Searching for Gravitational Waves

Iain Dorrington

A Thesis submitted to
Dalhousie University
for the degree of
Master of Science

February 2019

Declaration

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed: _____
Candidate

Date: _____

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by giving explicit references. A bibliography is appended.

Signed: _____
Candidate

Date: _____

Statement 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be available to outside organisations.

Signed: _____
Candidate

Date: _____

This is the optional
dedication page.
Break lines up
like this.

Abstract

An abstract is required in all theses. Make sure that it fits on one page!

Contents

List of Tables **2**

List of Figures **3**

1 Introduction **5**

2 Gravitational Wave Astronomy in a Nutshell **6**
2.1 General Relativity . . . 6
2.2 Gravitational waves . . . 6
2.2.1 Linearised Gravity . . . 7
2.2.2 Gauge Transformation . . . 7
2.2.3 Physical Effects of Gravitational Waves . . . 8
2.3 Gravitational Waves Sources . . . 10
2.4 How to Detect Gravitational Waves . . . 10
2.4.1 How to find something interesting in all that noise . . . 10

3 The Compact Binary Coalescence Search **11**
3.1 PyCBC overview . . . 11
3.2 PyGRB . . . 11
3.3 O2 PyGRB Search . . . 11
3.4 The O3 PyCBC search . . . 11

4 Undmodelled Search **12**
4.1 Xpipeline . . . 12
4.2 Subthreshold GRBs . . . 12

5 Machine Learning for GW Astronomy **13**
5.1 Introduction . . . 13
5.2 Xpipeline . . . 13
5.2.1 Burst Search Background . . . 14
5.2.2 Standard Likelihood . . . 17
5.2.3 Null Energy . . . 18
5.2.4 Incoherent Energy and Background Rejection . . . 19
5.3 Machine Learning . . . 19
5.3.1 Supervised Machine Learning . . . 20
5.3.2 Boosted Decision Trees . . . 20
5.3.3 Data Preprocessing . . . 23
5.3.4 Optimisation and Validation . . . 25

List of Tables

5.1 An example of MVA training data. Each event has a label and several attributes. 20

List of Figures

5.1 **Xpipeline Time-Frequency Map** This figure shows a time-frequency map from Xpipeline for a $1.4 - 10M_{\odot}$ NSBH merger. The top figure shows the E_{+} energy and the bottom figure shows the top 1% of pixels. 14
5.2 **Xpipeline Cut** This figure shows Xpipeline making a cut to eliminate many spurious signals. 15
5.3 **A decision tree example** To determine if a trigger is a signal or noise event the tree makes a series of cuts on the attributes x[N]. If the inequality in a node is true, then the next node is the branch to the left. Otherwise the next node is the one to the right. 21
5.4 **Visualising the Classifier** In the bottom plot you can see the value for log(Enull) and log(Inull) for all the signal and background training data used to build the classifier. We chose one of these events at random (indicated by the star) and varied Enull and Inull to see how it changed the MVA score, indicated by the colour in the top plot. As you can see, increasing Inull and decreasing Enull leads to the event being more likely to be classed as a signal. This is akin to the xpipeline cut shown in fig.5.2. 22

Chapter 1

Introduction

Chapter 2

Gravitational Wave Astronomy in a Nutshell

2.1 General Relativity

Christoffel Symbol

$$\Gamma^{\lambda}_{\mu\nu} = \frac{1}{2} g^{\lambda\rho}[\partial_\nu g_{\mu\rho} + \partial_\mu g_{\nu\rho} - \partial_\rho g_{\mu\nu}] \tag{2.1}$$

Riemann Curvature Tensor

$$R^{\mu}_{\lambda\alpha\beta} = \partial_\alpha \Gamma^{\mu}_{\lambda\beta} - \partial_\beta \Gamma^{\mu}_{\lambda\alpha} + \Gamma^{\mu}_{\nu\alpha}\Gamma^{\nu}_{\lambda\beta} - \Gamma^{\mu}_{\nu\beta}\Gamma^{\nu}_{\lambda\alpha} \tag{2.2}$$

Ricci Tensor

$$R_{\mu\nu} = g^{\alpha\beta} R_{\alpha\mu\beta\nu} = R^{\beta}_{\mu\beta\nu} \tag{2.3}$$

Ricci Scalar

$$R = g^{\alpha\beta} R_{\alpha\beta} = R^{\beta}_{\beta} \tag{2.4}$$

The Einstein Equations

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu} \tag{2.5}$$

Alternative form of the Einstein equations

$$R_{\mu\nu} = \frac{8\pi G}{c^4} \left(T_{\mu\nu} - \frac{1}{2} T g_{\mu\nu} \right) \tag{2.6}$$

2.2 Gravitational waves

General Relativity shows that spacetime can curve and move. From this, it may seem obvious that waves can travel though spacetime. But this is not a trivial fact; Einstein himself, having introduced the concept of gravitational waves in 1916, later claimed they could not exist. In this section, we will show that gravitational waves do exist. We will do this by considering a perturbation of the flat Minkowski metric. We

will then calculate the Ricci Tensor and Ricci Scalar for the perturbed metric and, with a clever choice of Gauge transformation, see that these yield a wave solution to Einstein's Equations.

2.2.1 Linearised Gravity

Let $g_{\mu\nu}$ be the metric given by adding a small perturbation $h_{\mu\nu}$ to the Minkowski metric $\eta_{\mu\nu} = \text{diag}(-1, 1, 1, 1)$. We write this metric as

$$g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}, \qquad |h_{\mu\nu}| \ll 1 \ . \tag{2.7}$$

To first order in h, we calculate the Ricci tensor (2.3)

$$R_{\mu\nu} = \eta^{\alpha\beta} R_{\alpha\mu\beta\nu} = \frac{1}{2}\left(\partial_\mu \partial^\alpha h_{\alpha\nu} + \partial_\nu \partial_\alpha h^\alpha_\mu - \Box h_{\mu\nu} - \partial_\mu \partial_\nu h\right) \tag{2.8}$$

and the Ricci scalar (2.4)

$$R = \eta^{\mu\nu} R_{\mu\nu} = \partial_\mu \partial_\alpha h^{\mu\alpha} - \Box h, \tag{2.9}$$

where $\Box = \partial_\mu \partial^\mu$ is the d'Alembertian operator and $h = h^\mu_\mu$ is the trace of h. From these, we find the resulting Einstein tensor is also linear in h

