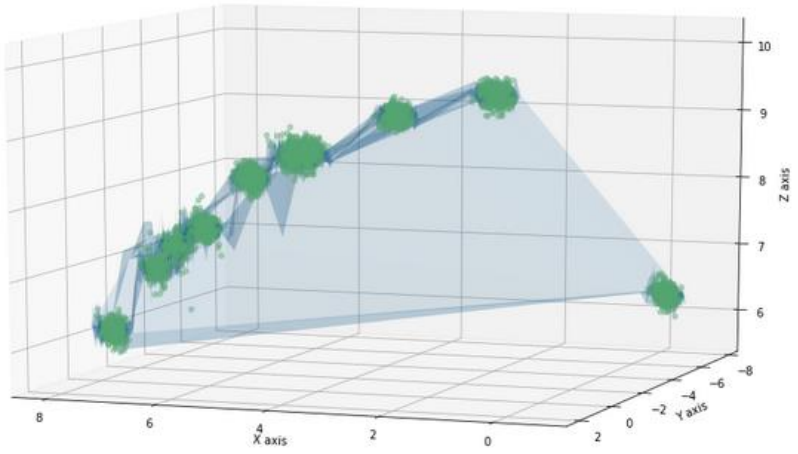

Figure 119. MLP 3D visualisation of test predictions for label 3 on ankle.

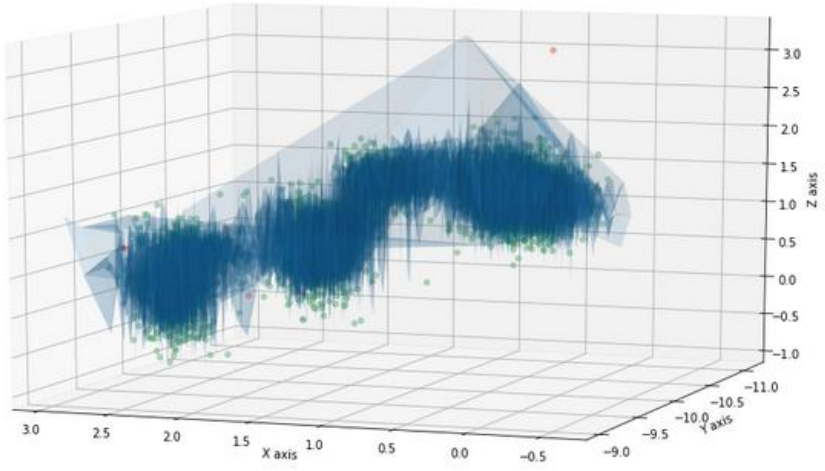

Figure 120. MLP 3D visualisation of test predictions for label 4 on ankle.

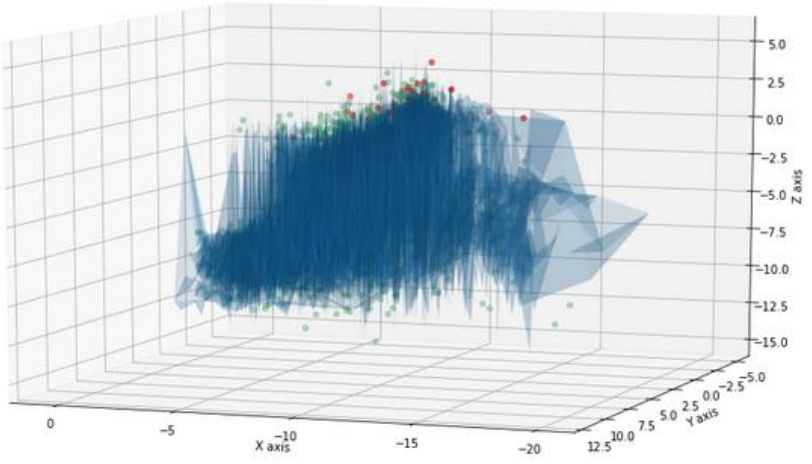

Figure 121. MLP 3D visualisation of test predictions for label 5 on chest.

Results for MLP with the second set of parameters.

