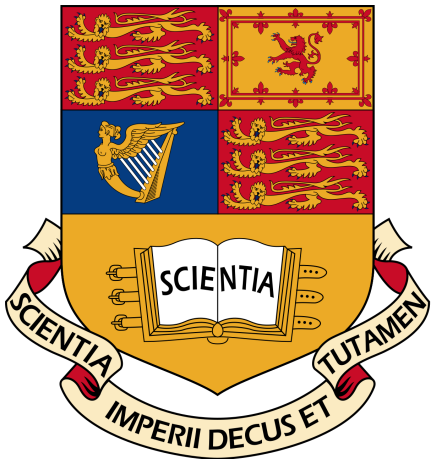
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2020



Project Title:	**Recognising the Sounds from Mars: Feature Identification from InSight**
Student:	**Rohan Tangri**
CID:	**01198239**
Course:	**4EM**
Project Supervisor:	**Professor Tom Pike**
Second Marker:	**Professor Richard Syms**

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, electronic copies of my final year project report to both Blackboard and the EEE coursework submission system.

I affirm that the two copies of the report are identical.

I affirm that I have provided explicit references for all material in my Final Report which is not authored by me and represented as my own work.

Abstract

This report follows the NASA InSight mission to Mars, which collects seismic data from the planet. Within the data, glitches known as 'donks' are required to be identified and eliminated from the time series. A multitude of different methods are investigated and comparisons presented to deduce the optimal algorithm for identifying such features with maximum accuracy. The technique consists of extracting the relevant labelled data, obtaining the main characteristics, and deriving important physical parameters that a machine learning algorithm can recognise well. The optimal performance is achieved with a Hidden Markov Model, achieving a perfect accuracy of 100% on cross-validated test sets, with encouraging results when applied to a whole Sol of data. A set of Google Collab notebooks are also provided as a sandbox environment and the techniques used have been consolidated into a set of python scripts that can be imported into future projects. Finally, the relationship between donk events and atmospheric temperature gradients is investigated and formalized using gamma distributions, reinforcing the link between high temperature change and donk occurrence.

Acknowledgements

First and foremost, I would like to thank Professor Tom Pike and for providing this opportunity, it has truly been an invaluable experience. Furthermore, I wish to express my deep gratitude to Alex Stott, who has been crucial in the development of this project and has always offered an ear to guide progress.

I would also like to thank my friends and family, both during and before university, without whom I would not be where I am today.

Contents

1 Introduction **6**
1.1 Motivation . . . 6
1.2 Report Structure . . . 6
1.3 List of Abbreviations . . . 7
1.4 Related Work . . . 7
1.5 Data Structure . . . 8

2 Background **9**
2.1 Preprocessing . . . 9
2.1.1 Short Time Fourier Transform . . . 9
2.1.2 Hilbert-Huang Transform . . . 9
2.1.3 Orthogonal Empirical Mode Decomposition . . . 11
2.1.4 Correlation Threshold . . . 12
2.1.5 Filterbanks . . . 13
2.1.6 Discrete Cosine Transform . . . 13
2.1.7 Delta Regressions . . . 13
2.2 Machine Learning Models . . . 14
2.2.1 Gaussian Mixture Model . . . 14
2.2.2 Hidden Markov Model . . . 16
2.2.3 Evolutionary Strategy . . . 18
2.2.4 Convolutional Neural Network . . . 19
2.3 Performance Metrics . . . 21
2.3.1 Binary Cross-Entropy . . . 21
2.3.2 Bayesian Information Criterion . . . 22
2.3.3 Stratified Cross Validation . . . 23
2.3.4 Precision and Recall . . . 23

3 Requirements Capture **24**

4 Analysis and Design **25**
4.1 High Level Diagram . . . 25
4.2 Data Extraction . . . 25
4.3 Visualization . . . 25
4.4 Preprocess . . . 26
4.5 Feature Extraction . . . 26
4.6 Training Models . . . 28
4.7 Benchmarking . . . 30
4.8 Validation . . . 30
4.9 Temperature Relationship . . . 30

5 Implementation **31**
5.1 Environment . . . 31
5.1.1 Resources . . . 31
5.1.2 Library List . . . 31
5.2 Data Extraction . . . 32
5.3 Visualization . . . 32
5.4 Preprocessing . . . 35
5.4.1 Binary Search . . . 35
5.4.2 Framing . . . 35
5.4.3 Negative Data Extraction . . . 35
5.4.4 Outliers . . . 36
5.5 Feature Extraction . . . 36
5.5.1 Time Series . . . 38
5.5.2 Periodograms . . . 38

5.5.3 Energy Ratios . . . 38
5.5.4 Spectrograms . . . 39
5.5.5 Hilbert Spectra . . . 40
5.5.6 Filterbank Coefficients . . . 41
5.5.7 Cepstral Coefficients . . . 41
5.5.8 Deltas . . . 42
5.6 Training Models . . . 42
5.6.1 Traditional Algorithms . . . 42
5.6.2 Convolutional Neural Network . . . 43
5.6.3 Gaussian Mixture Model . . . 44
5.6.4 Hidden Markov Model . . . 45
5.6.5 Evolutionary Strategy . . . 46
5.7 Benchmarking . . . 47
5.8 Validation . . . 48
5.9 Temperature Relationship . . . 48

6 Benchmark **50**
6.1 Traditional Algorithms . . . 50
6.1.1 Time Series . . . 50
6.1.2 Periodograms . . . 50
6.1.3 Energy Ratios . . . 51
6.1.4 STFT Filterbank Coefficients . . . 51
6.1.5 STFT Cepstral Coefficients . . . 52
6.1.6 HHT Filterbank Coefficients . . . 52
6.1.7 HHT Cepstral Coefficients . . . 53
6.2 Convolutional Neural Network . . . 53
6.3 Hidden Markov Model . . . 54
6.4 Evaluation . . . 55

7 Validation **56**
7.1 Naive Splitting . . . 56
7.1.1 K-Nearest Neighbours . . . 56
7.1.2 Hidden Markov Model . . . 58
7.2 Rolling Window Splitting . . . 61
7.2.1 Hidden Markov Model . . . 61
7.3 Post-Processing . . . 62

8 Temperature Relationship **66**

9 Conclusions and Further Work **68**

Bibliography **69**

Appendices **71**

A Negative Training Data **71**

B HMM Generated Donk **71**

C More Validation Examples - Sol 98 **71**

D Donk-Temperature Visualization **76**

E GitHub Link **77**

1 Introduction

1.1 Motivation

On November 26 2018, the NASA Atlas V-4-1 landed on Mars to stay for 728 days (709 Martian Sols). The InSight mission aims to study the interior of the planet for the first time in order to answer key questions about the early formation of rocky planets as well as the level of tectonic activity and meteorite impacts on the planet. To do this, the Mars vehicle has a multitude of sensors that send data back to Earth to be analyzed [1].

A 'donk' is a tapping sound perceived as noise that arises when parts of the rover itself cool/heat and expand/contract, sometimes resulting in two parts suddenly slipping across each other as the friction between them is overwhelmed. The sound is identical to a car engine cooling when it is turned off. Donks appear frequently in data, and ideally should be removed to improve the signal to noise ratio (SNR) and make analysis of other seismic features easier. The first step in this process is recognition of the donks in the time series. In order to do this, a machine learning driven approach is suggested. There are a few key challenges in accomplishing this task. Firstly, the time series is nonstationary and nonlinear. Secondly, the quantity of labelled data available is small. Finally, the donk features can appear to be very similar to other features (e.g. wind) and the framing is not consistent.

From what is understood, it is expected that donk events are caused by large changes in atmospheric temperature. This relationship is also investigated further for more appropriate analysis and understanding of donks in the future.

1.2 Report Structure

1. The **Background** section illustrates **what** tools are implemented to accomplish the goals of the project.
2. The **Requirements Capture** section illustrates the exact deliverable requirements to be met by the project.
3. The **Analysis and Design** section illustrates **why** tactics are used in the way they are to accomplish the goals set out in Requirements Capture whilst serving as a high level overview of the project.
4. The **Implementation** section illustrates **how** tactics are implemented in software with in depth discussion of parameter tuning.
5. The **Benchmark** section includes the raw benchmark results of the different algorithms used as well as an analysis of those results with rationale for the differences observed.
6. The **Validation** section includes analysis of the best models from the Benchmark section applied to real life Sols of data.
7. The **Temperature Relationship** section investigates the relationship between atmospheric temperature change and donk event occurrence.
8. The **Conclusions and Further Work** section compares the final deliverable to the Requirements Capture conditions and suggests how future improvements might be made.

1.3 List of Abbreviations

Data:

1. SP - Short Period
2. VBB - Very Broad Band

Preprocessing Techniques:

1. STFT - Short Time Fourier Transform
2. HHT - Hilbert-Huang Transform
3. FC - Filterbank Coefficient
4. CC - Cepstral Coefficient
5. DFT - Discrete Fourier Transform
6. FFT - Fast Fourier Transform
7. DCT - Discrete Cosine Transform
8. IMF - Intrinsic Mode Function
9. EMD - Empirical Mode Decomposition
10. OEMD - Orthogonal Empirical Mode Decomposition

Machine learning models and measures:

1. LR - Logistic Regression
2. BNB - Bernoulli Naive Bayes
3. SVM - Support Vector Machine
4. RF - Random Forests
5. KNN - K-Nearest Neighbours
6. GMM - Gaussian Mixture Model
7. CNN - Convolutional Neural Network
8. HMM - Hidden Markov Model
9. BIC - Bayesian Information Criterion

1.4 Related Work

The topic of automatic seismic signal recognition is a common one, with multiple techniques proposed to detect seismic signals on Earth. Furthermore, the recent industry push towards natural language processing (NLP) also provides some interesting parallels with similar signals being analyzed; the main difference being the bandwidth of those signals.

A series of seismic classification papers are found. The focus of these papers are for Mt. Merapi in Indonesia [2], Deception Island in Antarctica [3] and Nevado del Huila in Colombia [4]. Each paper demonstrates the use of sequential features to be identified using a hidden Markov model. This research reinforces the competence of such a technique, with each paper demonstrating results of around 90% classification accuracy.

There has also been research into which features best capture seismic signal information. There is an investigation into the Hilbert-Huang transform and its application in seismic signal processing which concludes that the transform can be used to remove noise from corrupt signals [5].

Finally, there has been extensive research into NLP techniques [6]. These are generally more modern and introduce the use of convolutional and recurrent neural networks to recognize speech as well as an accepted framework for feature extraction based on frequency cepstra.

1.5 Data Structure

The raw data itself is split up for each Sol. Furthermore, in each Sol there are two sets, SP and VBB. The SP data measures signals above 10Hz whilst the VBB data measures signals below 10Hz with interpolation applied such that the sampling frequency is uniform across both sets, $fs = 100Hz$. Finally, there are three axes, each detecting different directions of signals, given by Z, N and E.

2 Background

This chapter is split into three main sections, one on preprocessing, one on machine learning algorithms and one on evaluation metrics. An assumption is made that the reader is familiar with basic probability and stochastic properties (such as stationarity and Bayesian theory) as well as some of the more fundamental machine learning algorithms (such as regression).

2.1 Preprocessing

Here is an introduction for some of the algorithms used to obtain relevant features for the machine learning models. In particular, techniques for frequency representation, feature extraction and feature reduction are required.

2.1.1 Short Time Fourier Transform

One method of obtaining a time-frequency representation of data is to use the STFT to obtain a spectrogram. The frequency range is limited by the sampling frequency as given by Nyquist, $f_{max} = f_s/2$.

There are a few considerations when implementing the STFT. Firstly, the window width has to be decided. The data is inherently nonstationary; although, if a slice is chosen that is short enough, it can be assumed to be stationary. However, the uncertainty principle also informs the decision on frame width. This states that there is a tradeoff between time and frequency resolution. That is, for a narrower window width, the time resolution increases at the cost of the frequency resolution and vice versa. Typically, for speech and signal processing, the overlap between frames is also taken to be 50% of the frame width itself. Furthermore, values should ideally be multiples of two to take full advantage of the divide and conquer nature of the Cooley-Tukey FFT algorithm [7]. Another decision to make is the type of window to be used. Of course, a perfect low-pass filter would be ideal; however this is not feasible since it is non-causal. Instead, a Hanning window is a good approximation since it smoothes out the ends of the data frame, reducing spectral leakage. A Hanning window has the following form [8]:

$$w(n) = 0.5 - 0.5\cos(\frac{2\pi n}{N}) \quad \text{for } 0 \leq n \leq N-1 \tag{1}$$

$N =$ window length

2.1.2 Hilbert-Huang Transform

The HHT [9] is an adaptive, data driven alternative method to the STFT to generate time frequency plots. The HHT transform works by getting the instantaneous frequencies of a signal through time. In order to define instantaneous frequency, first analytic signals must be obtained. Suppose there exists a real valued time series, $x(t)$. The analytic signal can be extracted as:

$$x_a(t) = x(t) + j\widehat{x(t)} \tag{2}$$

Where $\widehat{x(t)}$ is the Hilbert transform of $x(t)$, defined as:

$$\widehat{x(t)} = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{x(\tau)}{t-\tau}d\tau \tag{3}$$

An analytical signal as in (2) recognises that the negative frequency components of a signal can be discarded with no loss of information due to the Hermitian symmetry of the spectrum of a real-valued function. Instead, one must deal with a complex-valued function. The magnitude of such a complex signal is the instantaneous amplitude and the argument is the instantaneous phase:

$$a_i(t) = |x_a(t)| \tag{4}$$
$$\phi_i(t) = arg[x_a(t)] = \arctan[x_a(t)] \tag{5}$$

Finally, the derivative of the unwrapped instantaneous phase gives the instantaneous angular frequency, so instantaneous frequency is defined as:

$$\begin{aligned} f_i(t) &= \frac{1}{2\pi}\frac{d}{dt}\phi_i(t) \\ &= \frac{1}{2\pi}\frac{d}{dt}\arctan[x_a(t)] \end{aligned} \tag{6}$$

The HHT employs an iterative algorithm known as the EMD which breaks a signal down into its IMFs which are characterized by two fundamental properties:

1. The number of extrema and the number of zero crossings differ, at most, by one.
2. The mean value of the envelopes defined by the local maxima and local minima is zero.

These properties allow the IMFs to have well-defined Hilbert transforms and consequently, meaningful instantaneous frequencies that can be plotted on a Hilbert spectrum. The EMD algorithm itself is as follows:

1. Find all local extrema in the signal.
2. Join all the local maxima with a cubic spline, creating an upper envelope. Repeat for local minima to create a lower envelope.
3. Calculate the mean of the envelopes.
4. Subtract the mean from the original signals.

5. Repeat steps 1-4 until the result is an IMF.
6. Subtract this IMF from the original signal.
7. Repeat steps 1-6 until there are no more IMFs.

The main problem with the HHT is embedded in the EMD algorithm: mode mixing [10]. Since EMD locally pulls out the highest frequency component as the current IMF, mode mixing in a signal means that the frequency tracked by a particular IMF will jump as an intermittent component begins or ends. This can result in an IMF having components of different frequencies. Suppose there exists an amplitude modulated sine wave:

$$x(t) = a_1 \sin(2\pi f_1 t) + a_2 \sin(2\pi f_2 t) \tag{7}$$

There will be significant mode mixing when $0.5 < f_1/f_2 < 2$, such that the algorithm identifies the amplitude modulated tone as a single component. Calling this a problem is a bit misleading, since in many scenarios the modulated interpretation may be the most appropriate and makes the raw EMD algorithm more likely to find components that are linked to the underlying process generating the signal.

2.1.3 Orthogonal Empirical Mode Decomposition

The EMD algorithm can be used as a feature extraction tool through the evaluation of the resultant IMFs. Instantaneous frequencies from such IMFs are only meaningful if they are mono-component orthogonal signals. Therefore, the orthogonality of the IMFs should be maximized. The orthogonal index, IO_T, can be used to measure the overall degree of orthogonality between all of the IMFs [11].

$$IO_T = \sum_{j=1}^{n} \sum_{k=1, k\neq j}^{n} \frac{\int_0^T c_j(t) c_k(t) dt}{\int_0^T x^2(t) dt} \tag{8}$$

$x(t) =$ original signal
$c_j(t) = j_{th}$ IMF
$c_k(t) = k_{th}$ IMF
$n =$ number of IMFs

Ideally, this number should be zero, although due to the way in which computers handle floating point numbers, two IMFs can be considered exactly orthogonal when the orthogonal factor is below a magnitude of 10^{-16}.

To improve orthogonality and reduce energy leakage, the backward orthogonalization algorithm can be applied, which makes use of the Gram-Schmidt method. In this, the first basic orthogonal function is taken as the last original IMF [11]:

$$d_1(t) = c_n(t) \tag{9}$$

From here, an iterative algorithm is applied:

$$\begin{aligned} d_{j+1}(t) &= c_{n-j}(t) - \sum_{i=1}^{j} \beta_{j+1,i} d_i(t) \\ \beta_{j+1,i} &= \frac{\int_0^T c_{n-j}(t) d_i(t) dt}{\int_0^T d_i^2(t) dt} \end{aligned} \tag{10}$$

However, there is a cost of applying such an algorithm. Namely, the output signals of such an algorithm directly influence the IMFs obtained by the EMD, so the resultant signals may not strictly be IMFs anymore. Fortunately, the orthogonal components generated by the backward algorithm are almost the same as the original IMF components except for in scale. Therefore; whilst strictly speaking the orthogonal components do not satisfy the first condition of IMFs, they can still be considered roughly accurate since the changes to the Hilbert spectrum are very small.

If further rigidity on the IMF conditions are required, a post-processing method using the EMD is proposed [12]. The result of the iterative algorithm will provide strict IMF components.

1. Suppose again that $d_i(t)$ represents the i_{th} orthogonal EMD component. Extract one IMF from $d_i(t)$ using the EMD algorithm. The IMF obtained is part of the final set, $c_i(t)$.
2. Sum the residue of the EMD, $r_i(t)$, with the next orthogonal component, $d_{i+1}(t)$.
3. Apply the EMD to the resultant sum, $r_i(t) + d_{i+1}(t)$ to obtain the next component $c_{i+1}(t)$.
4. Repeat until all the orthogonal features are processed.

2.1.4 Correlation Threshold

One of the advantages of the EMD is that the resultant IMFs can be interpreted as physically meaningful features. However, a criterion must be established to pick out the most relevant IMFs with the most useful information. The idea behind this selection process is that the original signal itself is useful; that is, the SNR is sufficiently high. Therefore, useful IMFs will be the ones that are most highly correlated with the original signal. Numerical experiments have been run in order to establish a reliable framework for retaining only relevant IMFs, the results of which give the following equation [13]:

$$\rho_{TH} = \frac{max(\rho_i)}{10 * max(\rho_i) - 3}, \quad i = 1, 2, ..., n \tag{11}$$

$\rho_{TH} =$ correlation threshold
$\rho_i =$ correlation coefficient of i_{th} IMF with the original signal
$n =$ number of IMFs

2.1.5 Filterbanks

Filterbanks are frequently used to reduce the feature space over the frequency domain; essentially capturing rough frequency energies. In seismic signal processing, these are equally spaced overlapping triangular filters, which can be modelled as follows [2; 3; 14]:

$$H(k) = \begin{cases} \frac{k-f(m-1)}{f(m)-f(m-1)} & \text{if } f(m-1) \leq k < f(m) \\ 1 & \text{if } k = f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)} & \text{if } f(m) < k \leq f(m+1) \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Where $f(m)$ is the frequency around which the filter is based and $f(m+1)$ and $f(m-1)$ are deduced based on the number of filters to be generated in a particular frequency range.

2.1.6 Discrete Cosine Transform

The DCT is a linear transformation that converts data into the summation of a series of cosine waves of differing frequencies and amplitudes [15].

Mathematically, the DCT of a discrete signal $x[n]$ of length N is given by:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cos(\frac{2\pi(2n+1)k}{4N}), \quad for\ k = 0 : N-1 \tag{13}$$

More intuitively, the reversed signal is appended to the original and zero interpolation is applied (a zero is inserted between each pair of samples). Then, the DFT of the $4N$ samples is taken, saving only the first N values as the result (since the output of the DFT is real, symmetric and anti-periodic).

This is useful since it is computationally cheap and produces uncorrelated features since the cosine basis functions are orthogonal [2; 3].

2.1.7 Delta Regressions

Delta regression is a differential technique that allows for the encoding of temporal information directly. Given an initial feature sequence [6]:

$$\Delta_t = \frac{\sum_{i=1}^{n} c_i * (v_{t+i} - v_{t-i})}{2\sum_{i=1}^{n} c_i^2} \tag{14}$$

$v_t =$ feature at time t
$c_i =$ regression coefficients

Where generally, $n = 2$. Delta-delta parameters (Δ_t^2) are obtained in the same way using the delta features, giving a new feature vector:

$$v_t^+ = [v_t, \Delta_t, \Delta_t^2]^T \tag{15}$$

The deltas can be thought of as velocity and the delta-deltas as acceleration of a feature sequence.

2.2 Machine Learning Models

Here, various machine learning algorithms will be introduced as methods to gain a deeper understanding of the data and as direct classification methods. In the analysis, more basic models are also used as a benchmark (e.g. Logistic Regression, Random Forests, Naive Bayes), only the more complex, less well known ideas are covered here.

2.2.1 Gaussian Mixture Model

A GMM is an unsupervised algorithm based on density estimation [16]. The goal of such a model is to estimate the distribution of a set of given data as a mixture of different Gaussian distributions.

Take the following notation:

$K =$ number of Gaussian distributions used
$\mu_k =$ mean of k_{th} Gaussian
$\Sigma_k =$ covariance matrix of k_{th} Gaussian
$\pi_k =$ mixing probability of k_{th} Gaussian

The overall GMM is still a probabilistic distribution and as such the sum of the areas of the mixtures must be 1. This is defined by the mixing probabilities:

$$\sum_{k=1}^{K} \pi_k = 1 \tag{16}$$

For each Gaussian used, the mean and covariance matrices must be optimized relative to the data. A maximum likelihood estimation (MLE) is done. A single multivariate Gaussian is given by:

$$\mathcal{N}(\boldsymbol{x}; \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \mu)^T \Sigma^{-1} (\boldsymbol{x} - \mu)\right) \tag{17}$$

Where $\boldsymbol{x}$ is the set of data points, D is the dimensionality of each data point and μ and Σ need be optimized such that $N(\boldsymbol{x}; \mu, \Sigma)$ is maximized. By taking the logarithm, the maxima of the function is easily obtained via differentiation. However, for a GMM there are multiple Gaussians.

Define the conditional probability of a particular Gaussian, k, given a data point $\boldsymbol{x}_n$:

$$p(z_{nk} = 1|\boldsymbol{x}_n) \tag{18}$$

The variable z is a latent variable that is binary; it is 1 when $\boldsymbol{x}$ comes from Gaussian k and zero otherwise. Likewise:

$$\pi_k = p(z_k = 1) \tag{19}$$

So the probability of observing a point that comes from Gaussian k is equivalent to the mixing coefficient for that Gaussian. Therefore, the bigger the Gaussian, the higher the probability an event comes from that distribution.