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu}. \tag{2.10}$$

2.2.2 Gauge Transformation

With the following Gauge transformation, we can simplify the expression for the linear Einstein tensor

$$x^{\mu'} = x^\mu + \chi^\mu(x), \qquad \chi \ll x \ . \tag{2.11}$$

As χ is small, we have $\partial_\mu \chi^\nu \ll 1$. This gives us

$$\frac{\partial x^\mu}{\partial x^{\alpha'}} = \delta^\mu_\alpha - \partial_\alpha \chi^\mu + \mathcal{O}(|\partial\chi|^2) \ . \tag{2.12}$$

Applying these results to our metric (2.7) we find

$$g_{\alpha'\beta'} = \frac{\partial x^\mu}{\partial x^{\alpha'}} \frac{\partial x^\nu}{\partial x^{\beta'}} g_{\mu\nu} = g_{\alpha\beta} - \partial_\alpha \chi_\beta - \partial_\beta \chi_\alpha \ . \tag{2.13}$$

Subtracting the Minkowski metric from each side, we find

$$h_{\alpha'\beta'} = h_{\alpha\beta} - \partial_\alpha \chi_\beta - \partial_\beta \chi_\alpha \ . \tag{2.14}$$

We have some freedom in choosing our χ, so we impose the *harmonic gauge condition*

$$\partial_\mu h^\mu_\nu = \frac{1}{2} \partial_\nu h, \tag{2.15}$$

where $h = h^\lambda_\lambda$. We can always choose a χ such that this is true. To see this, first note that $\partial'_\mu = \partial_\mu - (\partial_\mu \chi^\lambda)\partial_\lambda$. From this we find

$$(\partial'_\mu h^\mu_\nu - \frac{1}{2}\partial'_\nu h') \approx (\partial_\mu h^\mu_\nu - \frac{1}{2}\partial_\nu h) - \partial^2 \chi_\nu \ . \tag{2.16}$$

Thus, if we are given an h such that 2.15 is not true, we can choose a χ such that

$$\partial^2 \chi_\nu = (\partial_\mu h^\mu_\nu - \frac{1}{2}\partial_\nu h) \ . \tag{2.17}$$

Using 2.15, we can simplify the Ricci tensor and scalar

$$R_{\mu\nu} = -\frac{1}{2}\Box h_{\mu\nu} \tag{2.18}$$

$$R = -\frac{1}{2}\Box h \ . \tag{2.19}$$

Using these in the linearised Einstein equations (2.10) gives us

$$\Box h_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu}\partial^2 h = -\frac{16\pi G}{c^4} T_{\mu\nu} \ . \tag{2.20}$$

Alternatively, we can use 2.6 to write this as

$$\Box h_{\mu\nu} = -16\pi G \left(T_{\mu\nu} - \frac{1}{2}\eta_{\mu\nu} T \right) . \tag{2.21}$$

In a vacuum, the right hand side of this equation becomes zero, and we can recognise as the relativistic wave equation.

2.2.3 Physical Effects of Gravitational Waves

The plane wave solution for the vacuum wave equation is

$$h_{\mu\nu}(x) = \epsilon_{\mu\nu} e^{i k_\alpha x^\alpha} \tag{2.22}$$

where $\epsilon_{\mu\nu}$, the polarisation tensor for the gravitational wave, is symmetric and constant, and k^α is the 4-wave-vector given by $k^\alpha = (\omega, \vec{k})$. Subsituting this into the vacuum wave equation, we find

$$k^2 = -\omega^2 + \vec{k} = 0 \ . \tag{2.23}$$

Hence, gravitational waves travel at the speed of light.

The polarisation tensor is not arbitrary: It must satisfy the wave equation and the harmonic gauge condition. Putting the wave solution 2.22 into the harmonic gauge condition 2.15, we find that gravitational waves are transverse

$$k^\mu \epsilon_{\mu\nu} = 0 \ . \tag{2.24}$$

We can impose further gauge conditions as long as the harmonic gauge (and hence 2.24) is not violated. As any transformation with $\partial^2\chi = 0$ will satisfy 2.17, and hence harmonic gauge condition, we express χ as

$$\chi_\nu = X_\nu e^{ikx} \ . \tag{2.25}$$

Using 2.25 and 2.22 in the transformation equation 2.14, we find the transformation equation for the polarisation tensor

$$\epsilon'_{\mu\nu} = \epsilon_{\mu\nu} - ik_\mu X_\nu - ik_\nu X_\mu \ . \tag{2.26}$$

Taking the trace of this, we find

$$\epsilon'^{\mu}_{\mu} = \epsilon^{\mu}_{\mu} - 2ik^\mu X_\mu \ . \tag{2.27}$$

Thus, we can impose the further gauge condition that the polarisation matrix be traceless by choosing coordinates such that $\epsilon^\mu_\mu = 2ik_\mu X^\mu$, and so fixing one element of X^μ. We can fix the other elements of X_μ by setting $\epsilon_{i0} = 0$ for $i = 1, 2, 3$. We do this using 2.26 to find

$$\epsilon'_{i0} = \epsilon_{i0} - ik_i X_0 - ik_0 X_i \ . \tag{2.28}$$

Now we see that by setting $\epsilon_{i0} = ik_i X_0 - ik_0 X_i$, we have $\epsilon_{i0} = 0$. As the polarisaton tensor is symmetric, we have $\epsilon^0_i = 0$ as well.

These conditions give us that $\epsilon_{00} = 0$ as well. Using the fact that the polarisation matrix can be assumed traceless ϵ^μ_μ, the wave solution 2.22 reduces the harmonic gauge condition 2.15 to

$$\partial_\mu h^\mu_\nu = 0 \ . \tag{2.29}$$

As $\epsilon_{i0} = 0$, in the $\nu = 0$ case, only the $\mu = 0$ term is non-zero. Thus we are left with

$$\partial_\mu h^\mu_0 = \epsilon^0_0 ik_0 e^{ik_\alpha x^\alpha} = 0 \ . \tag{2.30}$$

Which implies $\epsilon^0_0 = 0$. As the polarisaton tensor is symmetric, we have $\epsilon^3_3 = 0$ as well.

