Understanding interferometer lock losses with machine learning

Laurel White[1]
Mentor: Jameson Rollins[2]
[1]Syracuse University
[2]LIGO, California Institute of Technology

Abstract. The Laser Interferometer Gravitational-Wave Observatory (LIGO) detectors are complex systems that must be extremely stable to detect gravitational-wave signals. Numerous control loops are used to maintain detector stability, or "lock." However, a detector can lose lock. A time-consuming lock acquisition process must be undertaken to regain it, reducing the amount of time during which the interferometer is recording data. The causes of some lock losses are unknown. In this project, we use machine learning to analyze time series data from the auxiliary channels in the LIGO Hanford detector, which can indicate changes in the states of various detector components as well as environmental factors such as seismic noise. We first determine features that characterize the data and then perform a regression to identify which of these features can distinguish between data preceding lock losses and data from stable times. We run a clustering algorithm on the predictive features to identify groups of similar lock loss events. The ultimate goal is to minimize the number of lock losses in the future by identifying the underlying causes of lock loss. We determine that this analysis will need to be repeated using different features in order to achieve this goal.

1. Introduction

The Laser Interferometer Gravitational-Wave Observatory (LIGO) operates two detectors in search of gravitational waves. In each detector, a laser beam is split into two separate beams which are sent down two perpendicular arms, reflected at the ends of each, and recombined such that there is near perfect destructive interference at the output port of the beam splittere. Passing gravitational waves distort spacetime, creating a phase difference between the two beams and affecting the interference pattern. In order for these detectors to be sensitive enough to detect the minute phase shifts caused by gravitational waves, they must be in a stable state known as "lock." The mirrors in the interferometers form optical cavities. In lock, the mirrors are controlled with extreme precision such that the laser beams resonate in each cavity, building up power and signal strength, and interfere destructively.[1] An automated control system, Guardian,[2] works through a lock acquisition process before an interferometer is stable and data collection can begin.

The loss of lock leads to a significant amount of downtime as the different componenets are realigned. While a "lock loss" is sometimes caused by a known source, such as an earthquake, there are other times when the reason is less apparent. There are thousands of auxiliary channels that collect data about the components

and environments of each interferometer.We hypothesize that there are features in the auxiliary data channels which signal the approach of lock loss events and which can be linked back to underlying causes of lock loss. We work to identify these causes and share the information with the detector commissioners so that they can work to prevent such events in the future.

In order to do this, we first have to identify features in the data which are most likely to predict lock loss. This is crucial, because selection of features that are not correlated to loss of lock will not yield meaningful results. We then implement an optimized regression algorithm, which needs to be able to reduce a large number of features to only those relevant to our analysis. Lastly, we implement a clustering algorithm on the lock loss samples using the features identified by the regression as indicators of lock loss in order to group lock loss events with common features. A successful end product is a set of features that are proven to precede lock loss events and whose underlying causes can be determined so that the lock loss problem can be approached at the root.

This project is based on previous work carried out by Jameson Rollins.[3] Rollins created a dataset of "lock loss" and "no lock loss" times, extracted amplitude spectral densities (ASDs) from auxiliary channel data during those times, reduced the ASD features to a smaller, more relevant set, and worked with a clustering algorithm that can be used to group related features. We follow the model used in this work towards its end goal of better understanding lock losses so that we can reduce lost time during future data collection.

2. Methods

The three main steps in this project are feature extraction, regression with regularization, and clustering. First, though, we must compile the dataset.

We limit our analysis to Advanced LIGO's second observing run, O2, and the Hanford interferometer. While there are thousands of auxiliary channels monitoring each of the detectors, we choose to focus on a reduced set to decrease the amount of data with which we are working. Our base channel list contains 5385 auxiliary channels. This list is generated by first identifying all channels that were recording at Hanford during O2 with the DQ label, indicating that they are intended for offline analysis, and removing the calibration channels. We then remove 23 channels that started recording in the middle of O2 and 1024 channels that contain completely flat data during our sample times, which we determine by finding the channels for which the difference between the maximum and minimum values reached is zero. We are left with 4338 channels.

The positive sample times are generated by locating the times of all lock losses that occurred during O2 while the Hanford detector was in its observing mode and then subtracting a buffer of 0.25 seconds from each time. The buffer is included to ensure that we do not capture any of the actual lock loss event. We do not want to include the lock loss data in our analysis because we are working to find features that are predictive of lock losses and therefore related to the causes of lock losses. We ultimately have 276 positive sample times.