Now, given the set of all possible latent variables, $\boldsymbol{z}$, the conditional probability of observing data given that it comes from Gaussian k can be defined:

$$p(\boldsymbol{x}_n|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k)^{z_k} \tag{20}$$

Marginalization can be used to obtain the probability function of the GMM, $p(\boldsymbol{x}_n)$:

$$\begin{aligned} p(\boldsymbol{x}_n) &= \sum_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{z})p(\boldsymbol{z}) \\ &= \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k) \end{aligned} \tag{21}$$

Again, the likelihood is the joint probability of all observations $\boldsymbol{x}_n$ which can be defined:

$$\begin{aligned} p(\boldsymbol{x}) &= \prod_{n=1}^{N} p(\boldsymbol{x}_n) \\ &= \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k) \end{aligned} \tag{22}$$

This expression is maximized with respect to μ_k, π_k and Σ_k to obtain these parameter values for the GMM probability functions. This is done iteratively using an expectation maximization algorithm since the log differentiation is difficult. Define the parameter $\gamma(z_{nk}) = p(z_k = 1|\boldsymbol{x}_n)$, then from Bayes rule:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n; \mu_j, \Sigma_j)} \tag{23}$$

Iterative parameter estimation at each step is as follows:

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N} \tag{24}$$

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk}} \tag{25}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \mu_k)(\boldsymbol{x}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})} \tag{26}$$

2.2.2 Hidden Markov Model

Strictly speaking, the HMM is an unsupervised machine learning method based on Markov chains. A Markov chain itself is a state machine where the the distribution of the next state, X_{n+1}, obeys the rule [17]:

$$P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, ..., X_0 = i_0) = P(X_{n+1} = j | X_n = i) \tag{27}$$

That is, the next state of the machine is only dependent on the current state, and previous states are irrelevant. Taking this further, the HMM represents a Markov chain where the states are unknown [18]. However, the chain is linked to some observable events which can then be used to gain information on the likelihood of being in a particular state. That is, a Markov chain can be modelled based on a sequence of observable events.

Consider the following notation:
$T =$ length of the observation sequence
$N =$ number of states in the model
$M =$ number of observation symbols
$A =$ state transition probability matrix
$B =$ observation probability matrix
$\pi =$ initial state distribution
$O =$ observation sequence
$\lambda = (A, B, \pi) =$ Markov chain
$b_j(k) =$ emmission probability of observation k at state j
$a_{i,j} =$ state transition probability from i to j

Consider a generic state sequence $X = \{x_0, x_1, ..., x_{T-1}\}$ and observables sequence $O = \{O_1, O_2, ..., O_{T-1}\}$. The probability of the state sequence with a sequence of observables is given by:

$$P(X, O) = \pi_{x_0} b_{x_0}(O_0) * a_{x_0,x_1} b_{x_1}(O_1) * ... * a_{x_{T-2},x_{T-1}} b_{x_{T-1}}(O_{T-1}) \tag{28}$$

By brute force, if the probabilities of all state sequences of length T are calculated, then the sequence with the highest probability can be found as the most likely state sequence of the hidden Markov chain.

Now, define $\alpha_t(i)$ as the probability of the partial observation sequence up to time t where the underlying Markov process is in state i at time t, we can determine the probability of a sequence of observations given a particular model:

$$\alpha_t(i) = P(O_0, O_1, ..., O_t, x_t = i|\lambda) \tag{29}$$

Then, applying some Bayesian theory:

$$\begin{aligned} P(O|\lambda) &= \sum_{i=0}^{N-1} \alpha_{T-1}(i) \\ &= \sum_{X} P(O|X, \lambda)P(X|\lambda) \\ &= \sum_{X} \pi_{x_0} b_{x_0}(O_0) * a_{x_0,x_1} b_{x_1}(O_1) * ... * a_{x_{T-2},x_{T-1}} b_{x_{T-1}}(O_{T-1}) \end{aligned} \tag{30}$$

It is useful to obtain $P(O|\lambda)$ as a function of $\alpha_t(i)$ since $\alpha_t(i)$ can be computed recursively and reduces the complexity to $O(N^2T)$ from $O(TN^T)$ using a naive approach. This is called the forward algorithm or α-pass.

The β-pass algorithm can also be defined as the probability of the partial observation sequence from time t where the underlying Markov process is in state i at time t:

$$\beta_t(i) = P(O_{t+1}, Ot + 2, ..., O_{T-1}, x_t = i|x_t = i, \lambda) \tag{31}$$

Now, define $\gamma_t(i)$ as the probability of being in state i at time t given an observation sequence and Markov chain parameters:

$$\gamma_t(i) = P(x_t = i|O, \lambda) \tag{32}$$

Since $\alpha_t(i)$ measures the relevant probability up to time t and $\beta_t(i)$ measures the relevant probability after time t:

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \tag{33}$$

Furthermore, a di-gamma, $\gamma_t(i, j)$, is defined as the probability of being in state i at time t and transitioning to state j at time $t + 1$:

$$\begin{aligned} \gamma_t(i, j) &= P(x_t = i, x_{t+1} = j|O, \lambda) \\ &= \frac{\alpha_t(i) * a_{i,j} * b_j(O_{t+1}) * \beta_{t+1}(j)}{P(O|\lambda)} \end{aligned} \tag{34}$$

$\gamma_t(i,j)$ and $\gamma_t(i)$ are related by:

$$\gamma_t(i) = \sum_{j=0}^{N-1} \gamma_t(i,j) \tag{35}$$

Using this information, an estimation of the best Markov model can be determined as follows:

$$\pi_i = \gamma_0(i) \tag{36}$$

$$a_{i,j} = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)} \tag{37}$$

$$b_j(k) = \frac{\sum_{t=0}^{T-2} \gamma_t(i,j)}{\sum_{t=0}^{T-2} \gamma_t(i)} \tag{38}$$

Thereby, the parameters of the model have been updated. A summary of the data fitting process is as below:

1. Initialize the model randomly such that $\pi_i \approx \frac{1}{N}$ and $b_j(k) \approx \frac{1}{M}$.
2. Compute $\alpha_t(i)$, $\beta_t(i)$, $\gamma_t(i,j)$ and $\gamma_t(i)$
3. Re-estimate the model λ.
4. If $P(O|\lambda)$ increases, go to 2.

Overall, this algorithm is known as the Baum-Welch method which is essentially an expectation maximization algorithm. Therefore, it is guaranteed to converge at least to a local maxima. However, there is also another algorithm available, known as the Viterbi algorithm, which is a dynamic algorithm with complexity $O(TN^2)$ at the cost of reduced accuracy by estimating the forward algorithm with the most likely state as opposed to summing over all states.

2.2.3 Evolutionary Strategy

Evolutionary strategy is a reinforcement learning algorithm which takes inspiration from evolution in nature [19]. As a reinforcement algorithm, this is essentially and optimization method for black box functions where traditional Newton or gradient methods are not suitable. Therefore, it can be used to optimize the model topologies of other algorithms (e.g. the number of states in the HMM).

Suppose the function $f(x)$ should be maximized, the function $f(x)$ then represents a reward function. A few hyper-parameters first need to be established. First, the population size should be chosen, N, being the number of guesses made at each iteration. A larger N will allow for a more complete search space, although this comes at the cost of increased computational complexity. Then, a learning rate must be chosen, α. Similar to other optimization methods, this defines the

shift in the set of solutions at each iteration. Therefore, a large α may diverge from the solution and a large α that converges will have a larger jitter around the optimal solution. This should be chosen to be small, although the smaller this gets the more iterations will be required to reach the solution. Finally, a noise variance, σ^2, should be chosen. This defines the spread of guesses at each iteration. Therefore, a larger σ^2 will allow for a larger search space and could push the algorithm out of a local maxima towards a global maxima. However, this comes at the cost of a less precise ascent direction. Based on these parameters, the algorithm is described below:

1. Randomly choose a single guess, x_μ^t.
2. Take the value of the guess as the mean for a solution generator. That is, $X \sim \mathcal{N}(x_\mu^t, \sigma^2)$. Take N points from the distribution X as the population of solutions to test.
3. Apply each solution in the population to the reward function to get a set of scores, A.
4. Update x_μ^t as in (25).
5. Repeat 2-4 indefinitely until a satisfactory optimal result is obtained.

$$\begin{aligned} x_\mu^{t+1} &= x_\mu^t + \frac{\alpha}{N\sigma}\sum_{i=1}^{N} A_i\epsilon_i \\ \epsilon_1, \epsilon_1, ..., \epsilon_N &\sim \mathcal{N}(0, I) \end{aligned} \tag{39}$$

2.2.4 Convolutional Neural Network

A CNN is a supervised deep learning algorithm. In order to understand this, first kernal convolution image processing must be introduced. As an overview, it consists of sliding a kernel over an image, applying a dot product at each iteration, to create an image with new pixel values. A kernel is essentially a grid of numbers. Depending on the kernel weights, a number of transformations can be applied to an image, including edge detection, image smoothing and sharpening. This is described mathematically below given that the top left of a 2D object is given by the origin with increasing direction both laterally to the right and vertically downwards [20]:

$$\begin{aligned} z[m,n] &= (x * w)[m,n] \\ &= \sum_{i}^{C}\sum_{j}^{R} w[i,j]x[m+i, n+j] \end{aligned} \tag{40}$$

$x[k,l]$ = image input at $[k,l]$
$w[i,j]$ = kernel weight at $[j,k]$
$z[m,n]$ = image output at $[m,n]$
R = number of kernel rows
C = number of kernel columns

At a high level, a CNN learns the best set of kernel weights that extract the most useful features of an image which can then be passed to a multi-layer perceptron (MLP) for image classification.

Each layer in such a network is made up of a convolutional layer and a pooling layer. At each layer K kernels are created with different weights and each is used to do a kernel convolution with the image, generating a matrix of K images. Each of the K output images have reduced dimensions due to the nature of the kernel convolution without padding. In a multi-layer CNN, the first ConvLayer is responsible for capturing the low-level features such as edges, colour, gradient orientation, etc. Additional layers extract more and more high-level features which gives the network an understanding of images in a dataset similar to how people do [21].

The max pooling layers slide a window over an image and take the maximum value of the image pixels within the window as the output for each step as opposed to average pooling which takes the average of the pixels in the window. This is useful in extracting dominant features which are rotational and positional invariant and reducing the spatial size of the data which reduces the computational power needed for processing [21].

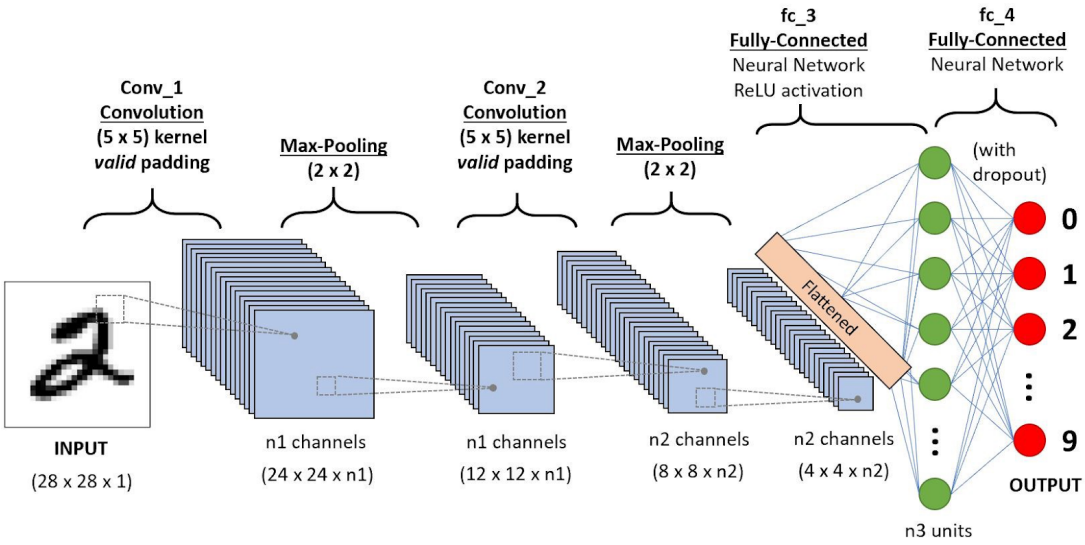


Figure 1: CNN Summary [21]

CNNs update their kernel weights via backpropagation as is standard for neural network architectures. Take the notation below combined with the notation of (40) [22]:

$a_k^l[m,n]$ = activation of the k_{th} kernel of layer l at $[m,n]$
$w_k^l[i,j]$ = weight of the k_{th} kernel of layer l at $[i,j]$
C_L = loss function
$\sigma(z)$ = activation function

Forward propagation is defined as in (40), that is, how the linear output of a kernel is generated from an input. However, this does not take into account the non linear activation function:

$$a_k^l[m,n] = \sigma(z_k^l[m,n]) \tag{41}$$

Applying this, the derivative of the loss function with respect to the kernel weight is obtained

using the chain rule:

$$\frac{\partial C_L}{\partial w_k^l[i,j]} = \sum_{m=1}^{C}\sum_{n=1}^{R} \frac{\partial C_L}{\partial a_k^l[m,n]} \frac{\partial a_k^l[m,n]}{\partial z_k^l[m,n]} \frac{\partial z_k^l[m,n]}{\partial w_k^l[i,j]} \tag{42}$$

The summations are necessary to travel across the activation output image since each pixel of the output image of the k_{th} in the l_{th} layer is a function of the kernel weight being optimized. The loss function at layer l by definition is a function of the activations of that layer and it is also clear that the activation is a function of the linear output $z_k^l[m,n]$ which in turn is a function of the weights. Therefore, each derivative is well defined for each iteration, since the values for the activations are obtained via the forward pass.

Knowing the gradient of the cost with respect to each kernel weight, the weights can then be updated via gradient descent:

$$w_k^l[i,j] = w_k^l[i,j] - \mu \frac{\partial C_l}{\partial w_k^l[i,j]} \tag{43}$$

$\mu =$ learning rate

Gradient descent is used as the optimization method since it is globally convergent so long as μ is small enough and it makes no assumptions on the Hessian. The main advantage Newton based algorithms have is that they have a higher order speed of convergence; however, this is not a big deal since a computer can run many iterations of descent in a small amount of time. Furthermore, the Newton method is only locally convergent and assumes a positive definite Hessian for invertibility [23].

The benefit of a neural network is it can model any function. Therefore, a neural network should be able to transform highly correlated features appropriately such that the correlation does not impact model performance. However, correlated inputs do impact backpropagation convergence speed, τ, which is controlled by the ratio between the largest and smallest (non-zero) eigenvalues of the covariance matrix of the input. Any correlation in the input data gives rise to an eigenvalue of the order of the input dimension as well as a continuous spectrum of order one, potentially massively increasing the training time [24].

2.3 Performance Metrics

2.3.1 Binary Cross-Entropy

For binary classification tasks, the binary cross-entropy is a useful loss function for neural networks. This is a narrower definition of the categorical cross-entropy loss function with simply two classes. There are two main ways of understanding this concept, using information theory or probability. Here, the derivation using probability will be used [25].

Suppose for classification a model obeys a Bernoulli distribution:

$$f(k,p) = p^k(1-p)^{1-k} \quad \text{for } k \in \{0,1\} \tag{44}$$

Ideally, when $k = 1$, the probability p should be high and when $k = 0$, the probability $(1-p)$ should be high. However, p itself is unknown, otherwise there would be no point to training since the distribution would be known exactly. Therefore, a maximum likelihood estimate (MLE) for the parameter p is done, with the estimate given by $\hat{p}$. The joint distribution given some data is given by:

$$f((x_1,y_1),(x_2,y_2),...,(x_n,y_n);\hat{p}) = \prod_{i=1}^{n} \hat{p}^{y_i}(1-\hat{p})^{1-y_i} \tag{45}$$

$(x_i, y_i) = (i_{th}$ feature, i_{th} target)
$n =$ number of samples

Maximizing a function is the same as minimizing the negative of that function. Furthermore, scaling and taking the logarithm of the function offers the same optimization task, such that the function to be minimized becomes:

$$-\frac{1}{n}\log[f((x_1,y_1),(x_2,y_2),...,(x_n,y_n);\hat{p})] = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{p}) + (1-y_i)\log(1-\hat{p}) \tag{46}$$

Which is the final binary cross-entropy loss function. Notice that $\hat{p} \in [0,1]$ is a probability. Therefore, often a sigmoid function, $s(x)$, is used to normalize the output to the same range:

$$s(x) = \frac{1}{1+e^{-x}} \tag{47}$$

2.3.2 Bayesian Information Criterion

When fitting probabilistic models (such as GMM) it is possible to increase the likelihood outcome by adding more parameters indefinitely. However, doing so may result in overfitting and increases computational complexity. Therefore, the goal of the BIC is to introduce a penalty term for the number of parameters in the model. In doing so, an optimal model becomes more clear. Formally, the BIC is defined below, where a smaller value is better [26]:

$$BIC = \ln(n)k - 2\ln(\hat{L}) \tag{48}$$

$\hat{L} =$ maximized value of the likelihood function of the model
$n =$ number of data points
$k =$ number of free parameters to be estimated

2.3.3 Stratified Cross Validation

A machine learning model must be able to generalize for the different variations of data available. However, with just one test set it is possible that a model will perform well on that one test set but not another. In order to counteract this, cross validation can be done, more specifically, k-fold cross validation. The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups or folds.
3. Take a group as the test data set.
4. Fit a model on the training set and evaluate it on the test set.
5. Retain the evaluation score and discard the model.
6. Repeat steps 3 to 6 until each group has been the test set once.
7. Average all the evaluation scores to obtain a final measure of the model's accuracy.

There is no formal rule for the number of folds to pick, k, although there is a bias variance trade-off associated with the its choice. As k increases, the number of samples in each fold decreases. As this happens, the bias of the folds becomes smaller but the score variance increases.

A stratified cross validation is the same as a standard cross validation more suited for classification problems. This is because stratification seeks to ensure that each fold is representative of all strata of the data. That is, each fold has the same proportion of different classifications. This has the effect of reducing the average bias and score variance of each set.

2.3.4 Precision and Recall

A confusion matrix is a classification matrix of results. That is, it illustrates how many test samples are correctly classified by a model and how many are misclassified into a different class. Taking the simple case of a binary classification task, with output classifications 0 and 1, the confusion matrix is defined as $M \in \mathbb{R}^{2\times 2}$:

$$M = \begin{pmatrix} \text{true positives} & \text{false positives} \\ \text{false negatives} & \text{true negatives} \end{pmatrix}$$

From such a matrix, precision and recall measures can be obtained:

$$precision = \frac{tp}{tp + fp} \tag{49}$$

$$recall = \frac{tp}{tp + fn} \tag{50}$$

3 Requirements Capture

As already discussed, this project is centred around the identification of donks in the time series given limited data and the investigation into the relationship between donk occurrence and atmospheric temperature change. More formally, the requirements are listed below:

1. **Classify donks reliably in the data**. Ideally a perfect machine is created that can identify only all the donks in a time series. However, failing this, the classifier should at least be able to identify a set consisting of almost entirely donks, even if it does not manage to identify entirely all the donks that exist in the set. That is, precision is valued over recall.

2. **Investigate the relationship between atmospheric temperature change and donk event occurrence**. Conclusions should be drawn about the likelihood of a donk event given some temperature change observed. if a relationship is seen, it should be formalized mathematically.

3. **Outline how strategies might change in the future**. Currently, not much labelled data exists to analyze. However, as time goes on this set will undoubtedly expand. Therefore, models trained now may not be adequate or optimal in the future, so possible strategy evolution should be investigated.

4. **Consolidate work into accessible notebooks and scripts**. There is as of yet no groundwork laid out in python. Therefore, one goal of this project is to establish some preliminary scripts that can be repeatedly used in future work. Furthermore, the notebooks will act as documentation on the functions used in those scripts as well as a sandbox for further investigation in the future.

4 Analysis and Design

This section serves as a high level overview of the project, with discussions on the different techniques used and why they are useful.

4.1 High Level Diagram

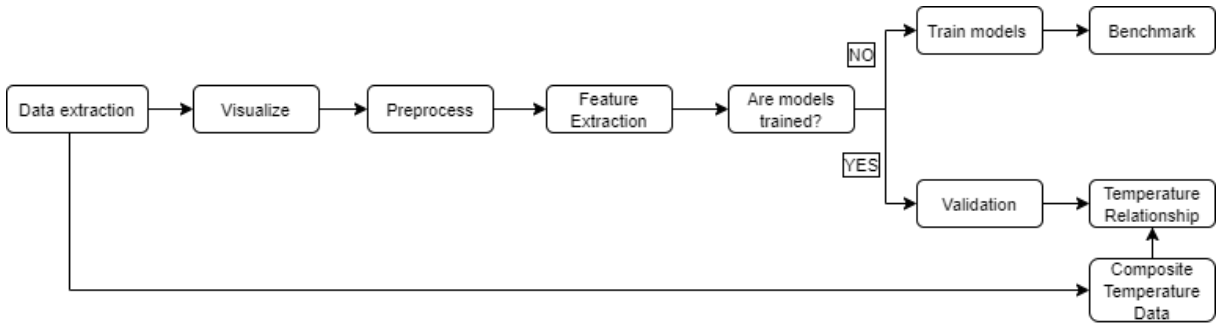


Figure 2: High Level Diagram

4.2 Data Extraction

It was decided that this project should be undertaken in python due to a pre-existing familiarity with the extensive machine learning library and support the language offers. However, the existing work surrounding the project exists in Matlab. Therefore, it is necessary to take the data from Matlab and convert it to a format amenable to python.

Furthermore, the label data exists as a single Excel spreadsheet which contains not only label dates for when the donks occur but also for when all different types of features are located across different days. Therefore, it is also necessary to reduce this set to only the relevant donk label data.

Finally, atmospheric temperature data is taken to generate differential temperature distributions; which when combined with classified donk events can be used to analyse the donk-temperature relationship.

4.3 Visualization

This refers to the plotting of different formats of the data. It is useful to visualize data to get a deeper understanding of the problem and to initially hypothesize appropriate features which can be used to classify the data and to validate the existing labelled set.

Two forms of visualization are needed, the first is to concentrate on the donk events themselves to better characterize what it is that needs to be classified. Secondly, the overall Sols in which the donks sit are also plotted in order to see what other features exist in the data and how they compare to the donk events to be identified.

4.4 Preprocess

This refers to the process of generating data in the appropriate format to start extracting features. Up to this point, the signal data is still in chunks of Sols. This is transformed into an array of both donk frames and negative frames. The number of donk and negative data frames collected are equal to generate a balanced dataset to train on.

The donk label data also consists of a list of times detailing from when and when to donks occur across all the Sols of data. These are not uniform in length so are formalised such that the data can be passed onto the machine learning models which require standardized inputs. There is a tradeoff to be had in this decision; a longer frame length allows for the capture of more contextual information in each frame, however, it also reduces the resolution of the classification. That is, multiple events can be bundled into a single frame, potentially misleading as to the number of events classified in the Sol. Furthermore, as frames become too large, the actual event to be classified becomes an increasingly insignificant portion of the sample (SNR decreases).

4.5 Feature Extraction

Here, the time series can be transformed to obtain appropriate features that the machine learning algorithms can classify on. A summary of the process is illustrated in Figure 3.

