Gravitational Wave Detection of Compact Binaries through Multivariate Analysis*

Dany Atallah†
Department of Physics & Astronomy, Cardiff University
The Parade, Cardiff CF24 3AA, United Kingdom
Department of Physics & Astronomy, California State University Long Beach
Long Beach, CA 90840, USA
Department of Physics, University of Florida
Gainesville, FL 32603, USA

Patrick J. Sutton and Iain Dorrington
Department of Physics & Astronomy, Cardiff University
The Parade, Cardiff CF24 3AA, United Kingdom

(Dated: August 9, 2016)

We utilize a technique called Multivariate Analysis (MVA) in an attempt to improve LIGO sensitivity to Gravitational Wave (GW) signals sourced from merging binaries composed of any combination of black-holes and neutron-stars; otherwise known as "Compact-object Binary Coalescence" or CBC. MVA is being used by researchers at Cardiff University to look for un-modelled GW signals and by others to look for CBC signals, with promising results. The MVA pipeline used for the un-modeled search can theoretically use more data than the MVA in use by other researchers to search for CBCs, potentially making a more powerful classifier. In principle, this extra information could improve the sensitivity of the analysis. For this reason, we have attempted to adapt the MVA pipeline used in the un-modelled search to classify triggers which are characterized by the parameters which identify them in a templated search for CBC signals.

I. INTRODUCTION

Within the framework of Einstein's Theory of General Relativity, Gravitational Waves (GW) are described to be small oscillations within the fabric of space-time. These oscillations have been shown to be detectable by Earth-based GW detectors, specifically the Livingston, L1, and Hanford, H1, interferometers. Each interferometer is set up with two arms in an "L-shape" such that the fluctuations of spacial geometry due to GWs can be fully exploited and measured. A beam of light is shined through a mirror angled at 45 degrees which splits the original beam into two beams perfectly out of phase with each other. These beams travel down the corridors of each arm and bounce off a mirror at the end of each arm. They propagate back up the arms to a detector which can record when the beams of light change phase. When a GW passes through the detectors, the arms compress and elongate in a rhythmic motion in accordance with the polarization of the GW. The compression and elongation of the arms distorts the distance light must travel down each arm and thus alters the phase of light beams traveling down each arm.

The interferometer can measure these minuscule changes in arm length by analyzing how out of phase the light is by the time it reaches the detector. The change in interferometer length is known as strain and

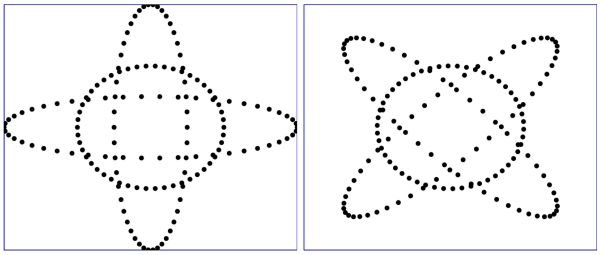

FIG. 1. Plus polarization h_+ (left) and cross polarization h_- (right) of gravitational waves

is modeled computationally by the statement:

$$h^\alpha(t) = F_+^\alpha h_+^\alpha(t) + F_-^\alpha h_-^\alpha(t)$$

Here, F_+^α and F_-^α represent the antenna response functions which describe the sensitivity of the detector α to the plus and cross polarizations of a GW while $h_+^\alpha(t)$ and $h_-^\alpha(t)$ represent the propagation of a GW of plus and cross polarizations.

The detection of GW150914 in conjunction with the second detection, GW151226, has ushered in the era of GW astronomy and heightened the possibility of testing detection methods by using binary-black-hole (BBH) mergers as reference points. Up to this point though, detection of GWs from binary systems, whether black-hole/black-hole, neutron-star/neutron-star, or black-hole/neutron-star depends on a fully templated search.