LABEL	PRECISION	ERROR			TIME
		MSE	RMSE	MAE	
0	0.81				
1	0.93				
2	0.95				
3	1.00				
4	0.98				
5	0.94				
AVERAGE/TOTAL	0.93	0.17	0.41	0.11	3min 6s ± 12.2 s per loop

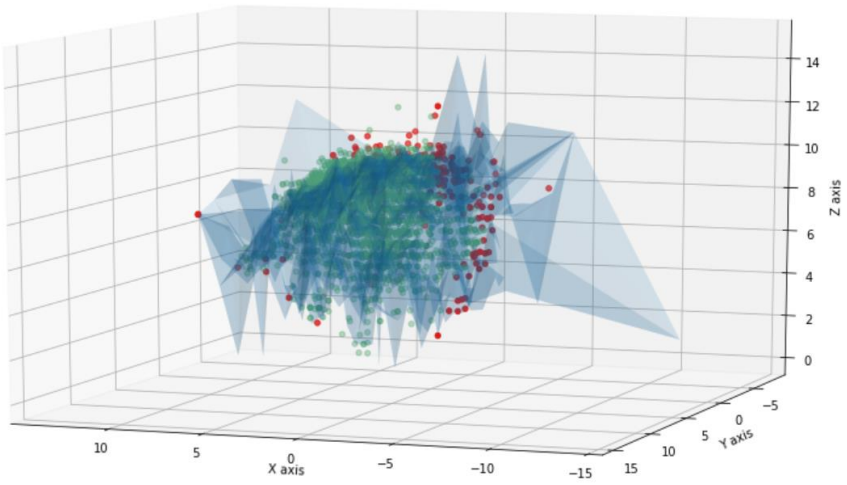

Figure 122. MLP 3D visualisation of test predictions for label 0.

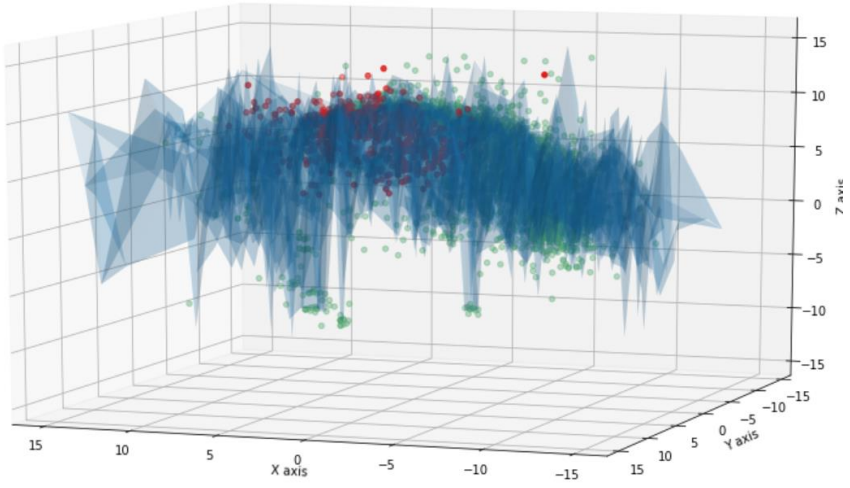

Figure 123. MLP 3D visualisation of test predictions for label 1.

