# Data Mining Using the Grand Tour Algorithm

*Dominic Dent*

*9567718*

*In Collaboration With Karan Muhki*

*Supervised by Prof. Stephen Watts*

University of Manchester

School of Physics and Astronomy

MPhys Project

December 2018

## Abstract

The Grand Tour algorithm produces a series of 2D projections of a high dimensional data-set. The algorithm was implemented in Python. Machine learning algorithms were then applied to the 2D projections and their performances were analysed. An alternative algorithm that finds optimal 2D projections for maximising accuracy was developed. This new algorithm successfully finds projections that produce higher accuracies, in a faster time when compared to the Grand tour.

## Contents

# 1 Introduction

From new ways of thinking about cancer prognosis to personalised political advertisements, data mining has a massive influence in todays world. Due to the increase in computational power and the amount of readily available data, machine learning techniques have seen a huge increase in use across academia and industry.

Data visualisation is an important tool when investigating patterns in data. It is also important for communication of information in data to people (especially to clients who are less well-versed in the data science techniques). In two dimensions, humans can easily interpret a plane that separates classes, but as dimensionality increases, problems in intuition quickly arise. The motivation behind the Grand Tour algorithm is to simplify data. One example of the utility of this algorithm is in finding 2D projections of a data-set when a high dimensional hyperplane (that separates two classes of data) can be simplified to a straight line separation.

The key aims of this project were to see if the grand tour could be used to gain an understanding to which machine learning algorithms may be best suited to a data-set. After finding projections that the machine learning models gave high accuracies, we decided to attempt to find new rotations that maximised the accuracy. A new technique to generate rotation matrices was constructed with the aim of maximising accuracy. This technique can also be used to maximise any other single-valued measure of a 2D data-set.

Throughout this report, the vector space containing the full dimensionality of the data is known as the feature space.

## 2 The Grand Tour Algorithm

The grand tour is an algorithm that was developed as a tool to visualise high-dimensional data. Beyond three dimensions, visualising multi-dimensional spaces becomes problematic and unintuitive for most people. The grand tour proposes to get around this problem by producing a series of rotation matrices. The infinite set of these rotation matrices applied to the data points show the multidimensional space from every possible 2D projection. Due to the practical infeasibility of having an infinite number of rotation matrices, the Grand Tour algorithm attempts to quickly explore distinct 2D projections of the feature space, and as the number of time iterations increase, the closer the algorithm gets to producing every possible projection.

A set of key features for the Grand Tour to have were proposed in the original 1985 paper by Daniel Asimov [1]. These include:

1. The sequence of planes should be dense in the space of all planes. A subset (the sequence of planes) is called dense in a set (space of all planes) if any member of the set can be arbitrarily well-approximated by elements of the subset. This motivates the use of the Grassmannian manifold $G_{2,n}$. The general Grassmannian $G_{k,n}$ is a compact smooth manifold that parameterises all $k$-dimensional linear subspaces of a $n$-dimensional real or complex vector space.

2. The sequence of planes should become dense in $G_{2,n}$ rapidly. For any practical use of the Grand Tour, the algorithm needs to be computationally efficient in satisfying some reasonably dense coverage of the space of all planes.

3. Ideally, the sequence of planes would be uniformly distributed in $G_{2,n}$.

4. * For human visualisation reasons, the Grand Tour should be continuous in the limit of infinitesimal step-size.

5. * Again, for human visualisation purposes, the sequence of planes should be approximately straight. In other words, the curve in $G_{2,n}$ should approximately follow a geodesic.

6. There should be some degree of flexibility in customising the grand tour's parameters.

7. The sequence of planes should be deterministic, that is to say, any Grand Tour should be able to be repeated.

* For the aim of this project, these features are not required for the majority of this project due to the reason that it is less important for human visualisation when data mining.

The grand tour also has the flexibility that the data can be projected into any number of dimensions less than $n$.

There are various methods to implement the grand tour algorithm, in this project the Asimov-Buja Winding Algorithm in $d$-space was used [1][8].

## 2.1 The Asimov-Buja Winding Algorithm or Torus Method

Each data point is represented as an $n$-dimensional vector in the feature space

$$\boldsymbol{x} = \sum_{i=1}^{n} x_i \boldsymbol{e}_i \tag{1}$$

where $x_i$ is the value of the data for the $i^{th}$ dimension and $\boldsymbol{e}_i$ is the $i^{th}$ unit basis vector. The data is transformed by multiplying the successive rotation matrices produced from the Grand Tour by the data using

$$\boldsymbol{x}' = \boldsymbol{Q}(t)\boldsymbol{x} \tag{2}$$

where $\boldsymbol{Q}(t)$ is the rotation matrix given by the Grand Tour at timestep $t$, the details of which will be described below. After the transformation has been made, $\boldsymbol{x}'$ is still a $n$-dimensional vector. To make the projection into 2D, two of the orginal $\boldsymbol{e}_i$ basis vectors need to be chosen. If $j$ and $k$ are the chosen basis directions, the 2D space can now be represented by the respective $x'$ and $y'$ values

$$\begin{aligned} x' &= \boldsymbol{x}' \cdot \boldsymbol{e}_j, \\ y' &= \boldsymbol{x}' \cdot \boldsymbol{e}_k. \end{aligned} \tag{3}$$

To construct the generalised rotation matrices in the Grand Tour, we need to choose $\boldsymbol{Q}(t)$ to be an element of the special orthogonal group $SO(n)$ which has the properties of being a square $n \times n$ matrix with a determinant of $+1$. $SO(n)$ is also equivalent to the space of all rotations of the unit sphere in $\mathbb{R}^n$, and is a Lie group.

