Detecting Continuous Gravitational Waves

Hamouda Mtir
EURECOM
Antibes, France
hamouda.mtir@eurecom.fr

Marco Sorbi
EURECOM
Antibes, France
marco.sorbi@eurecom.fr

Gabriele Spina
EURECOM
Antibes, France
gabriele.spina@eurecom.fr

***Abstract*—In this project we propose a possible pipeline for the problem of detecting Continuous Gravitational Waves in interferometers' data. The proposed approach includes new features based on the SOAP algorithm and an ensemble model with two base classifiers and a Logistic regression on top of them. Different models were compared, with some of them obtaining good results. The code for the project can be found at our GitHub repository [1].**

I. Introduction

The task idea has been inspired by the Kaggle competition "G2Net Detecting Continuous Gravitational Waves" [2] [3], which aims to find Continuous Gravitational Waves (CGWs) in measurements taken by LIGO interferometers. Scientists detected the first class of gravitational waves in 2015. There are four classes, yet at present only signals from merging black holes and neutron stars have been detected. Among those remaining are Continuous Gravitational Wave signals, which are weak and long-lasting signals emitted by rapidly spinning neutron stars. There could potentially be many continuous signals from neutron stars in our own galaxy, but the current challenge for scientists is to make the first detection. Detecting CGW is a difficult task for several reasons:

- the typical amplitudes of the signals are not known, but they could be one or two orders of magnitude lower than the amplitude of the detector noise [2];
- the shape of the signal is not known a priori, as there are no information about the source (position, ellipticity) [4];
- the signals are theoretically expected to be quasi-monochromatic[1]. However the Doppler effect shifts the frequency of the signal over time, because of the relative motion between the CGW source and the Earth [5];
- the frequency of the CGW usually decreases over time, since the CGW source looses rotational energy over time due to the emission of CGWs and electromagnetic radiation. This effect is known as spin-down [5].

For this project we propose a new approach to the problem using traditional Machine Learning techniques and completely new features. In particular, we employed data augmentation and preprocessing techniques to extract meaningful data from the spectrograms of the measurements, and we used a model ensemble of classifiers to predict the probability of each record of containing a CGW in it.

[1]signal whose spectrum consists only on a single frequency or at most on very fine band of frequencies

II. Related Work

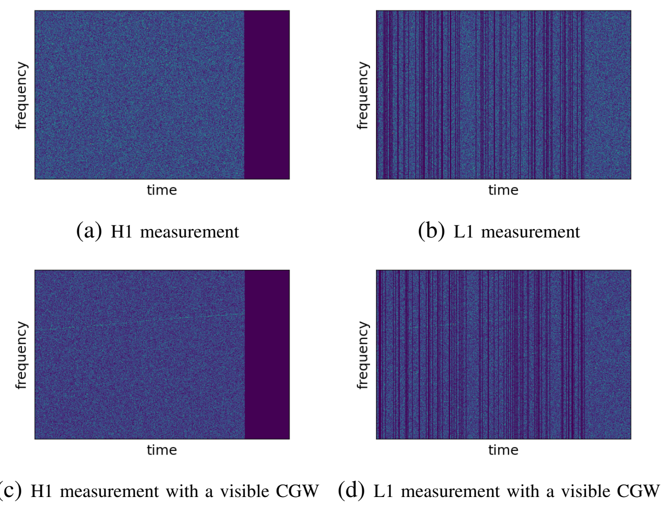
The search for CGW can be divided into different categories, based on the amount of known information, spanning from *targeted* searches, for which the source parameters are well-know, to *all-sky* searches for which the entire portion of the parameter space is explored [4] [6], and which is also the setting of this project.
A traditional approach to the problem uses *matched filtering*, which is an optimal method for this task. A matched filter is a linear filter used for maximizing the Signal-to-Noise Ratio (SNR) of the original signal by correlating a known signal template with the data [7]. A visual demo of the technique is available at [8]. However, as said, it is required that the shape of the signal is known, which is not the case for CGW. The method ends to be computationally expensive and unfeasible for all-sky searches: it would need to evaluate the correlation between the detector signal and all the possible templates that can be created by varying the source parameters.

