Supervised Learning Algorithms for Gravitational Wave Data Classification

by Rakibul Chowdhury

Supervised Learning Algorithms for Gravitational Wave Data Classification

Pioneer Academics 2021

Professor Eric Myers

Abstract

This paper approaches gravitational wave detection from a machine learning perspective, with an emphasis on supervised learning algorithms. A monomial logistic regression algorithm is utilized to study the signal to noise ratio of various events in an optimal matched filter search as a predictor variable to identify compact binary coalescences. The study produced an unfavorable classification success rate of 50%. Alternative predictor variables are explored.

2 Aug 2021 Rakibul Chowdhury rakibul@mit.edu

Contents

I. Background and Introduction

A. Gravitational Waves and Detection

Gravity is not often associated with waves, as it is traditionally viewed as the force of attraction between bodies of mass. However, in the early 1900's, with his theory of general relativity, Albert Einstein theorized that bodies of mass cause distortions, reminiscent of geodesics drawn on the earth's surface (Einstein's multi-dimensional model of reality combines the passage of time with relative velocity - hence, general relativity). From his theory, Einstein further predicted that massive, accelerating bodies would cause rhythmic distortions of space-time that can be described as gravitational waves (GW)[1].

To measure these disturbances in space-time, physicist Rainer Weiss invented the interferometry technique by which scientists now search for gravitational waves. Light is sent through a beam splitter that sends light through perpendicular 4km long vacuums to detect distortions in either beam. The key to this technique is that the beams do not experience the same distortions as each other, making them detectable; because of the length of the arm, the premise is that even miniscule disturbances will be detectable when analyzing the difference between how light travels in one tunnel versus the other, measured as 'strain'.

B. Optimal Matched Filtering vs Machine Learning

Optimal Matched Filtering

Optimal matched filtering is the current dominant method by which researchers search for gravitational wave events in interferometer data. The Laser Interferometer Gravitational Wave Observatory's (LIGO) technical note T1400342-v2 details the recovery of burst injections from LIGO's 5th science run. These signals, which were made by manipulating the hardware of the interferometer, were intended to simulate astrophysical events for the purpose of testing and calibration[2]. Because burst waveforms are crudely modeled[2], the hardware injections were created with a number of known waveforms for the purpose of calibration.

The authors employed optimal matched filtering and adapted the process to find hardware injections and simulate bursts in data from LIGO's 5th science run (S5). This proved to be a very effective method of looking through a large amount of data to recover more than 60,000 clearly defined waveforms. However, their use of optimal matched filtering exposes a weakness in the general process of using matched filtering to recover GW events; namely, the inability to ambiguously search through data without an explicitly defined template.

Supervised learning

Contrary to the matched filter approach, machine learning (ML) is an algorithmic field that aims to create active prediction without explicit human instruction. ML algorithms are built with the goal of reaching "correct" predictions based on inputs. Supervised learning is an ML technique that utilizes given training data in the form of input-output pairs to create a classification system for new data[3]. Given the data, algorithms must utilize some form of inductive bias to prioritize one possible solution over another. A logistic regression model is one such example of inductive bias and is seen as the "baseline supervised machine learning algorithm for classification"[3]. The mathematical model is used by the algorithm to analyze training data and "learn" how to interpret new unseen inputs.

Similar to linear regressions, logistic regressions create equations based on weighted parameters in an effort to model an observable pattern between input and output. However, this implementation feeds the result into a sigmoid function in order to reach a value between 0 and 1. Logistic regressions use known labels of either 0 or 1 for each instance of an input to produce weighted parameters that bring each input as close as possible to its true label[4]. The interpretation of this result depends on the implementation, but the simplest interpretation is that the resulting value can be used for classification by interpreting it as the probability that a given input produces a value of 1. This concept can be implemented on simple binary classifications in addition to multinomial classifications where an algorithm must make decisions between more than two outcomes.

C. Modeling CBC Events

The S5 hardware injections, though limited in the scope of phenomena they simulate, provide a powerful, large set of training data to create models of certain events and demonstrate the efficacy of this approach. The training data will be used to analyze the properties of compact binary coalescence (CBC) events using a logistic regression model for classification. As an extension of Cole and Kanner's study, this paper will analyze the signal to noise ratio (SNR) of CBC and non CBC events in addition to noise samples when analyzed using a CBC matched filter search to demonstrate the creation of predictor variables for classification.