We require the sequence of planes in the grand tour to be dense in the space of all planes. To satisfy this, consider a curve on a $n$-torus in $p$-dimensions, $\mathbf{T^p}$, where $p = \frac{n(n-1)}{2} = \binom{n}{2}$. The curve $\alpha$ will be dense if the components of $\alpha$ are all linearly independent real numbers [1]. In this context, real numbers $\lambda_i$ with $i = 1, \ldots, p$ are said to satisfy the condition of linear independence if $\sum_{i=1}^{p} \lambda_i n_i = 0$ (where $n_i$ can be any set of integers) is only satisfied by the set containing $n_i = 0, \forall i$. This gives the mapping

$$\begin{gathered} \alpha : \mathbb{R} \to \mathbf{T^p} \\ \text{via} \\ \boldsymbol{\alpha}(t) = (\lambda_1 t, \ldots, \lambda_p t) \end{gathered} \tag{4}$$

where $t$ is the time-step multiplied by the constant step-size. The motivation behind using a $p$-torus is due to its inherent cyclical nature, this property combined with the linearly independent coefficients means that it fufills the denseness condition.

A mapping now needs to be defined from $\mathbf{T^p}$ onto $SO(n)$. The mapping will also need to be surjective to preserve the denseness feature of the $\alpha$ curve. Consider the rotation matrix $R_{i,j}(\theta)$ to be the element of $SO(n)$ that rotates the $\boldsymbol{e}_i\boldsymbol{e}_j$ plane by an angle of $\theta$ with the elements of the matrix being

$$R_{i,j}(\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sin(\theta) & \cdots & \cos(\theta) & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix} \tag{5}$$

with the cosines and sines appearing on the $i^{th}$ and $j^{th}$ rows and columns. The mapping can now be defined as

$$\begin{gathered} \beta : \mathbf{T^p} \to SO(n) \\ \boldsymbol{\beta}(\theta_{1,2}, \theta_{1,3}, \ldots, \theta_{n-1,n}) = R_{1,2}(\theta_{1,2}) \times R_{1,3}(\theta_{1,3}) \times \ldots \times R_{n-1,n}(\theta_{n-1,n}) \end{gathered} \tag{6}$$

where the the angles $\theta_{i,j}$ are equal to the elements of $\boldsymbol{\alpha}(t)$. The full rotation matrix is thus defined as $\boldsymbol{Q}(t) = \boldsymbol{\beta}(\boldsymbol{\alpha}(t))$, and when used in conjunction with equation (2) the 2D Grand Tour projections are produced.

## 2.2 Implementation

The algorithm was implemented with Python, and animated with the libraries PyQt and Matplotlib. To generate the $p$ independent real numbers, the first $p$ prime numbers were found and then the square root was taken. To keep all the values between 0 and 1, the actual number used was the square root of a prime modulo 1.

In practice, the time $t$ was calculated by using $t = t'\delta$, where $t'$ would go up in integers for each time step, and $\delta$ would be the step size. The value for $\delta$ was determined by seeing how smooth the grand tour looked after rendering the animation. It should be noted that the dimensionality of the feature space has an impact on the choice of $\delta$.

## 3 Machine Learning

Machine learning is an area of study involving making predictive computational models using statistics. There are two key types of machine learning algorithms - supervised and unsupervised. This project concerns the supervised type where there is access to a set of training data. Training datasets contain both input and output data, and by applying machine learning algorithms a model is trained to predict the output of new input data. Unsupervised learning consists of using input data to find structure in the given dataset.

Three different machine learning techniques were looked at, Support Vector Machine (SVM), Decision Tree (DT), and Neural Network (NN). By looking at the set of 2D plots over all calculated time steps, the 2 dimensions were used as inputs in three different machine learning algorithms. Each point on the 2D plot had a corresponding label or class. By using the 2 dimensions produced by the Grand Tour, and the corresponding class label, machine learning models were trained to predict the class. An accuracy measurement was taken to quantify how well the different models performed.

Overfitting is a problem in machine learning where the model has a complicated decision boundary that possess a heavy bias towards the training data set [4]. This has the effect of causing the models to not generalise well to new data samples. A randomised split of training data and test data was made to get a good indication of how well a model will perform at predicting new samples. However, due to the limited number of training samples in some of the data sets, this was not feasible for every data-set. To further check for overfitting, the decision boundaries were also plotted.

To implement the machine learning algorithms, various python libraries were used. Scikit-learn was primarily used for the SVM and decision tree classifiers, and TensorFlow/Keras was used for the neural network.

## 3.1 Support Vector Machine (SVM)

The support vector machine, or SVM, is a supervised learning model finds a decision boundary that maximises the margin between two classes.

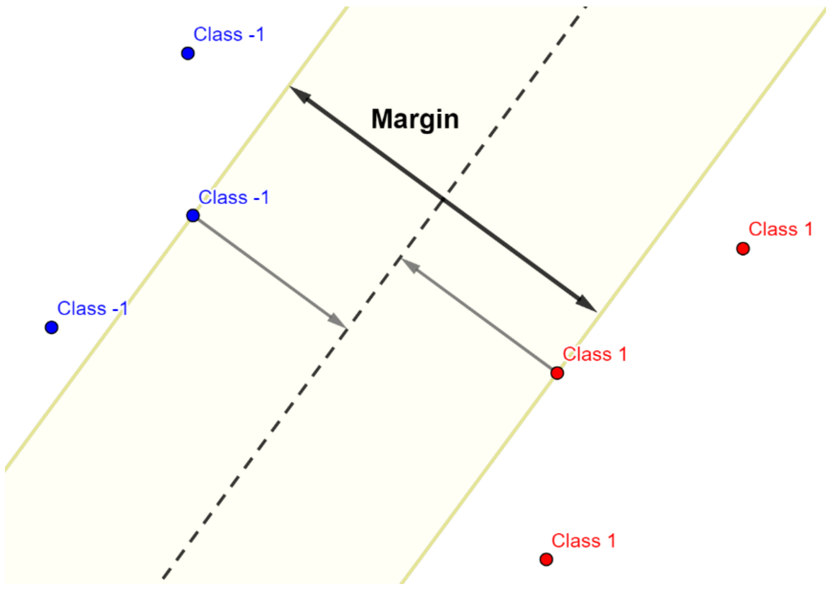


**Figure 1:** Figure showing a margin (black line) between two classes of data. The variable M is shown by the grey lines.