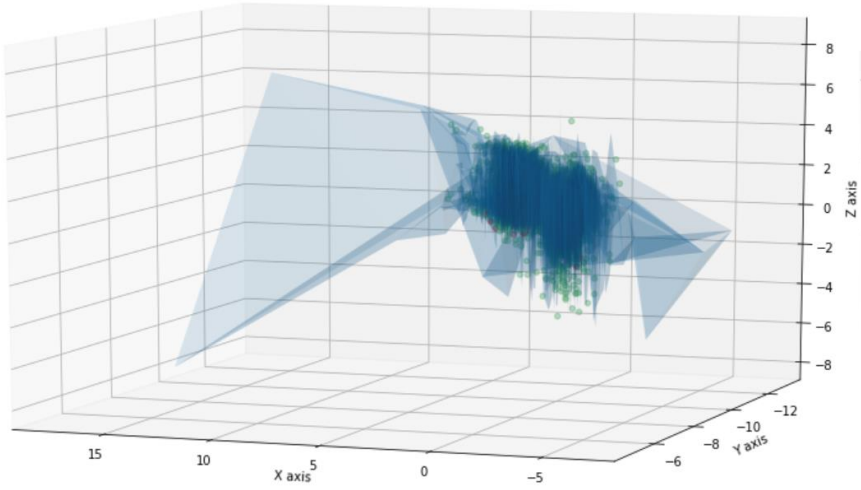

Figure 124. MLP 3D visualisation of test predictions for label 2 on ankle.

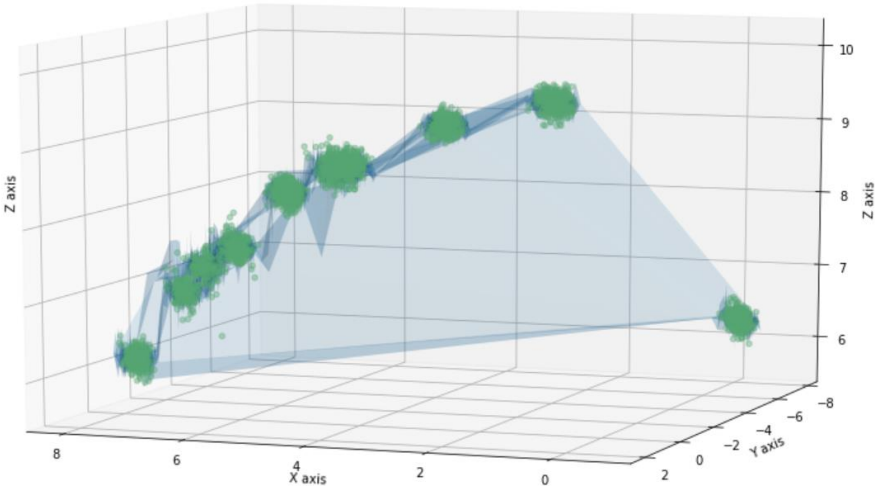

Figure 125. MLP 3D visualisation of test predictions for label 3 on ankle.

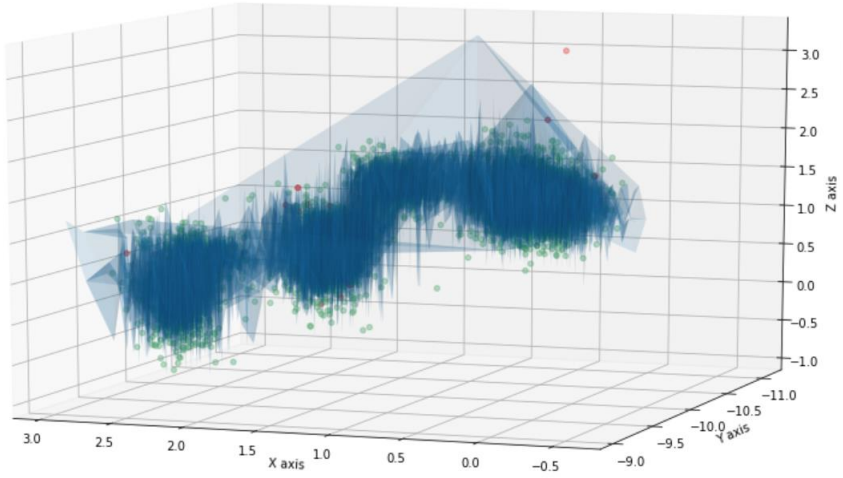

Figure 126. MLP 3D visualisation of test predictions for label 4 on ankle.