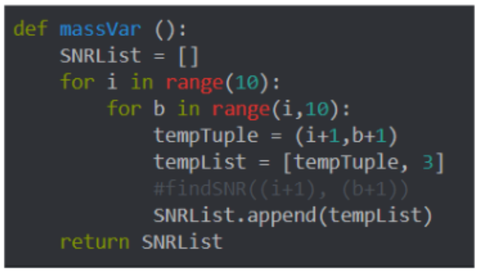
It may seem counterintuitive to use matched filtering in a method that aims to provide an alternative to matched filter searches; however, an important distinction is that matched filtering is not being used to *recover* the event; rather, the *output* of the matched filtering algorithm is being used as an element of a model that represents CBC events. This paper aims to outline the process by which models of phenomena observable in interferometer data can be made in an effort to conduct a more effective analysis. It is also critical to acknowledge that utilizing artificially modeled injections to create such a model introduces bias, and that this model is merely a proof of concept.

II. Methodology

A. CBC Events

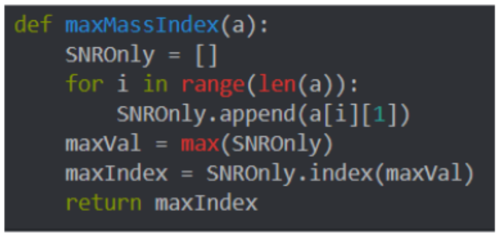
The program utilizes a modified version of the Gravitational Wave Open Science Center (GWOSC) *Find a Hardware Injection* example code to collect the SNR of events stored in hdf5 data files[5]. The program makes use of data quality channels to locate periods of noise and periods of possible activity. It uses FINDCHIRP, a template generation algorithm, to create a template and subsequently performs the necessary steps to do an Inverse Fast Fourier Transform on the data, compute the power spectral density, matched filter, and normalization to compute the data's SNR.

The program is modified to perform a mass variational study to find the maximum SNR of a particular event by using templates of various mass combinations. It is written to create templates of every combination of solar masses between 1 and 10, including pairs of the same mass, as shown in *figure 1*. This result is stored in a list of lists that stores tuples containing the masses alongside their respective SNR. The index of the maximum mass is then found and recorded as shown in *figure 2*. These functions are called by a main function that feeds in event data, specifying both the file itself and whether or not the event is a CBC. The results are outputted onto a comma separated value (CSV) file that records the SNR of a piece of data along with its classification for training, with 1 representing a CBC and 0 representing non CBC data.



```
def massVar ():
    SNRList = []
    for i in range(10):
        for b in range(i,10):
            tempTuple = (i+1,b+1)
            tempList = [tempTuple, 3]
            #findSNR((i+1), (b+1))
            SNRList.append(tempList)
    return SNRList
```

figure 1 - Mass variational study



```
def maxMassIndex(a):
    SNROnly = []
    for i in range(len(a)):
        SNROnly.append(a[i][1])
    maxVal = max(SNROnly)
    maxIndex = SNROnly.index(maxVal)
    return maxIndex
```

figure 2 - Max SNR index

B. Non CBC events

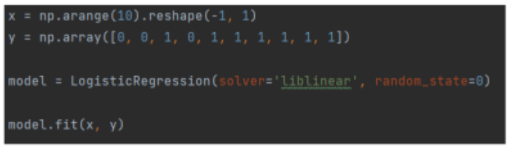
Non CBC events require the program to look at different data quality (dq) channels to locate the segment. For example, to analyze a hardware burst injection, the program must use the HW_BURST key to track the correct dq channel for activity. For data that is only random noise, the program is modified to simply analyze 136 seconds of the data without searching for an injection segment. Similarly to CBC events, the SNR is recorded, but the only difference is that this SNR will be tagged with a value of 0 instead of 1 to flag that it is a non CBC SNR value. The SNRs of each individual event are printed onto a CSV file in addition to its 1/0 flag. One possible source of bias to note is that only events that were 100 percent contained in a single frame file were analyzed.

C. Logistic regression model

Logistic regressions are an example of a linear classifier. This implementation pairs a linear function with coefficients for independent variables with a sigmoidal function. As such, the outputs will be close to 0 or 1 for most of the domain, with a transitional region in between that is roughly linear[6].

The specific solver utilized in this implementation is the liblinear solver built by scikit-learn[7] that utilizes a coordinate descent algorithm built on the C++ LIBLINEAR library, suitable for our monomial classification. Such a model must make use of regularization methods to avoid overfitting, where the intricacies of training data are learned too well to the point where accuracy is hindered[7].