We can make the polarisation tensor simpler by assuming the wave is traveling in the z-direction, that is $\vec{k} = (0, 0, \omega)$. The transverse condition 2.24 then forces all the z-components of the polarisation matrix to be zero, $\epsilon^3_i = \epsilon^i_3 = 0$. The wave solution 2.22 now looks like

$$h_{\mu\nu}(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & a & b & 0 \\ 0 & b & -a & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} e^{(i\omega(z-t))} \ .$$

2.3 Gravitational Waves Sources

- GW generation
- Sources
  - BBH - Multiple found already
  - BNS - One found and Hulse-Taylor
  - NSBH
  - GRB
  - SN
  - Unknown

2.4 How to Detect Gravitational Waves

Can cite soaulson and the noises paper

- Michelson Interferometer (cite R.Weiss 1972 paper to look smart)
- Measure change in output power due to interference in the arms
- Noise sources and how they are mitigated
  - Fundamental noise
  - Environmental noise
  - Instrumental noise
  - Noise budget

2.4.1 How to find something interesting in all that noise

- Autocorrelation function
- PSD and ASD (tie back to the noise budget)
- Matched filtering (for 1 detector)
- Unmodelled search (for 1 detector)
- Network statistics (coincident and coherent)

Chapter 3

The Compact Binary Coalescence Search

3.1 PyCBC overview

- Overview of pipeline based on technical paper
- Consistency checks
- The O2 search (coincident)

3.2 PyGRB

- Has sky position
- Coherent

3.3 O2 PyGRB Search

3.4 The O3 PyCBC search

- More detectors strengthens the case for coherent search
- Hierarchical search, using coincident search to find location

Chapter 4

Undmodelled Search

4.1 Xpipeline

4.2 Subthreshold GRBs

Chapter 5

Machine Learning for GW Astronomy

5.1 Introduction

The Laser Interferometer Gravitational Wave Observatory (LIGO) and Virgo collaborations operate ground-based Gravitational Wave (GW) detectors. They have detected signals of astrophysical origin, including the merger of a Binary Neutron Star (BNS) system and multiple Binary Black Hole (BBH) merger signals. As these sources have a known signal morphology, they can be found using a highly sensitive matched-filter search, as discussed in chapter 3. But some of the most interesting possible sources, such as core collapse supernova, do not have a known waveform morphology. For this reason, it is important to develop unmodelled searches as well.

In unmodelled searches we look for coherence between the data streams of multiple interferometers. We will consider the case of a Burst search where the sky position of the candidate source is known. This allows us to calculate the relative time of arrival in each detector and the relative signal power in each detector as well.

In this chapter, we will first discuss how an existing targeted Burst search, called *X-pipeline*, searches for Gravitational Wave Bursts (GWBs). We will then see how we can improve this pipeline by using *Multivariate Analysis* (MVA) to apply cuts and rank triggers.

5.2 Xpipeline

Xpipeline is a data analysis pipeline for the Burst search. It begins by using the known sky location of the GRB to time-shift the data from each detector so that the GW arrives simultaneously in each data stream. Xpipeline then makes coherent data-streams from the individual detector data. There are two types of coherent data-streams: *signal streams*, which increase the power of a GW signals, and *null streams*, which reduce the power of true GW signals but not noise. *Triggers* are groups of neighbouring pixels that are above threshold in a time-frequency map of a signal stream (see fig.5.1). Xpipeline then removes triggers based on cuts of the various network data streams (see fig.5.2). The position of these cuts is set to give the

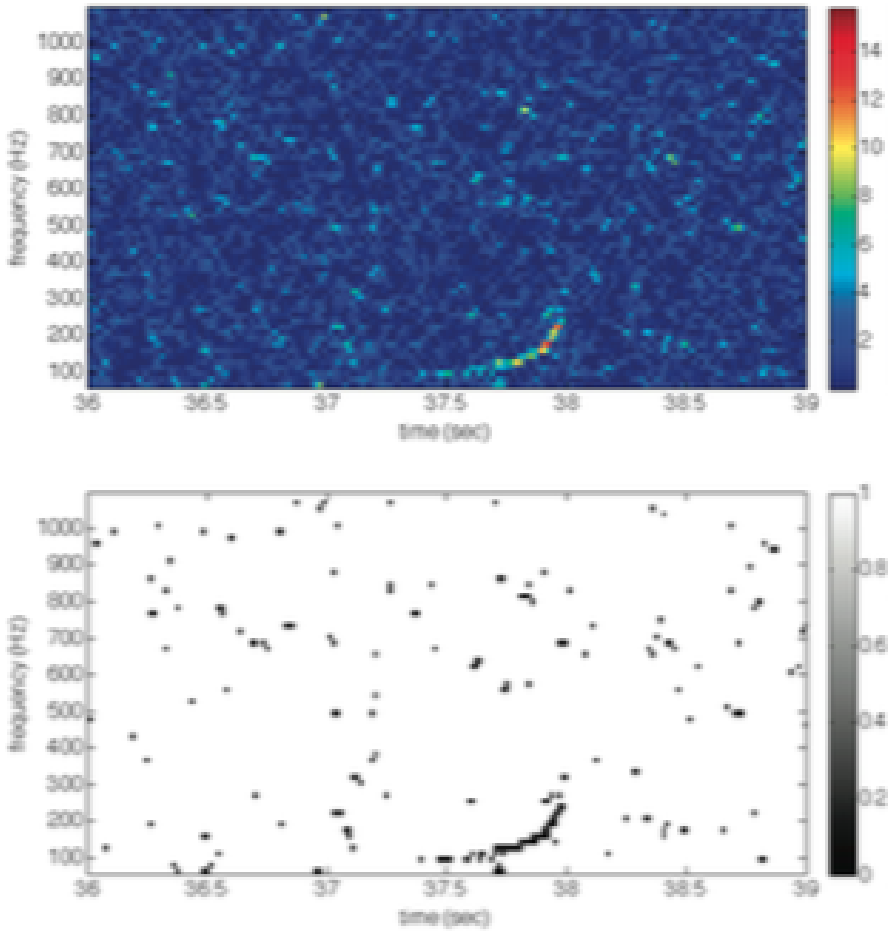

Figure 5.1: **Xpipeline Time-Frequency Map** This figure shows a time-frequency map from Xpipeline for a $1.4 - 10M_{\odot}$ NSBH merger. The top figure shows the E_{+} energy and the bottom figure shows the top 1% of pixels.

best performance on a subset of the triggers that are used for tuning. For the MVA pipeline, the position of these cuts is instead set by a machine learning algorithm (as we will see in 5.3).

In this section, we will formulate two of the coherent data-streams generated by Xpipeline, as illustrative examples. The first is a signal-stream, called the *standard likelihood*. We will then use the standard likelihood to create a null-stream.