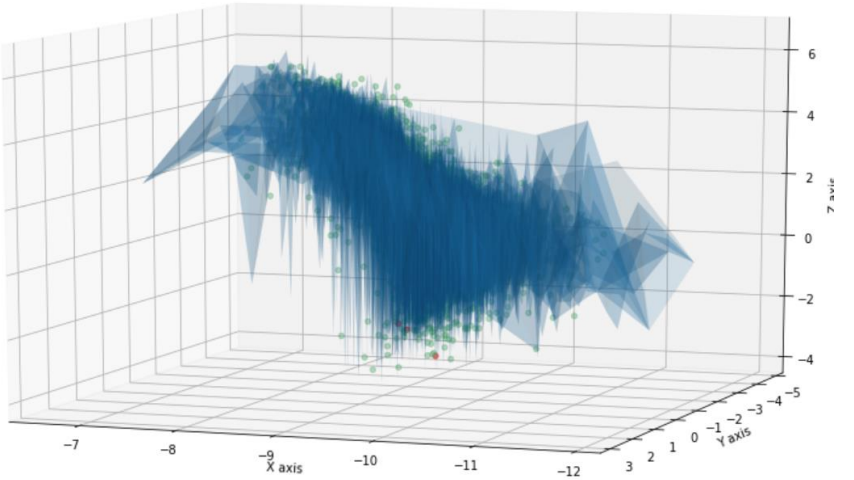

Figure 127. MLP 3D visualisation of test predictions for label 5 on chest.

6.3.5. Validation

In this section, results from the validations with the best models in the implementation, after evaluating the previous results are shown. Evaluation and discussion of the results can be seen in the conclusions section. Nevertheless, we must bear in mind that validation only takes into account the G-forces obtained from the wrist accelerometer, as the dataset was created with only one accelerometer allocated in that position. Therefore, the model with best results in section 6.1 will be used to validate the dataset from the SISMO project.

The dataset for validation is not rather large and it is standardised for its values to be between -1.5g and 1.5g in order to provide homogeneity. The values should be on the same scale as the training and test sets.

Being MLP the best algorithm for the ML models, we observe that:

LABEL	PRECISION	ERROR		
		MSE	RMSE	MAE
0	0.81			
1	0.61			
2	0.68			
3	0.94			
4	0.92			
5	0.78			
AVERAGE/TOTAL	0.79	1.97	1.40	0.67

7. Conclusions

In this document the presentation of state-of-the-art ML algorithms have been presented, as well as techniques to resolve time series forecasting problems. The algorithms dealt with are Support Vector Machines – linear and non-linear; Gaussian Naïve Bayes; Decision Trees; and Multilayer Perceptron. Also, the importance of feature selection and pre-processing of data is explained, so as to obtain reliable predictions from a robust dataset.

In this section, the discussion of the results from the previous section is written. All performance measures are taken into consideration, being these the success rate in the model output predictions, as well as the error rates and the time spend for each set of predictions. Time is in an important factor in this project, because, if taken further, it can be applied to near-real-time applications, where motions of ICU patients can be seen and predicted in short periods of time.

Being the best case scenario, we find that an ML model with MLP configured to work with an quasi-Newton method (‘*lbfgs’*) optimized solver and hyperbolic tangent activation function is the most reliable predictor. Its accuracy is spread out as it can manage to predict simple and complex movement with a 93% accuracy, 0.17 MSE, 0.41 RMSE and 0.11 MAE. Even further, simple mobilizations have greater accuracy, reaching a 100% prediction rate. This must not be judged as a perfect predictor, as fresh data from subjects with higher deviations can produce inaccurate outputs, although it demonstrates the contrary with the given test set. It is also seen that the increase of feature in the samples provide even more reliable predictions for MLP, as more bias weights are considered within the algorithm. Nevertheless, MLP is not the fastest model in the implementation, and the model would not be trained fast enough for real-time applications. In the case of using the dataset with the need of high performance in terms of time, DTs with entropy criterion and defined random state have proven to have almost the same accuracy as MLP in a shorter period of time. The ML model with a DT algorithm achieves a 92% accuracy in only three seconds, being this a considerable result.

Gaussian Naïve Bayes proves to be the fastest ML model, training a model with a fairly large data is less than a second. This could be used for real-time applications. DTs, as aforementioned, are also very fast ML models in terms of training and predictions. The simplicity and lack of correlation between variables is the reason why these models output results so quickly, although predictions are not as good for time series as MLP or SVM with a smaller dataset. On the other hand, SVM can be quite slow and can train a model with the time series data in a long period of time. SVM could not be considered as a good predictor if we are looking at time performance variables.