An alternative approach is represented by the usage of Convolutional Neural Networks, since matched filtering can be regarded as a convolutional layer with a set of predefined kernels [7]. These techniques allow to achieve real time analysis and detection [9] with a speed up of two or three orders of magnitude. However, CNNs perform well in the detection of gravitational waves from merging binary systems of black holes or neutron stars, for which the signal amplitude is way higher than our task, while they struggle on the analysis of signals with low SNR. Moreover it would need a lot of data augmentation and resources to train a proper CNN, which becomes unfeasible for our scope, even if they are most probably the best approach to this task.
Therefore, given the very strict limitations and high complexity of the project, we will focus mostly on feature engineering and we will try to apply traditional ML approaches to the problem.

III. Dataset

A. Description

The entire dataset available on Kaggle [2] contains a labeled train set of 15 GB and an unlabeled test set of 212 GB, used by Kaggle for evaluations. The dataset includes records of measurements taken by the two LIGO interferometers, H1 and L1, split into chunks in time. Each record includes a timestamp and Short-time Fourier Transform. Some records also include simulated CGW or simulated noise, and a binary



(a) H1 measurement (b) L1 measurement (c) H1 measurement with a visible CGW (d) L1 measurement with a visible CGW

Fig. 1: H1 and L1 spectrograms of two records

label indicating whether a CGW was added. The goal of the algorithm is to determine the probability of a CGW to be present in the input. The dataset includes 603 records, 400 of which labeled as 1 (positive records containing a CGW), 200 as 0 (negative records not containing a CGW), and 3 labeled as -1, which are useless as they just represent an easter egg from the creators of the challenge [10], so they were dropped. From this dataset we created our train set and test set by performing a standard 80%-20% split. The former was used to train and validate the models, the latter to test them.

B. Data Exploration

To inspect the data we plotted the spectrogram of each signal for the H1 and L1. All the spectrograms present time gaps and discrepancies between the measurements, maybe due to the interferometers not working together all the time and being shut off independently for multiple reasons (e.g. maintenance).

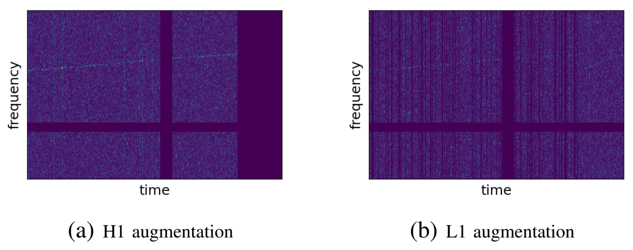
Since CGWs are quasi-monochromatic, we tried to look for quasi-horizontal thin lines, mostly with negative slope for the spin down effect, but also considered other shapes due to the effects described in Section I. Most of the spectrograms appeared to be pure noise, making it difficult to detect the presence of CGWs. Fig. 1a and Fig. 1b show the two H1 and L1 noisy measurement of one example record.

However there are also records which clearly show the presence of a quasi-horizontal line and that are labeled as 1. These correspond to those signals having such a high SNR to be recognized by visual features: we managed to find 21 of these records out of 600. Fig. 1c and Fig. 1d show an example of one of them.

IV. Methods

A. Data Augmentation

As the initial dataset has only 600 samples, we employed a data augmentation technique to generate more samples and



(a) H1 augmentation (b) L1 augmentation

Fig. 2: Example of augmentation of a record

increase the robustness of the final model. The augmentation approach adopted is inspired by SpecAugment [11] presented by Google Brain. It is a data augmentation method for automatic speech recognition, but that fits our needs as we are working with spectrograms. The spectrum augmentation policy consists of three kinds of deformation of the spectrogram:

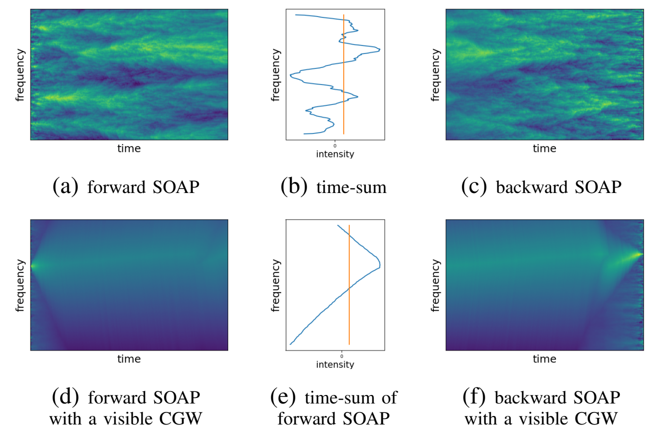
- *Time Warping*: a technique used in signal processing and time series analysis to establish deformation of the time-series in the time direction. Random points on the horizontal time axis shifted left or right by a random distance in the range $[0, W]$ where W is the time warp parameter.
- *Frequency Masking*: f consecutive frequency channels $[f_0, f_0 + f)$ are masked (set to 0), where f is selected randomly from a uniform distribution on the interval $[0, F]$ (where F is the frequency mask parameter) and f_0 is selected from the range $[0, \nu - f)$ where ν is the total number of frequency channels.
- *Time Masking*: t consecutive time steps $[t_0, t_0 + t)$ are masked (set to 0) where t is selected randomly from a uniform distribution on the interval $[0, T]$ (where T is the time mask parameter), and t_0 is chosen from $[0, \tau - t)$ where τ is the total number of time units.