5.2.1 Burst Search Background

Suppose we have a network of D detectors. A gravitational wave, described by $h_{+}(t)$ and $h_{\times}(t)$, passes through the Earth from direction $\hat{\Omega}$. We describe the sensitivity of detector $\alpha \in \{1, ..., D\}$ to the plus and cross polarizations using the *Antenna Response Functions*, denoted $F_{\alpha}^{+}(\hat{\Omega})$ and $F_{\alpha}^{\times}(\hat{\Omega})$. The position of detector α

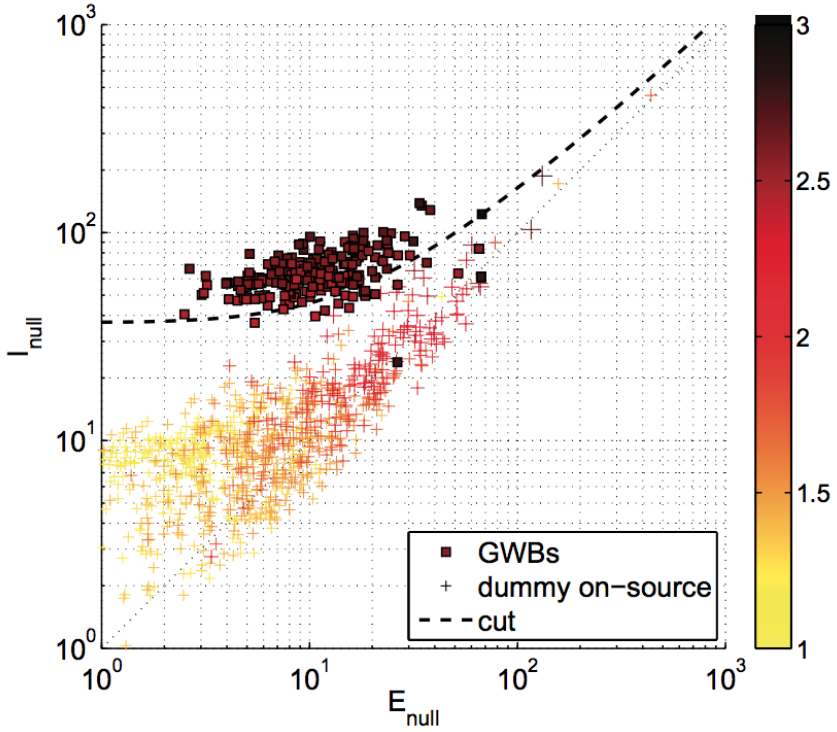

Figure 5.2: **Xpipeline Cut** This figure shows Xpipeline making a cut to eliminate many spurious signals.

is denoted by $\mathbf{r}_\alpha$ and the noise in this detector is given by $n_\alpha(t)$. The detector output $d_\alpha(t)$ is then given by

$$d_\alpha(t + \Delta t_\alpha(\hat{\Omega})) = F_\alpha^+(\hat{\Omega})h_+(t) + F_\alpha^\times(\hat{\Omega})h_\times(t) + n_\alpha(t + \Delta t_\alpha(\hat{\Omega})) \ . \tag{5.1}$$

Here Δt_α is the time taken for the GW to reach the detector from some arbitrary reference point[1] $\mathbf{r}_0$

$$\Delta t_\alpha(\hat{\Omega}) = \frac{1}{c}(\mathbf{r}_0 - \mathbf{r}_\alpha) \cdot \hat{\Omega} \ . \tag{5.2}$$

From now on we will suppress explicit mention of the reference point $\mathbf{r}_0$ or the time delay Δt_α on the understanding that detector outputs need to be time-shifted by an appropriate amount.

In reality, detector outputs are not continuous but sampled discretely. The discrete Fourier transform $\tilde{x}[k]$ of the time series $x[j]$, and its inverse, are given by

$$\tilde{x}[k] = \sum_{j=0}^{N-1} x[j]e^{-i2\pi jk/N}, \qquad x[i] = \frac{1}{N}\sum_{j=0}^{N-1} \tilde{x}[k]e^{i2\pi jk/N} \ . \tag{5.3}$$

For sampling rate is f_s and N data points in the time domain, we convert continuous to discrete notation by using

$$x(t) \to x[j] \tag{5.4}$$

$$\tilde{x}(f) \to f^{-1}\tilde{x}[k] \tag{5.5}$$

$$\int dt \to f_s^{-1} \sum_j \tag{5.6}$$

$$\int df \to f_s N^{-1} \sum_k \tag{5.7}$$

$$\delta(t - t') \to f_s \delta_{jj'} \tag{5.8}$$

$$\delta(f - f') \to N f_s^{-1} \delta_{kk'} \ . \tag{5.9}$$

For example, the one-sided noise spectral density for a detector with noise $n(t)$ can be written in continuous form as

$$\langle \tilde{n}^*(f)\tilde{n}(f')\rangle = \delta(f - f')\frac{1}{2}S_n(f) \tag{5.10}$$

where the angle brackets indicate an average over the noise. In the discrete notation listed above, this becomes

$$\langle \tilde{n}_\alpha^*[k]\tilde{n}_\beta[k']\rangle = \frac{N}{2}\delta_{\alpha\beta}\delta_{kk'}S_\alpha[k] \ . \tag{5.11}$$

[1]The center of the Earth is a fairly intuitive choice for a worldwide detector network.

We will be working with the *noise-spectrum-weighted* quantities, defined by

$$\tilde{d}_{w\alpha}[k] = \frac{\tilde{d}_\alpha[k]}{\sqrt{\frac{N}{2}S_\alpha[k]}} \tag{5.12}$$

$$\tilde{n}_{w\alpha}[k] = \frac{\tilde{n}_\alpha[k]}{\sqrt{\frac{N}{2}S_\alpha[k]}} \tag{5.13}$$

$$F^{+,\times}_{w\alpha}(\hat{\Omega}, k) = \frac{F^{+,\times}_\alpha(\hat{\Omega})}{\sqrt{\frac{N}{2}S_\alpha[k]}} \tag{5.14}$$

The normalisation of the whitened data is

$$\langle \tilde{n}^*_\alpha[k]\tilde{n}_\beta[k']\rangle = \delta_{\alpha\beta}\delta_{kk'} \ . \tag{5.15}$$

In vector notation, we can write 5.1 as

$$\tilde{\mathbf{d}} = \mathbf{F}\tilde{\mathbf{h}} + \tilde{\mathbf{n}} \tag{5.16}$$

where $\mathbf{F} = [\mathbf{F}^+ \ \ \mathbf{F}^\times]$ and $\tilde{\mathbf{h}} = [\tilde{h}_+ \ \ \tilde{h}_\times]^T$.