Principal Component Analysis can allow us to simplify the problem by reducing the dataset to two principal components. PCA enables us to analyse a data with many differences between labels by evaluating the core dependencies. The dataset, although robust, does not always contain values for the G-forces in the areas covered by ankle and chest accelerometers, and PCA allows models to do faster predictions on a more standardised dataset. Also, it proves to be very useful in terms of visualisation, as the margins that separate data points from being predicted to fall into one label or another are easily seen in 2D. 3D visualisations are a bit more cumbersome and it is easier to visualise the data points with a single label area, as many data in the visualisation can overlap and it becomes much less defined to see what predictions fall into which areas. Nevertheless, the results obtained from PCA are not extremely relevant when being

compared to the analysis with the features from the wrist accelerometer or the features from all three accelerometers.

By far, the most interesting part of the implementation is when a dataset a fresh set of data is created from an accelerometer and software from the SISMO project. Modifying the code to obtain a dataset that is found in the training and test set parameters proves that *Scikit-learn* is a very interesting tool that can be used in many ways – the data was scaled by use of its packages. Also, this provides insight of further development for the project, as it can be integrated in real-life applications for early mobilizations tracking. Thus, in cooperation with health professionals, the data obtained from patients in the ICU can be analysed and fed into a ML model, most probability with a DT or MLP algorithm. Following the movements of a patient can distinguish between early mobilization patterns under supervision of such professionals or motions done non-therapeutically, e.g. movements while sleeping or eating. Even further, if the ML models are trained regularly throughout the recovery of a patient, improvements can be tracked to obtain more accurate models against movements with increased complexity and assess which movement are done at different stages of recovery. The lack of time for further development leaves an open discussion for this topic, as it would be positive to test the models with different movements and taking into account different features, that would be obtained from more input sources i.e. accelerometers.

It is demonstrated in the results section that predictions become more inaccurate when the data points in the time series dataset become more complex. As an example, it is observed that the motion labelled under “5”, crouching by bending knees, always has de worst predictions. This can be due to the fact that G-forces obtained from accelerometers that are not located directly on the points of movement had greater variations. The bending knees in different subjects have similar G-forces in that area, although subject can move their arms, chest or ankles very differently when applying the movement. This is strongly reflected in the validation section, as movements such as drinking from a glass (label “1”) or lying down (label “3”) have good predictions because of the location of accelerometers on the wrist and chest, respectively. The fresh dataset provided has proven that the MLP model can do fairly good predictions when subject input does not fluctuate too widely. The rescaling of the set helps strongly for these predictions to be more accurate. Nevertheless, improvements can be done as it is not known exactly how the movement where done and with what speeds or the exact bending of limbs, for example.

8. Further Work

Analysis of time series can be done in many ways and ML problems do not always have one specific solution for each problem. This study proposes one possible solution for time series forecasting after using several ML models and evaluating the outputs. Nevertheless, further work can be done in order to provide a solution with stronger predictive features. Listed below we find several proposal for this purpose:

a. Usage of deep neural networks. Deep learning cover ML algorithms, more specifically artificial neural networks, where multiple layers of nonlinear processing units are found. This means that several hidden layers transform and extract features by cascading in different layers of abstraction. Deep generative models – models with joint probability distribution – organise the layers such that data inputs increase in complexity in terms of composition and abstraction. The higher the number of layers, the deeper the neural network becomes. Deep learning models are a lot more complex to deal with regarding propagation and recurrence, although representation redundancy is avoided as they extract principal components from the feature set.
b. Having more data. More data from other mobilizations, being these simple or more complex can train models even further. The usage of data points makes it easy to obtain new samples, although standardisation and data labelling should always be considered. The implication of more data, nevertheless, being for new mobilizations or even more sensors, can affect ML models in terms of overtraining. Overtraining of models can imply the decrease of prediction accuracy. Also, the amount of samples requires more control over iterations, especially for models with iterations, as well as computational power, as ML models can become slower when huge amounts of data have to be processed.
c. Evaluation of cycles or repetitive patterns by using groups of data points as samples. As read in section 2, time series forecasting strives to find patterns, cycles and repetitions within data. By training models on single data points this can be achieved, although grouping time series data can determine these characteristics more accurately and offer better predictions. This can be very beneficial but complexity in the resolution of problems increases greatly. The appearance of variables such as the frequency of cycles, for example, requires the usage of assumptions, or complex algorithms to find approximations, that would make the groupings possible to use without decreasing prediction accuracy. The shifting of periods can be complex to assist with time series data, although it would definitely be a very interesting problem to solve.