Finally, by combining the three previous techniques, every sample was independently augmented 9 times. Thereby, the final dataset we created was composed of 6000 samples, including the orignal records. Fig. 2 shows an example of a record created with this procedure.

B. Preprocessing

a) Time alignment: The first part of the preprocess consists in the resolution of time discrepancies between H1 and L1 measurements, that was done by filling with zeros the missing timesteps. After this, the multiple detectors version of Snake On A Plane [12] algorithm, based on the Viterbi algorithm [13], was applied. This algorithm is an iterative approach used to find the most likely sequence of states, the Viterbi path, in the context of a Markov process. SOAP has been designed by its authors to detect the most probable track for a continuous gravitational wave assuming its presence, and we based our search on its output. The following steps were done:

- Independent forward and backward in time applications of SOAP, as in Fig. 3, to look for correlations afterwards



(a) forward SOAP (b) time-sum (c) backward SOAP (d) forward SOAP with a visible CGW (e) time-sum of forward SOAP (f) backward SOAP with a visible CGW

Fig. 3: forward and backward SOAP of two records, with forward time-sum and peak threshold

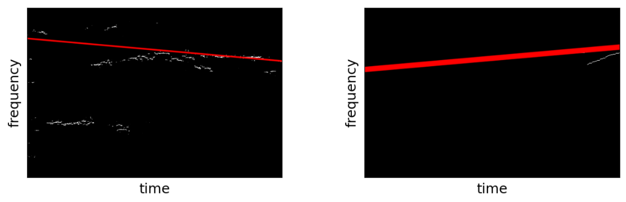


Fig. 4: Lines detected from forward SOAP of two records, the red line width is proportional to the number of votes

- Standardization of the output for each timestep, to make likelihoods comparable at different timesteps.

From the algorithm output we extracted the following features:

1) Lines: The most important line was extracted using the Hough transform, which is a common technique for line detection. Each output is first thresholded using its 0.995 quantile as threshold, then the Hough transform is applied to extract information about the lines in the image. Among them, only the most important one is extracted, using the number of votes in the transform procedure as indicator of importance. Fig. 4 shows the detected lines on two thresholded SOAP outputs. We expect the line importance to be representative of the SNR ratio of the measurement: positive records should be more likely to have a very marked line in the SOAP output with respect to negative records. Each line is described by the polar coordinates (ρ, θ). The following features have been extracted:

- `imp_fw`: line importance of the forward output
- `imp_bw`: line importance of the backward output
- `rho_fw`: ρ of the line of the forward output
- `rho_bw`: ρ of the line of the backward output
- `delta_rho`: $|\rho_{\texttt{fw}} - \rho_{\texttt{bw}}|$

The line slopes θ were not used as most of them had the same value, and their variance would have been too low for this feature to be significant.

2) Peaks: As the frequency change of a wave over time is rather small, to identify tracks we computed the sum along the time axis for each of the forward and backward outputs, obtaining the 1-dimensional functions depending on frequency shown in Fig. 3b and Fig. 3e. Then we detected the peaks of these functions and we extracted the following features:

- `peaks`: the ratio between the second highest peak and the first one, this should be low if a CGW is present because only one relevant track should be detected
- `areas`: setting a threshold at one fifth of the highest peak value, the ratio between the area above the threshold contiguous to the highest peak and the total area above the threshold is computed, this should be low if only noise is present, as there should be more regions.

3) Mixed features: Starting from the sums made in IV-B2, the forward and backward outputs were combined to obtain the following features:

- `correlation`: scalar product of the normalized sums
- `delta_peaks`: absolute distance in frequency position between forward and backward highest peaks
- `peaks_mixed` and `areas_mixed`: the two features in IV-B2 were computed also on the pointwise product of the sums, only on the points where both sums are positive.