In order to make the new era of Gravitational Wave detection a reality, Data Analysis techniques must continue

* Work supported by NSF grant to University of Florida IREU program
† Mentored by Dr. Patrick Sutton

to grow in robustness and efficiency. One such idea could be the usage of Multivariate Analysis: a kind of Machine Learning. Using the ROOT toolkit developed for CERN and the PyCBC pipeline, the goal of this project is to figure out how to bridge the data from the two programs and train ROOT on the characteristics that identify a Gravitational Wave. The hope for this project is that a fully trained ROOT will be capable of classifying potential Gravitational Wave signals, also known as *triggers*, from background noise transients with far greater accuracy than current methods.

II. MODELED SEARCH USING PYCBC

We have chosen to use the PyCBC pipeline to produce the data sets to be used for supervised machine learning. PyCBC specializes in modeled searches for GWs and has proven to be a powerful pipeline, as it is credited with being the first pipeline to detect GW150914. In essence, PyCBC works by correlating detector data with waveform templates that model an expected signal. If a candidate signal (trigger) is found in both detectors within a ±15 ms time delay–10 ms for inter-site propagation time and 5 ms for uncertainty in the arrival time of weak signals–then triggers which ring off the same template in both detectors are considered "coincident" and recorded.

The PyCBC analysis is capable of quantifying the background noise within LIGO data. Background noise transients are found by artificially shifting the time stamps of the individual detector data streams by far more than the 10 ms inter-site propagation time–on the scale of 100 ms to hours–such that a real GW event cannot exist in collected coincident data. The coincident triggers isolated by PyCBC after applying a heavy time shift are guaranteed to be glitches.

Coincident triggers found in both, the injection and background search, are quantified by the masses, spins, and the duration of the CBC templates they ring off, the end-times of the signal in each detector, and the SNR and chi-squared statistic the signal is found to have in each detector. Traditionally, the SNR and chi-squared statistic have been the only parameters used by the PyCBC analysis to classify of signal vs. background. With MVA, the full range of trigger properties can be explored, potentially creating a more powerful classifier.

III. DETECTOR BACKGROUND NOISE

A source of great trouble for classifying GW candidate triggers is the ever present background noise within the data collected by the interferometers. The output of the detector α is a linear combination of the signal and noise n^α and is quantified by the statement:

$$d^\alpha(t+\Delta t^\alpha(\Omega)) = F_+^\alpha(\Omega)h_+^\alpha(t) + F_-^\alpha(\Omega)h_-^\alpha(t) + n^\alpha(t+\Delta t^\alpha(\Omega))$$

Again, the F_+^α and F_-^α represent the antenna response functions which describe the sensitivity of the detector α to the plus and cross polarizations of a GW while $h_+^\alpha(t)$ and $h_-^\alpha(t)$ represent the propagation of a GW of plus and cross polarizations. Ω represents the direction of the GW source relative to the location of the detector.

In order for a trigger to be considered a gravitational wave, the trigger must be present in both the H1 and L1 detectors within the bounds placed on inter-site propagation time. The noise background is formed by cross-correlating the detector streams after time-shifting the data by multiples of 100 ms.

If the noise n^α is as loud as an incoming GW within a frequency band, then it becomes far more difficult to differentiate signal form noise and completely impossible if the incoming signal is quieter than the noise. A trigger's significance is measured by how often the background noise in a detector can produce a glitch with the same strain. A weak signal would be one with a low detection significance. With MVA, it can be possible to greatly increase the possibility of accurately classifying weak signals otherwise lost to low significance measurements by engaging the full parameter space PyCBC uses to quantify a trigger with a possible template.

IV. MACHINE LEARNING

Machine Learning is a field of computation which is proving itself to be a very useful tool for analyzing immensely large data sets. To give an example, a scenario in which machine learning would be useful is betting on a football match. Each experienced gambler would have their own rule of thumb for what football team to bet on in a given lineup. In many cases, the opinions of a select few experienced gambler line up, but in others their opinions diverge. Machine Learning is most useful in these kinds of situations–where many data samples exist which are weakly correlated to an outcome. A machine learning algorithm can be fed the statistics correlated to an outcome and discern the best possible way to combine each "rule of thumb" to make accurate predictions and distinguish the signal from the noise.