This model is shown implemented in *figure 3*, adapted from Mirko Stojiljković' basic logistic regression model built with sci-kit learn[6,7]. A numpy array of input values



```
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 1, 0, 1, 1, 1, 1, 1, 1])

model = LogisticRegression(solver='liblinear', random_state=0)

model.fit(x, y)
```

figure 3 - Logit code

between 0 and 9 is paired with an output array of 0s and 1s. Then, an instance of the LogisticRegression class is made by specifying the solver (in this case, liblinear) in addition to the pseudo-random number generator. A linear function f(x) is created with coefficient b_0 and intercept b_1:

$$f(x) \ = b_0 x + b_1$$

The logistic function is then defined as:

$$g(x) \ = \ \frac{1}{1+e^{-f(x)}}$$

The graph of f(x) and g(x) from the array in *figure 3* are shown in *figure 4*, providing a visual representation of the predictions that the model produces when the original input array is fed into the function:

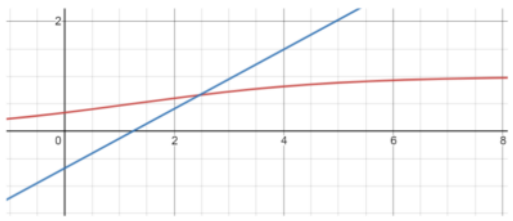


figure 4 - Example logistic regression

The output array [0 0 1 1 1 1 1 1 1 1] aligns with the graphical representation, with inputs greater than 2 producing high probabilities and thus generating outputs of 1 in the array.

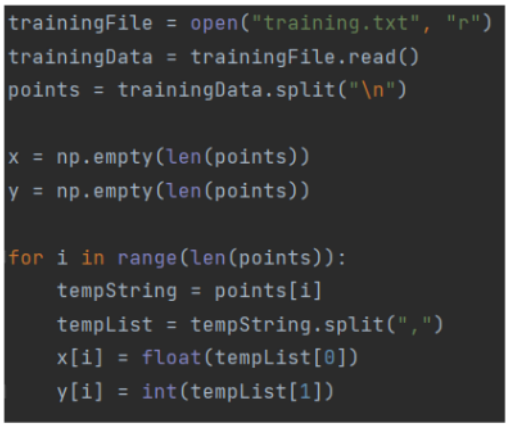
D. Hardware injection training

HDF5 files were taken from LIGO's S5 data archive[8]. Data files for CBC events were first analyzed; files were collected in order of earliest GPS time and analysed using the methods detailed in section II.B. The resulting SNR value is printed onto a CSV file as pictured in *figure 5*. An identical study was done on hardware burst injection data in addition to files with only random noise.

```
78.39035701787279,1
24.419307917399365,1
12.36449998719692,1
14.389424569618791,0
11.169021442014806,0
31.85630475785578,0
10.411940802060752,0
54.83946209071395,0
16.26104153547201,0
```

figure 5 - CSV file snippet

After the data is collected, the program reads the file and feeds the data into an input and output numpy array to store the values. This is shown in *figure 6*.



```
trainingFile = open("training.txt", "r")
trainingData = trainingFile.read()
points = trainingData.split("\n")

x = np.empty(len(points))
y = np.empty(len(points))

for i in range(len(points)):
    tempString = points[i]
    tempList = tempString.split(",")
    x[i] = float(tempList[0])
    y[i] = int(tempList[1])
```

figure 6 - File reading

After the data is stored in the respective arrays, the regression algorithm computes the coefficient and intercept as done in *figure 3*.

III. Results and interpretation

A. Coefficients

After performing the analysis, it was found that higher SNR values in CBC matched filtering do correlate with higher probabilities of an event being a CBC. However, even the lowest SNR values signaled a response of 1 in the predictor. With a b_0 of 0.0087934 and b_1 of 0.0020092, almost all of the domain was associated with a higher than 50% likelihood of being a CBC as shown in the regression (blue) in *figure 7*. This result likely means that, though higher SNRs signal a higher likelihood of an event being a CBC, it is not an especially reliable predictor variable. It could also signal flaws in the training data used.