5.2.2 Standard Likelihood

Assume that the noise in our detectors is Gaussian. As a gravitational wave $\tilde{\mathbf{h}}$ passes through the detector from a known direction, the probability of attaining whitened detector output $\tilde{\mathbf{d}}$ in one time-frequency pixel is given by

$$P(\tilde{\mathbf{d}}|\tilde{\mathbf{h}}) = \frac{1}{(2\pi)^{D/2}} \exp\left[-\frac{1}{2}\left|\tilde{\mathbf{d}} - \mathbf{F}\tilde{\mathbf{h}}\right|^2\right] \ . \tag{5.17}$$

For a set $\{\tilde{\mathbf{d}}\}$ of N_p time-frequency pixels, we have

$$P(\{\tilde{\mathbf{d}}\}|\{\tilde{\mathbf{h}}\}) = \frac{1}{(2\pi)^{N_pD/2}} \exp\left[-\frac{1}{2}\sum_k \left|\tilde{\mathbf{d}}[k] - \mathbf{F}[k]\tilde{\mathbf{h}}[k]\right|^2\right] \ . \tag{5.18}$$

By comparing this value to the probability that the detector produces this output in the absence of any GW signal, we can calculate a likelihood of the signal being a real GW signal.

The *Likelihood Ratio* L is the log of the probability that the detector network will have output $\tilde{\mathbf{d}}$ in the presence of GW $\tilde{\mathbf{h}}$ divided by the probability of obtaining the same output in the absence of a gravitational wave ($\tilde{\mathbf{h}} = 0$)

$$L = \ln\frac{P(\{\tilde{\mathbf{d}}\}|\{\tilde{\mathbf{h}}\})}{P(\{\tilde{\mathbf{d}}\}|\{0\})} = \frac{1}{2}\sum_k\left[\left|\tilde{\mathbf{d}}\right|^2 - \left|\tilde{\mathbf{d}} - \mathbf{F}\tilde{\mathbf{h}}\right|^2\right] \ . \tag{5.19}$$

For the above analysis to be applied, we would need to know the waveform $\tilde{\mathbf{h}}$ in

advance. For GRBs and other unmodelled searches, this is not possible. One way to handle this problem is to fit the waveform in each time-frequency pixel to the data in such a way as to maximise the likelihood ratio. Hence we have

$$0 = \frac{\partial L}{\partial \tilde{\mathbf{h}}}\bigg|_{\tilde{\mathbf{h}}=\tilde{\mathbf{h}}_{\mathbf{max}}} . \tag{5.20}$$

Solving this, we find

$$\tilde{\mathbf{h}}_{\mathbf{max}} = (\mathbf{F}^\dagger \mathbf{F})^{-1} \mathbf{F}^\dagger \tilde{\mathbf{d}} \tag{5.21}$$

where the superscript dagger † denotes the conjugate transpose.

Calculating the likelihood ratio for $\tilde{\mathbf{h}}_{\mathbf{max}}$ gives us the *Standard Likelihood* E_{SL}

$$E_{\mathrm{SL}} = 2L(\tilde{\mathbf{h}}_{\mathbf{max}}) = \sum_k \tilde{\mathbf{d}}^\dagger \mathbf{P}^{\mathrm{GW}} \tilde{\mathbf{d}} \tag{5.22}$$

where

$$\mathbf{P}^{\mathrm{GW}} \equiv \mathbf{F}(\mathbf{F}^\dagger \mathbf{F})^{-1} \mathbf{F}^\dagger . \tag{5.23}$$

We can see from equation 5.1 that the contribution made to the data output by a passing GW from fixed sky location is restricted to the subspace spanned by $\mathbf{F}_+$ and $\mathbf{F}_\times$. Therefore the energy in this subspace is the energy that is consistent with a GW from a given sky location. We can show that $\mathbf{P}^{\mathrm{GW}}$ is a projection operator, projecting the data into this same subspace. The standard likelihood maximises the energy in this subspace, and so is the maximum energy contained in the whitened data that is consistent with a GW from the given sky location.[2] This is an example of the coherent signal-streams that Xpipeline uses.

5.2.3 NULL ENERGY

We can use the standard likelihood to find a null-stream. First consider the *total energy*, given by

$$E_{\mathrm{tot}} = \sum_k |\tilde{\mathbf{d}}|^2 . \tag{5.24}$$

This is an incoherent statistic as it contains only auto-correlation terms, and no cross-correlation terms. If we subtract the standard likelihood from the total energy, we obtain the *null energy*

$$E_{\mathrm{null}} \equiv E_{\mathrm{tot}} - E_{\mathrm{SL}} = \sum_k \tilde{\mathbf{d}}^\dagger \mathbf{P}^{\mathrm{null}} \tilde{\mathbf{d}} . \tag{5.25}$$

This is the energy that is inconsistent with a GW from given sky location, and must therefore be associated with noise. This is the minimum amount of energy in our whitened data that is inconsistent with our GW. It is an example null-stream used by Xpipeline.

[2] In practice, some of this energy will be due to noise. We say that the energy due to noise is *inconsistent* with a GW signal. The rest of the energy must be due to the GW, so we say it is *consistent* with the GW.

This shows one of the key advantages of coherent analysis. If we analysed our data incoherently, we would be working with just the total energy. But using coherent methods, we can project the whitened data into the subspace spanned by $\mathbf{F}^+$ and $\mathbf{F}^\times$, thus removing some fraction of the noise without removing any of the signal. The drawback is that if the sky position is not known in advance, then the analysis needs to be repeated for a set of directions that span the entire sky ($\gtrsim 10^3$ directions), each with different antenna response functions $\mathbf{F}^+$ and $\mathbf{F}^\times$. This will slow down analysis and increase the false alarm probability.

5.2.4 Incoherent Energy and Background Rejection

The diagonal elements of 5.25 are auto-correlation terms, and the other elements are cross-correlation terms. The auto-correlation part of the null energy are called the *incoherent energy*, and denoted by

$$I_{\text{null}} = \sum_k \sum_\alpha P^{\text{null}}_{\alpha\alpha} |\tilde{d}_\alpha|^2 \ . \tag{5.26}$$

Background triggers are typically not correlated between the different detectors of the network, so the cross-correlation terms are small relative to the auto-correlation terms. This means that for glitches, we have

$$E_{\text{null}} \approx I_{\text{null}} \ . \tag{5.27}$$

Compare this to the case of a GW signal. This will be correlated between the detectors. By construction, the energy of the correlated energy does not appear in the null-stream. Therefore, in the presence of a strong GW, the incoherent energy is much larger than the null energy

$$E_{\text{null}} \ll I_{\text{null}} \ . \tag{5.28}$$

Using 5.27 and 5.28, we see that the ratio of E_{null} and I_{null} is very different in the case of a glitch as opposed to a GW signal. We can use this to make the following cut to remove noise triggers from our sample

$$I_{\text{null}}/E_{\text{null}} > C \tag{5.29}$$

for some constant $C > 1$. This test does not work as well for small amplitude glitches, where the statistical fluctuations can lead to E_{null} being smaller than I_{null}. For this reason, Xpipeline varies C with the size of the trigger. The precise position of the cut is set to maximise performance on a set of injections.