4) Local Binary Patterns (LBP): It is a type of visual descriptor used for classification. For each record, two LBP histograms were extracted from both the SOAP outputs and they were used as feature vectors, respectively called `fw_lbp` and `bw_lbp`. Each vector describes the main patterns present in the image: SOAP outputs can be compared by comparing their feature vectors, and we can use their similarities for the classification task.

C. Machine Learning

The extracted features were split into two separate sets, one containing only the LBP feature vectors, the other containing all the remaining. The LBP features were used to train a distance-based model (`model_lbp`), while the others were normalized and then used to train a second model (`model_feat`). A third model of Logistic Regression was used over the predictions of the two base classifiers to combine them into the final predictions. We will refer to this architecture, as "ensemble model". The idea comes from the `StackingClassifier` of the `scikit-learn` library [14] [15], not used in our project but re-implemented to better match our requirements with the data augmentation.

V. Experiments / Results / Discussion

A. Baseline

A simple baseline was computed preliminarly, without using data augmentation, by extracting two sets of simple features from both the spectrograms of each record: statistical and autocorrelation. These features were lately substituted with the ones explained in IV-B, since the features extracted from SOAP constitute a more informative generalization of the statistical features, while the autocorrelation features resulted to have a low importance in the tested model.

1) Statistical features: Since we are looking mostly for fine horizontal lines, a simple strategy could be projecting each spectrogram on the frequency axis by summing each row, obtaining two one-dimensional arrays of 360 elements (one for each measurement). From each array we extracted mean value, standard deviation, minimum value, maximum value and the 25%, 50%, 75% quantiles.

2) Autocorrelation features: For each measurement, the signal autocorrelation at 24 hours was computed. We expect to have a non-autocorrelated signal if there is only noise, and a positive autocorrelation if the CGW is present. The autocorrelation is computed at 24 hours because this counteracts the Doppler effect due to the rotation of the Earth during the day.

3) Evaluation: For the baseline evaluation the features were normalized to have zero mean and unitary standard deviation, and then fed into a model of Logistic Regression, balancing the weights of each class. The model was validated using a 5 fold cross-validation over our train set, using AUC-ROC as measure, as it is the method used on Kaggle, obtaining a score of 0.668.

B. Model selection

Various classification models were trained to predict the class probabilities:

- `model_lbp`:
  - *K-nearest Neighbors (KNN)*: probabilities are computed by comparing the record with its nearest neighbors. The metric used to measure similarity between the LBP feature vectors was the KL-divergence. In particular, to obtain a Symmetrized KL-divergence, we averaged the KL-divergences between two records:

$$SKL(p,q) = \frac{KL(p||q) + KL(q||p)}{2} \tag{1}$$

    Moreover, we also averaged the measures between the forward and the backward feature vectors:

$$\overline{SKL}(p,q) = \frac{SKL(p_{\mathtt{fw}}, q_{\mathtt{fw}}) + SKL(p_{\mathtt{bw}}, q_{\mathtt{bw}})}{2} \tag{2}$$

  - *Support Vector Machine (SVM)*: tries to find the optimal hyperplane that better separates the two classes. It was tested using a custom kernel [16] computed as:

$$K(x_1, x_2) = e^{-a\overline{SKL}(x_1,x_2)} \tag{3}$$

    with a normalizing factor.
- `model_feat`:
  - *K-nearest Neighbors (KNN)*
  - *Random Forest Classifier (RF)*: an estimator that fits an ensemble of decision tree classifiers and uses averaging to improve the predictive accuracy.
  - *Multi-Layer Perceptron (MLP)*: neural network composed of multiple layers of interconnected perceptrons, known for their ability to learn non-linear relationships in data.
  - *Support Vector Machine (SVM)*: with RBF and polynomal kernels.
  - *Adaptive Boosting (AdaBoost)*: a boosting technique which fits a classifier by updating the weights of the misclassified records of the previous classifier.
  - *Extreme Gradient Boosting (XGBoost)*: a boosting technique which fits a classifier on the residuals of the previous classifier, minimizing the loss function.