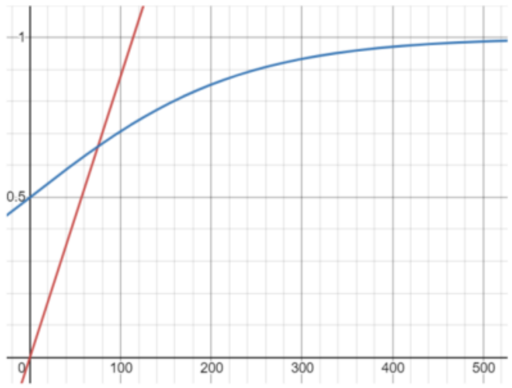


figure 7 - Graphical representation of regression

When the input data was fed into the regression, it signaled every input as a 1, giving it a failure rate of ~50%, as the data had equal parts CBC and non CBC data. A study of SNR *on its own* does not create an effective model for classification. This does not, however, invalidate the predictor variable for use along with other variables. It is possible that the variable can still be relevant in perhaps a fuzzy logic study (see section IV).

B. Outliers

The data file of GPS time 821731328 returned an SNR of 1071.61 when analyzed with the optimal matched filtering process.

IV. Future implementations

A. Further CBC modeling

Choosing additional predictor variables requires thorough understanding of the phenomenon being modeled in order to choose values that are relevant to prediction. This would involve a deep study of the event, related events, and extensive testing of the classifier algorithm's performance to verify the relevance of variables. Many different tests can be made to create models of phenomena, and can be run in simultaneous subroutines for prediction. In the context of CBC events, some additional variables of interest include:

1. In an optimal matched filtering search, the width of peaks in the SNR time series.
2. Properties of time domain cross correlation peaks, such as size and width
3. Range in frequency of the event, or even where the range specifically begins and ends.
4. Length of the event
5. Spectrogram curve-fitting

- An analysis could be done on a 2-dimensional graph of time, frequency, and normalised energy, where a curve-fit is done on the graph of time vs location of peak frequency. Such a spectrogram is pictured in *figure 8*.

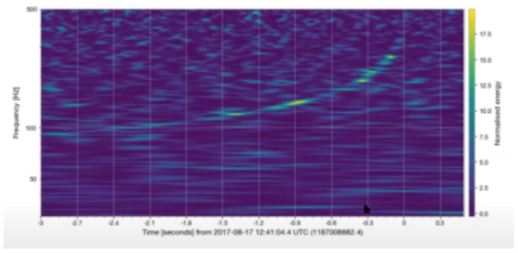


figure 8 - Time versus frequency and normalised energy

B. Multivariable models

Distinguishing between different events is another powerful extension of classifier algorithms. After finding enough relevant predictor variables and creating models of various events, models could be combined to make multivariable predictions about data; in other words, the individual algorithms could be utilized in conjunction to create a program that can make inferences between multiple outputs. This could be done simply by running each individual regression and choosing the output with the highest probability.

Fuzzy logic can also be applied by utilizing the 0-1 output of each individual test to make greater inferences about the data rather than choosing the test with the highest output. This can be done through employing "defuzzification" methods on the fuzzy (non-boolean) set to get an accurate, well defined output. Such methods include the use of logic operators, centroid methods, center of sums, or mean of maximum[9].

C. Modeling noise instead of astrophysical events

The original premise of this project was to model various forms of noise rather than astrophysical events. The idea behind this method was to model noise, which is constantly occurring and in abundance, rather than astrophysical events that can be more difficult to accurately model for various reasons. Theoretically, if noise can be accurately identified, conversely, astrophysical events can be found more effectively; in simple terms, everything that isn't noise is a gravitational wave event. Because of the prevalence of noise, intricate models could be made that account for many predictor variables, identify various types of noise, or even utilize non-boolean logic to create powerful inferences about data.

[1] LIGO Caltech, https://www.ligo.caltech.edu/page/what-are-gw

[2] Recovering S5 Burst Injections, Cole and Kanner, LIGO Technical Note, 2014

[3] Supervised Learning, IBM Cloud Education, August 2020

[4] Speech and Language Processing, Daniel Jurafsky & James H. Martin, Stanford University, 2020

[5] Find a Hardware Injection, GWOSC

[6] Logistic Regression in Python, Mirko Stojiljković, in realpython.com

[7] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[8] Archive for S5 dataset, GWOSC, https://www.gw-openscience.org/archive/S5

[9] Optimal Placement and Sizing of Shunt Capacitor Banks in the Presence of Harmonics, Mohammad A.S. Masoum and Ewald F. Fuchs, in Power Quality in Power Systems and Electrical Machines, 2015