In order for Machine Learning to be a viable tool for solving a problem, it requires three rules to be satisfied. First, a pattern must exist. This means that we must be able to initially recognize a signal from the background. Gravitational Wave detectors depend on the fact that a GW signal can be isolated from the noise background and so we can rest assured that a pattern does in fact exist which differentiates a GW signal from background noise transients (glitches).

Second, machine learning will only work if the user has data to be analyzed. The more data that can be fed to a machine learner, the more accurate the classifier will be. Luckily for the LIGO collaboration, there is a plenty of data collected by pipelines such as PyCBC. The Py-

CBC pipeline outputs statistics on triggers within LIGO data, such as the possible masses and spins of a trigger and the SNR and chi-squared statistics describing the trigger in each detector. In the case of machine learning for the sake of GW classification, the data used to train our classifier is formed by PyCBC analysis of GW injections superimposed within a routine science run and the PyCBC analysis of glitches labeled as "false alarms".

Third, machine learning is most viable in a system where an exact solution cannot be exacted mathematically. For this reason, machine learning is an incredibly viable tool in noisy environments. The noise within LIGO data is non-Gaussian and will inherently obstruct GW signals within the data in random and unpredictable ways, but a good machine learning algorithm will be able to identify the patterns that qualify a data set as either signal or noise if strong patterns exist.

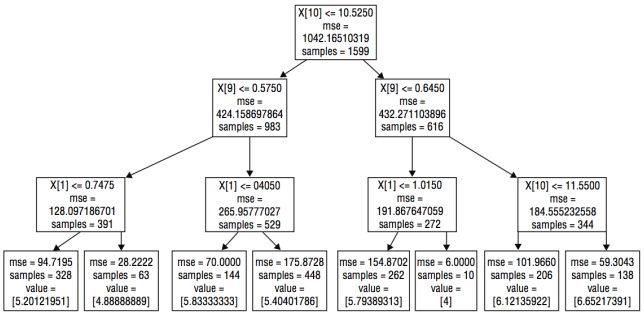

FIG. 2. An example of a trained decision tree with 10 attributes, denoted X[1] to X[10]. Each data point in the testing set is classified based on these nodes starting from the top and working downwards. The final row (the leaf nodes) is the predicted result for the data, given by value. Each node also contains the number of samples (data points) in that node and the mean square error. The tree is trained to minimize the average error.

A. Multivariate Analysis

The branch of Machine Learning employed in this project is called Multivariate Analysis (MVA). Literally translated, Multivariate Analysis is a "Multiple Variable Analysis". MVA is used in situations where you wish to train a classifier on a system which can be quantified with numerical sets of data. The data sets must be predefined by the user, representing signal or noise. We can use the parameters of a priori known injections and a priori known glitches found by PyCBC to train an MVA classifier.

B. Boosted Decision Tree MVA

Boosted Decision Tree (BDT) MVA is the kind of Multivariate Analysis we employ to discriminate GW signals from glitches in a sample of triggers. BDT MVA uses a large number of simple classifiers called binary decision trees to classify data. A trained binary decision tree classifies data by passing it through a series of yes/no questions.

The power behind BDT is that it works by taking advantage of a principle called *Ensemble Methods*. If there is a single classifier which can be trained, at best, to predict whether a trigger is signal or noise to 55% accuracy, then the precision of this classifier is just a little better than flipping a coin. But, let us train 100 classifiers, each with a 55% chance of correctly predicting whether a trigger is signal or noise. The probability of more than 50 of the classifiers accurately classifying signal from noise is much higher than a single classifier of 55% accuracy doing the classification on it's own.

This is the guiding principle behind Ensemble Methods: If we use many classifiers, an *ensemble*, then each individual classifier–otherwise known as *base learners*–only needs to be a little more accurate than 50% for the consensus vote of the ensemble to be impeccably accurate.