5.3 Machine Learning

We have seen how Xpipeline makes cuts on coherent statistics to distinguish between noise and GW signals. You can see from 5.29 that the cut only uses two

Label	loghbayesiancirc	standard	circenergy	circinc
Background	12.3128	58.3523	44.7196	24.9015
Background	12.0349	67.5344	41.7045	22.4848
Signal	18.2145	59.8136	53.3320	22.0601
Signal	43.7113	123.9194	118.9774	43.9234
Signal	6422.1467	14426.9124	14167.2933	4991.7876

Table 5.1: An example of MVA training data. Each event has a label and several attributes.

statistics, the null energy and the incoherent energy. Xpipeline generates many coherent statistics that could used in conjunction to make a more powerful cut. In this section we discuss how to use Machine Learning to make achieve this. The software we use is the Toolkit for Multivariate Analysis package in the ROOT data analysis framework.

5.3.1 SUPERVISED MACHINE LEARNING

The type of Machine learning we use is called *supervised machine learning*. Supervised machine learning algorithms are trained on data which has already been classified. They can then be shown a new, not-classified data point and determine the appropriate classification. Supervised machine learning requires data to be in a particular format (see table 5.1). It is a list of *events*, each with a *label* and corresponding *attributes*. The machine learning algorithm builds a *classifier* that can determine the label of an event when given the event's attributes. In the case of our GW search, the events are the triggers returned by Xpipeline (see section 5.2). The labels are *signal* or *background*, and the attributes are the values of the signal and null data streams for those triggers as well as some statistics describing the trigger, e.g. peak frequency, trigger duration. The signal triggers are generated by injecting signal waveforms into the data. Background triggers are triggers that do not coincide with an injected signal.

The signal and background trigger sets are each divided into two subsets: The *training set* and the *testing set*. The training set is used to build the classifier, while the testing set is used to measure the accuracy of the trained classifier. If the classifier performs much better on the training set than on the testing set, then the classifier is *overtrained*. This means that the classifier has learned exactly the properties of signals and noise in the training set, rather than learning the general properties of signals and noise. When an overtrained classifier is used on a different data set with different noise properties (such as the testing set), it will perform poorly.

5.3.2 BOOSTED DECISION TREES

A *Decision Tree* is a simple type of classifier. It is a flowchart of true/false statements about a trigger's attributes to determine the correct label. For example,

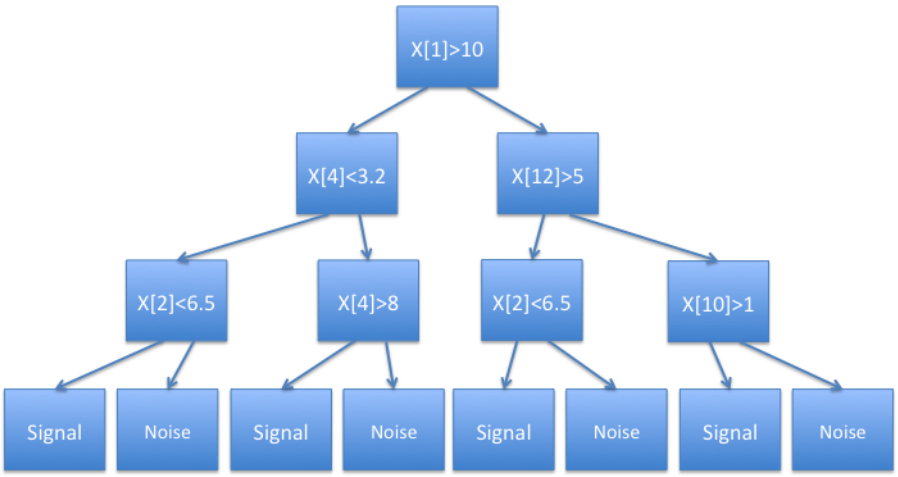

Figure 5.3: **A decision tree example** To determine if a trigger is a signal or noise event the tree makes a series of cuts on the attributes x[N]. If the inequality in a node is true, then the next node is the branch to the left. Otherwise the next node is the one to the right.

consider the decision tree shown in fig.5.3. Here the attributes are labeled as the components of a vector x. We start at the top node and work downwards. If the node is true then we follow the branch to the left and if not then we follow the branch to the right. We then consider the statement at the end of whichever branch we follow. We continue this process until we reach a *leaf node*, which has no branches and contains a final classification for the trigger.

We can improve the performance of the classifier by using an *ensemble* (or *forest*) of trees. This means training multiple distinct trees, and each tree independently classifies the trigger. The final classification of each trigger is a normalised sum of the outputs of each tree, with +1 corresponding to signal, and -1 corresponding to background. This leads to different regions of the parameter space having different MVA scores, as can be seen in fig.5.4. The higher the score, the more likely an event is to be a signal.

To train a decision tree, we need methods to find the variable to cut on and the position of the cut for each decision node, and the label in each leaf node that best discerns between signal and background. Each of these values is set by brute force: Trying each possible combination of cuts and labels to get the best performance on the training events. Ensemble methods work best when each classifier in the ensemble is independent of the others, so training every tree on the same events is not going