The variable M (shown as the grey lines in Figure 1) is equal to half the margin distance and is the measurement of the closest point to the boundary line. The SVM algorithm finds the hyperplane that gives the maximal margin. The value $M$ is maximised subject to the contraint

$$y_i(x_i^T\beta + \beta_0) \geq M,\ \forall i \tag{7}$$

where $y_i$ is the target variable for a single data point $i$, formally $y_i \in \{-1, 1\}$. $\beta$ and $\beta_0$ are the coefficients of the hyperplane where $M = \frac{1}{||\beta||}$[3]. In the case where the data is not linearly separable, some points are allowed to be on the wrong side of the margin. These points are then given a slack variable $\varepsilon_i$ measured as the distance inside the margin, and new constraints are formulated:

$$\begin{gathered} y_i(x_i^T\beta + \beta_0) \geq M(1-\varepsilon_i),\ \forall i \\ \sum_{i=1}^{m} \varepsilon_i = \text{constant} \end{gathered} \tag{8}$$

Depending on the dimensionality of the feature space, and the type of kernel used, the geometry of the decision boundary varies. For a linear kernel the decision boundary will be a hyperplane, in 2D this is a straight line.

The kernel trick is used to find non-linear boundaries by using a transformation to add a dimension to the feature space, and then find a hyperplane in a new feature space [2]. The kernel describes the type of function used to create the transformation to form another dimension. A good example of how the kernel trick works would be using a tranformation (on a 2D feature space) of the form $z = x^2 + y^2$ to classify a circular decision boundary centered at 0. A 2-plane would then be found parallel to the $xy$-plane at some constant $z$ value, and would translate to a circular decision boundary.

When an SVM is used to classify more than two classes, there has to be some simplification to make it a binary classification problem. Often what is used is a "One-Vs-Rest" strategy, in which one class is treated as normal but the remaining classes are combined together.

## 3.2 Neural Network

A neural network is a machine learning technique that can be used for classifying data. The theory of computational neural networks was originally formulated in 1943 [5], and one of the first implementations was completed in 1957 [6]. However, due to the computational power needed, models were not commonly used until later on in the 20th century.

Neural networks are constructed with layers of nodes with directed links between nodes in subsequent layers as seen in Figure 2. The first layer has a node for each input dimension or feature, whereas the last layer has a node for each possible output class or label. Between the first and last layers there can be a number of hidden layers, with the number of nodes in each layer being varied depending on the complexity of the model. Every layer excluding the output layer will have a bias node that will always be active (i.e. have a value of 1). The reason for the bias node is obvious if you consider the input $x = 0$ and $y = 0$, because without the bias node, the outputs would all be 0. Each node from the input layer is connected to all the nodes in the subsequent layer with a forward directed and weighted link. This structure repeats in the same manner to the final output layer.

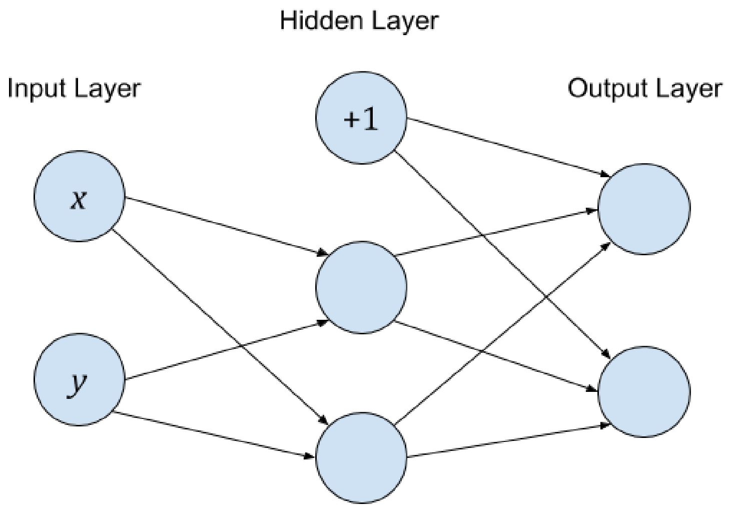


**Figure 2:** Figure showing a simple neural network. The network contains two input nodes, one hidden layer containing a bias node and two other hidden layer nodes, and an output layer with two output nodes.

To compute the activation $z$ of a node $j$ in layer $l$, the input nodes are multiplied by their relative weights and all summed

$$z_j^{(l)} = \sum_{i=1}^{N} \theta_i a_i^{(l-1)} \tag{9}$$

where $N$ is the number of nodes that have links connected towards node $j$. The value of this sum is then inputted into an activation function that will return a value between 0 and 1. One example of an activation function commonly used is the sigmoid function,

$$h(z) = \frac{1}{1+e^{-z}} \tag{10}$$

To train a neural network, the weights are randomly initialised and a training sample is inputted at the first layer of nodes. The values are then fed-forward through the network to give a reading at each output node between 0 and 1, which can be heuristically interpreted in a trained model as the probability that the input is in a given class.

At this point, the outputted values are compared to the data points given layer and the error of the predicted output is inputted into a cost function. A cost function that is commonly used is the square mean loss, given by

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} ||y_i - h(z_i^{output})||^2 \tag{11}$$

where $y_i$ is the target output vector for data point $i$ with $m$ training samples.

To train the neural network, the weights are altered so that the cost function is minimised. This is known as backpropagation. Using the derivative of the cost function, the weights are adjusted by using an optimisation method, for example, gradient descent to minimise the cost as a function of the weights. For this method we want to see how the cost $J$ changes with respect to a change in the weight $\theta$, this is given by

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial \theta} \tag{12}$$

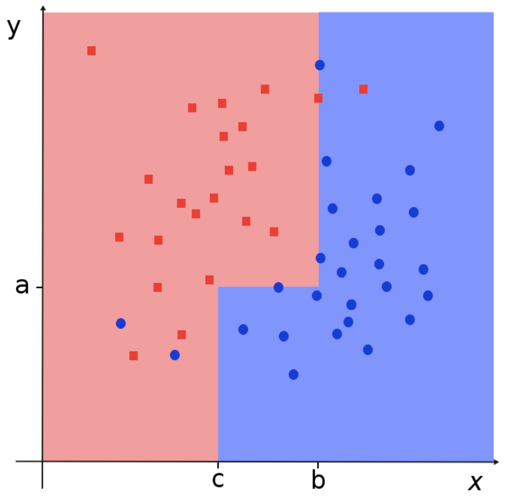
where $h$ is the sigmoid function, and $z$ is the activation of a node. This can be calculated for every weight in the network, and by applying a gradient descent algorithm, the weights are altered for each training sample. For the earlier layers in the network, the network can be followed backwards with the cost term giving an error for a specific activation node, the same method of using a gradient descent is then used to alter the weights.