In BDT, the base learners are binary decision trees. A binary decision tree is shown in the above figure. A binary decision tree makes a series of yes/no decisions to arrive at a final conclusion. It is composed of two fundamental components: a leaf node and a decision node. Decision nodes divide the data based on a simple true/false condition. Trials that satisfy the condition go down one branch of the tree, trials that do not go down another branch. Leaf nodes do not divide the data, but assign a prediction to the data set–signal or noise–according to the leaf node that is reached.

Boosting a decision tree creates an ensemble of binary decision trees which have uniquely weighted votes. During the training process, each tree is weighted according to how accurately it can discern signal from noise. The process trains each binary decision tree on a different subset of the training sets. The weight of the binary decision tree determines how important its vote is.

C. ROOT Toolkit for Multivariate Analysis

Our weapon of choice for supervised machine learning is the Toolkit for Multivariate Analysis (TMVA) package created with the ROOT data analysis framework, a software tool created for CERN. TMVA contains different MVA methods including our preferred choice of BDT MVA. TMVA requires an input of data sets representing known signal and data sets representing known noise. The user must split the sets randomly into two child sets; one child set is used for training the classifier and the other for testing the classifier's performance. The signal and background child sets that are used for testing the classifier do not contain events from the training sets, thus producing an unbiased assessment of the classifier's

performance.

The training results of the classifier are stored in a "weight" file containing all the information required to classify future data sets. The MVA classifies a data set by assigning a number ranging from -1 to +1. If an event is classified with a +1, it means that the MVA is absolutely certain that the event is a signal while a -1 represents an absolute certainty of noise. How well you trust the MVA can be determined by inspecting the testing results.

V. TRAINING A BDT CLASSIFIER

Machine Learning depends on the parameters of a priori known data type, in this case signal and background. We start with the parameters of known injections and glitches, then split the samples randomly into training and testing child sets. The MVA is told what kind of data it is being trained on–signal or background–but is unaware of the data type during testing.

A previous project involving the classification of GW emissions from Gamma Ray Bursts, or GW bursts worked on by Patrick Sutton at Cardiff University utilized a script of code written in C++ called *RootLoader.C*. RootLoader accepts 4 .txt files as input–signal training, signal testing, background training, and background testing–and organizes the data into files recognizable by ROOT. The xtmvaClassification.py module is the MVA trainer within the TMVA package. A set of commands are executed by the user telling xtmvaClassification.py what form of MVA the user desires to utilize, BDT in this case. The default BDT configuration options are used for the sake of prioritizing the development of the methodology for creating a classifier.

The goal of this project was to apply a similar methodology in order to create a BDT classifier trained on the signals of CBC injections. In order to accomplish this, a method of analyzing and selectively extracting trigger parameters quantified by PyCBC had to be developed.

I successfully created several functions in Python from scratch capable of tackling this task; they are contained within a module named *gwx3k.py*. The module contains four functions, each of which completes a set of tasks within the data analysis procedure. The data files analyzed are in the HDF format and so the H5py Python module was required to do the analysis. HDF files function like file cabinets: the file contains cabinets which either contain data or folders which contain data or possibly more folders. The base folder will contain raw numerical data which can be extracted with a set of commands within the H5py library.

A. gwx3k()

The gwx3k() function accepts 4 input arguments–a list of PyCBC outputs of injection runs from H1, a list of PyCBC outputs of injection runs from L1, a list of coincidence files which allows a user to easily find triggers coincident between the two detectors, and a start time for corresponding to where in the detector data the user wishes to begin analysis. gwx3k() can be configured to reject any trigger which does not meet a minimum SNR value in any of the detectors–by default a minimum SNR of $\geqslant 6.0$ has been chosen. The analysis searches through a coincidence file and isolates known injection triggers which meet the minimum SNR requirement. The triggers which survive the cut are then corresponded to template paramters which correctly quantified the signal–the masses and spins of the source binary system. The statistical parameters quantified during the PyCBC analysis, such as the SNR and chi-squared statistic, are extracted from the individual H1 and L1 data files. The parameters describing the CBC are stored in a list of lists which can be accessed and manipulated to the user's desire. The list of lists is then printed to a text file for the next part of the procedure.