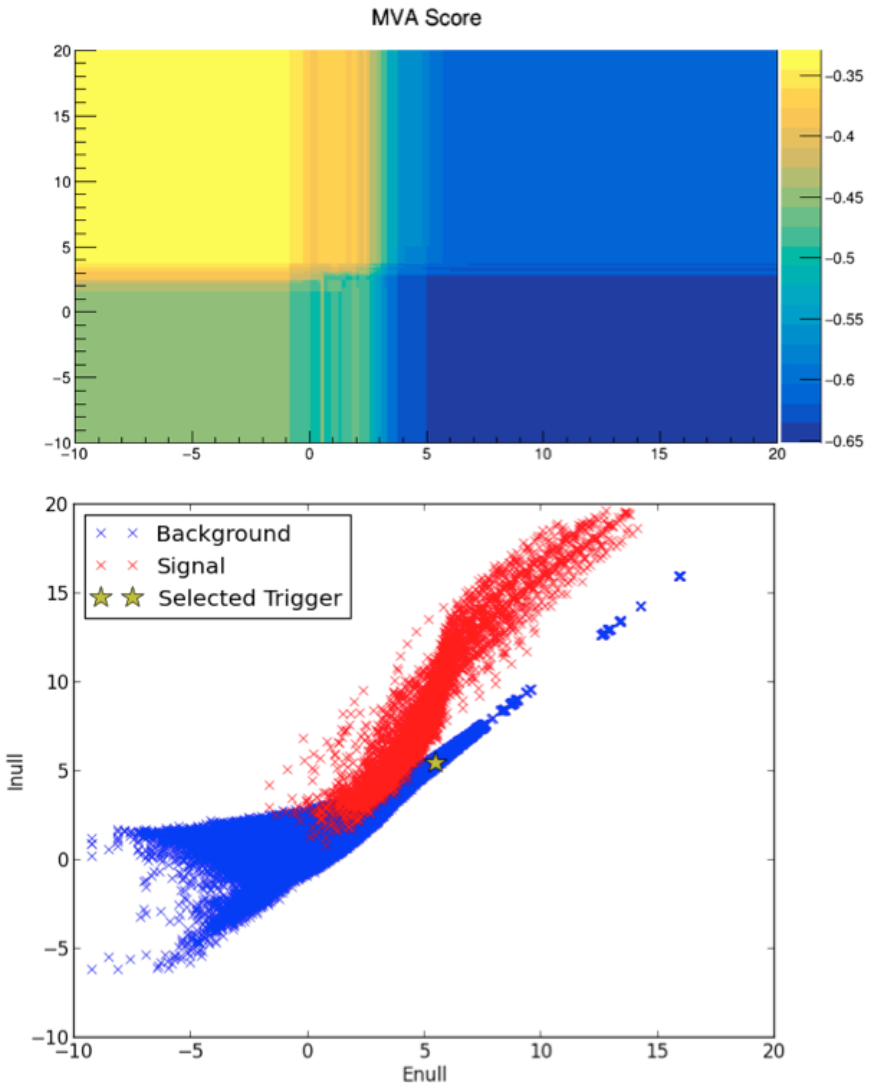

Figure 5.4: **Visualising the Classifier** In the bottom plot you can see the value for log(Enull) and log(Inull) for all the signal and background training data used to build the classifier. We chose one of these events at random (indicated by the star) and varied Enull and Inull to see how it changed the MVA score, indicated by the colour in the top plot. As you can see, increasing Inull and decreasing Enull leads to the event being more likely to be classed as a signal. This is akin to the xpipeline cut shown in fig.5.2.

to give optimal results. For this reason each tree is trained on some subset of the training set. We could pick events at random to form these subsets, but a more powerful method is to use *Adaptive Boosting*.

When using adaptive boosting, each event in the training set is given a probability that it will be selected to train the next tree. Initially each event in the training set has the same probability of being selected. Then after each tree is trained the probability of each event being selected to train the next tree is updated such that misclassified events are more likely to be included in the training set for the next tree. The misclassifed events have their probability of being selected for training the next tree updated by the *boost weight*, given by

$$\alpha = \frac{1 - \text{err}}{\text{err}} \tag{5.30}$$

where err is the misclassification rate. The weights are then renormalised.

Note that the boost weight is greater than one for[3] err$< 1/2$, and that as the error gets smaller, the boost weight increases. The effect of this is that each new tree is more likely to be trained on the events that are most difficult to classify.

The ensemble output is also changed, so that it is weighted rather than being a simple sum. If the output of the ith tree is given by $h_i(\mathbf{x})$, with $\mathbf{x}$ being the event attributes, then the ensemble output is given by

$$H(\mathbf{x}) = \frac{1}{N} \sum_{i=0}^{N} \ln(\alpha_i) h_i(\mathbf{x}) \tag{5.31}$$

where N is the number of tree in the ensemble. Small values indicate background events, while large values indicate a signal.

5.3.3 Data Preprocessing

Extra data preprocessing is required for the training set. Suppose a small amplitude signal injection is added close to a large amplitude glitch in the data. This trigger will be labeled as a signal, due to the injection, but the properties of the trigger will resemble a glitch, as the glitch has a much larger amplitude than the signal injection. It is essentially a background trigger labeled as a signal. This has two affects: It reduces our ability to detect real signals by making the properties of signals harder for the algorithm to learn, and it can lead to background triggers being misclassified as a Gravitational Wave signal.

For these reasons, it is important to make sure that signal injections do not overlap in time or frequency with background triggers. To prevent this from happening we *clean* the data. This means finding all of the triggers in the data for the smallest signal injection scale. The injected waveforms are too small to be detected so all of these triggers must be background. We then increase the injection amplitude and look for triggers again. Any triggers that overlap in time or frequency with the noise triggers

[3]The misclassification rate is always less than half when the labels only take two possible values as the algorithms we use are always better than chance alone.

are then identified and removed from the from the signal set. **This description of cleaning may need to change if we change the cleaning codes.**

We must also not include injections in our signal set that are too small to be detected. If we do then we would again be including triggers labeled as signal that have the properties of a noise trigger, even if there is no glitch present. For this reason we apply a threshold on the amplitude of the signal set, so any injection below that amplitude is removed. The level of the threshold is set experimentally; If it is too low then we will increase the chance of a false positive and hurt our sensitivity to real signals, but if it is too high then we will limit the classifiers ability to detect low amplitude signals.

We also use *Generalised Clustering*. This is a change in the way Xpipeline defines a trigger. Without using generalised clustering, triggers are groups of neighbouring pixels in the time-frequency maps such as in fig.5.1. Generalised clustering allows these pixels to be separated by a user-specified number of pixels. This can boost the signal power of long triggers, as long triggers tend to contain breaks in their time-frequency maps that causes Xpipeline to list a single long trigger as multiple short triggers. The downside to using generalised clustering is that it can cause noise triggers to be grouped together as well, boosting the power of noise. This then causes our sensitivity to short triggers to be reduced slightly. Experimenting with generalised clustering in both the standard Xpipeline analysis and the MVA analysis suggests that the benefits outweigh the costs, with an improvement of between 7% and 50% for long inspiral and adi waveforms at the cost of a decrease of 3% to 9% for short sine-gaussian waveforms.

As XTMVA is to be used for the unmodelled search, it is also important to ensure that the search can find waveforms that are not in the training set. There are several tools that we use to achieve this. The first is to limit the amount of information the classifier is given about the waveform morphology. The classifier cannot know the precise morphology of any waveform because the only attributes that the classifier trains on is the time, peak-frequency, and various measurements of the coherent and null energy between the detectors in the network. This forces the classifier to use the coherence of the triggers to make a classification, rather than the waveform morphology.