## 3.3 Decision Tree

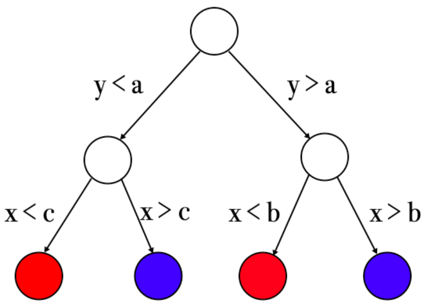
A decision tree classifier works by using simple decision rules to identify the class of some inputted data [7]. The feature space is partitioned repeatedly through these decision rules until the classes are grouped together or a *max depth* is reached, as shown in Figure 3.

The rules used to partition the space are chosen by computing the Gini impurity, which is calculated for a set of points with $J$ classes by

$$I_{Gini}(p, J) = 1 - \sum_{i=1}^{J} p_i^2 \tag{13}$$



**(a)** The decision boundary produced by the decision tree graph (b)



**(b)** The decision tree graph for inputs $x$ and $y$

**Figure 3:** Figure showing the decision tree graph (b) and respective boundary (a).

where $p_i$ is the fraction of items in class $i$. Gini inpurity is a measure of how mixed a subset of elements from the various classes are. To choose a rule, consider a parent set of $N$ points containing $J$ classes, and a rule that produces two child branches containing $J_0$ classes with $N_0$ data points, and $J_1$ classes with $N_1$ data points. We then measure the information gain $IG$ by looking at the parent Gini impurity, and the Gini impurity of the two child branches, and take the weighted mean

$$IG = N \cdot I_{Gini}(p, J) - (N_0 \cdot I_{Gini}(p, J_0) + N_1 \cdot I_{Gini}(p, J_1)) \tag{14}$$

Information gain is then measured for all possible splits, and the split that gives the highest information gain is chosen. This process is then recursively made until either a chosen *max depth* is reached or all training data has been separated.

# 4 Optimisation

When trying to find a 2D projection of the data that maximises the accuracy of a machine learning model, it is generally not efficient to use the Grand Tour algorithm. Instead, new transformations should be constructed without the use of the Grand Tour algorithm. This is because the Grand Tour's purpose is to give a sequence of rotation matrices that can be combined to create a smooth animation of the data from as many angles as possible. By constructing rotation matrices without the constraints of the Grand Tour, the time to find local and global maxima is decreased.

## 4.1 Mathematical Structure

Each rotation matrix $\boldsymbol{A}$ applied to a dataset, will have a corresponding value for an accuracy measure. In an $n$-dimensional feature space, all $n \times n$ rotation matrices will need to be elements of the special orthogonal group $SO(n)$. Therefore, new rotation matrices that are created will need a determinant of $+1$. This requirement puts constraints on the elements in the matrices:

$$\sum_k A_{ik} A_{jk} = \delta_{ij} \tag{15}$$

where $A_{ij}$ represents the element in the $i^{th}$ row and $j^{th}$ column of the rotation matrix $\boldsymbol{A}$, and $\delta_{ij}$ is the Kronecker delta.

Consider the transformation equation given by equation (2), given that the projection wanted is 2D, only two rows of the rotation matrix need to be quantified. To simplify the problem the following vectors are used:

$$\boldsymbol{a}_1 = \begin{bmatrix} A_{11} \\ A_{12} \\ \vdots \\ A_{1n} \end{bmatrix} \qquad \boldsymbol{a}_2 = \begin{bmatrix} A_{21} \\ A_{22} \\ \vdots \\ A_{2n} \end{bmatrix} \tag{16}$$

The constraints given by equation (15) can now be re-written as

$$\begin{aligned} \boldsymbol{a}_1 \cdot \boldsymbol{a}_1 &= 1 \\ \boldsymbol{a}_2 \cdot \boldsymbol{a}_2 &= 1 \end{aligned} \tag{17}$$

$$\boldsymbol{a}_1 \cdot \boldsymbol{a}_2 = 0 \tag{18}$$

Using the contraints from equation (17), $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ can be considered as vectors pointing to the edge of an $(n-1)$-sphere in the vector space $\mathbb{R}^n$. From the constraint given by equation (18) the two vectors are required to be orthogonal to each other as shown in Figure 4.

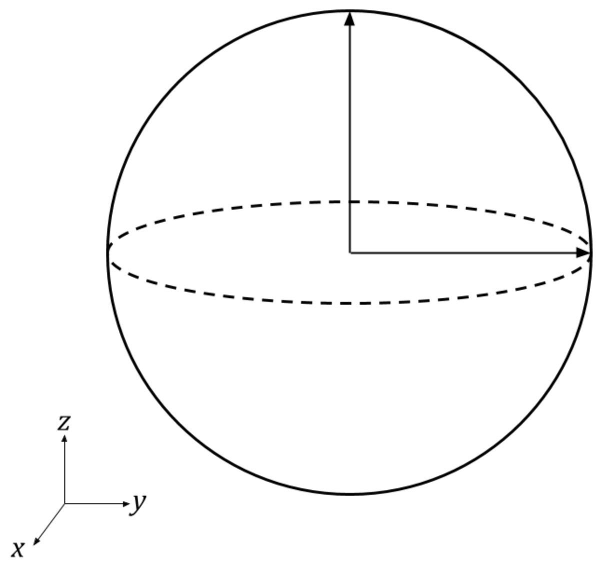


**Figure 4:** Figure showing the $n = 3$ case with a sphere. The second horizontal vector can point anywhere on the dotted "equator" line to satisfy the orthogonality condition.