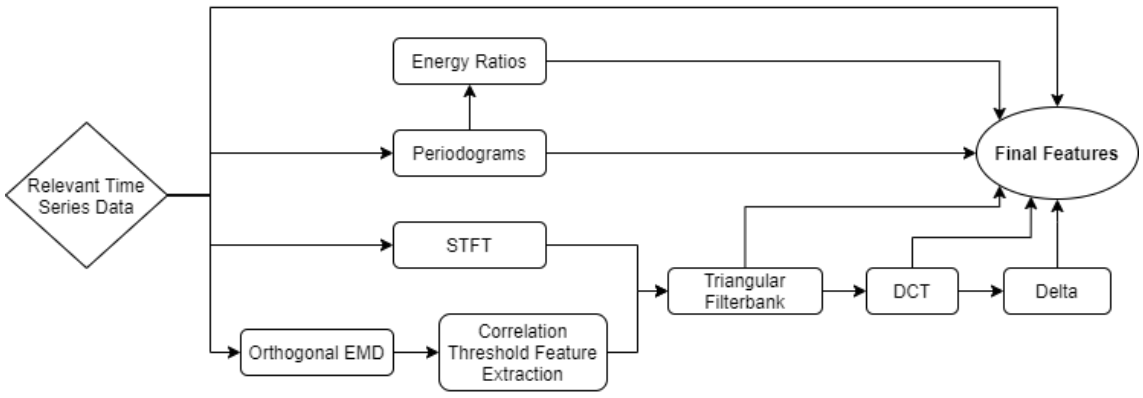


Figure 3: Feature extraction overview

As a naive starting point, the raw time series features are collected. This will allow analysis into the significance of the isolated temporal information for classification.

A simple periodogram set is then obtained, such that all time information in the frame is lost and only frequency information remains. This is done with dB scaling so that small variations in the PSD are better separated and a prior detrend is also applied to the time series to remove any large DC response. The periodograms act as a test to directly observe the significance of the frequency information for classification.

Furthermore, energy ratios are also acquired from the periodograms, the ratio being the total energy above a threshold frequency to the energy below the same threshold frequency. This is

done as a basic method that theoretically captures the most useful information in a very condensed format, minimising the curse of dimensionality.

Time frequency representation is also acquired via the STFT and HHT methods described in Section 2.11 and Section 2.12. A summary of the comparisons of the two approaches is given by Table 1.

	STFT	**HHT**
Basis	a priori	adaptive
Frequency Method	Convolution	Differentiation
Nonlinear	No	Yes
Non-stationary	No	Yes
Feature Extraction	No	Yes
Theoretical Base	Theory Complete	Empirical

Table 1: Summary of STFT and HHT methods

One failure of the STFT is its difficulty in distinguishing dominant frequencies with respect to time of non-stationary signals. The reason for this is if a frame is chosen with multiple frequency components (as is fundamental in non stationary data), then multiple components are mixed during a single operation of the kernel. Furthermore, since the STFT has an a priori basis it suffers from problems like the uncertainty principle. On the other hand, the fact that the EMD is adaptive means that for many signals the algorithm does not converge to a set of finite IMFs.

The STFT has existed for a very long time as the standard in generating time frequency plots. This well established presence allows it to benefit from the existence of algorithms designed to speed up computation times (Cooley-Tukey FFT) which are well documented in professional level libraries with mass market usage. Therefore, this provides the foundation for a reliable method which can easily be implemented with fast computation times, which cannot be said of the HHT. Furthermore, across short time frames the seismic signal can be approximated as stationary so it would appear to be suitable.

The OEMD can be used to achieve orthogonalization between IMFs as discussed in Section 2.1.3. The mode mixing phenomena is undesirable in this scenario since the bandwidth of all feature signals is small (as a reminder, $f_s = 100Hz$), and the frequencies of such signals are in close proximity. Therefore, by separating the separate frequencies there is a higher likelihood of separating the different features of the signal. In order to remedy this, two approaches were investigated. Firstly, a masking signal can be used. That is, based off the frequency information of the original data, a masking signal can be created such that it will pull only the desired signal away from the other components by the same mode mixing phenomena as previously discussed. The masking signal can then be removed from the relevant IMF leaving behind the attenuated desired component. However, this is not feasible since the bandwidth of the feature signals is equal to the frequency range available, so a masking signal of sufficiently high frequency cannot be generated without interpolation first being applied which could introduce artifacts. Another approach is to use a noise assisted method called the Ensemble EMD algorithm (EEMD). This defines the true IMF components as the mean of an ensemble of trials, each consisting of the signal plus a white noise of finite amplitude. However, this introduces randomness to the algorithm which leads to inconsistent

results such that the same data does not give the exact same IMFs over multiple implementations. In the end, the OEMD was deemed sufficient to gather uncorrelated signal components.

The resultant time frequency representations are passed to an overlapping triangular filterbank where FCs are obtained. This is done since two closely spaced frequencies are difficult to discern. Only the rough energy at each frequency is required. Whilst the frequency resolution of the time-frequency representations themselves could simply be reduced; the FCs take a fraction of neighbouring frequency energies as well, giving a better indication of the rough energy at a particular frequency. Since the filters are overlapping, the filterbank features are highly correlated.

For each representation, decorrelated FCs, also known as CCs, are also evaluated with the application of the DCT. This is important since many machine learning models (e.g. Naive Bayes, HMM) assume uncorrelated features. However, as a linear transformation, the DCT also tends to throw away a lot of information in the seismic signal. Therefore, comparing both CCs and FCs will indicate which feature the tradeoff favours.

In a heuristic attempt to compensate for the feature decorrelation assumption made by some machine learning models, first order (delta) and second order (delta delta) regression coefficients can be appended to CCs to directly incorporate differential temporal information between frames, which is otherwise lost with the DCT.

4.6 Training Models

	Logistic Regression	**Support Vector Machine**	**Bernoulli Naive Bayes**	**Random Forests**	**K-Nearest Neighbors**	**CNN**	**HMM**
Lazy/Active	Active	Active	Active	Active	Lazy	Active	Active
Binary Classification	Yes	No	Yes	No	No	No	No
Uncorrelated Features	Yes	No	Yes	No	No	No	Yes/No
Feature Standardization	No	Yes	No	No	Yes	Yes	No
Suitability for Small Datasets	High	High	High	High	Low	Low	Varies
Scalability	Low	Low	Low	Low	High	High	High

Table 2: Summary of machine learning methods

There are many different machine learning models available to fit the data features on. Each algorithm has its own advantages and disadvantages which must be tested to determine the best model for classification. A summary of the different algorithms is given by Table 2

The CNN should be optimal as more samples are obtained and the population becomes larger,

since it is able to account for niche features in the data. However, CNNs are computationally complex; often with hundreds of thousands of trainable parameters such that the number of curves in the coordinate space is much higher. Therefore, with small quantities of data, regression to the mean is not ensured and more data is needed for reliable backpropagation. Furthermore, since CNNs are made to take image-like data as an input, only spectral features with dense pixel data are used for training (HMM and STFT features). In theory, feature correlation should not affect the CNN performance; in fact, application of the DCT removes some differential information between features, so the algorithm could perform worse. On the other hand, the decorrelated features should run faster, although this is less impactful with GPU acceleration used. In order to obtain a direct applied comparison, both correlated and decorrelated features will be tested.

On the other side, the RF technique is an entropy based algorithm with few assumptions on the underlying data, even when compared to the other traditional algorithms, making it one of the most robust algorithms available. This makes it very well suited to small datasets, and so is implemented as one of the core basic algorithms the HMM and CNN compete with; however, as an ensemble method, it takes longer than the other traditional algorithms to train.

The LR and SVM algorithms are both active distance metric based. The LR algorithm is more basic with the assumption of a sigmoid decision boundary which the SVM is not subject to. Both are included since it is useful to see how close the sigmoid decision boundary is to the more generally generated decision boundary of the SVM.

The BNB is the simplest conditional probability based algorithm used. A naive bayes can be described as a HMM with one fixed state. Therefore, BNB is included since it is a very quick way of determining which features are likely to work best with the longer HMM.

As a compromise between complex and simple systems, the HMM is sufficient as it is highly tuneable and can be made to be as complex or simple as desired with the ability to trade the number of assumptions on the data for runtime complexity. This will be elaborated in Section 5.6.3. However, unlike the other algorithms, it is not included in any mass market libraries, reducing its reliability. Furthermore, there is potential for the number of trainable parameters to grow quickly, although it never reaches the complexity of the CNN. This means that it can take a long time to train, especially without GPU support, so only features that already perform well on the faster, more basic algorithms (especially BNB) will be taken to train on the HMM. In order to observe the variety of donks and negatives in the dataset, a GMM will be applied, the results of which will help dictate the use of GMMs as emission distributions for the HMM. If the whole data does not fit well to the GMM, then when probabilistically split up in the HMM each section will definitely not fit well either since the distribution space is even smaller.

The KNN is included as the only lazy algorithm to be tested. It is important to investigate since a lazy algorithm is especially useful for a rapidly growing dataset as there is no defined training phase. Instead, the new data points are simply passed to the feature space and the algorithm will take into account these points to help with classification without any further steps necessary. This comes at the cost of high memory usage; however, the dataset would need to become exceptionally large for this to become a serious issue.

For CNNs, HMMs and KNNs, topologies need to be optimized. An evolutionary strategy algorithm can be used for such a task. For HMMs and KNNs, the topology is controlled only by one or two parameters (number of states and potentially number of Gaussians to model the emissions for

HMM and only the K value for KNN), so this is simple to apply and the algorithm converges in few iterations. For the CNN; however, the topology is controlled by a great number of parameters, such as the number of kernels in each layer, the size of the kernels, the number of layers, etc. This makes the runtime of the evolutionary strategy for the CNN exceptionally long due to the larger dimension search space. Therefore, a more conventional method of trials following well known guidelines for the structure is used to obtain a suitable, roughly optimal topology.

4.7 Benchmarking

A benchmark is formalized in order to compare the different model performances. A stratified cross validation is done to ensure model performance across the whole dataset. Furthermore, precision, recall and confusion matrices are taken into account as well as simple classification accuracy, giving hints as to what aspects of the model perform well or poorly. Simple accuracy is suitable since the dataset used is unbiased between classes.

4.8 Validation

The benchmark alone is a necessary but insufficient measure of the overall performance of the model in real life because the dataset used for training is very small, and so does not capture much of the variance of real life events. Therefore, as a final validation, an entire Sol's worth of data is passed to the optimal model(s) and the results will act as a final determining factor of the competence of the strategies employed.

4.9 Temperature Relationship

It is hypothesized that donks occur as a result of large atmospheric temperature changes. In order to validate this, the conditional probability, or likelihood, of a donk given temperature difference is taken using Bayesian theory:

$$f_{D|T}(t) = \frac{f_{D,T}(t)}{f_T(t)} \tag{51}$$

$f_{D|T}(t) =$ likelihood distribution of donk given temperature gradient
$f_{D,T}(t) =$ joint distribution of temperature gradient and donk events
$f_T(t) =$ distribution of temperature gradients

The validation set and temperature data extracted are used to generate the joint and temperature gradient distributions. The logarithm of the likelihood distribution is then taken to get a better visual separation between small changes. Once this is done, the resultant distributions can be appropriately formalized mathematically.

5 Implementation

This section will illustrate in detail the implementation steps of the project, with rationale given for parameter tuning decisions.

5.1 Environment

The scripts written thus far are done using Matlab. However, with the machine learning aspect of the project, python appears to be a better selection given the availability of various libraries dedicated to machine learning and data processing. In particular, a Google Collaboratory document is an ideal IDE to develop in. This is essentially a cloud python notebook with the ability to call on GPU resources for free, particularly useful for neural network acceleration. The only disadvantage is that after a few hours of inactivity the connection terminates on its own, although even here a local runtime can be used if prolonged continuous usage is expected.

5.1.1 Resources

In particular, the free version of the Google Collab environment includes 12.72GB of RAM with 15GB of permanent disk storage and further access to an Nvidia Tesla P100 16GB GPU with an Intel Xeon CPU.

5.1.2 Library List

This project will also make use of a great many different libraries:

1. *NumPy* - A scientific computing package with excellent linear algebra capabilities.
2. *pandas* - A useful library for visualizing data in a tabular format.
3. *Matplotlib* - A useful library for visualizing data graphically.
4. *seaborn* - Built on Matplotlib for specialized statistical visualization.
5. *SciPy* - A package containing useful signal processing transforms (e.g. FFT).
6. *math* - A library that allows for multiple basic mathematical transforms (e.g. logarithm).
7. *datetime* - Allows for manipulations of data representing dates and times.
8. *random* - Allows for generation of random processes.
9. *librosa* - A package for automatic and reliable filterbank generation.
10. *PyHHT* - A more individual package containing tools to implement the raw EMD algorithm.
11. *Keras* - Built on TensorFlow to allow for easy neural network manipulation.
12. *scikit-learn* - Contains many traditional machine learning techniques with high abstraction.
13. *hmmlearn* - A specialized package for HMMs emulating scikit-learn commands.
14. *pickle* - Allows for saving and loading data structures in their native format.

5.2 Data Extraction

A Matlab script is provided that obtains the signal data of a particular Sol. This data is too large to be put into an Excel spreadsheet so is instead extracted as a .txt file with CSV formatting that can then be imported into python. Before this is done, the Sols of the labelled donk events must be identified. A separate Excel spreadsheet is also provided which contains over 10,000 labels for all the different features, identifying when each particular event starts and ends in the time series.

Out of the 10,000+ labels, it turns out only 259 of those labels are donks, with very few donks actually being labelled on each day. There is limited storage on the cloud and the Matlab scripts used to get the data have a long runtime, so it is decided to extract data from three dates overall. There are two peaks where the majority of donks have been labelled, on 16-02-2019 and 14-06-2019, as well as these, the date 2019-03-07 is taken as it still has significant donk data, giving a total of 167 labelled donk events.

The date 16-02-2019 is represented by Sol 80, the date 14-06-2019 is spread over Sol 194 and Sol 195 and the date 2019-03-07 is represented by Sol 98. For details on the data structure, refer to Section 1.3. For each Sol, a separate combination of SP and VBB data is extracted, the largest in size being from Sols 195 and 98, each with a combined memory usage of 1459MB, whilst the other two Sols only take up around 960MB each.

5.3 Visualization

In order to visualize the data, spectrograms are used alongside the time series since they are robust and quick to compile. To generate these, the logarithm of the square root PSD is calculated. This square root logarithm is used to squish the range of values closer together, which overall provides a better visual separation of the frequency energies. Furthermore, for appropriate resolution, frame width and overlap are chosen $[width, overlap] = [0.64s, 0.32s]$ and a Hanning windowing function is used. Since the main donk energies exist at high frequencies, the frequency scale is kept linear so high frequency energy separations can be more clearly observed. The 'spectrogram' function from scipy allows for such specifications.

Figure 4 shows an example of a donk event visualization. From the time series, it can be seen that the donk appears as a glitch like event with a very short duration. The spectrograms illustrate that the energy of the event is mainly contained in the higher frequencies above $20Hz$. Closer analysis reveals there are generally two distinct high energy bands, one at around $25Hz$ and one at around $40Hz$.

The Sol data sets are very large and a python notebook offers little space to properly visualize such large datasets. Furthermore, the interactive plotting function takes too long to run using the cloud resources. Therefore, it was decided that a separate Python script would be used to visualize the Sol data, allowing for faster runtimes and more space to examine interactive plots. A class 'LSG' is created, standing for 'Labelled Spectrogram Generator'. This takes in the data from the data extraction step and generates spectrograms which are labelled using another input label data to illustrate where events have been labelled.

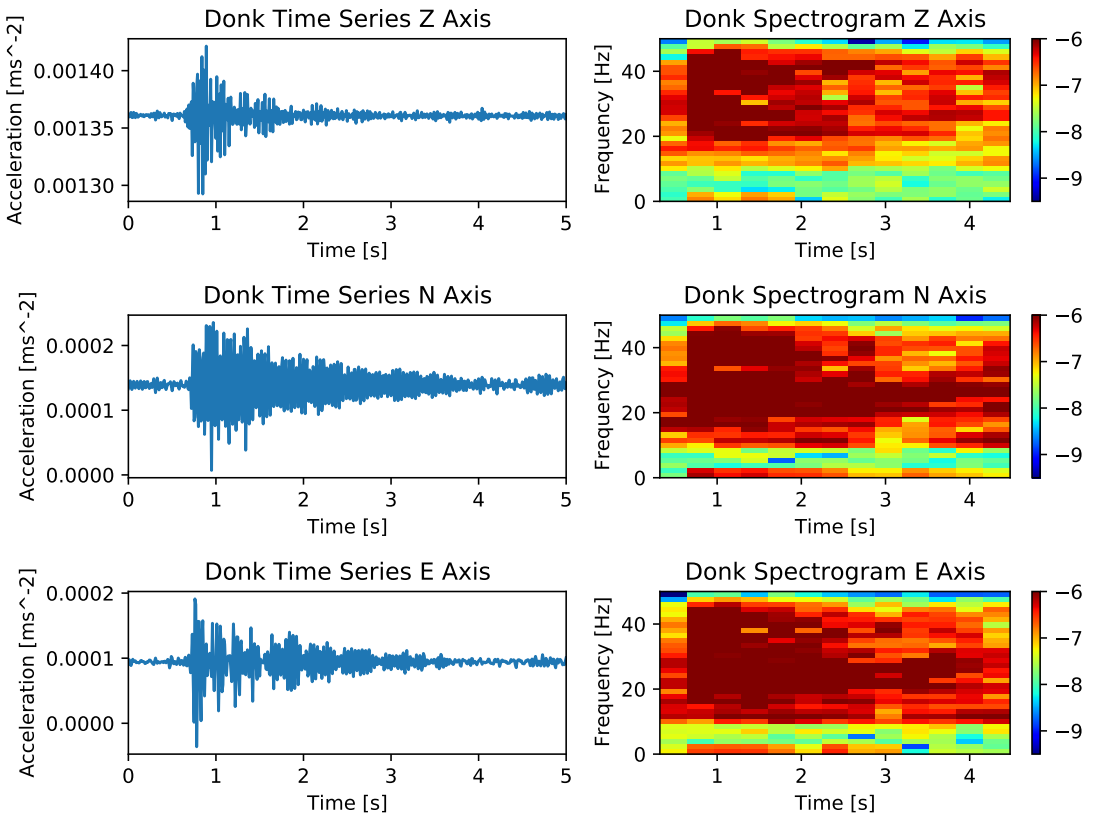


Figure 4: Visualization of a donk event

For plotting a Sol spectrogram, either a new one can be generated or one can simply be loaded from a previously saved pickled plot. This allows for time saving since the same plots need not be generated from scratch repeatedly. The spectrograms themselves are calculated in the same way they are for the individual donk events, with the exception that the frame width and overlap are chosen as $[width, overlap] = [2.56s, 1.28s]$ to adjust to a new resolution. The starts of the labelled events are labelled on the spectrogram with a black line and the ends are labelled with a white line for clarification.

A zoomed in plot of a labelled spectrogram is given by Figure 5. Here, two labelled donk events exist at 48,150s and 48,510s. Just from this section, it is clear to see that not every donk in the Sol has been labelled; in fact, the majority have been left unlabelled. Furthermore, it can also be seen that there are other features with similar frequency patterns to the donks, such as the event starting at 47,800s. For the most part, these events are wind, which appear to be a smudged donk across time.

A concluding hypothesis is formed that both time and frequency information should be important in appropriately identifying donk events and separating them from the rest of the Sol data.

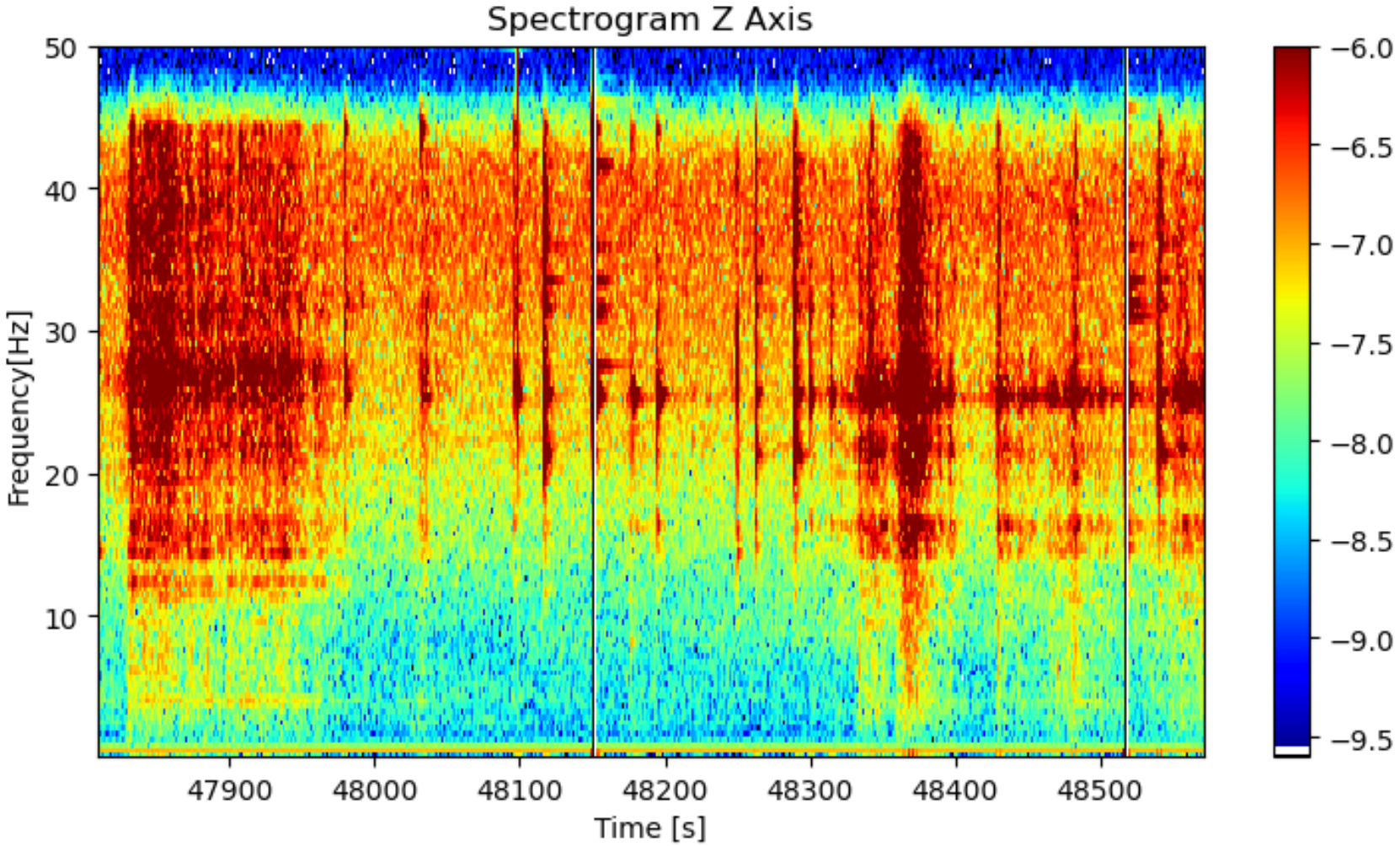


Figure 5: Zoomed in labelled spectrogram of Sol 80 on the Z axis.

5.4 Preprocessing

All preprocessing functions are collected into a python script under a class called 'preprocess' as well as being illustrated in a Google Collab document. As a general rule, the preprocessing algorithms convert data into numpy arrays for fast processing before converting the processed data into a pandas dataframe for a more structured view.