9. References

1. Bontempi, G., Ben Taieb, S., & Le Borgne, Y.-A. (2013). Machine Learning Strategies for Time Series Forecasting. Discovery, (July 2015), 62–77. https://doi.org/10.1007/978-3-642-36318-4_3

2. Ahmed, N. K., Amir; Atiya, F., Neamat, ;, Gayar, E., El-Shishiny, H., … Gayar, N. El. (2010). An Empirical Comparison of Machine Learning Models for Time Series Forecasting. *Econometric Reviews*, *29*(29), 594–6215. https://doi.org/10.1080/07474938.2010.481556

3. Bansal, D., Chhikara, R., Khanna, K., & Gupta, P. (2018). Comparative Analysis of Various Machine Learning Algorithms for Detecting Dementia. *Procedia Computer Science*, *132*, 1497–1502. https://doi.org/10.1016/j.procs.2018.05.102

4. Seligman, B., Tuljapurkar, S., & Rehkopf, D. (2017). Machine Learning Approaches to the Social Determinants of Health in the Health and Retirement Study. *SSM - Population Health*, *4*(November 2017), 95–99. https://doi.org/10.1016/j.ssmph.2017.11.008

5. Khaleghi, A., Ryabko, D., Mary, J., & Preux, P. (2016). Consistent Algorithms for Clustering Time Series. *Journal of Machine Learning Research*, *17*(3), 1–32. Retrieved from http://jmlr.org/papers/v17/khaleghi16a.html

6. Van der Heijden, M., Velikova, M., & Lucas, P. J. F. (2014). Learning Bayesian networks for clinical time series analysis. *Journal of Biomedical Informatics*, *48*, 94–105. https://doi.org/10.1016/j.jbi.2013.12.007

7. Smola, A. J. (2000). Using Support Vector Machines for Time Series Prediction. *GMD FIRST*, (May 2014), 1–13. https://doi.org/10.1515/9783110915990.1

8. Spiegel, S., Gaebler, J., Lommatzsch, A., Luca, E. De, & Albayrak, S. (2011). Pattern Recognition and Classification for Multivariate Time Series Categories and Subject Descriptors. *Time*, 34–42. https://doi.org/10.1145/2003653.2003657

9. Analytics, B. (2017). A comprehensive beginner's guide to create a Time Series Forecast (with Codes in Python). Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/

10. MHEALTH Dataset. ICS UCI EDU. Retrieved from https://archive.ics.uci.edu/ml/datasets/MHEALTH+Dataset

11. Dataset for ADL Recognition with Wrist-worn Accelerometer. ICS UCI EDU. Retrieved from https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer

12. How to Predict Multiple Time Series with Scikit-Learn with sales Forecasting Example. Mario Filho. Retrieved from http://mariofilho.com/how-to-predict-multiple-time-series-with-scikit-learn-with-sales-forecasting-example

13. Time Series Forecast Study. Jason Brownlee on February 20, 2017. Retrieved from https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-french-champagne

14. In Depth: Naïve Bayes Classification. Python Data Science Handbook. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/05.05-naive-bayes.html

15. Gaussian Processes. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/gaussian_process.html

16. Support Vector Machines. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/svm.html

17. Naïve Bayes. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/naive_bayes.html

18. Decision Tree. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/tree.html

19. Feature Selection. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/feature_selection.html

20. Neural Network models (supervised) . Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/neural_networks_supervised.html

21. Preprocessing data. Scikit-learn developers 2007-2017. Retrieved from http://scikit-learn.org/stable/modules/preprocessing.html