Due to the vectors being contained on a $n$-sphere (from equations (17)), it is natural to transform the elements of the vectors into $n$-dimensional spherical polar coordinates.

$$\boldsymbol{a} = \begin{bmatrix} \cos(\theta_1) \\ \sin(\theta_1)\cos(\theta_2) \\ \sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \\ \vdots \\ \sin(\theta_1)\ldots\sin(\theta_{n-2})\cos(\theta_{n-1}) \\ \sin(\theta_1)\ldots\sin(\theta_{n-2})\sin(\theta_{n-1}) \end{bmatrix} \tag{19}$$

where $\theta_1 \in [0,\ \pi]$ and $\theta_i \in [0,\ 2\pi]\ \forall\ i > 1$.

## 4.2 Finding an Optimum Projection

To fufill the orthogonality constraint, two intial vectors are set to

$$\boldsymbol{a}'_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad \boldsymbol{a}'_2 = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \tag{20}$$

To optimise for machine learning accuracy, we now want a rotation $\boldsymbol{R}$ to be applied to both vectors (this ensures the orthogonality is conserved). To construct the rotation $\boldsymbol{R}$ that produces the maximum accuracy, we multiply all the rotation matrices that rotate the $i^{th}$-axis towards the $(i+1)^{th}$-axis in the form

$$\boldsymbol{R} = \boldsymbol{O}_{n-1} \times \boldsymbol{O}_{n-2} \times \ldots \times \boldsymbol{O}_1 \tag{21}$$

where $\boldsymbol{O}_i$ is defined by

$$\boldsymbol{O}_i(\phi_i) = \begin{pmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & \cos\phi_i & -\sin\phi_i & \cdots & 0 \\ 0 & \cdots & \sin\phi_i & \cos\phi_i & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix}. \tag{22}$$

The starting values for the angles used in the rotation matrices $\phi_i = \frac{\pi}{2}$ where $i = 1, \ldots, n-2$, and $\phi_{n-1}$ is varied between $[0, 2\pi]$ by dividing the interval into equal parts. The values taken by $\phi_{n-1}$ are each used to generate $\boldsymbol{R}$ in equation (19) and then transformed by transformation equations

$$\begin{aligned} \boldsymbol{R}\boldsymbol{a}'_1 &= \boldsymbol{a}_1 \\ \boldsymbol{R}\boldsymbol{a}'_2 &= \boldsymbol{a}_2 \end{aligned} \tag{23}$$

giving the first two rows of the transformation matrix used on the data to perform a projection. The new $x'$ and $y'$ values that the machine learning trains a model on is then given by

$$\begin{aligned} x' &= \boldsymbol{a}_1^T\, \boldsymbol{x} \\ y' &= \boldsymbol{a}_2^T\, \boldsymbol{x} \end{aligned} \tag{24}$$

where $\boldsymbol{x}$ is a data point in $n$-dimensional feature space and $\boldsymbol{a}_i^T$ represents the transpose of $\boldsymbol{a}_i$. The machine learning accuracy is then evaluated in each of the new 2D spaces. Whichever value for $\phi_{n-1}$ returns the best accuracy is kept. The process is then repeated for $\phi_{n-2}$ while keeping $\phi_{n-1}$ the same. When this step has been repeated for every value of $\phi_i$ (while keeping all $\phi_j$ the same, where $j > i$) we have now fully quantified $\boldsymbol{a}_1$. It is also true that each angle $\phi_i$ used in equation (22) are the spherical polar angles for $\boldsymbol{a}_1$ in the form shown in equation (19).

To optimise $\boldsymbol{a}_2$, we now alter each angle of the spherical polar coordinates of $\boldsymbol{a}'_2$ (excluding $\theta_1$ which is set to $\frac{\pi}{2}$ to ensure orthogonality), while keeping $\boldsymbol{R}$ the same. As with the $\phi_i$ angles, the angle $\theta_{n-1}$ is altered first by testing different values uniformly between $[0, 2\pi]$. With each of these values for $\boldsymbol{a}'_2$ the transformations in equations (23) and (24) are used to produce a 2D space to find the accuracy measure given by the machine learning model. Whichever angle produces the highest accuracy is kept, and the next angle in descending order is now altered. This is repeated until every angle has been optimised.

After these steps, an optimum projection has been found.

## 5 Results

After producing the Grand Tour projections for a data-set, the machine learning algorithms were applied at each time-step to give a sense of how "useful" a projection is. For the most part, the algorithm was applied to 1000 iterations of the Grand Tour.

### 5.1 Machine Learning Performance on the Grand Tour

The Grand Tour was performed for 1000 iterations. At each iteration, the three machine learning models were trained on the $x$ and $y$ values. The accuracy for each model was then found. An example of how the machine learning model accuracy varied over the Grand Tour is given below for the room data-set.