5.4.1 Binary Search

In order to combine the SP and VBB files for each Sol, the two sets can simply be added together. Taking the donk label set, an algorithm can then be applied to the combined seismic data file to isolate the donks of each Sol. A binary search is chosen as the fastest ordered list search algorithm with a total runtime of $O[Mlog(N)]$ where N is the number of samples in the Sol and M is the number of donk events to be found in that Sol. The function has a tuneable parameter 'acc' which has already been set to be functional at a value of 5. This essentially sets the precision of the search which is traded for convergence of results. Since the data is discrete, it is possible that the binary search algorithm will not find the exact value being searched for. For example, the time being searched for may be 13:14:12.465 whereas the time samples of the data may in reality go from 13:14:12.461 to 13:14:12.471 (100Hz sampling rate). Therefore, 'acc' represents a range of values to be searched for at each step, more specifically, the number of milliseconds either side of the current search time which the target time may sit within. Therefore, to guarantee convergence with the minimum range, the value should be chosen according to:

$$acc = \frac{1000T}{2} \tag{52}$$

$T =$ time period

5.4.2 Framing

The donks found in the labelled set are not all the same length. Eliminating outliers, it is discovered that the average length of each donk is 500 samples, or 5s. However, data is frequently labelled such that the main donk lobe exists right at the beginning or end of the 500 sample frame. It is hypothesized that the donks might be more easily classified by taking the 500 samples before and after the set originally identified to better centre the event such that there is greater consistency in the features to be learnt. Therefore, a frame of 1500 samples, or 15s, is used.

There still exists; however, the option to extract each donk with frame lengths as specified taking the label end times into account, as is done in the visualization methods in Section 5.3.

5.4.3 Negative Data Extraction

For each Sol, a certain number of donks, D_i, are obtained by the binary search algorithm. The same number of negatively labelled data samples, X_i, are also to be gathered from the same Sol.

However, there exist no labels for such data. Furthermore, from visualization it is found that many donks exist in each Sol that are not labelled in the set, so randomly choosing a chunk of data is likely to result in the negative data also having donks within it. Therefore, again using the results from the visualization, periods of time are found that are suitable to be included as negative data when split into 15s frames. In particular, areas of windy data are used as approximately half the negative data set since these are most similar to the donk features to be classified. Therefore, it is useful to include these as negative samples for the models to train with. For example, for Sol 80 a suitable time period is found starting 12,000s into the Sol. These negative sets combined with the donk data form the total feature set of each Sol, with N_i total samples:

$$N_i = D_i + X_i = 2D_i \tag{53}$$

The label given to each sample is also adjustable, although it is automatically set such that each donk is labelled as the integer 1 and each negative data sample is labelled as the integer 0. Therefore, if there are multiple classes that need be identified, this can be done with proper labelling by passing different label sets to be identified.

5.4.4 Outliers

There exist two donks from Sol 80 where the labelled end time is the same as the start time. Upon further investigation, it is found that these are still indeed donks and there has clearly been a human labelling error; therefore, these are included in the set. However, if the exact length donks are desired, these outliers should be removed since they will give a single point, invalid signal. This can be done by passing to the binary search algorithm an optional minimum limit for the length of the donks (measured in time samples) that are desired to be included in the set.

Furthermore, 18.8% of the labelled donk events of Sol 80 are falsely labelled (16 events). These are simply removed from the set by passing to the binary search algorithm again an optional list of indeces correlating to the indeces of the label set that should not be extracted.

5.5 Feature Extraction

Figure 6 illustrates a visual summary of the physical features obtained for comparison. Furthermore, Table 3 offers a higher level summary of the data tensors where $N = \sum_{i=1}^{Sols} N_i$ and N_i is defined as in (53). All feature sets are saved in permanent memory as numpy arrays.

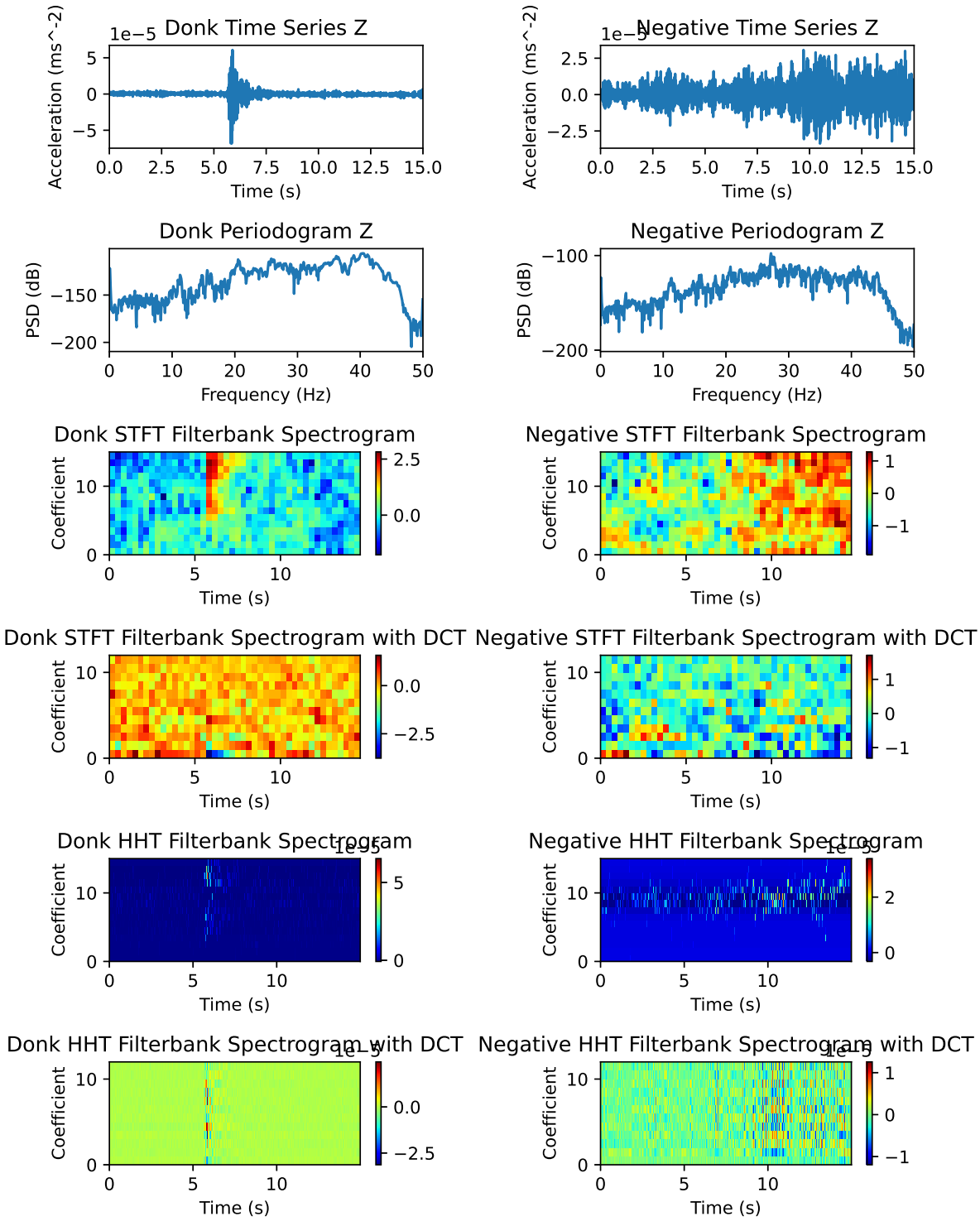


Figure 6: Summary of features on the Z axis, with a donk (left) and negative (right)

	Time Series	Periodogram	Energy Ratios	STFT	HHT
Tensor Order	3	3	3	4	4
Shape	(N,3,1500)	(N,3,751)	(N,3,3)	(N,3,45,33)	(N,3,1500,201)
Shape with Filterbank	N/A	N/A	N/A	(N,3,45,16)	(N,3,1500,16)
Shape with DCT	N/A	N/A	N/A	(N,3,45,13)	(N,3,1500,13)
Shape with Delta Regression	N/A	N/A	N/A	(N,3,45,39)	(N,3,1500,39)

Table 3: Summary of feature set dimensions

5.5.1 Time Series

From Figure 4, it can be observed that each axis of the time series has a different DC offset. This is undesirable and should be normalized since the DC offset provides no information for signal classification and makes training harder, so for each axis the DC component is removed.

5.5.2 Periodograms

Methods for obtaining periodograms through the Cooley-Tukey FFT are well established with preexisting libraries available to draw upon with high abstraction. The 'periodogram' function from scipy generates a periodogram with a prior detrend applied to the time series. However, the function does not have dB scaling in-built, so this is done after the periodograms are generated.

5.5.3 Energy Ratios

Energy ratios can be obtained from the periodogram features with a threshold frequency, $f_{th} = 10Hz$:

$$\begin{aligned} E_{low} &= 10\log\left(\sum_{f=0Hz}^{10Hz} S_{xx}\right) \\ E_{high} &= 10\log\left(\sum_{f=10Hz}^{50Hz} S_{xx}\right) \\ ratio &= \frac{E_{high}}{E_{low}} \end{aligned} \tag{54}$$

S_{xx} =PSD of a frame axis

This is done for each 500 sample segment since the donk should reside primarily in the middle 500 samples so the difference in the ratio between this segment and the other segments surrounding it should be highlighted.

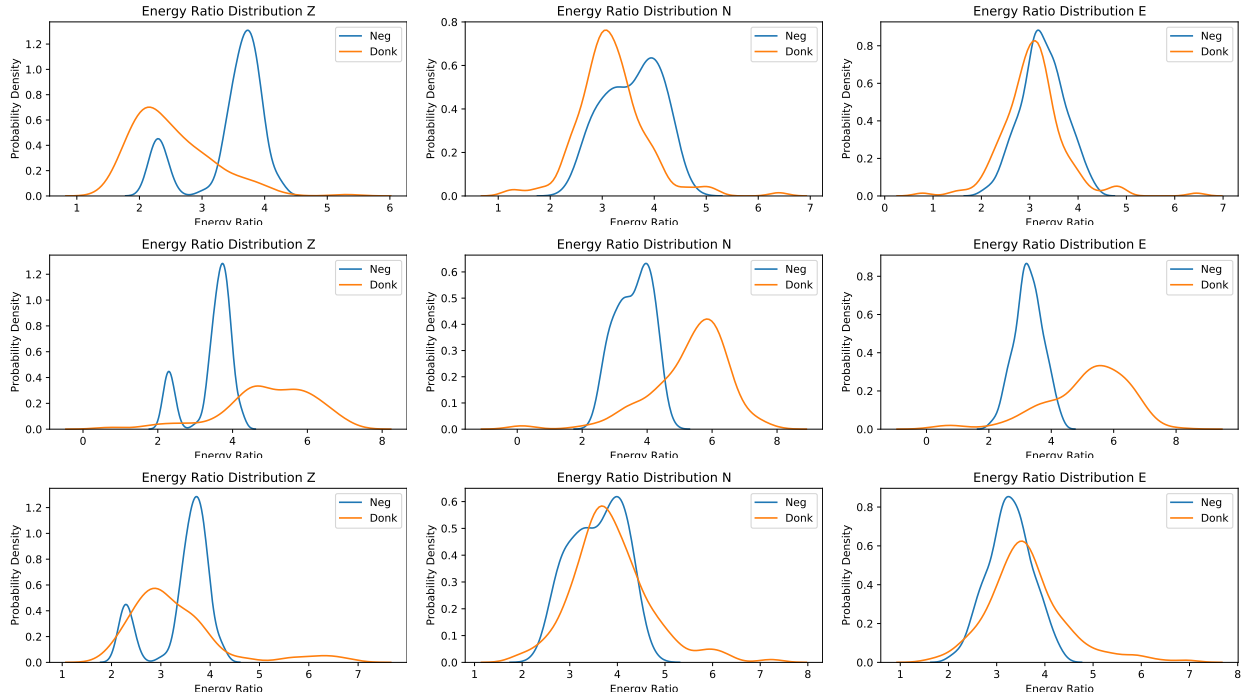


Figure 7: Distribution of energy ratios across all dimensions (first row: first ratio, second row: second ratio, third row: third ratio)

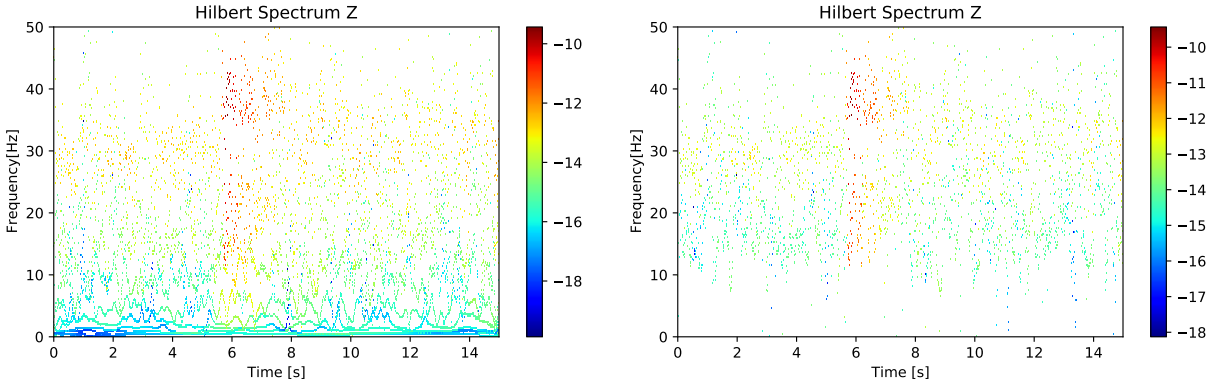
The distribution of each ratio within each axis can be obtained as shown in Figure 7. The results validate the donks being in the middle of the frame with the second row of distributions seeing a higher average energy for the donks than for the negatives, which is a result of the high frequency energy components observed in the donks from visualization. The other two rows see very similar distributions between the donk and negative energies, indicating little difference. It is also observed that both the donk and negative sample energy ratios appear to follow a roughly Gaussian distribution; althought this should become more apparent with more data by the Central Limit Theorem.

5.5.4 Spectrograms

The STFT spectrograms are calculated in the same way as in the visualization spectrograms of Section 5.3.

5.5.5 Hilbert Spectra

Unfortunately, there currently exists little support on python for HHT transformations. Therefore, all functions in this section are collected into a class called 'marsHHT' which can be imported into any future python projects for access. An individual package, PyHHT, is found that implements the naive EMD algorithm sufficiently well. However, methods for the OEMD, post-processing and Hilbert spectrum generation are created organically. Using the PyHHT methods to obtain initial IMFs, the OEMD is applied to the IMFs as defined in (10). The average orthogonal factors of both sets of IMFs is given by $[IO_T^{EMD}, IO_T^{OEMD}] = [0.01335, 6.722 \times 10^{-18}]$. Clearly, $IO_T^{OEMD} \ll IO_T^{EMD}$ to the point where the OEMD IMFs are considered perfectly orthogonal. Having applied the post-processing algorithm defined in Section 2.1.3, the instantaneus frequencies and log instantaneous amplitudes can be obtained to plot a Hilbert spectrum, as in Figure 8a. This is done by collecting the instantaneous amplitudes into frequency bins and summing them for each time-frequency bin before taking the logarithm of the resultant values. Since no scaling is done for distance between bins and actual instantaneous frequencies found, the number of frequency bins used should be set large for the best representation, although this comes at the cost of increased memory usage and runtime of the feature reduction methods.



(a) Full Hilbert spectrum

(b) Hilbert spectrum with feature extraction

Figure 8: Donk Hilbert spectra on the Z axis

It has already been established that the most significant components of the donk occur at high frequencies, but clearly in the Hilbert spectrum there are many frequency components taken below 10Hz. These components add complexity to the spectrum without adding much information. Therefore, relevant features are extracted from the spectrum using (11) to obtain the reduced spectrum in Figure 8b. Now, many of the low energy components of the signal are removed giving a clearer feature space with less information complexity for models to learn. Overall, the HHT provides much higher resolution in both the frequency and time domains when compared to the STFT.

5.5.6 Filterbank Coefficients

Both the Hilbert spectra and STFT spectrograms can be passed through a filterbank to obtain FCs. The package librosa has functionality to generate certain types of filterbanks; the closest one to the needs of the project being the mel filterbank. This is made of triangular overlapping filters; however, they are not equally spaced but instead obey logarithmic spacing [14]:

$$m = 2595 \log(1 + \frac{f}{700}) \tag{55}$$

$m =$ mel scale
$f =$ frequency

These filters are designed for speech signals, and as such the spacing occurs over a large bandwidth. Since the bandwidth of the seismic signals is much lower, the filterbank generated can be assumed to be linearly spaced as required. As a summary, 16 triangular overlapping filters are generated in the range $f_r = [0Hz, 50Hz]$ which are assumed to be equally spaced. As such, when applied to the time-frequency representations, each frame axis consists of a sequence of feature vectors defined, $V_{fbank} \in \mathbb{R}^{16}$, the resultant FCs. Furthermore, mean normalization is applied to balance the spectrum and also improve SNR.

Obtaining FCs has a larger effect on the HHT features since these start with a much higher frequency resolution when compared to the STFT features. Overall, the number of features in the HHT set is reduced by 92% whereas the STFT set is only reduced by 50%.

To get an idea of the level of correlation between features, the sum of the correlation matrix is taken and scaled by the square total number of features per sample (number of components in the symmetric square matrix), the results of which are, $[corr^{FC}_{STFT}, corr^{FC}_{HHT}] = [0.305, 0.075]$. As expected, the correlation between the HHT features is lower since the STFT itself is an overlapping method.

5.5.7 Cepstral Coefficients

The DCT can be applied to decorrelate the coefficient vectors. Here, only the first 13 features are retained since the higher coefficients represent fast changes in the filterbank energies and it turns out that these actually degrade model performance. Therefore, each frame axis consists of a sequence of feature vectors defined, $V_{CC} \in \mathbb{R}^{13}$, the resultant CCs. This reduces the FC features by another 18.75%.

In the same way as with the FCs, the average correlation between the CC features are obtained $[corr^{CC}_{STFT}, corr^{CC}_{HHT}] = [0.0815, 0.102]$. As expected, the DCT has reduced the correlation between the CCs significantly for the STFT samples. However, the HHT CC feature correlation actually increases slightly. This is surprising, although it could be argued that the correlation between features was already so low that there would be limited gain from this method anyway.

5.5.8 Deltas

Finally, deltas and delta-deltas are obtained to construct another feature set. Each delta is essentially a difference equation, so adds another 13 coefficients to the feature vector. This is applied once to the stationary coefficients to get the deltas and then once again to the deltas to get the delta-deltas. Therefore, each frame axis consists of a sequence of feature vectors defined, $V_{deltas} \in \mathbb{R}^{39}$. Hence, this increases the number of features by a factor of 3 from the CCs.

5.6 Training Models

There are three main libraries which will cover all the machine learning algorithms described in Table 2; Keras, scikit-learn and hmmlearn. Therefore, a different set of functions is required to interface with each of these libraries. In particular, scikit-learn implemets all of the algorithms apart from the CNN and HMM (classed as the traditional machine learning algorithms) which are handled by Keras and hmmlearn respectively. All training is implemented in a Google Collab notebook.

5.6.1 Traditional Algorithms

All the traditional machine learning algorithms accept only two dimensional arrays, in the format (number of samples, number of features). From Table 3; however, all of the datasets obtained are of order three or four. Naively, these can simply be flattened such that the axes are concatenated. However, it is also noted that an assumption of the BNB and LR methods is uncorrelated features. It is also clear that the axes are correlated since each responds to the same stimulus at the same time. Therefore, concatenating axes creates correlated features.

Figure 9 shows the correlations between the first feature and all other features (first row of the covariance matrix) of the flattened time series. This can be extrapolated to other feature correlations. For the donk samples in particular, there is a strong periodic pattern every 1500 features, coinciding with the length of each axis frame. The spikes are less prevalent for negative data; however, they are still noticeable.

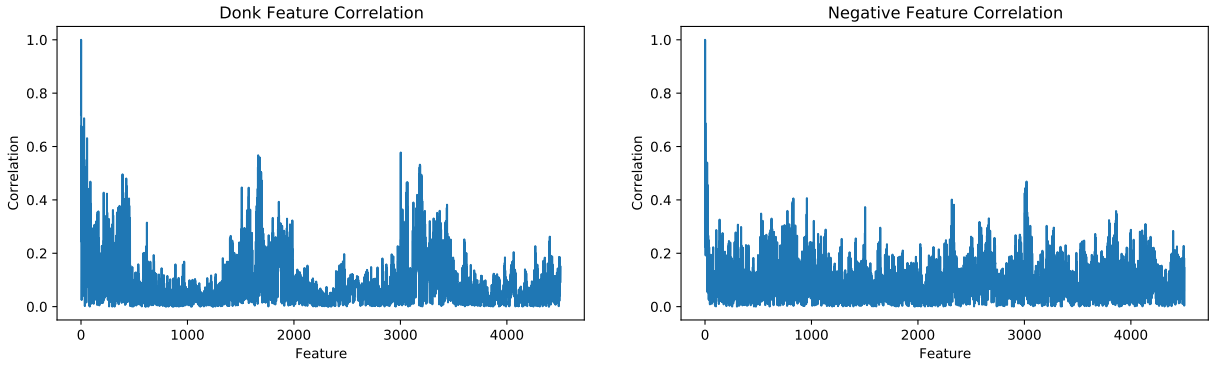


Figure 9: Flattened time series correlations

To combat this, each axis is trained independently of each other. Each trained axis model will then be used to obtain a predicted probability of the existence of a donk for each test sample, and the average of these will be taken and rounded to obtain a final classification. However, the RF, SVM and KNN do not make the assumption of uncorrelated features, so the flattened features can be passed directly to a single model.

5.6.2 Convolutional Neural Network

Feature standardization is the process whereby features are transformed such that their mean is 0 and their standard deviation is 1. This is generally done to improve distance metric based models since features with different scales can bias certain features in the space. For this project, since all features are generated by the same process, feature scaling is undesirable since features are directly comparable. However, for CNNs, each image (sample) should be standardized. A CNN learns by continually adding gradient error vectors multiplied by a learning rate computed from backpropagation to various weight matrices throughout the network as training examples are passed through. If image scaling is not applied, the ranges of the distributions of feature values would likely be different, and so the fixed learning rate would cause biases in each dimension that would differ from one another. To counter this, the learning rate can be made adaptive for each weight parameter, adding another set of hyperparameters to be trained, or the input can simply be standardized prior to training.

For the CNN, Keras provides a sequential structure that allows for the stacking of layers one by one. Since deeper layers are meant to capture higher level features, the size and number of the kernels shrink as the model gets deeper. Furthermore, since the feature set is an order 4 tensor, each sample is three dimensional. Therefore, three dimensional kernels are used for both convolution and pooling layers. A popular choice for activation function is reLU since it is piecewise linear so is not so computationally complex and it helps with vanishing and exploading gradients in backpropagation. However, the model itself is not very deep and the GPU acceleration in Google Collaboratory helps with swift execution; so a more complex tanh activation function is chosen to get a slight increase in performance. Max pooling layers are chosen over average pooling since these also act as a noise suppressant by discarding noisy activations altogether.