TABLE I: Validation results

feat	RF	SVM	MLP	KNN	AdaBoost	XGBoost
lbp						
KNN	0.819	0.776	0.794	0.776	0.802	0.811
SVM	**0.820**	0.780	0.807	0.797	0.805	0.816

C. Validation

To validate the models a cross-validation strategy with 5 partitions was applied. A first layer of cross validation was used to generate predictions of `model_feat` and `model_lbp` over the entire augmented train dataset, while a second layer was used for the final estimator, which was trained using the predictions of the base estimators. For the second layer of cross validation, for each partition, the classifier was trained using the augmented records' predictions, and tested only on the original non-augmented records' predictions. This was done to ensure the effectiveness of the validation procedure: since the model is going to be tested on a non-augmented dataset, it should be tested against the same type of data also during the validation phase. Table I reports the best result obtained for each combination of `model_lbp` and `model_feat`. The best performing combination is, respectively, SVM with RF, and it is the one used in the following experiments. The best hyperparameters set, in the `scikit-learn` implementation [15], is:

- SVM:
  - `kernel`: `"precomputed"`
  - `C`: 0.1
- RF:
  - `n_estimators`: 10000
  - `criterion`: `"entropy"`
  - `max_depth`: 5
  - `max_features`: `n_features`
  - `class_weight`: `"balanced"`

D. Augmentation

The augmentation technique has been validated on the chosen models, SVM and RF, repeating the same procedure as in V-C using only the original records. Without augmentation, the cross-validation average result drops to 0.806.

E. Feature selection

To determine which features are relevant among the ones we computed, we looked at the empirical importances computed by the logistic regressor, to compare LBP to the other features,

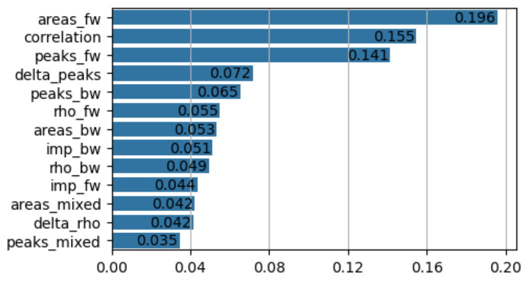


Fig. 5: feature importances

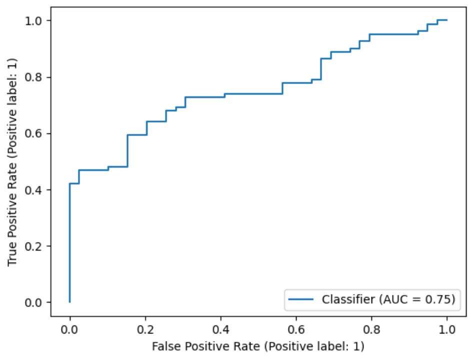


Fig. 6: Testing ROC

and by the random forest. The logistic regressor's coefficients are 2.89 for LBP and 3.20 for the feat model predictions, and the random forest feature importances are reported in Fig. 5. It is visible that the model prefers the forward SOAP application rather than the backward one, and that there are three features that together achieve almost 0.5 (`areas_fw`, `correlation` and `peaks_fw`) and are considered very important to determine if a track is a CGW or just noise. Anyway, as we have a small number of features and the less relevant has an importance greater than 3%, we decided to keep all of them.

F. Results

The best validation result obtained, **0.820**, is much higher than our baseline's result, **0.668**. The final local testing result is **0.751**, corresponding to the ROC in Fig. 6, which is lower than the one obtained in validation as expected, and may indicate slight overfitting over the models' hyperparameters.

The Kaggle submission scores **0.673** as private score, which corresponds to position 564 out of 937. This is lower than the local test: a possible reason is that the train set may be not representative for the test set [17].

VI. Conclusion

In this project we proposed a new pipeline with augmentation techniques and new features for the problem of detecting CGW, and we obtained satisfying results by using a model ensemble of two base classifiers and a Logistic Regression on top of them. We also compared different base classifiers, with the couple Support Vector Machine and Random Forest performing better than the others. Considering the work, the available time, the resources and the limitation to traditional ML techniques without employing CNNs, we can overall be happy with the results, as our features revealed to be meaningful and allowed us to obtain a good final model.