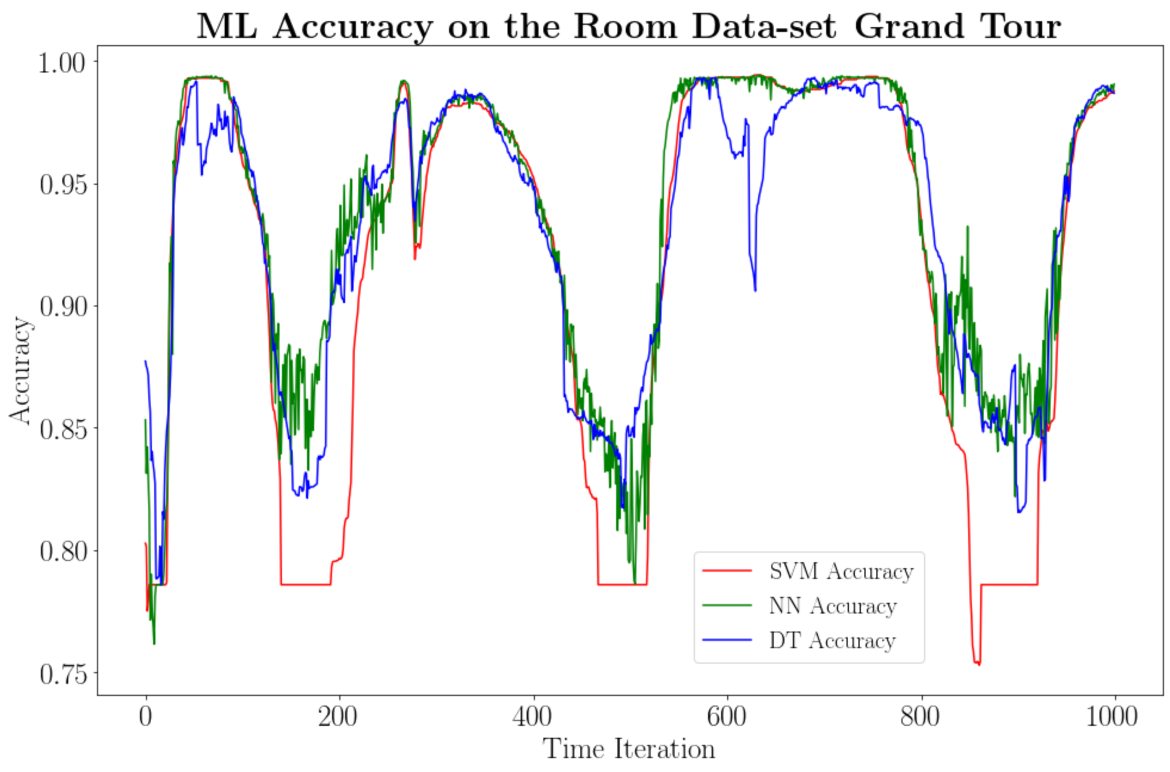
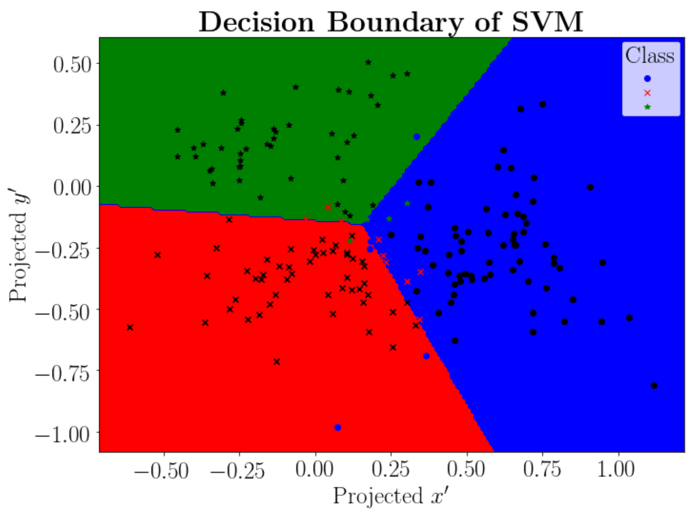
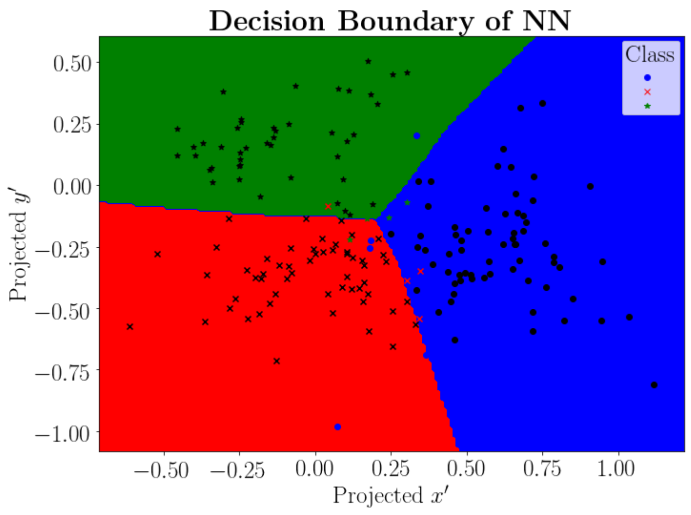


**Figure 5:** Figure showing the graph of relative performances of the machine learning techniques over the room data-set.

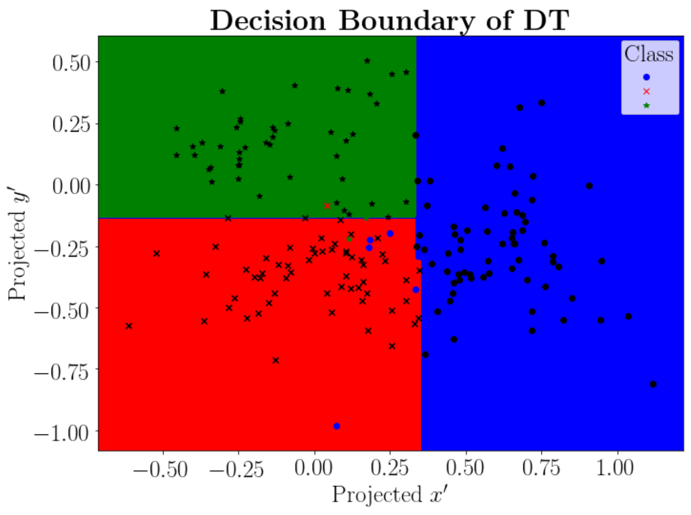
For a particular projection from the Grand Tour on the "Wine" data set, the decision boundaries are shown in Figure 6 for each algorithm. Note the relative similarity of the decision boundaries, even though the different algorithms have very different methods to construct the decision boundary. The accuracy on this particular projection in the Grand Tour was 92.7% for the SVM, the NN was 95.5%, and the DT was also 95.5%.



**(a)** The decision boundaries produced by the SVM.



**(b)** The decision boundaries produced by the NN.



**(c)** The decision boundaries produced by the DT

**Figure 6:** Figure showing the three different machine learning algorithms on the same projection of “Wine” data set. A black marker indicates a point that has been correctly identified. The legend in the top right corner shows the marker type for each class and the corresponding colour.

A comparison of all the machine learning model's performances is given in Table 1 below for some of the data-sets used.

**Table 1:** Table showing the machine learning performances of the data-sets used. Measure A is the maximum accuracy found in the 1000 iterations on the Grand Tour. Measure B is the number of iterations where that machine learning model performed better than the other two.

| | **Dataset** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Room | | Particle | | Bank | | Blood | | Spine | | Iris | |
| | A | B | A | B | A | B | A | B | A | B | A | B |
| **SVM** | 99.42% | 145 | 85.05% | 26 | 99.34% | 265 | 77.51% | 0 | 75.73% | 81 | 96% | 15 |
| **NN** | 99.42% | 472 | 86.19% | 152 | 100% | 529 | 81.52% | 637 | 80.58% | 329 | 98% | 534 |
| **DT** | 99.32% | 319 | 90.76% | 266 | 98.91% | 138 | 81.52% | 180 | 80.58% | 469 | 98% | 192 |

## 5.2 Optimisation Performance

The optimisation did very well in finding projections that maximised accuracy measurements. The accuracy found was better than the maximum Grand Tour accuracies over 1000 iterations.