Two CNN topologies are required, one for the HHT and one for the STFT features. A summary of the CNN topology for the HHT features is given by Table 4, whilst Table 5 shows a summary of the topology for the STFT features.

The output of the last layer is flattened and passed to a two layer fully connected network for final classification. The first layer has 500 neurons and a reLU activation whilst the final layer has two neurons each representing the probability of the sample being a donk or a negative feature given the target data is one-hot encoded. A binary cross-entropy loss function is used for binary classification, so the outputs must also be scaled between 0 and 1. To do this, a sigmoid is used as the activation function of the last layer.

Layer	Layer Type	Number of Kernels	Kernel Shape	Activation
1	Conv3D	64	(3,3,3)	tanh
2	MaxPooling3D	1	(1,3,3)	N/A
3	Conv3D	32	(1,3,3)	tanh
4	MaxPooling3D	1	(1,3,2)	N/A
5	Conv3D	16	(1,3,1)	tanh
6	MaxPooling3D	1	(1,3,1)	N/A

Table 4: CNN topology for HHT features

Layer	Layer Type	Number of Kernels	Kernel Shape	Activation
1	Conv3D	64	(3,3,3)	tanh
2	MaxPooling3D	1	(1,3,3)	N/A
3	Conv3D	32	(1,3,2)	tanh
4	MaxPooling3D	1	(1,2,2)	N/A

Table 5: CNN topology for STFT features

In training, a single batch is used, since the data set is small, which improves performance. Furthermore, callbacks are used to avoid overfitting. When overfitting occurs, the learning curves for the training and test sets diverge. A callback can be used to save the model at the point just before the learning curves begin to diverge, capturing the model with the best generalization.

5.6.3 Gaussian Mixture Model

The package scikit-learn is used to implement the GMM on the HHT CCs. The suitability of this method is somewhat dependent on frame size. With a longer frame, more features are used to make up the higher dimension distribution (an entire sample makes up a single input for the distribution), so there is a greater likelihood that more Gaussians will be suitable to model the overall higher dimension space. On the other hand, in this case, the utility of the long frames is chosen to better centre the donks in the frame. Therefore, there is relatively less variety between samples (which, again, are the input for the distribution), implying fewer Gaussians are necessary for appropriate modelling. Furthermore, the negative data aught to require more Gaussians than the donk data since there are more events encompassed in the negative data. For example, one Gaussian might model the wind features whereas there might be another Gaussian to model light tremor events.

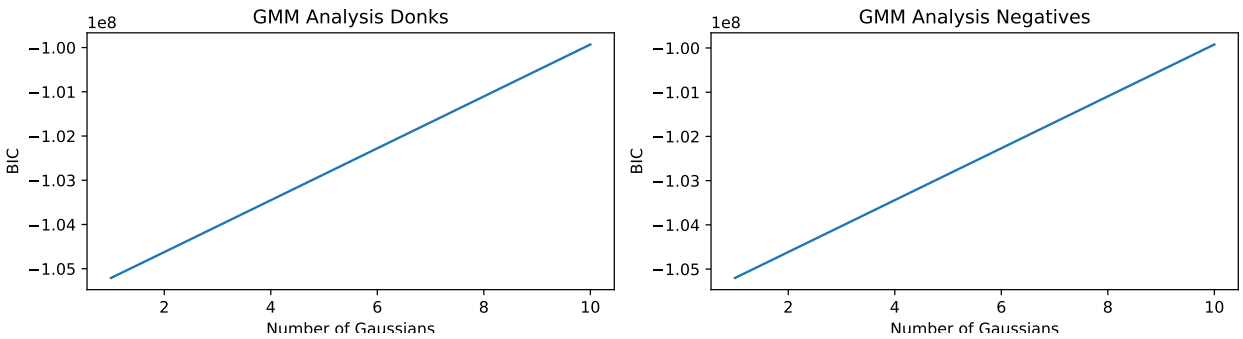


Figure 10: BIC scores of different GMMs (left = donk models, right = negative models)

The BIC is built into the scikit-learn GMM class and is used to determine the suitability of the model for different numbers of Gaussians in the GMM. The results are plotted in Figure 10. It can be seen that in fact a single Gaussian is sufficient for fitting both the donk and negative data since the optimal model is chosen with the lowest BIC; and increasing the number of Gaussians linearly increases the BIC here. One explanation for this is a lack of data collected, such that not enough variety of donks and negatives have been collected to model. Another explanation is linked to the frames still being too short to create a high enough dimension space for sufficient variety, or the deliberate attempt to reduce variety between samples by using a longer frame; although this last case is unlikely since it does not explain the negative GMM results. In any case, the conclusion of this is that a single Gaussian is sufficient for modelling the HMM emission distributions.

5.6.4 Hidden Markov Model

Similar to the traditional methods, the HMM only accepts two dimensional inputs. However, it has the capability of handling three dimensional samples by the ability to define sequence lengths. That is, axes of the input data can be stacked to give feature vectors of length $L = 3X$ where X is the length of the feature vectors for each axis. For example, given the HHT with DCT features of shape $(N, 3, 1500, 13)$, the input can be reshaped into $(N \times 1500, 39)$. Another input is also defined as the lengths of each sequence. That is, it would be defined as a vector of length N with each of the N components corresponding to the length of the sequence at that index. Therefore, for this example, it would be $[1500, 1500, 1500, ..., 1500]^T \in \mathbb{R}^N$. In essence, this allows the model to conclude that each sample in this case consists of 1500 feature vectors of length 39.

The HMM itself assumes GMM emission distributions (just Gaussian if there is only 1 Gaussian in the model), these being the distributions of observing a feature given being in a certain state. Since each observation is multivariate (feature vectors), each state has a multivariate GMM distribution associated with it. Therefore, in training, parameters for the initial state distribution, state transitions and the multivariate Gaussian mean vectors and covariance matrices for each state must be learnt:

$$P = (S) + (S^2) + (SK(L + L^2)) \tag{56}$$

$P =$ number of trainable parameters

$S =$ number of states defined for the hidden Markov chain
$L =$ length of the feature vectors
$K =$ number of Gaussians in the GMM emissions $(= 1)$

As S grows, P grows massively, often increasing the runtime beyond acceptable levels and requiring larger amounts of data to accurately estimate parameters. To counter this, the covariance matrices of the Gaussian emissions can be assumed to be diagonal. If this is done, then the input features must be decorrelated using the DCT. This reduces the total number of trainable parameters to:

$$P = (S) + (S^2) + (SK(L + L)) \tag{57}$$

To further reduce the runtime, the viterbi algorithm is used in training to get a good approximate of how well the model performs.

The HMM is technically an unsupervised model that simply obtains the most likely state distribution of a Markov chain given some observable data. Therefore, to classify data, two models must be trained. One model is fitted with the donk data and one model with the negative data. A test sequence can then be passed to both trained models to determine the log likelihood of observing that observation sequence in each model. The test sequence is then assigned to the model that gives a higher log likelihood. This process is summarised in Figure 11.

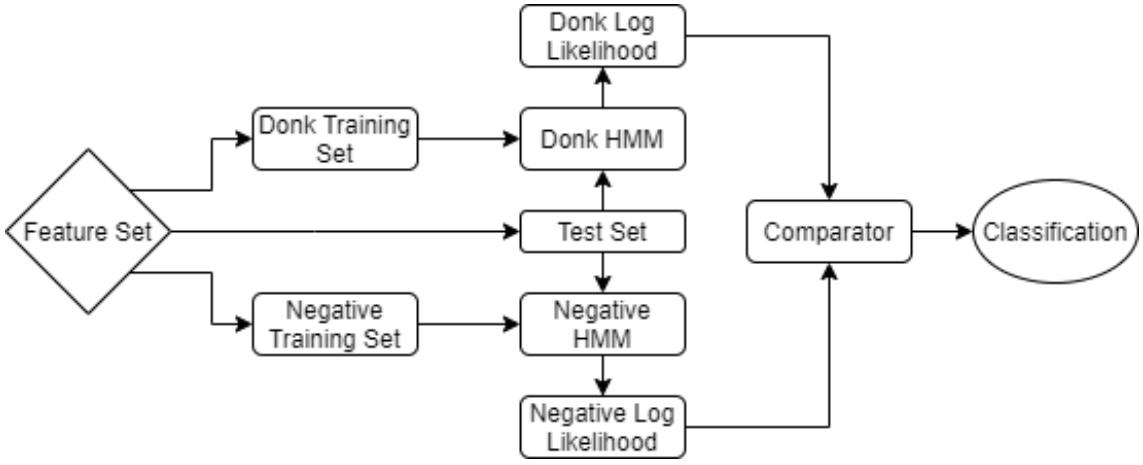


Figure 11: Summary of HMM training

5.6.5 Evolutionary Strategy

The evolutionary strategy algorithm is used to optimize the number of states in the HMM and the K value of the KNN algorithm. The K value is a single scalar; however, as already noted, two HMMs are trained for classification, one for the donk and one for the negative samples. Therefore, the individuals of the population for the HMM evolutionary strategy are vectors, $I_{HMM} \in \mathbb{R}^2 =$ [donk state, negative state]. Therefore, the search space for the HMM algorithm is of a higher dimension, so more iterations of the algorithm are needed for convergence.

For both the KNN and HMM, the reward function is the average cross validation accuracy of the model. Each iteration of the HMM evolutionary strategy takes a long time to run since it has trainable parameters that require a training phase to learn. In order to make it run a bit faster, each score is memorized in a dictionary, such that repeated state scores need not be recalculated.

The number of states and the K value need to be integers; however, the values obtained from the normal distribution to create the population are continuous floating point numbers. Therefore, the population created is discretized such that the individuals can be passed onto their respective algorithms. This leads to two phenomena. Firstly, if the variance of the distribution, σ^2 is chosen too small, then a small range of values are picked from it which when discretized lead to many repeated values. Therefore, σ^2 should be large enough for decent variance in the population and is larger than if the population were continuous. Secondly, if the learning rate, α, is chosen too small, then the algorithm will never move away from the initial guess chosen. This is because a significant movement needs to push the next guess such that when rounded it moves to a new number or vector. Therefore, the α value is also larger than if the population were continuous. Table 6 shows the final hyperparameters chosen for each algorithm. The, σ^2 is larger for the KNN algorithm since the search space has fewer dimensions.

	HMM	**KNN**
Trials	20	10
Population Size	10	10
Learning Rate (α)	5	5
Population Variance (σ^2)	1.5	2

Table 6: Evolutionary strategy hyperparameters

Finally, boundary conditions are set. For the KNN algorithm, $K > 1$. Therefore, each individual of the population is reset to 1 if it goes below it. For the HMM, the number of states, $S > 2$. Therefore, any parameter of any vector in the population that goes below 2 is also reset to 2.

Whilst the algorithm is guaranteed to converge to a local maxima, it is not guaranteed to converge to the global maxima. Therefore, in each case, the algorithm is run twice from different start points in the search space to increase confidence in the obtained final result.

5.7 Benchmarking

The Keras package is dissimilar from the scikit-learn package when compared to the hmmlearn library. Therefore, the benchmark for the CNN is also dissimilar compared to the other models. For the other models, a stratified cross validation is done. Each validation has 10 splits which is shown empirically to yield data sets that do not suffer from excessively high bias and test results that do not suffer from very high score variance. As a result of this, for each validation, the test train split is $1:9$. The root of the score variance, otherwise known as the standard deviation, is also taken and a confusion matrix is generated from which recall and precision scores can be obtained, each metric being offered by scikit-learn. However, for the CNN, the scikit-learn stratified cross validation function does not support one-hot encoded target data. Therefore, a standard cross

validation is done, still with the same 10 splits. Furthermore, instead of obtaining the confusion matrix metrics, the binary cross entropy loss value is tracked together with the accuracy.

5.8 Validation

Once the models have been compared, the models that perform well are saved to permanent memory and applied to all the data of Sol 80, consisting of 5,549,593 sample points.

One issue with validation is how best to split the data. In a naive method, the Sol is simply split into sequential, fixed size, independent chunks. Following this naive framing convention could lead to under classification (poor recall) since the fixed framing would mean that some donks appear on the boundary between one frame and another; so are more difficult to classify. In order to get around this, a rolling window is used such that frames are overlapping. However, now there is an issue with memory. With the naive splitting arrangement, the whole Sol can be split and have feature extraction done without exceeding the RAM availability of the cloud. However, using a rolling window massively increases the total number of frames taken. For example, assume frames of 1500 samples are desired, with the total data having length 150,000. The naive approach leads to 100 frames each with 1500 samples whilst the rolling window approach leads to 148,500 frames each with 1500 samples. However, this assumes the period between windows is 1. In reality, a period of around 500 should suffice such that each 1500 sample is visited three times at different positions. This massively cuts down on the total size of the data to store and process. Formally, the number of frames extracted from a segment using both techniques are as below:

$$X_n = \left\lfloor \frac{L}{F} \right\rfloor \tag{58}$$

$$X_w = \left\lfloor \frac{L - F}{P} \right\rfloor \tag{59}$$

X_n = number of segments from naive framing
X_w = number of segments from rolling framing
F = desired frame length of a single segment
L = length of the Sol
P = period between frames

5.9 Temperature Relationship

First, the raw composite temperature data is converted into differential temperature data. To do this, it is naively split up into 1500 sample frames alike to how the seismic data is split up into frames. The temperature difference within each frame is then taken to model differential temperature aligned with the framed seismic data.

The seaborn package is very useful for automatically plotting kernel density estimations for distributions of sample data. However, the x-axis is fixed to 128 bins. Therefore, if two distributions have slightly different input ranges, the values plotted won't match based on the x-axis. The joint

distribution and temperature gradient distributions have to be plotted with the same temperature gradient x-axis samples such that division of the distributions can be done meaningfully. To accomplish this, linear interpolation is applied:

$$P(T = t_1 \cap D) = P(T = t_0 \cap D) + \left.\frac{\partial f_{D,T}(t)}{\partial t}\right|_{t=t_0} (t_1 - t_0) \tag{60}$$

From this, the joint and temperature gradient distributions are plotted with the same x-axis, so can now be divided to obtain the likelihood of a donk with respect to temperature difference. The logarithm of the resultant distribution is taken to get the final log likelihood.

6 Benchmark

This section lays out the raw benchmark results of the different machine learning algorithms on the different features and an overall comparison is provided comparing feature sets as well as more in depth comparisons between the models used with each feature set.

6.1 Traditional Algorithms

6.1.1 Time Series

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	53.0	65.2	89.4	96.3	94.0
Precision	56.4	71.3	100	100	91.1
Recall	53.8	52.3	78.8	92.7	98.7
Standard Deviation	8.32	11.2	4.17	3.48	4.64

Table 7: Time Series Results

For the time series samples, the entropy based RF algorithm is the clear winner, as expected given the small amount of training data, with highest overall accuracy, perfect precision and the lowest standard deviation. The only competition is the KNN which has higher recall as well as perfect precision; however, it is less reliable given its larger standard deviation

The poor performance of the LR algorithm with respect to the SVM and KNN indicates that whilst distance based metrics are suited to separating the features, the sigmoid function is a poor decision boundary. On the other hand, the poor performance of the BNB indicates a conditional probability model least suited for the time series.

6.1.2 Periodograms

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	99.3	49.7	98.7	97.0	99.7
Precision	100	4.83	100	100	100
Recall	98.7	10	97.3	94.0	99.3
Standard Deviation	2.00	0.645	2.21	2.30	1.00

Table 8: Periodogram Results

The KNN algorithm provides the highest accuracy and low standard deviation as well as perfect precision as the optimal method for the periodogram samples.

It is clear from the SVM, LR and KNN performance that distance based metrics are best able to separate the samples. The LR actually beats the SVM, showing that the sigmoid function is an excellent decision boundary for the frequency data. As expected, again, the RF also performs well as an entropy based model; whilst on the other hand, the BNB performs poorly. The extremely low recall, precision and standard deviation imply the algorithm simply assigns classifications randomly to samples, indicating the set does not suit a probabilistic model.

6.1.3 Energy Ratios

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	92.4	49.7	98.7	98.0	98.3
Precision	98.7	4.83	100	100	100
Recall	86.1	10	97.4	96.0	96.7
Standard Deviation	3.65	0.645	2.20	2.66	2.24

Table 9: Energy Ratio Results

Overall, the SVM performs optimally with top scores across the board bar standard deviation using the energy ratios. However, there is very little distinction between the performance of the SVM with that of the RF and KNN algorithms.

For the energy ratio features, the distance based metrics outperform the probabilistic ones. In fact, the BNB classifies events randomly as shown by the extremely low precision, recall and standard deviation. The entropy based RF algorithm also is a competing metric.

6.1.4 STFT Filterbank Coefficients

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	97.3	89.7	99.7	99.3	99.02
Precision	98.7	93.8	100	100	100
Recall	96.0	86.1	99.3	98.7	98.0
Standard Deviation	1.95	4.31	1.00	1.33	2.08

Table 10: STFT FC Results

The SVM is the optimal model here with the best accuracy, recall and standard deviation as well as perfect precision.

Overall, the features favour the distance based metrics for separation. However, The large disparity between the LR and SVM indicates that a fixed decision boundary is not suited. As usual, the entropy based RF algorithm also performs very well whilst the BNB performs worst, indicating a probabilistic model is unsuitable.

6.1.5 STFT Cepstral Coefficients

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	97.4	87.4	99.7	97.7	98.7
Precision	100	97.9	100	99.3	100
Recall	94.7	76.9	99.3	96.0	97.4
Standard Deviation	2.90	3.13	1.00	3.28	2.19

Table 11: STFT CC Results

The SVM algorithm is optimal for the STFT CCs, again outcompeting at every metric.

Overall, the features favour the distance based metrics for separation. However, The large disparity between the LR and SVM indicates that a fixed decision boundary is not suited. As usual, the entropy based RF algorithm also performs very well whilst the BNB performs worst here, indicating a probabilistic model is again unsuitable.

6.1.6 HHT Filterbank Coefficients

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	63.7	94.4	71.5	95.7	76.2
Precision	73.6	100	66.5	100	81.5
Recall	42.7	88.8	90.0	91.3	67.5
Standard Deviation	10.8	3.95	5.01	5.17	9.27

Table 12: HHT FC Results

In the end, the BNB algorithm comes out ahead for the HHT FCs. Whilst the RF has a higher accuracy and recall, it also has a significantly higher standard deviation. This high standard deviation makes it much less unreliable, and a high recall is not an essential characteristic for model selection.

Distance based metrics do not perform so well for these features, although the SVM performs better than the LR indicating that the naive boundary condition is particularly insufficient. Instead, the HHT FCs appear to favour probabilistic and entropy based models.

6.1.7 HHT Cepstral Coefficients

	Logistic Regression	Bernoulli Naive Bayes	Support Vector Machine	Random Forests	K-Nearest Neighbours
Accuracy	63.0	93.4	68.2	96.7	74.2
Precision	72.2	100	63.1	100	84.2
Recall	43.3	86.8	91.4	93.3	60.3
Standard Deviation	9.32	4.23	6.94	3.27	6.74

Table 13: HHT CC Results

The RF surpasses all other algorithms with the best accuracy, recall and standard deviation along with perfect precision.

Distance based metrics, again, do not perform so well for these features, although the SVM performs better than the LR indicating that the naive boundary condition is particularly insufficient. Instead, the HHT CCs instead appear to favour entropy and probabilistic models.

6.2 Convolutional Neural Network

	HHT Filterbank Coefficients	HHT Cepstral Coefficients	STFT Filterbank Coefficients	STFT Cepstral Coefficients
Accuracy	99.3	99.7	99.3	97.8
Standard Deviation	1.33	1.00	1.31	2.12
Binary Cross Validation Loss	0.0593	0.00959	0.0203	0.0396

Table 14: CNN Results

The STFT CCs clearly perform worst, but for the other features, the accuracy scores are all very close. However, the binary cross-validation loss provides more insight showing that the HHT CCs are clearly best for the CNN. Interestingly, whilst the STFT FCs perform better than the STFT CCs, HHT CCs perform better than the HHT FCs. The STFT result is expected; however, the HHT result is not since the DCT should remove information from the signal. One explanation is that the DCT mainly removes inter-feature information, and the HHT FCs are already very uncorrelated. Therefore, the DCT does not remove much information and the information it does remove might be irrelevant and a hindrance to classification.

6.3 Hidden Markov Model

	Gaussian HMM (No Deltas)	Gaussian HMM (Deltas)
Accuracy	100	100
Precision	100	100
Recall	100	100
Standard Deviation	0	0

Table 15: HMM Results

The HMM is the only model that achieves a perfect score, using the HHT CCs. The model without the delta regressions appended is simpler with fewer parameters so is defined as optimal.

An interesting note from the evolutionary strategy, the maxima are not strict. That is, instead of there being a point being the optimal model, there is a surface surrounding a point which all have optimal scores. Furthermore, generally it is observed that as the number of states increases, the recall of the model also increases at the cost of precision. Overall, the model without the delta regressions was optimal around 6 states both for the donk and negative models. With delta regressions applied, the models become optimal with around 4 states.

Information from the donk model can be extracted to better identify what exactly the model has learnt in order to classify the events. Figure 12 shows the most likely state transition of the model given a donk sample. It can be observed that most of the time, the model stays in state 5, which can be identified as modelling the period of time before and after the main donk event has occurred. There are five other states which are used for when the main donk event occurs, suggesting five different identifiable characteristics that a donk possesses. What exactly these are; however, is unknown; although it is hypothesized that state 2 describes the long tail of the donk.

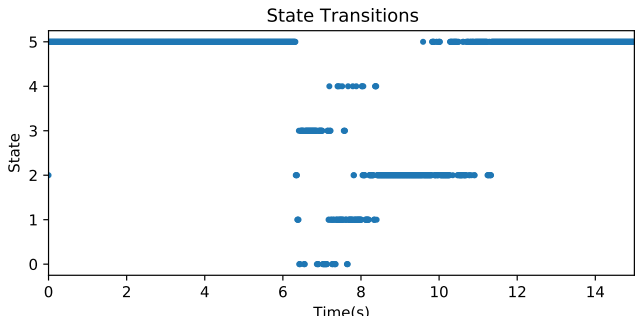


Figure 12: Donk State Transitions

Closer investigation into the state transition matrix shows an irreducible, aperiodic Markov chain; such that the transition matrix itself is primitive. Therefore, it can be neatly described by a stationary distribution, $\pi = [0.0123, 0.0534, 0.292, 0.0395, 0.0503, 0.553]$. This reinforces the earlier conclusion in that the last state is where the model spends the most time with the other states being used to define the main donk event; which is mainly described in state 2.

6.4 Evaluation

The HMM with HHT CCs is clearly the best instrument here achieving perfect classification. Many different algorithm feature combinations also offer impressive results. The most basic are the periodogram features using the KNN, although the STFT features with the SVM and the HHT CCs with the CNN provide the same results as well. The energy ratios fail to achieve the same success, suggesting too much useful information has been removed.

Furthermore, across almost every metric, precision scores are higher than recall scores. That is, the algorithms might not differentiate all the donks, but the ones that are taken are highly likely to actually be donks. This suggests some overlapping decision boundary, with more negatives in the overlap. The recall can interpreted as the level of skepticism the algorithm has.

For the KNN algorithm across all features, the optimal K value obtained via the evolutionary strategy is given by $K = 1$. Generally, a smaller K value is more susceptible to noise. Therefore, this optimal value suggests that the boundary between the classes is very distinct and there are few outliers to confuse classification.