Among the parameters, two "spectator values" are included which can be passed through the xtmvaClassification pipeline without machine learning being done on them. In other words "in one ear, out the other". These spectator values will allow the user to correlate the results of MVA classification to the data set classified. This is mandatory in order for a user to see what the MVA classified an individual trigger as.

B. noiseminer()

The noiseminer() function operates identically to the gwx3k() function except that noiseminer() is configured to search for, isolate, and extract glitches resulting form heavy time shifts as quantified by the PyCBC analysis. For both functions, the extraction process has been vectorized using the numpy module. Vectorization greatly increases the efficiency of the analysis procedure compared to traditional for loops. The vectorization allows a final text file containing rows of selectively isolated data sets from a sample of 82 million triggers to be created within 5 minutes.

The list of parameters the classifier is to be trained on are as follows: mass 1, mass 2, spin 1, spin 2, Template ID, Chi-squared DOF, Template Duration, L1 Chi-squared, L1 CoA Phase, L1 end time, L1 Sigma-squared, L1 SNR, H1 Chi-squared, H1 CoA Phase, H1 end time, H1 Sigma-squared, and H1 SNR.

C. itemsplitter()

RootLoader.C requires four text files containing data sets grouped for signal training, signal testing, background training, and background testing–itemsplitter() creates these child sets from the parent sets created by gwx3k() and noiseminer(). First, the injection and