**Table 2:** Table showing the performance of the decision tree accuracy (with *max depth* = 3) when used in conjunction with the optimisation algorithm which finds optimal 2D projections. The "$n$-D DT Accuracy" is how well the decision tree performed when acting in the $n$-dimensional feature space.

| Data Set: | Wine | Particle | Iris | Pulsar | Bank | Blood | Spine |
|---|---|---|---|---|---|---|---|
| $n$-D DT Accuracy | 99.4% | 83.5% | 90.6% | 97.8% | 98.5% | 76.3% | 82.2% |
| Optimisation Accuracy | 100% | 95.8% | 100% | 98.2% | 100% | 83.1% | 92.2% |

When applying both algorithms to the "Wine" data set, the Grand Tour had a maximum accuracy of 92.1% over 1000 time steps, whereas the optimisation algorithm found a projection of 100% accuracy. The projection produced from the optimisation algorithm is shown in Figure 7.

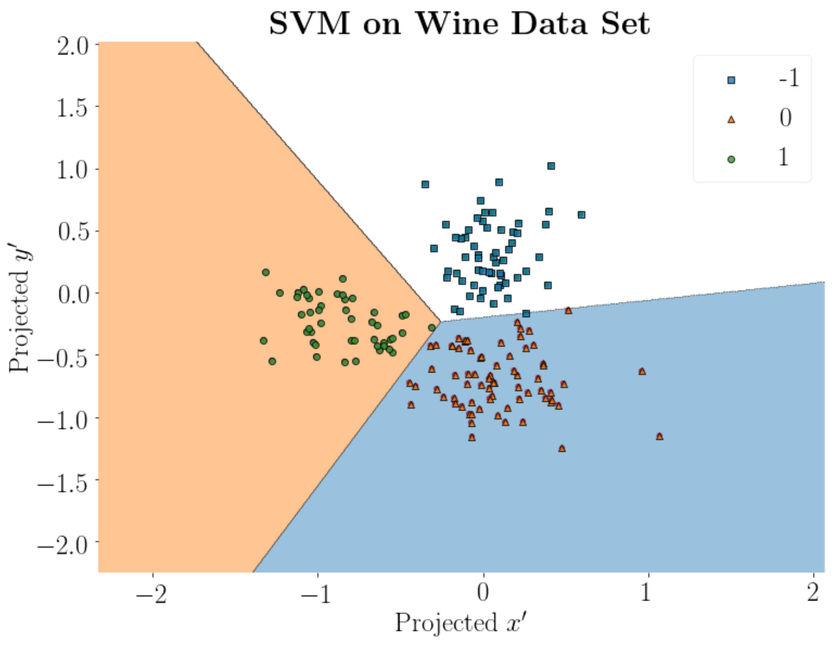


**Figure 7:** Figure showing 100% classification with a SVM on the "Wine" data set. With the legend showing the class value for each marker.

It should be noted that there is no guarentee that the optimisation algorithm will reach the global maximum. However, empirically the optimisation algorithm has found better 2D projections than the Grand Tour.

An example of how much each dimension from the feature space appears in the optimal projection for the "Particle" data set is shown in Figure 8. This indicates that variables 1, 2 and 6 could be removed while conserving the classification accuracy.

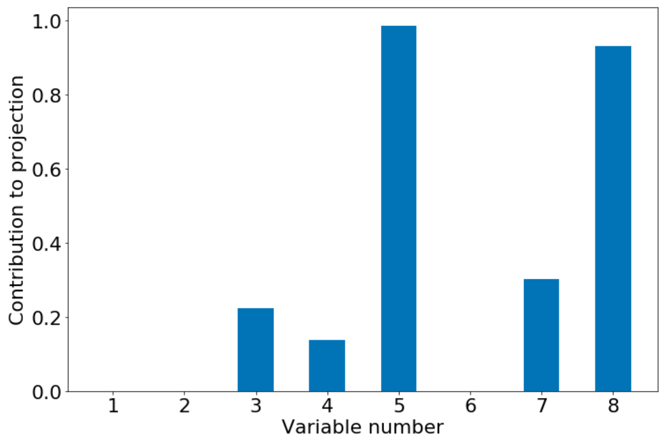


**Figure 8:** Figure showing the contributions of each variable in the maximum accuracy projection from the "Particle" data set. This was found by using a decision tree classifier and producing an accuracy of 95.8%.

# 6 Conclusion and Discussion

In this project, an implementation of the Grand Tour was successfully made in Python. The performance of three machine learning algorithms used on the projections was also looked at. Then a new algorithm was developed to find optimal projections to maximise accuracy of a multi-dimensional data set.

Using the Grand Tour and the optimisation algorithm for data mining has some interesting properties. Insights into dimensionality reduction can be made by finding the projections that maximise the various machine learning algorithms. Looking at the relevant rotation matrices, how much of each feature or dimension appears in the new convoluted basis vectors gives insight to which features are important in distinguishing for a certain class. In the future, it would be interesting to pursue a methodical implementation of these reductions, and how they would compare to current techniques. It would also be interesting to benchmark how many iterations of the Grand Tour it would take to reach an equivalent max accuracy projection found by the optimisation algorithm.

With the optimisation algorithm, there are some definite avenues for potential improvement. There is a vast amount of research into generic global optimisation problems, some of which will have techniques that can be used to speed up optimisation or reduce chances of hitting local maxima.

One suggestion was to look into if the Hilbert curve could be used to look at the data from many different unique projections, as it uniformly covers the space. In 3D, a vector from a Hilbert curve node to the centre of the data uniquely defines a plane. This is because the vector can be used to restrict one dimension and use two orthogonal unit vectors to create a basis for a subspace. However, when the feature space exceeds three dimensions there is a problem with this method. In a 4D feature space, two unique vectors are needed to define a unique plane. After using the vector defined from the Hilbert node to the centre of feature space, the subspace still needs to be reduced by one dimension to produce a 2D subspace.