The STFT features appear more competent than the HHT features using the traditional algorithms; with an exception, the HHT features perform better with the BNB and seem to be the only features suited for a probabilistic model. Therefore, these are taken forward for training with the HMM. One potential explanation for the better STFT performance is that the low frequency components of the signal are more important than initially hypothesized; another is that the higher resolution HHT features require more parameters to train, which in turn may require more data to reinforce.

It has already been concluded that the STFT CCs are less correlated in comparison to the STFT FCs whereas there is little difference in the HHT features. Given the condition of uncorrelated features required by the LR and BNB, it is expected that the more decorrelated features should perform better with these algorithms than the more correlted features. For other models, the FCs are expected to perform better since the DCT removes information from the signal. The STFT features generally follow this pattern, with the exception of the BNB where recall plummets significantly and the RF where there is an improvement in performance.

On the other hand, the HHT features do not obey this rule to the same extent. Comparing the FCs and CCs, there is a performance drop for the BNB as expected; however, the LR is difficult to distinguish. Furthermore, the other distance based metrics (KNN and SVM) also see an unexpected performance drop whereas the entropy based RF sees an improvement. Interestingly, this is the exact opposite to how the STFT models compare. In the CNN analysis, it is suggested that for the HHT features, the FCs are already very uncorrelated and so the DCT may not discard much information, with any information it does remove being noisy and a hindrance to classification.

In general, the excellent scores observed are promising but are likely indicative of the low volume of labelled data collected, so potentially do not represent real life performance.

7 Validation

The peridogram KNN and HHT CC HMM are taken for validation on Sol 80. The periodogram KNN is taken since it is the simplest feature set that provides the joint second highest score in the benchmark whilst the HMM is taken as the overall highest score model.

Both the naive splitting and rolling window methods will be used to directly compare which method allows the models to pick out more valid donks, and a post-processing algorithm is developed to boost precision.

7.1 Naive Splitting

With the naive splitting method, 3697 frames are extracted such that the split Sol has the shape $(3697, 3, 1500)$. This splitting arrangement will mainly give a good indication of the precision of the models tested in real life.

7.1.1 K-Nearest Neighbours

Periodograms are extracted from each frame and passed to the KNN for classification. This classified 891 of these frames as donks, so 24.1% of the total frames. On the surface this might seem like an excellent result, in the benchmark the model had 100% precision so this would suggest this model must have simply found a lot of donks. However, Figure 13 shows many problems using the same period of time as in Figure 5. Firstly, not all the clear donks in the period have been labelled. Secondly, the windy features all have significant labelling suggesting the model easily confuses donks for other features.

Overall, when applied to a real life scenario, this model has terrible precision; highlighting the case that the small dataset used for training is not entirely representative of the problem seen in reality. Clearly, the decision criteria that lead to such excellent performance on the benchmark have failed to translate to more general cases for this model.

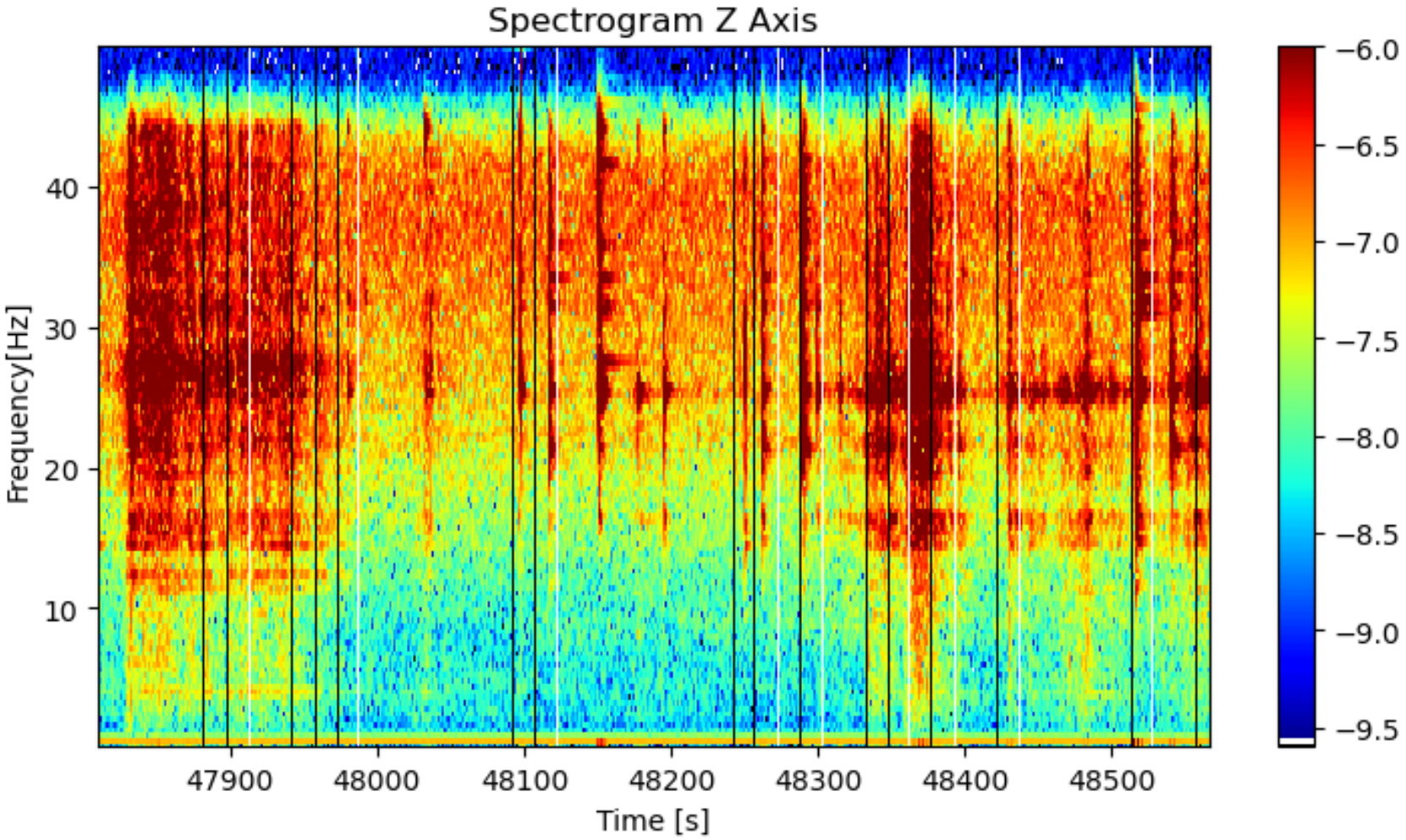


Figure 13: Zoomed in labelled spectrogram of Sol 80 on the Z axis.

7.1.2 Hidden Markov Model

HHT CCs are extracted from each frame and passed to the HMM for classification. This classified 403 of these frames as donks, so 10.9% of the total frames. Figure 15 shows the same period of time in Figure 5. It can be observed that every donk in this period has now been correctly identified.

The first donk identified at the beginning of the set appears to be potentially misclassified. However, upon further investigation, it is seen that this appears to be separated enough from the wind features that it embodies many of the criteria desired in a donk. The time series and spectrogram of this donk classification are shown in Figure 14. This illustrates the main weakness of the model, if a frame clips the beginning or end of a wind like feature, it is often misinterpreted as a donk since without the wider contextual view they look very similar.

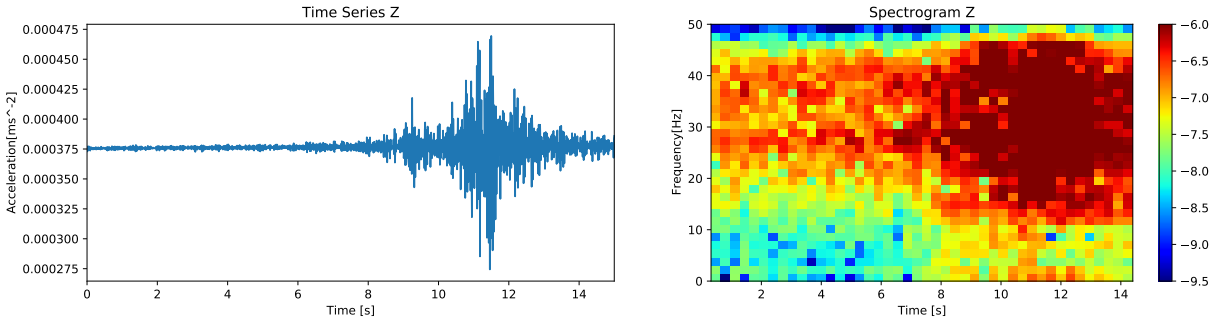


Figure 14: Visualization of frame 3186

Importantly, the body of the longer wind like events after the fist donk classification and around 48,350s have not been labelled. From this, it can be concluded that the model is able to appropriately separate donks from continuous, longer segments of wind like data as desired. Just from this, it is clear to see that the probabilistic decisions of the HMM are able to generalize for wider context cases, and training has been largely successful.

The model is also able to recognise donks where just looking at the Sol spectrogram they may have been missed. For example, in Sol 80 there is a donk classified at 20,350s, in the middle of what appears to be a windy section as shown in the top subfigure of Figure 16; however, when the frame is isolated, it can be seen that the event is indeed a donk. This is potentially the fault of the colourbar scale used which may not have a large enough range to appropriately show the donk in the Sol.

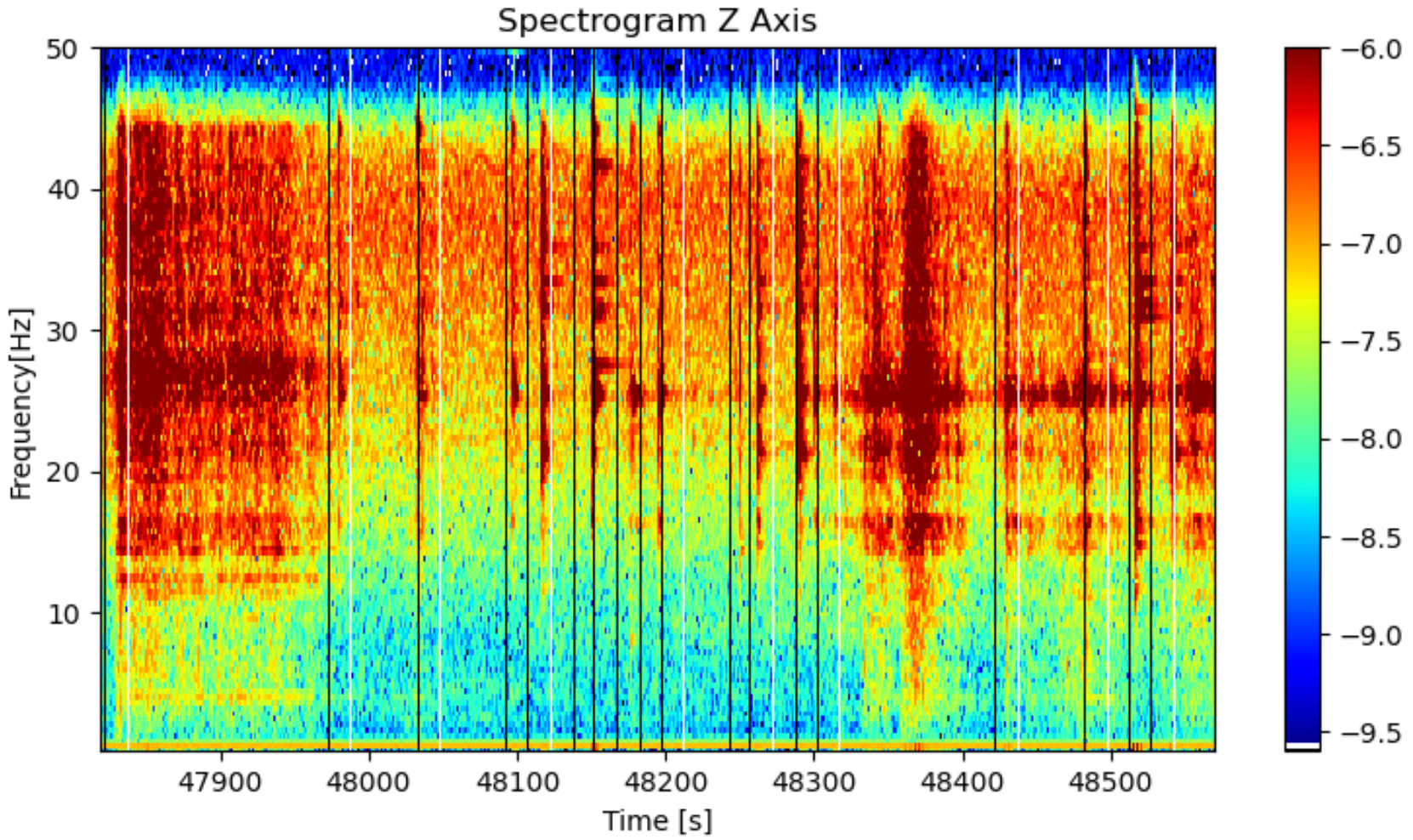


Figure 15: Zoomed in labelled spectrogram of Sol 80 on the Z axis.

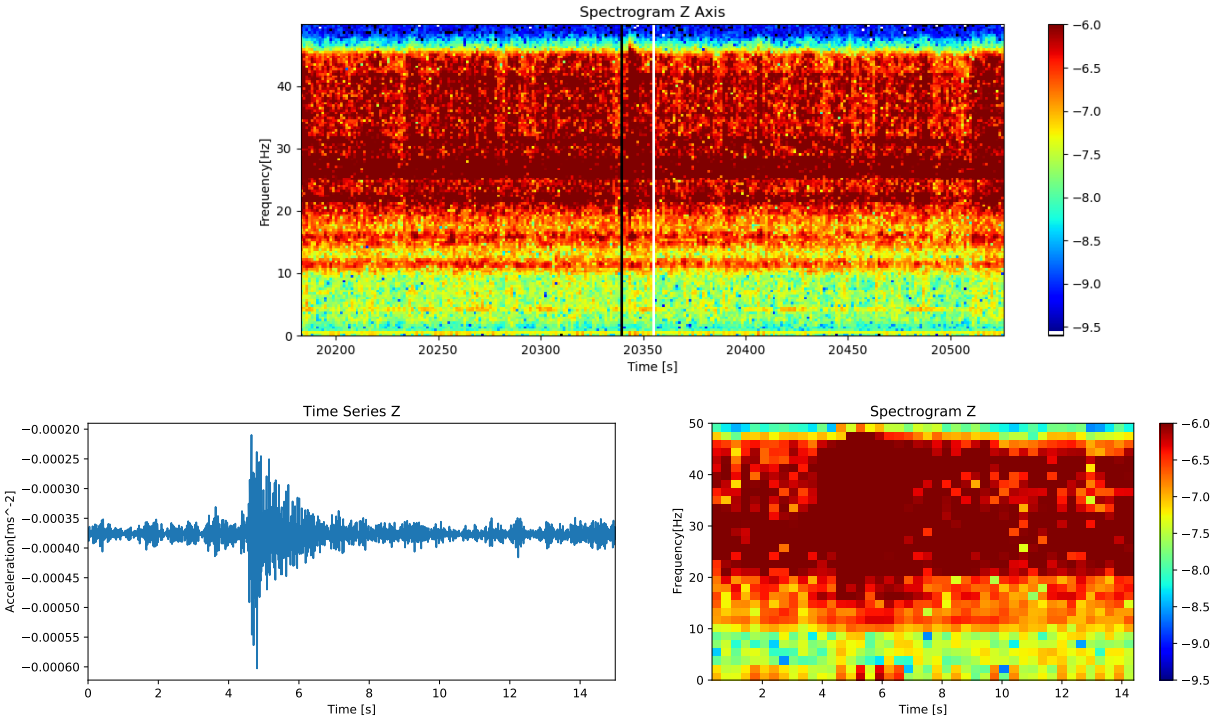


Figure 16: Visualization of frame 1355, in environment (top) and isolated (bottom)

The model also does well in identifying donks in a more noisy environment. Figure 17 shows the visualization of frame 33, which the HMM also identified as a donk. In this period, the SNR is lower than in the training set. Therefore, it is somewhat surprising the model was still able to identify this event appropriately.

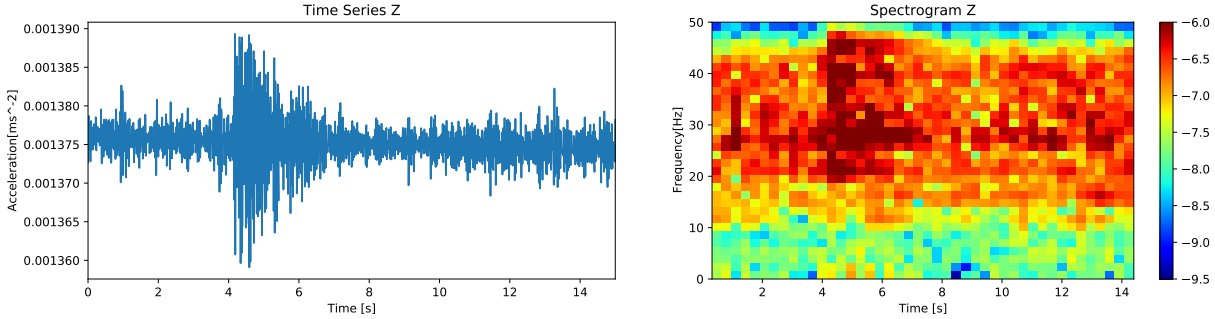


Figure 17: Visualization of frame 33

Furthermore, the model seems adept at identifying donks that exist between frames. Both frames 196 and 197 shown in Figure 18 are an example of this, both being identified as donks where the donk itself appears to occur at the boundary. This is surprising since one of the reasons for choosing the longer frame length was to better center the donks in training, with the tradeoff theoretically being worse classification of donks in the extremems of the frame.

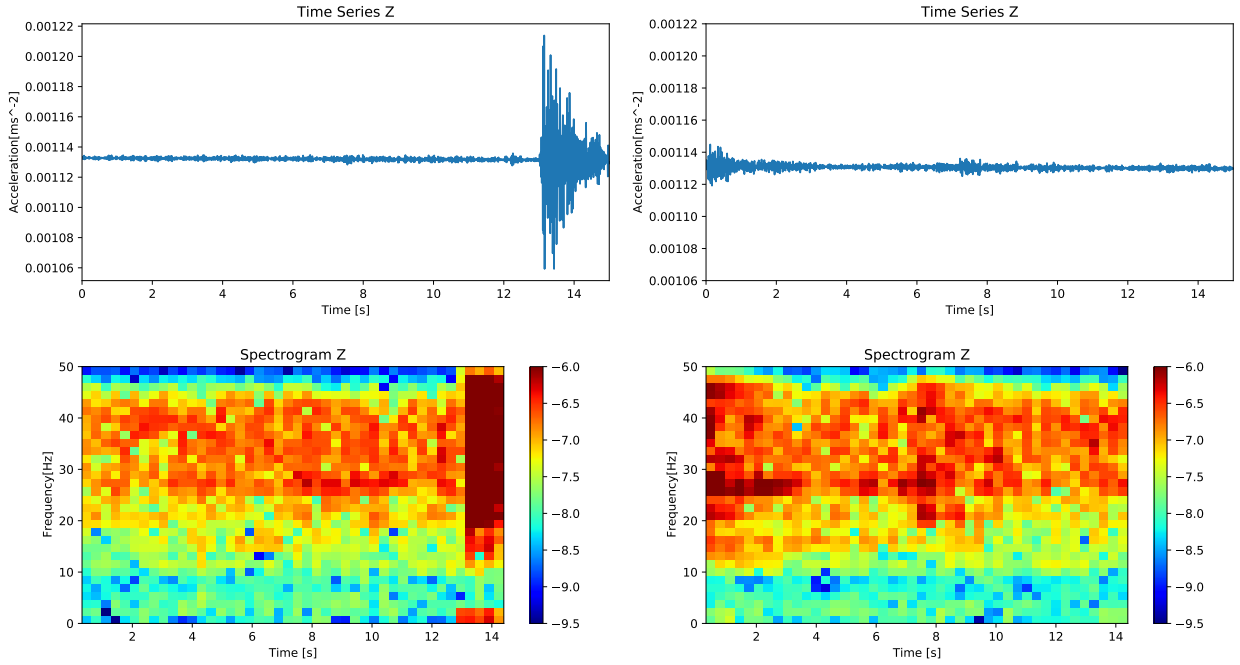


Figure 18: Visualization of frames 196 (left) and 197 (right)

7.2 Rolling Window Splitting

With the rolling window splitting method, 11096 frames are extracted such that the split Sol has the shape $(11096, 3, 1500)$. This splitting arrangement will aim to push the recall of the models tested even higher. However, since the KNN has already shown a poor precision, only the HMM will be tested with this new splitting arrangement.

7.2.1 Hidden Markov Model

Now, 1242 donk frames are identified by the model, 11.1% of the total frames. On the surface, this seems like an improvement over the naive splitting scheme; however, the increase in the number of frames extracted is mainly as a result of the same donks being classified multiple times at different positions. In fact, on average each donk is classified roughly three times, such that the actual number of donks identified is basically the same as with the naive splitting arrangement. An example of this is given by Figure 19 showing three consecutive frames classifying the same donk.