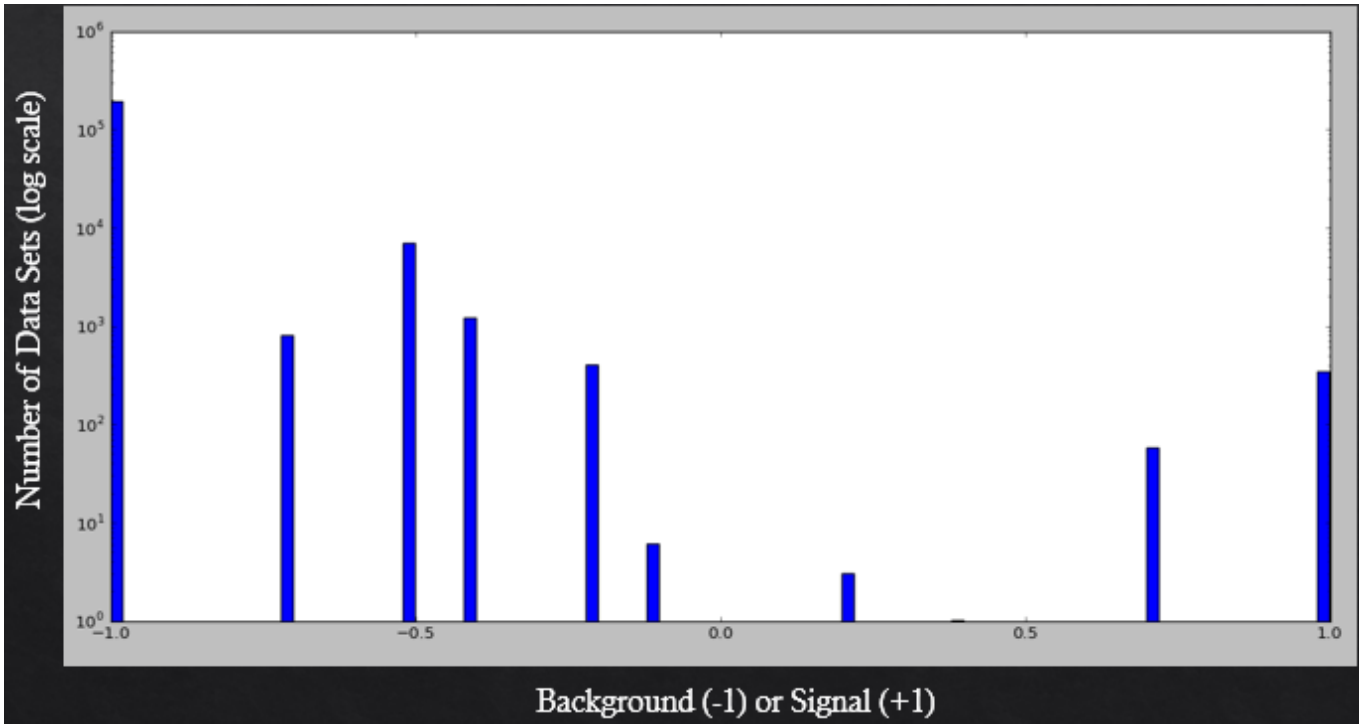

FIG. 3. The y-axis represents the amount of triggers rated on a logarithmic scale; the x-axis represents the rating itself.

glitch/noise data sets are shuffled and then split in half to form two child data files from the injection file and two child data files from the glitch/noise files. These files are fully prepared for passing on to RootLoader.

D. dataswap()

The final portion of the classifier training procedure requires the extraction of the testing results. After training, the BDT classifier is tested with trigger parameter sets. The classifier rates the triggers on a scale from -1 to +1. A -1 means that the classifier is certain that the trigger is a glitch while a +1 means that the classifier is certain the trigger is a signal. The results of the test are output to a ROOT file which can be interacted with through a set of C++ commands or usage of the Python module PyROOT.

The dataswap() function extracts the data from the ROOT file alongside "spectator values" that the MVA tool does not do machine learning on. These spectator values are the indices which correlate a BDT score with the trigger which earned that score. The scores are organized into a list of lists alongside their respective indices.

Within the original HDF trigger coincidence files, there is a branch called 'stat' which is used to rate the performance of the PyCBC classifier. Each trigger's 'stat' is replaced with the BDT score in a new copy of the trigger coincidence file. In short, we can use the same programs used to rate the 'stat' branch to rate the BDT performance if the BDT scores are simply replaced with the 'stat' scores. The BDT classifier is rated based on how accurately it can isolate the signals from the glitches. For example, if the classifier has a tendency to rate injections with scores < 0 and/or glitches with scores > 0, then the creation of that classifier is a terrible waste of server time.

VI. NEXT STEPS

Up till now, the classifier has been run once due to time running out in the project period. The run was successful, but produced "dummy results" because the trigger indices were not marked as "spectator values" during the training process and as a consequence, the results were incredibly accurate. The injections were correlated with indices < 10000 while the glitches were correlated with indices ranging from 0 up to 82 million. This caused the indices of the training sets to be the most important decision node within each binary decision tree in the BDT classifier.

The MVA run which produced the figure above had a testing set of about 200,000 sets of glitch parameters and about 400 sets of injection parameters. The histogram dictates that about 350 parameter sets were classified as definite signal and about 150,000 classified as definite background. Again, these are dummy results from a test run including training on indices which correlate triggers to their location in their coincidence HDF file.

The next part of the project will first involve working out the bugs in the known set of commands running xtmvaClassification.py so that spectator values can be cor-

rectly left unanalyzed by the TMVA package. Next, we will begin experimenting with tuning specific computational parameters which affect the *boosting* process of the MVA to maximize the classifiers accuracy from training. Furthermore, we will see which parameters are seen as by the BDT as being the most characteristic of singling out the signals form the glitches. Figuring out which parameters are most important for the classification process may bring new insight into what kinds of CBC configurations are most likely to occur due to the Template ID being a parameter of the training process. The MVA process also includes detection times, so if there happens to be a correlation between signals and/or glitches being more prevalent at certain times than others, it might lead to an interesting way of gauging issues with the detector instrumentation.