Something that would be interesting to investigate further is to see if a new unsupervised learning clustering algorithm could be developed to group data into classes. By looking at how much each data point moves relative to another data point during the Grand Tour (travel matrix), could this give insight to classification groupings. After looking at the matrix colourmap of this variable in the "Wine" data set, three rectangular shaped structures can be seen in the ordered data. By using network community detection algorithms or some other clustering analysis, would this method work, and more importantly, how would it compare to current unsupervised learning algorithms.

## References

[1] D. Asimov. The Grand Tour: A Tool For Viewing Multidimensional Data. Technical report, Stanford University, 1985.

[2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*, 1992.

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[4] D. M. Hawkins. The Problem of Overfitting. *Journal of Chemical Information and Computer Sciences*, 2004.

[5] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 1943.

[6] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 1958.

[7] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology. *IEEE Transactions on Systems, Man and Cybernetics*, 1991.

[8] E. J. Wegman and J. L. Solka. On some mathematics for visualizing high dimensional data. *Sankhya*, pages 429–452, 2002.

## A The Grand Tour Code

```
import numpy as np

def getAlpha(d):
    """
    produces d(d-1)/2 array of position in grand tour
    """
    p = d*(d-1)/2
    primeList = []
    count = 1
    while len(primeList) < p:
        count += 1
        primeBool = False
        for i in range(2, count - 1):
            if count % i == 0:
                primeBool = True
        if primeBool == False:
            irrational = (np.sqrt(count)%1)
            primeList.append(irrational)
    primeList = np.asarray(primeList)
    primeList = primeList.dot(stepSize)
    """
    Irrational number generation using exponentials, not being used
    p = int(d*(d-1)/2)
    alpha = np.zeros(p) #alpha(t) parameters defining grand tour in G2,d
    for i in range(0,p):
        alpha[i] = (np.exp(i) % 1) * 2 * np.pi
    alpha = alpha.dot(0.001)
    """
    return primeList


def getAngles(alpha,d):
    """"""
    Inputs:
    alpha = 1xd(d-1)/2 array defining position on grand tour
    d = dimensions of data
    Outputs a dxd array of angles required for the transformation
    """
    theta = np.zeros((d,d));
    i = 0;
    k = 0;
    while i < d-1:
        j = i + 1;
        while j < d:
            theta[i][j] = alpha[k];
            j += 1;
            k += 1;
        i+= 1;
    return theta;

def RotationMatrix(i, j, d, theta):
    """
    Inputs:
    i = first indicie of rotating plane
    j = second indicie of rotating plane
    d = dimensions of data
    theta = dxd array of angle of rotation of rotating plane
    Outputs a rotating matrix to rotate plane of ixj plane by theta_ij
    """
    R = np.identity(d)
    R[i,i] = np.cos(theta)
    R[i,j] = -1*np.sin(theta)
    R[j,i] = np.sin(theta)
    R[j,j] = np.cos(theta)
    return R

def BetaFn(d, theta):
    """
    Inputs:
    d = dimensions of data
    theta = dxd array of angle of rotation ixj plane
    Outputs the full matrix transformation for all rotations
    """
    b = RotationMatrix(1, 2, d, theta[1,2])
    i = 1
    j = 2
    for i in range(d):
        for j in range(d):
            if j <= i:
                continue
            if i==1 and j==2:
                continue
            b = np.matmul(b, RotationMatrix(i, j, d, theta[i,j]))
    return b
```

```python
def GrandTour(data, nSteps):
    """
    Inputs:
    data = array of data points, dimensions x npoints
    Outputs a 3D array number of points x t x dimensions, where t
    the time step at that point in the tour
    """
    d = np.shape(data)[0] #dimensions of data
    nPoints = np.shape(data)[1] #number of data points
    #initialise 3d matrix to store transforemd data at each timestep
    tData = np.zeros((nSteps,d,nPoints))
    tBeta = np.zeros((nSteps,d,d))
    Alpha = getAlpha(d)
    for t in range(0, nSteps):
        alpha = Alpha.dot(t)
        theta = getAngles(alpha, d)
        b = BetaFn(d, theta)
        a = np.matmul(b, data)
        tData[t,:,:] = a
        tBeta[t,:,:] = b
    return tData, tBeta

tData, tBeta = GrandTour(data, nSteps)
```

## B Data Sets

### Wine

Variables:
13 variables
Classes:
3 classes defining region of wine origin

### Particle

Variables:
8 variables
Classes:
2 classes for signal and background

### Bank note authentication

Variables:
1. variance of Wavelet Transformed image (continuous)

2. skewness of Wavelet Transformed image (continuous)
3. curtosis of Wavelet Transformed image (continuous)
4. entropy of image (continuous)
5. class (integer)

Classes:
1 Fake
0 Real
https://archive.ics.uci.edu/ml/datasets/banknote+authentication

## Iris

Variables:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

Classes:
Iris-setosa
Iris-versicolor
Iris-virginica
https://archive.ics.uci.edu/ml/datasets/Iris

## Room Occupancy

Variables:
Temperature, in Celsius
Relative Humidity
Light, in Lux
CO2, in ppm
Humidity Ratio
Classes:
1 Occupied
0 Unoccupied
https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

## Pulsars

Variables:
1. Mean of the integrated profile.
2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.

8. Skewness of the DM-SNR curve.
Classes:
0 Negative
1 Positive
https://archive.ics.uci.edu/ml/datasets/HTRU2

**Blood**
Variables:
Recency (months)
Frequency (times)
Monetary (c.c. blood)
Time (months)
Classes:
0 Donated since March 2007
1 Not donated since March 2007
https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center

**Spine**
Variables:
pelvic incidence
pelvic tilt
lumbar lordosis angle
sacral slope
pelvic radius
grade of spondylolisthesis
Classes:
DH (Disk Hernia)
SL Spondylolisthesis
NO Normal
AB Abnormal
https://archive.ics.uci.edu/ml/datasets/Vertebral+Column