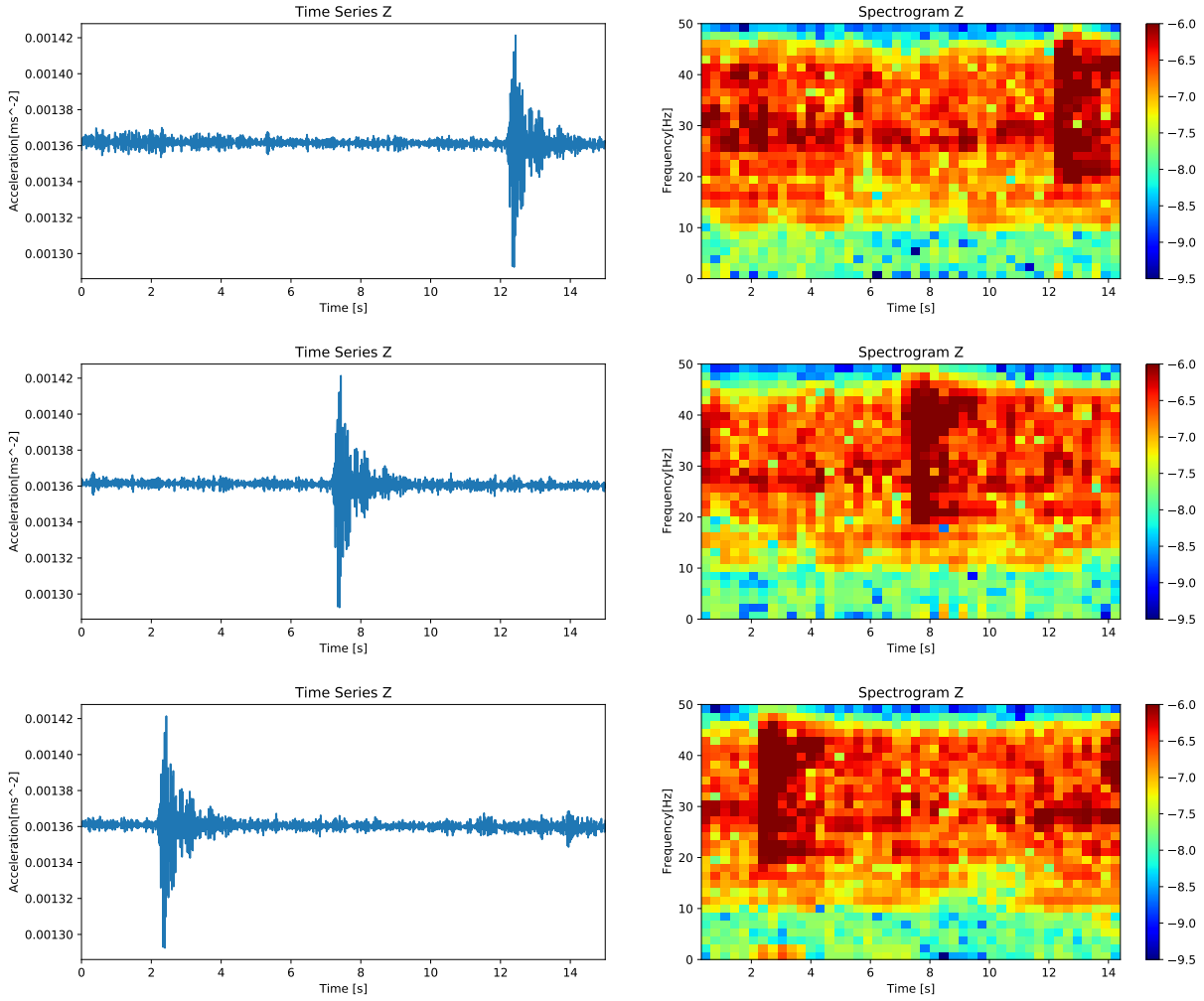


Figure 19: Visualization of frame 126 (top), 127 (middle) and 128 (bottom)

Overall, the rolling window is an unnecessary tool since the recall of classification is not improved. This reinforces the earlier conclusion that the HMM is able to adequately identify donks across different stages of the event, not just when the event is centered in the frame. Furthermore, the rolling window becomes actively undesirable when taking into account the significant extra processing required for the 200% increase in data as well as the increased storage requirements since the processed feature set now takes up 5GB compared to the 2GB of the naive splitting feature set.

7.3 Post-Processing

The conclusion of the validation thus far is that the HMM with HHT CCs performs well with excellent recall and decent precision; although it was noted that it struggles to ignore frames clipping wind like features since these share many similarities with donks; appearing as a short glitch like event with the same concentration of frequency energies. For example, Figure 20 and Figure 21 are both naively classified as donks by the HMM. Both illustrate the same problem, clipping a

wind like feature at the beginning and end of the frame.

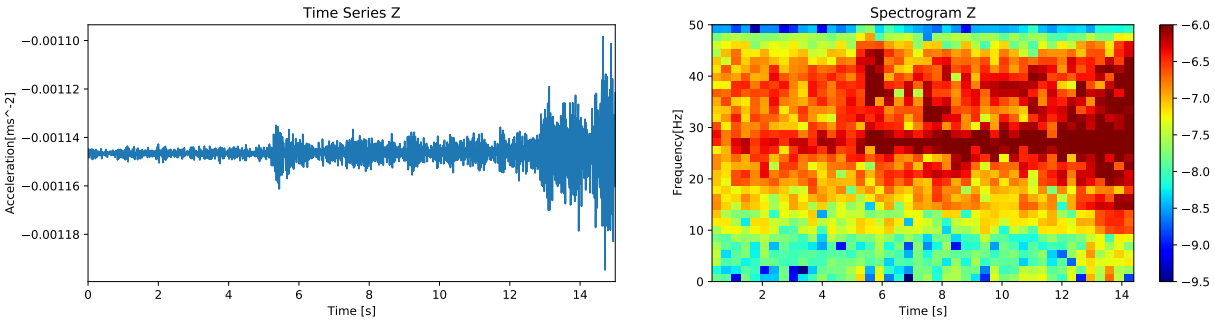


Figure 20: Visualization of frame 2322

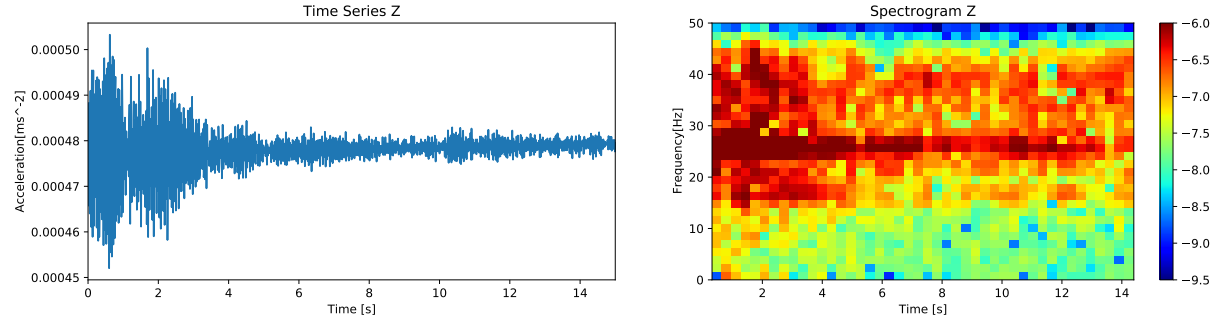


Figure 21: Visualization of frame 3230

In an attempt to alleviate this and improve precision further, a post-processing algorithm is suggested:

1. Find the index of the maximum absolute value of the amplitude of the i_{th} donk frame, D.
2. If this index exists within the first 200 samples of D, shift the frame back by 700 samples.
3. Else, if this index exists within the last 200 samples, shift the frame forwards by 700 samples.
4. Obtain the HHT CCs of the new frame.
5. Pass the new feature frame to the HMM for classification.
6. If classified negatively, remove D from the positive set.
7. Repeat for all donk frames originally classified by the HMM.

Applying this, the clipped event is centred in the frame so the HMM can confirm whether it is a donk or not. Overall, the set of classified donks is reduced to 373, removing 30 (7.44%) of the classifications. Figure 22 and Figure 23 show the adjusted donk classified frames under the post-processing algorithm. In both cases, it is much clearer that the event observed is not a donk, and both are examples of events removed from the donk set.

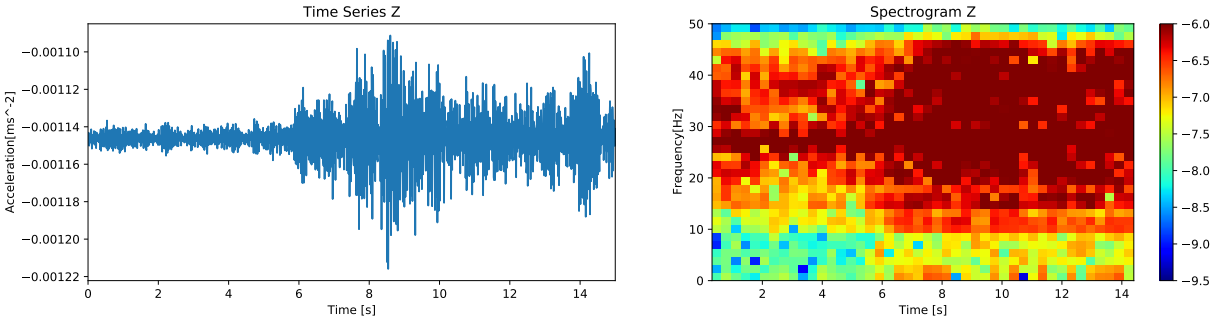


Figure 22: Visualization of adjusted frame 2322

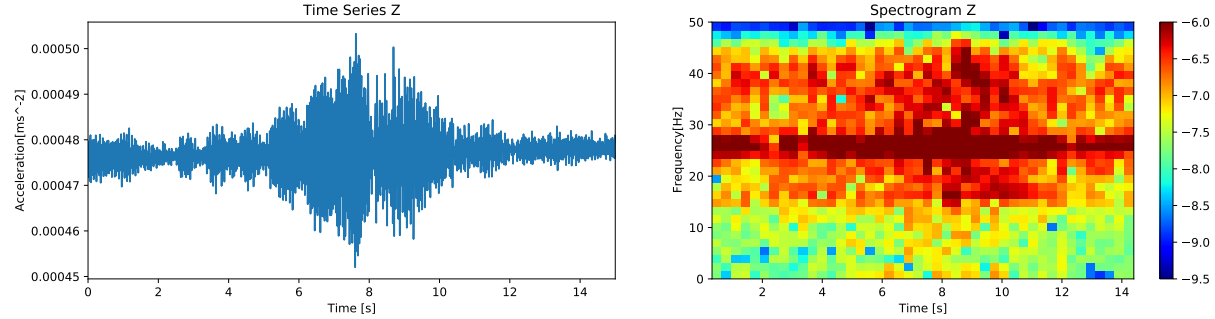


Figure 23: Visualization of adjusted frame 3230

Unfortunately, as a side-effect, three frames with actual donks in them have been removed; in one case because the donk event was right next to a windy feature (Figure 24) and in the other two cases, the donk events were not particularly clear (e.g. Figure 25). Despite this, overall precision has been improved greatly with little impact on recall, a desirable result.

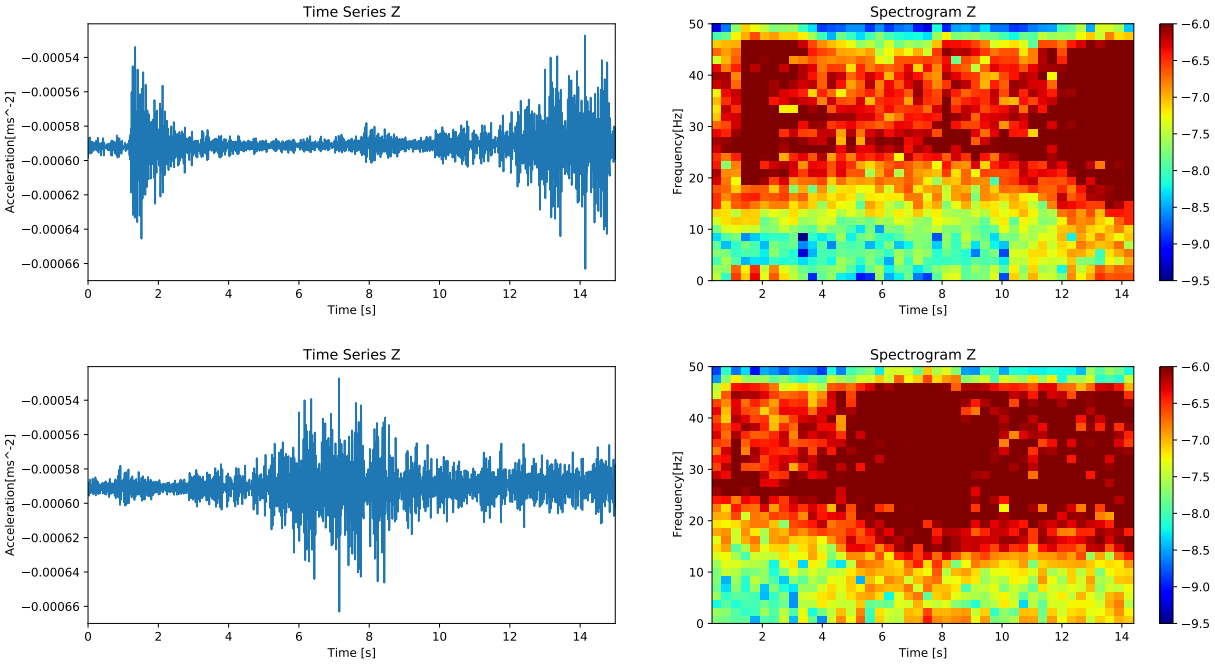


Figure 24: Visualization of frame 2744, original (top) and adjusted (bottom)

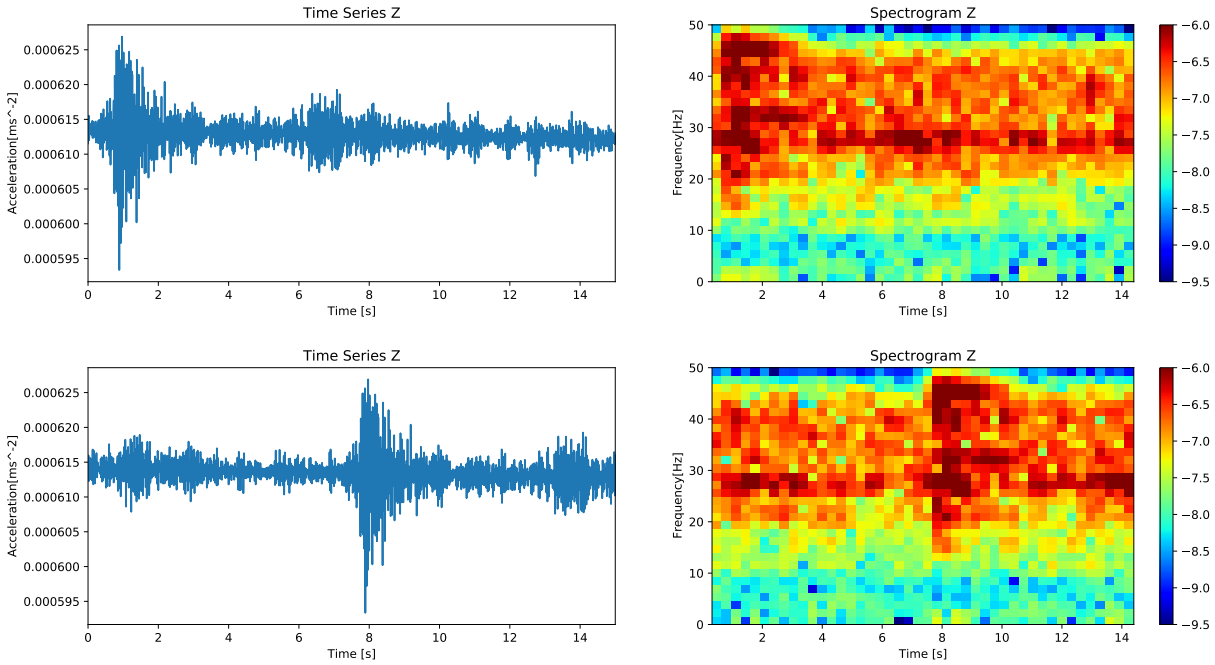


Figure 25: Visualization of frame 569, original (top) and adjusted (bottom)

8 Temperature Relationship

Figure 26 shows the different distributions associated with differential temperature using data from Sol 80. The left subfigure is the raw temperature distribution, whilst the middle is the joint donk differential temperature distribution and the right is the log conditional probability, or likelihood, distribution of a donk given temperature change.

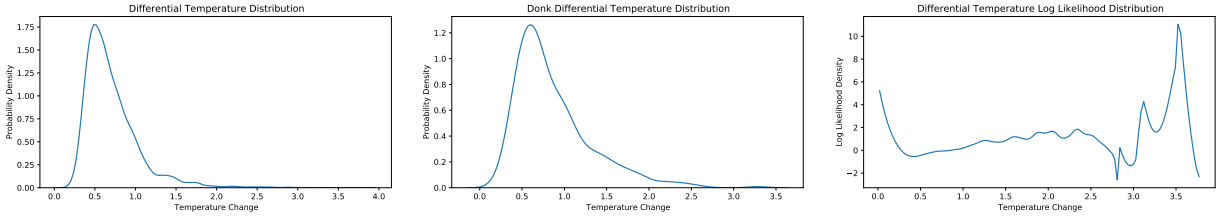


Figure 26: Practical: differential temperature distribution (left), joint differential temperature donk distribution (middle) and donk log likelihood distribution (right)

Now, the goal is to formalize these plots. Just from observation, it can be seen that the gamma distribution appears to be a good fit for the temperature difference distribution, $f_T(t)$, and the joint donk temperature difference distribution, $f_Y(t)$, $Y = D \cap T$. In order to obtain values for the shape, k, and scale, θ, parameters, first the unbiased sample means and variances of both distributions are obtained:

$$[\mu_T, \sigma_T^2] = [0.706, 0.113] = [k_T\theta_T, k_T\theta_T^2] \tag{61}$$

$$[\mu_Y, \sigma_Y^2] = [0.870, 206] = [k_Y\theta_Y, k_Y\theta_Y^2] \tag{62}$$

Therefore:

$$[k_T, \theta_T] = [4.41, 0.16] \tag{63}$$

$$[k_Y, \theta_Y] = [3.68, 0.237] \tag{64}$$

These values are sufficient to define the gamma distributions:

$$f_T(t) = \frac{t^{k_T-1}e^{-t/\theta_T}}{\Gamma(k_T)\theta_T^{k_T}} \tag{65}$$

$$f_Y(t) = \frac{t^{k_Y-1}e^{-t/\theta_Y}}{\Gamma(k_Y)\theta_Y^{k_Y}} \tag{66}$$

The gamma function is defined below:

$$\begin{aligned}\Gamma(k) &= \int_0^\infty x^{k-1}e^{-x}dx \\ &= \frac{1}{k}\prod_{x=1}^{\infty}\frac{(1+\frac{1}{x})^k}{1+\frac{k}{x}}\end{aligned} \tag{67}$$

With this, the final likelihood function can be defined, $f_{D|T}(t)$, from which the log likelihood distribution can be taken:

$$\begin{aligned}f_{D|T}(t) &= \frac{f_Y(t)}{f_T(t)} \\ &= \frac{t^{k_Y-1}e^{-t/\theta_Y}}{\Gamma(k_Y)\theta_Y^{k_Y}} \Bigg/ \frac{t^{k_T-1}e^{-t/\theta_T}}{\Gamma(k_T)\theta_T^{k_T}} \\ &= Ct^{k_Y-k_T}e^{\frac{t}{\theta_T}-\frac{t}{\theta_Y}}\end{aligned} \tag{68}$$

Where:

$$C = \frac{\Gamma(k_T)\theta_T^{k_T}}{\Gamma(k_Y)\theta_Y^{k_Y}} \tag{69}$$

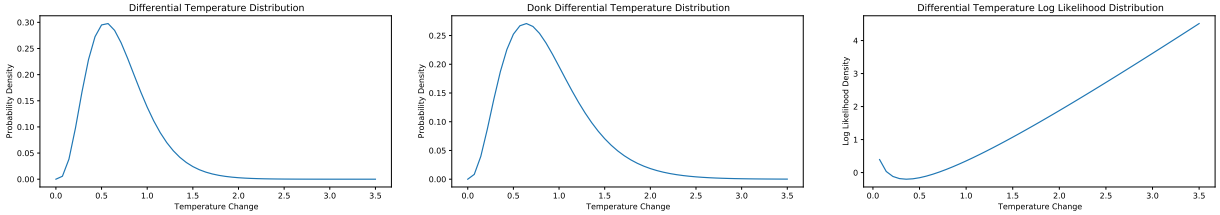


Figure 27: Theoretical: differential temperature distribution (left), joint differential temperature donk distribution (middle) and donk log likelihood distribution (right)

Overall, it can be seen that indeed a large temperature difference increases the likelihood of a donk event occurring.

9 Conclusions and Further Work

In conclusion, the HMM trained with HHT CCs with post-processing offers an impressive model for adequate classification of donk events in the time series using naive splitting given the constraints on the data supplied. Furthermore, a model is formalized that describes the likelihood of donk events given temperature difference; which supports the claim that donks are more likely given a high temperature change observed. Finally, a set of scripts and notebooks have been collected and are ready for use in any future projects.

Most importantly, in order to further improve classification, the next step should be to obtain more labelled data. This will give greater confidence to decision boundaries made by the machine learning techniques and the larger dataset would also be more representative of the variance of events in real life.

In order to remove donk glitches from the time series, a method using HMMs is also suggested. The HMM trained with the donk data can be used to generate appropriate new donk data not before seen. Along the same idea, it is proposed that a HMM can be trained on the frame directly preceding a donk frame to learn the statistical properties of the underlying process without the donk. A frame of data can then be generated by the trained HMM to replace the donk frame. There are a few assumptions required by this method. The seismic data is inherently nonstationary; therefore, it is assumed that a period of two 15s frames is short enough to be considered stationary such that the statistical properties of the underlying process in the preceding frame are the same as the properties of the underlying process excluding the donk event in the donk frame. For the same reason, a single model cannot be trained to generate new data for all the donk frames since these occur throughout the data. Therefore, due to the data being nonstationary, an appropriate model for one portion of the Sol would be inappropriate for another portion.

Bibliography

[1] NASA, "Overview | mission – nasa's insight mars lander." https://mars.nasa.gov/insight/mission/overview/.

[2] M. Ohrnberger, "Continuous automatic classification of seismic signals of volcanic origin at mt. merapi, java, indonesia," *Dissertation*, 07 2001.

[3] C. Benitez, J. Ramírez, J. Segura, J. Ibáñez, J. Almendros, A. Garcia-Yeguas, and G. Cortés, "Continuous hmm-based seismic-event classification at deception island, antarctica," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 45, pp. 138 – 146, 02 2007.

[4] N. Trujillo-Castrillón, C. M. Valdés-González, R. Arámbula-Mendoza, and C. C. Santacoloma-Salguero, "Initial processing of volcanic seismic signals using hidden markov models: Nevado del huila, colombia," *Journal of Volcanology and Geothermal Research*, vol. 364, pp. 107 – 120, 2018.

[5] R. Berešík, "Hilbert-huang transform and its application in seismic signal processing," pp. 1–6, 10 2016.

[6] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[7] C. S. Burrus, *The Cooley-Tukey Fast Fourier Transform Algorithm*. OpenStax CNX, 2009.

[8] B. Boashash, *Time-Frequency Signal Analysis and Processing (Second Edition)*. Oxford: Academic Press, second edition ed., 2016.

[9] N. E. Huang and S. S. P. Shen, *Hilbert–Huang Transform and Its Applications*. WORLD SCIENTIFIC, second edition ed., 2013.

[10] R. Deering and J. F. Kaiser, "The use of a masking signal to improve empirical mode decomposition," in *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 4, pp. iv/485–iv/488 Vol. 4, 2005.

[11] T.-L. Huang, M.-L. Lou, H.-P. Chen, and N.-B. Wang, "An orthogonal hilbert-huang transform and its application in the spectral representation of earthquake accelerograms," *Soil Dynamics and Earthquake Engineering*, vol. 104, pp. 378 – 389, 2018.

[12] Z. Wu and N. E. Huang, "Ensemble empirical mode decomposition: a noise-assisted data analysis method," *Adv. Data Sci. Adapt. Anal.*, vol. 1, pp. 1–41, 2009.

[13] A. Ayenu-Prah and N. Attoh-Okine, "A criterion for selecting relevant intrinsic mode functions in empirical mode decomposition.," *Advances in Adaptive Data Analysis*, vol. 2, pp. 1–24, 01 2010.

[14] H. Fayek, "Speech processing for machine learning: Filter banks, mel-frequency cepstral coefficients (mfccs) and what's in-between." https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html, 2016.

[15] M. Brookes, "Dsp and digital filters." http://www.ee.ic.ac.uk/hp/staff/dmb/courses/DSPDF/00300_Transforms.pdf, 2016.

[16] O. C. Carrasco, "Gaussian mixture models explained." https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95, 2019.