VII. Future Work

During the implementation of the task, we thought about some other things that we could do, but that were not done mainly due to time availability. Some of them are:

- Trying to modify the behavior of SOAP: as the original algorithm assumes that a CGW is present to detect its most probable track, we could change the way the two measurements are combined to improve the noise reduction, for example by making pointwise products between the two and softmaxes. We started to study this approach but we reverted to the original algorithm to respect the deadlines.
- Trying different data augmentations techniques, maybe by creating from scratch new records using the tools of the library PyCBC [18], or by better tuning the hyperparameters of the augmentation. Moreover it would be interesting to find how the size of the augmented dataset impacts the results.
- Trying many more different features. The lack of literature for this task didn't allow us to find meaningful descriptors of the records, and even if our features perform quite good, there is for sure margin of improvement.
- Trying another metric for the `model_lbp`.
- Trying different architectures for the models ensemble or using a single model to perform the prediction.

Moreover a Deep Learning approach using *Convolutional Neural Networks* could be tried, as it is the most standard approach to the task. In this context, data augmentation would have been critical to significantly improve the score.

VIII. Contributions

In a first moment we searched for and analyzed online documentation and previous literature for the task and then we put all the things together to come up with a pipeline to follow. Then we assigned the following tasks for each component:

- Hamouda: PyCBC [18], data exploration, utility functions (data loading, plots, ...), data augmentation and its integration to the pipeline with validation
- Marco: autocorrelation for baseline, SOAP algorithm implementation, Peaks and Mixed features, feature selection and model testing (local and kaggle)
- Gabriele: baseline, Lines and LBP features, metric for LBP model, ML pipeline implementation

General strategies and details, such as the ensemble structure and models to try, have been discussed among all the components.

REFERENCES

[1] *g2net_detecting_cgw*. URL: https://github.com/marS24-0/g2net_detecting_cgw.

[2] *G2Net Detecting Continuous Gravitational Waves*. URL: https://www.kaggle.com/competitions/g2net-detecting-continuous-gravitational-waves.

[3] Rodrigo Tenorio, Michael J. Williams, and Chris Messenger. *Learning to detect continuous gravitational waves*. Tech. rep. LIGO-P2200295. 2022. URL: https://dcc.ligo.org/P2200295.

[4] Pia Astone et al. "Method for all-sky searches of continuous gravitational wave signals using the frequency-Hough transform". In: *Phys. Rev. D* 90 (4 Aug. 2014), p. 042002. DOI: 10.1103/PhysRevD.90.042002. URL: https://link.aps.org/doi/10.1103/PhysRevD.90.042002.

[5] *Search for gravitational waves from rotating neutron stars in our Galaxy with LIGO and Virgo O3 data*. URL: https://www.ligo.org/science/Publication-O3AllSkyCW/.

[6] *10 - The search for continuous gravitational waves (Cristiano Palomba)*. URL: https://www.youtube.com/watch?v=J0Y2MdBBJ-o.

[7] *Matched-filtering Techniques and Deep Neural Networks for Gravitational Wave Astronomy*. URL: https://www.youtube.com/watch?v=p-wocRl9Be0.

[8] *Gravitational-wave Data Analysis: Matched Filtering GW151226*. URL: https://www.youtube.com/watch?v=bBBDR5jf9oU.

[9] *Deep Learning for Real-time Gravitational Wave Detection - Daniel George*. URL: https://www.youtube.com/watch?v=2EPK42cgSTY.

[10] *Regarding Those -1 Labels*. URL: https://www.kaggle.com/competitions/g2net-detecting-continuous-gravitational-waves/discussion/363734.

[11] Daniel S. Park et al. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". In: *Interspeech 2019*. ISCA, Sept. 2019. DOI: 10.21437/interspeech.2019-2680. URL: https://doi.org/10.21437%2Finterspeech.2019-2680.

[12] Joe Bayley, Chris Messenger, and Graham Woan. "Generalized application of the Viterbi algorithm to searches for continuous gravitational-wave signals". In: *Phys. Rev. D* 100 (2 July 2019), p. 023006. DOI: 10.1103/PhysRevD.100.023006. URL: https://link.aps.org/doi/10.1103/PhysRevD.100.023006.

[13] A. J. Viterbi. *IEEE Transactions on Information Theory 13, 260 (1967)*.

[14] *sklearn.ensemble.StackingClassifier*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html.

[15] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[16] Pedro Moreno, Purdy Ho, and Nuno Vasconcelos. "A Kullback-Leibler Divergence Based Kernel for SVM Classification in Multimedia Applications". In: 16 (Mar. 2004).

[17] *High validation accuracy (0.77), low LB score (0.56)*. URL: https://www.kaggle.com/competitions/g2net-detecting-continuous-gravitational-waves/discussion/363810.

[18] *PyCBC*. URL: https://pycbc.org/.