There is still a possibility that the classifier will become too specialised to the waveforms in the training set, as the classifier is given peak frequency data and certain waveform morphologies may have particular characteristics than become apparent from the coherent and null streams. For this reason we also trained the classifier on a variety of different waveforms. The training set included long and short waveforms, and a variety of different morphologies. Some of the signals are astrophysically motivated, such as compact binary coalescence signals, while some are artificial, such as the white noise burst.

The final tool we use to ensure the classifier is sensitive to generic waveforms is to test the classifier on waveforms that are not included in the training set. If the classifier can find waveforms not in the training set, then we can be reasonably confident that it is sensitive to generic waveforms. We also try removing certain waveforms from the training set to test the robustness of the classifier. This will lead to a drop in sensitivity for that waveform, but if the drop in sensitivity is small, then

we can be confident that the classifier is robust.

5.3.4 Optimisation and Validation

With any machine learning algorithm, there are *hyperparameters* that must be tuned. The number of trees in the ensemble, the maximum depth of the trees, and many more hyperparameters must be chosen. We optimise the hyperparamters by repeatedly running an example analysis with different hyperparameters, trying to increase the number of injected signals we could recover from the testing set.

Setting our hyperparameters using the testing set can cause *data leakage*. Data leakage is when data from outside the training set is used to build a classifier. As we tune our classifier on the testing set, it is possible that we will implicitly tune our classifier to only work well on this training and testing set. This is very similar to overtraining discussed in section 5.3. To avoid this, once we have tuned the hyperparameters on a single GRB, we test the classifier on several other GRBs. If the performance drops significantly on these other GRB analyses, then we have had data leakage and we need to retune our classifier. This process of testing on previously unused data is called *validation*. If there is evidence of data leakage, then we must retune our classifier and validate it again, this time using a different (previously unused) GRB so as to avoid data leakage from the validation GRB.

Optimisation is a somewhat cyclical process, as once we have changed one hyperparameter, we must then go back and test that other parameters do not now need changing. But optimisation is not pure guess work, and in the rest of this section we will see the strategy we used to optimise our classifier. We first discuss optimising the training data to use. As mentioned in section 5.3.3, there are several choices to make regarding what data is used for training, such as setting the amplitude threshold. We then discuss optimising the BDT classifier itself.

Training Set

Many of the hyperparameters mentioned in section 5.3.3 had to be optimised. Consider the amplitude threshold applied to triggers before they are included in the training set. If this is too low then noise triggers contaminate the signal training set, but if it is too high then we limit the sensitivity of our classifier to only higher amplitude waveforms.

We also optimised the waveforms that are included in the training set, whether to use cleaning or not, and whether to use generalised clustering. **Write more on this once we've done it.**

- Had to optimise the threshold as well
- Training waveforms
- cleaning or not
- Gen clustering or not

BDT Parameters

There are many hyperparameters that need to be set for a BDT analysis. In this section we discuss some of these hyparameters and the method we used to optimise them for our analysis.

We began by setting values for *NTrees*, the number of trees in the ensemble, and the learning rate, discussed in section 5.3.2. These two parameters are set first to ensure the machine learning algorithm will converge in a reasonable amount of time. Setting the learning rate too low causes the classifier to take longer to converge. Setting the learning rate too high can cause the classifier to never converge. Similarly, using too many trees takes too long for the classifier to finish training, but too few and the training will terminate before the algorithm has converged. Setting these first ensures that we have a classifier that gives sensible results in a reasonable amount of time. While optimising, we set these values slightly high, to ensure that our classifier converges and does so quickly while we optimise our other parameters. Once the other parameters are set, we again optimise NTrees and the learning rate, increasing the number of trees and decreasing the learning rate to ensure the algorithm reaches its optimum performance, even if it increases the time taken for training.

We now tune the tree-specific parameters. Unlike the number of trees and the learning rate which are primarily tuned to ensure the classifier will converge in a reasonable time, these parameters are set to ensure that the classifier is accurate but does not overtrain. Overtraining can happen when the trees are allowed to make cuts that are too fine, carving out regions of parameter-space around anomalous events in the training set rather than finding cuts that generalise beyond the training data. The way to prevent this is to limit how fine the cuts made by the decision trees are allowed to be, while allowing cuts that are fine enough to pick out the general features of signal and background events in our data. There are several hyperparameters we can set to do this, which must all be tuned.

The first of these is the maximum depth of the trees, which is the maximum number of cuts a tree can make before it reaches a leaf node. Each cut divides the parameter-space into ever smaller regions which it labels as background or signal. Setting the maximum number of cuts too low will therefore cause the classifier to be too coarse in dividing up the parameter-space, resulting in poor accuracy. Increasing the maximum depth allows the classifier to pick out smaller features in the parameter-space. If the maximum depth is too high then the classifier will overtrain; dividing the parameter-space into precisely the regions that work for the training set and losing generality. As we are using adaptive boosting, it is recommended **cite TMVA userguide** to use trees with fewer cuts. For this reason we tried values from 2-16, finding that for our problem a maximum depth of 8 was optimal.

A related hyperparameter is the minimum number of events that we allow in a leaf node. If we allow the training algorithm to have any number of events in a leaf node, then it will occasionally find cuts that results in a small number of events in one or more of the leaf nodes. The again allows for the carving out of anomalous events in the training set rather than finding cuts that can generalise beyond the training set. Conversely, setting the minimum number of events allowed in the leaf nodes to be too high does not allow the classifier to pick out the key features in the data. We

used a grid search over the values 100-1600 for the minimum number of events per leaf node and found the optimal value to be 400.

The final hyperparameter we set is the number of cuts that the training algorithm scans over to find the best cut. When the classifier is training it searches for the best way to cut the parameter-space into a signal subspace and a background subspace. To do this we can try every possible cut on every parameter. This can be very slow and lead to overtraining. To speed up our analysis and reduce the chance of overtraining, we can choose the number of cuts to try on each parameter. For example, we may decide to use 10 cuts for each parameter. In this case the algorithm will tune the cut on a parameter which ranges from 0-100 by trying cuts at 0,10,20,... and selecting the best of these cuts. To tune the number of cuts we wanted to use, we tried values from 10-160 as well as allowing the classifier to try every possible cut. We found that 80 cuts was optimal for our purposes.

- This can definitely be phrased better

Result of Optimisation

- Show nice plots of improved sensitivity due to optimisation
- Show improvement when compared to Xpipeline