[17] C. Ling, "Probability and stochastic processes." Department of Electrical and Electronic Engineering, Imperial College London.

[18] M. Stamp and A. Professor, "A revealing introduction to hidden markov models," 10 2013.

[19] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017.

[20] R. B. Paranjape, "1 - fundamental enhancement techniques," in *Handbook of Medical Imaging* (I. N. BANKMAN, ed.), Biomedical Engineering, pp. 3 – 18, San Diego: Academic Press, 2000.

[21] S. Saha, "A comprehensive guide to convolutional neural networks — the eli5 way." https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[22] S. Rai, "Forward and backpropagation in convolutional neural network." https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e, 2017.

[23] A. Astolfi, "Optimization." Department of Electrical and Electronic Engineering, Imperial College London.

[24] S. Halkjær and O. Winther, "The effect of correlated input data on the dynamics of learning," in *NIPS*, 1996.

[25] R. Khan, "Where did the binary cross-entropy loss function come from?." https://towardsdatascience.com/where-did-the-binary-cross-entropy-loss-function-come-from-ac3de349a 2019.

[26] A. Datalab, "What is bayesian information criterion (bic)?." https://medium.com/@analyttica/what-is-bayesian-information-criterion-bic-b3396a894be6, 2019.

Appendices

A Negative Training Data

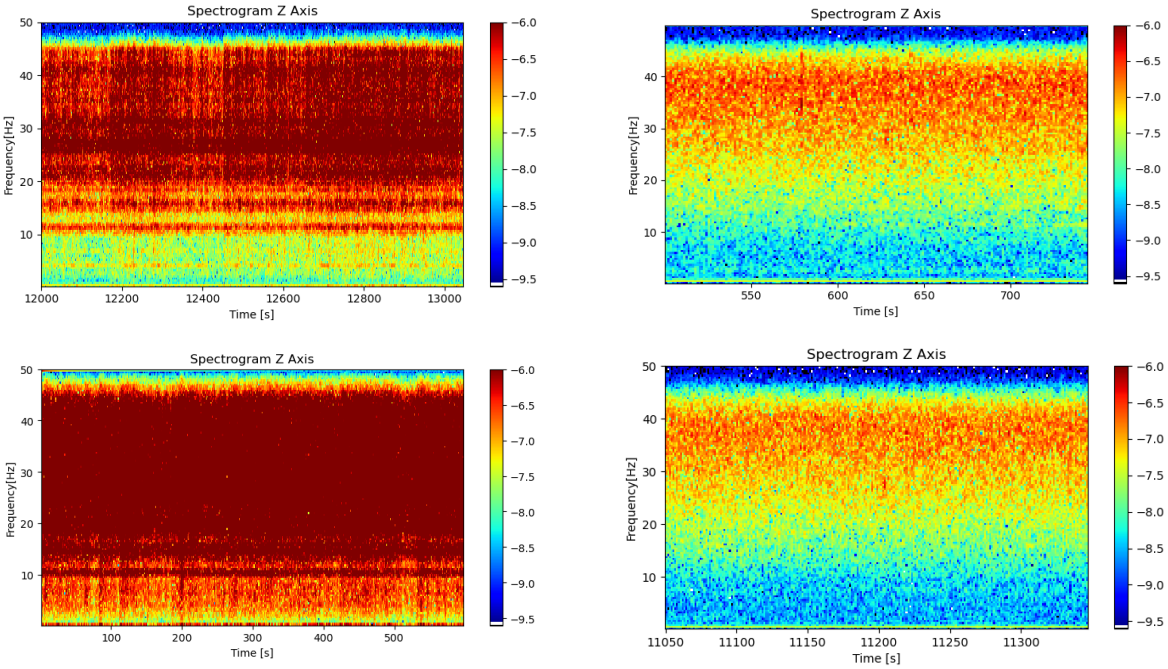


Figure 28: Negative data from Sol 80 (top left), 98 (top right), 194 (bottom left), 195 (bottom right)

B HMM Generated Donk

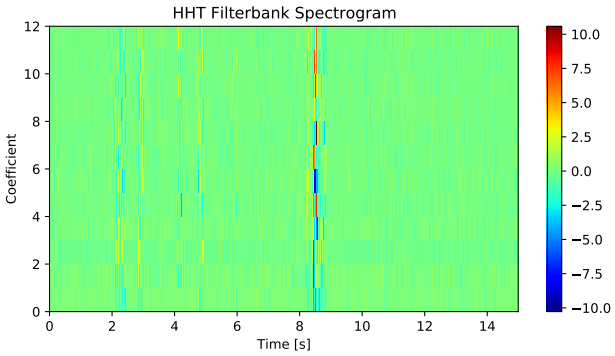


Figure 29: Generated donk HHT CC sample

C More Validation Examples - Sol 98

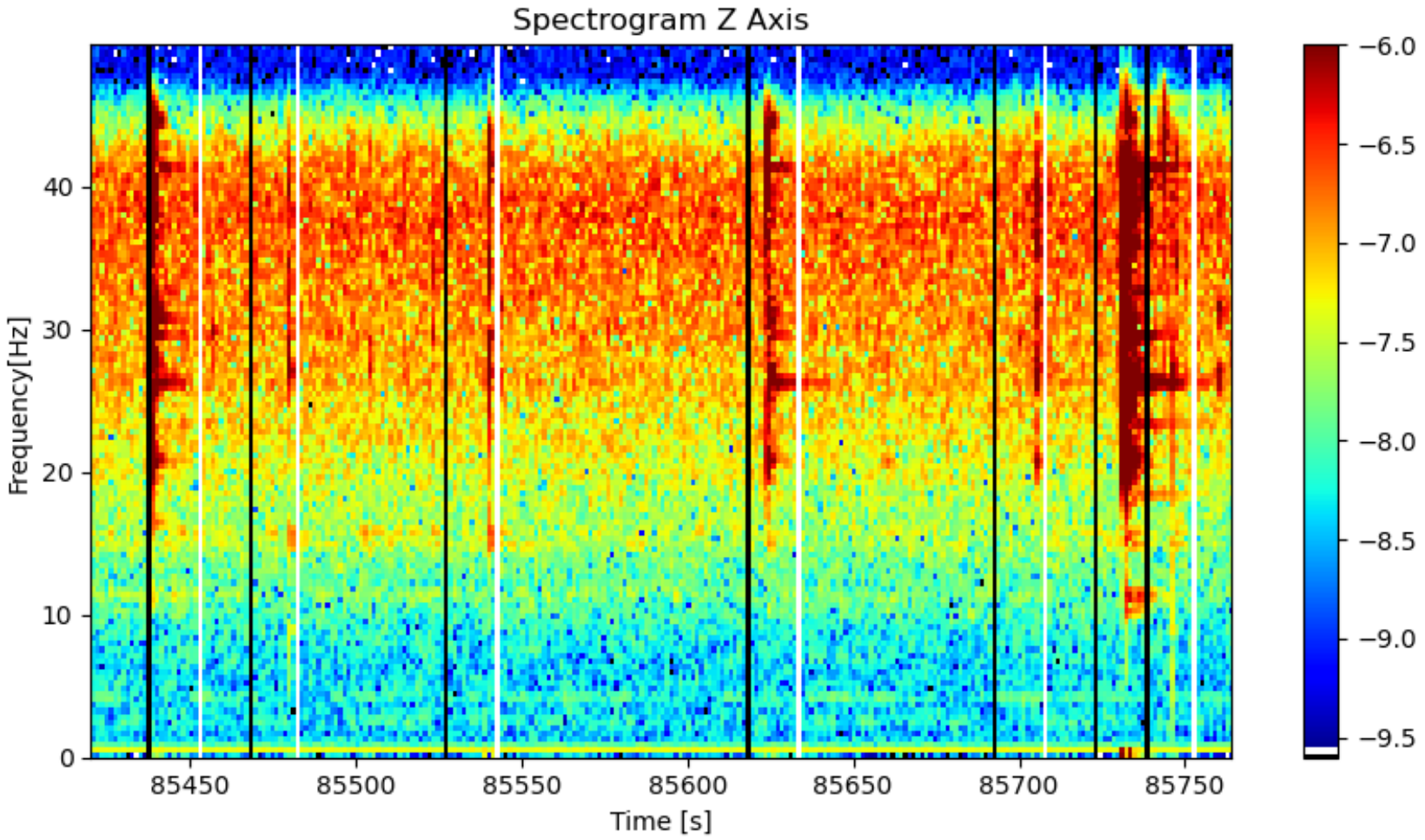


Figure 30: Zoomed in labelled spectrogram of Sol 98 on the Z axis (easy)

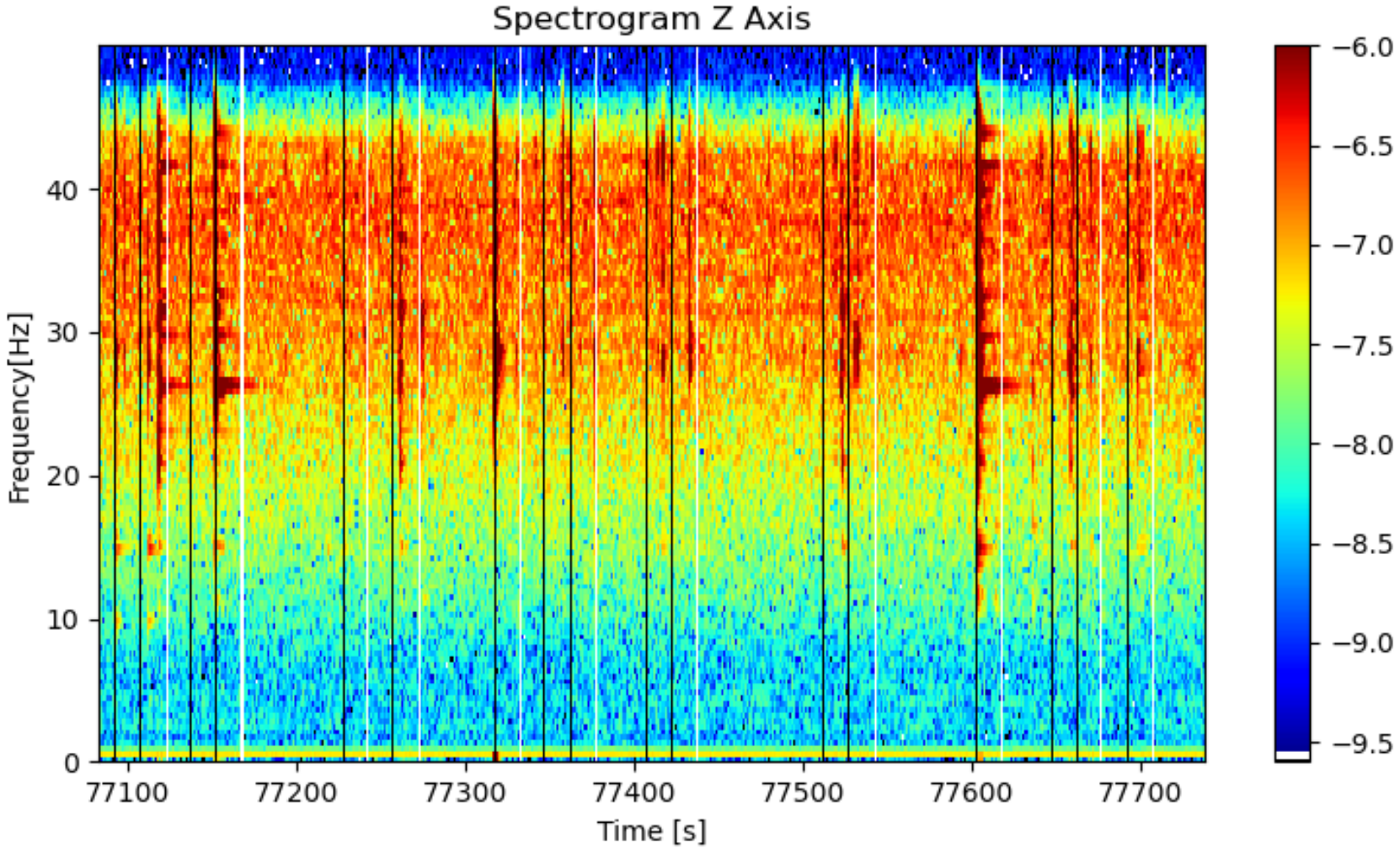


Figure 31: Zoomed in labelled spectrogram of Sol 98 on the Z axis (medium)

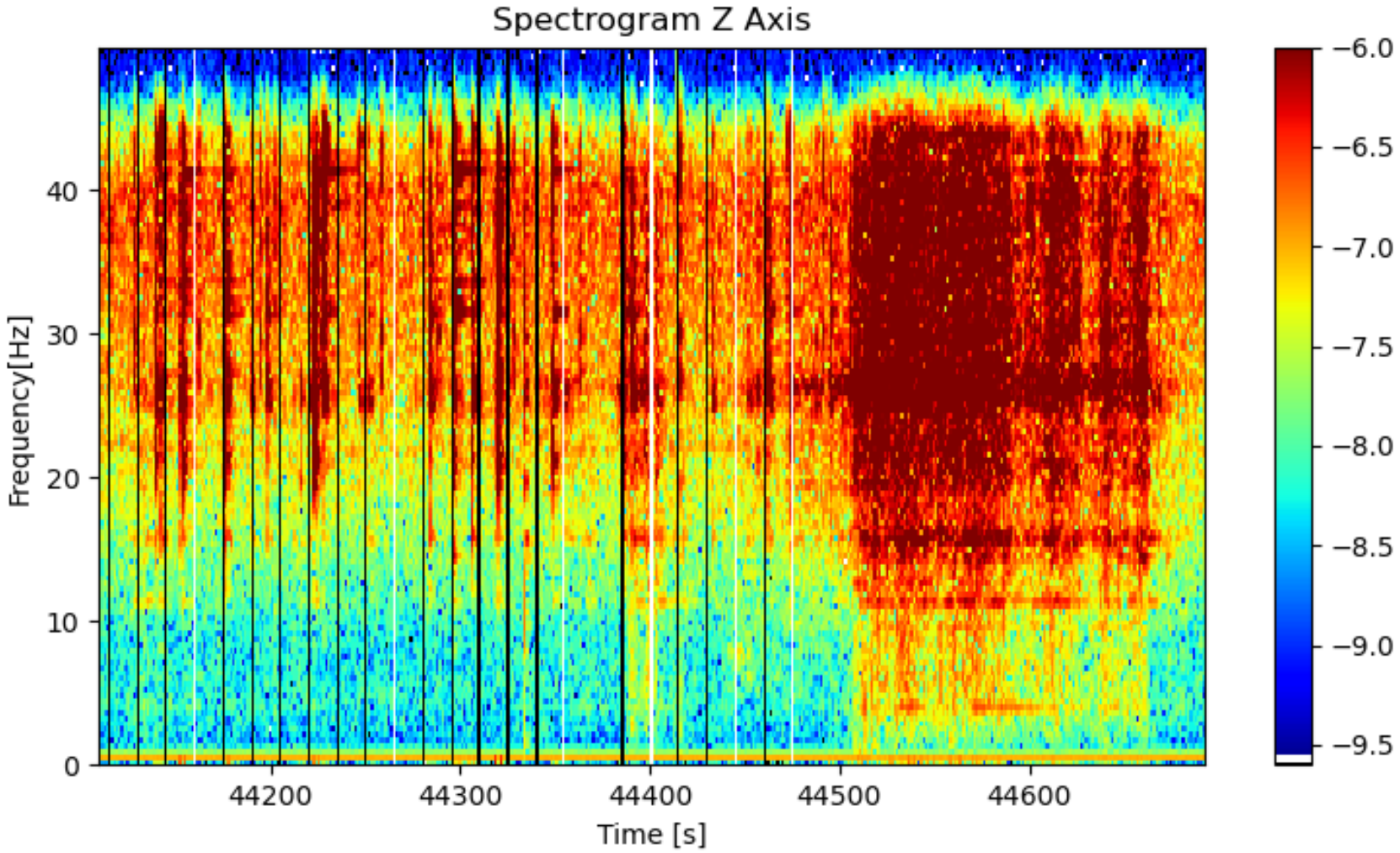


Figure 32: Zoomed in labelled spectrogram of Sol 98 on the Z axis (hard)

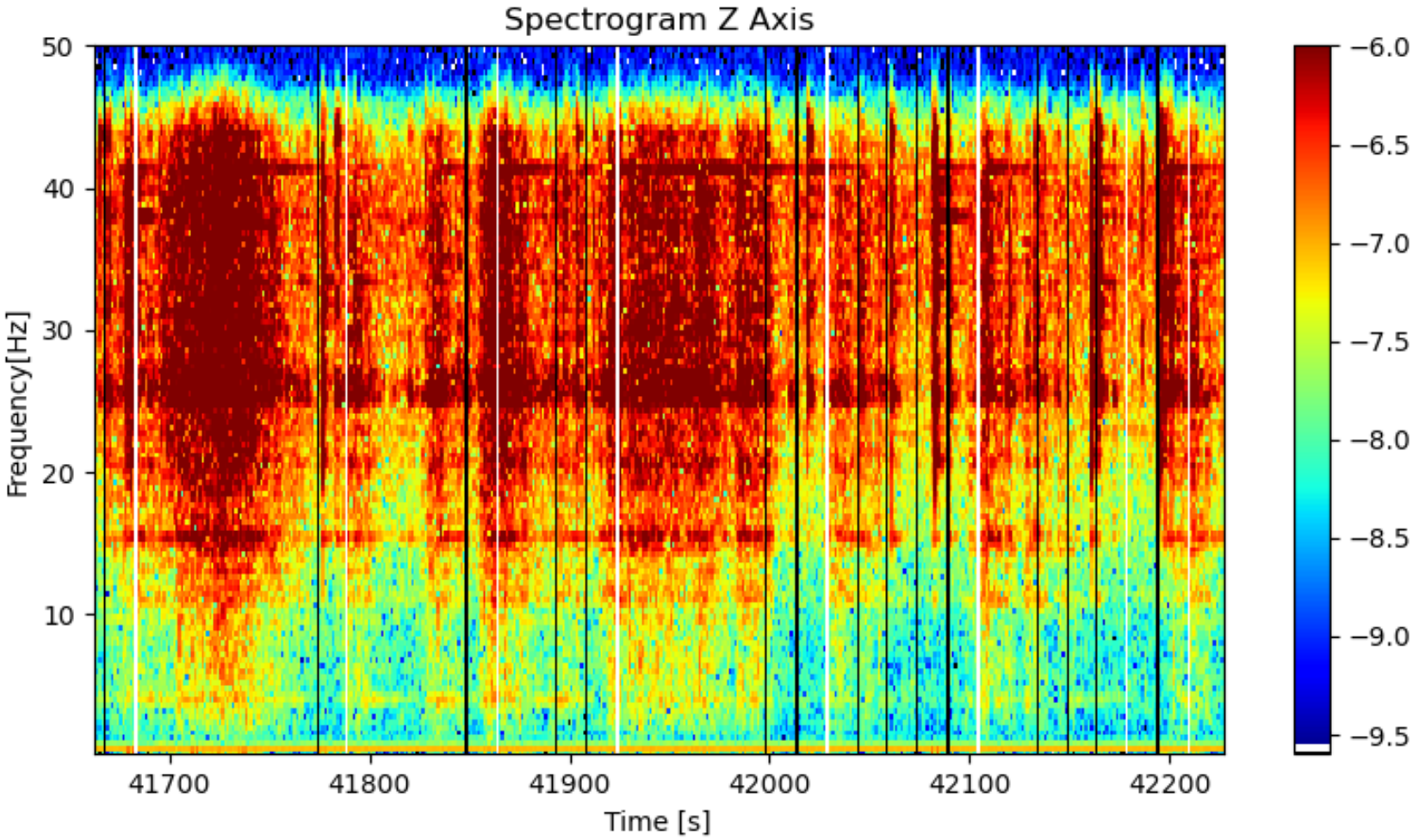


Figure 33: Zoomed in labelled spectrogram of Sol 98 on the Z axis (very hard)

D Donk-Temperature Visualization

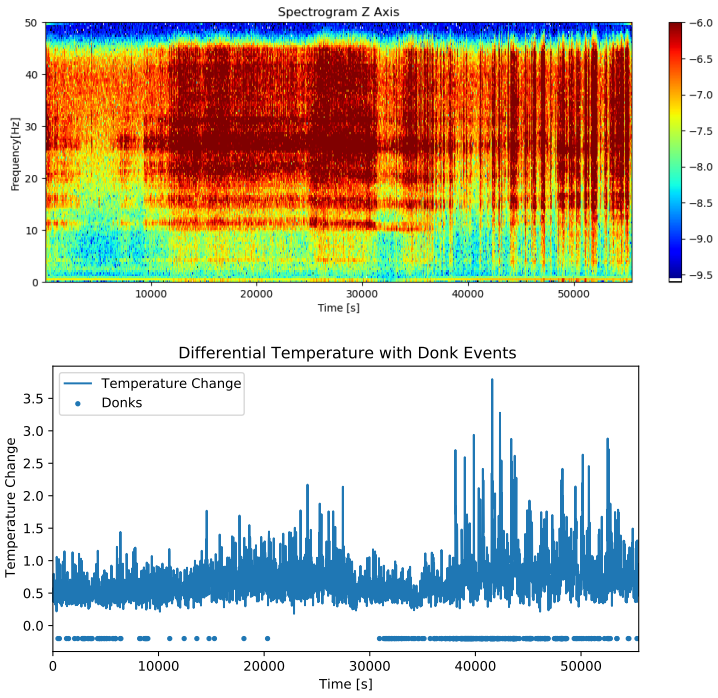


Figure 34: Sol 80 combined visualization

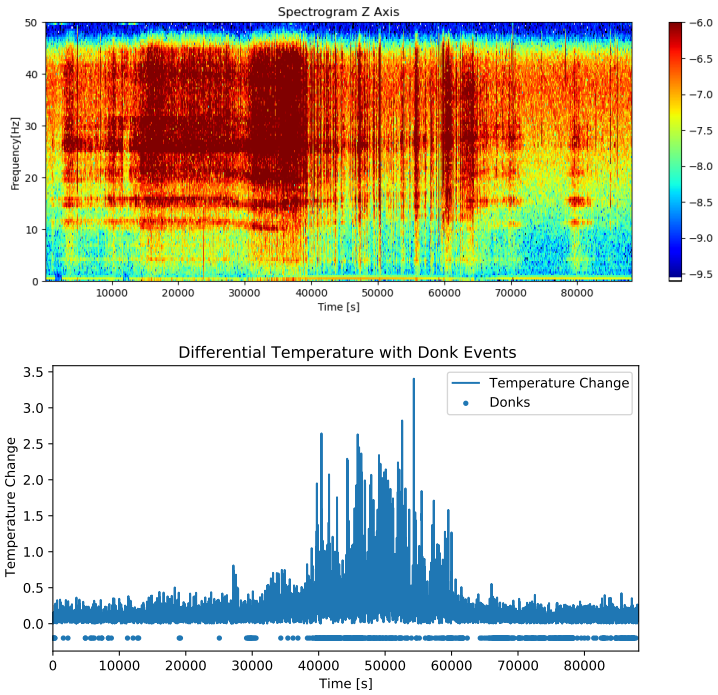


Figure 35: Sol 98 combined visualization

E GitHub Link

https://github.com/09tangriro/InSight-FYP