The negative sample times are generated by identifying the segments during which the Hanford detector was in its observing mode during O2. From each of these segments, a sample is taken every 10000 seconds starting 1000 seconds after the detector entered the observing mode and ending 1000 seconds before it exited the

observing mode. This ensures that the negative samples are distinct and well removed from lock losses. We have 1086 negative sample times.

The length of our samples is a parameter which can be tuned between different iterations, called "runs," of our feature extraction procedure.

2.1. Feature Extraction

To begin our first run, we look for features in the data samples that we believe could potentially be indicative of impending lock loss. We cannot perform the regression using the raw time series data–in which the amplitude of each channel is stored versus time–because that number of feature data points would cause our regression to be time-consuming as well as severely under-determined. The results would not be meaningful. The solution to this problem is to perform calculations on the samples that reduce the number of data points while preserving any characteristics that may allow us to distinguish between positive and negative samples.

The previous work used ASDs, which are calculated by using Fourier analysis to determine the power in each frequency bin that is present in an auxiliary channel at a given time. An unstable oscillation in a control loop, which is a proposed cause of lock losses, would appear as increasing power over time in a specific bin of the ASDs for the relevant channels. However, the channels are samples at rates of up to 16384 Hz. In order to significantly reduce this amount of data, the frequency bins must be large, which may hide interesting features in the data. Instead, we choose to perform various calculations that return single values to extract the features for our runs. Each calculation is performed on data from every channel for every sample.

We use HTCondor to parallelize the feature extraction for our dataset. Once the extraction is complete, we create histograms to analyze the ability of our features to distinguish between positive and negative samples. A histogram is generated for each combination of channel and feature. It plots the values calculated using that channel and feature for the positive and negative samples separately, and they are normalized to create probability density functions. Ideally, certain features calculated for certain channels will show a clear separation between the typical values obtained for positive versus negative times, and these will be the features that the regression identifies as predictive of lock losses.

The following extracted features are adapted from the documentation pages of the tsfresh feature extraction package,[4] which relies heavily on the numpy package[5] and the scipy package.[6] The calculations are done on a time series x containing n data points.

- Complexity

$$\sqrt{\sum_{i=0}^{n-2lag} (x_i - x_{i+1})^2} \tag{1}$$

  Complexity is a measure of how quickly the data in a time series move up and down. We normalize it for our time series so that it depends on the frequency of oscillation of the data more so than the magnitude. It is important to note that this definition of complexity does not take into account the overall shape of the time series, so it may assign a high complexity value to a quickly oscillating sine wave even though this would not typically be considered "complex."

- Absolute energy

$$E = \sum_{i=1}^{n} x_i^2 \tag{2}$$

Absolute energy measures the energy in a time series based on the magnitudes of the values of the data points.

- Mean value

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{3}$$

The mean is a simple calculation of the average value of the data points in the time series.

- Standard deviation

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2} \tag{4}$$

The standard deviation calculates the average distance of the data points in the time series from their mean value.

- Variance

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{5}$$

The variance is the square of the standard deviation.

- Fisher kurtosis[7]

$$g_2 - 3 = \frac{m_4}{m_2^2} - 3 \tag{6}$$

where

$$m_r = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^r \tag{7}$$

Kurtosis describes the sharpness of the peak of the distribution of data in the time series. Specifically, Fisher kurtosis assigns a value of zero to a normal distribution.

- Skewness[7]

$$g_1 = \frac{m_3}{m_2^{3/2}} \tag{8}$$

Skewness describes the symmetry of the peak of the distribution of data in the time series. A normal distribution has a skewness of zero while a positive skewness indicates that there are more data on the right side of the distribution.

- Absolute sum of changes

$$\sum_{i=1}^{n-1} |x_{i+1} - x_i| \tag{9}$$

The absolute sum of changes sums the magnitudes of the differences between each set of consecutive data points in the time series.

- Mean absolute change

$$\frac{1}{n} \sum_{i=1}^{n-1} |x_{i+1} - x_i| \tag{10}$$

The mean absolute change measures the average difference between the magnitudes of consecutive points in the time series.

Together, these features should characterize the magnitude, distribution, and variability of the data in a time series. We also calculate the difference between the maximum and minimum values in each time series, which should capture large spikes or dips in the data.

2.1.1. Run 0 The first run uses a sample length of 4 seconds, meaning that, for each sample time, a sample is created using data starting 4 seconds before the time up until the sample time. Errors in retrieving the data necessitate the removal of 3 positive sample times and 30 negative sample times, leaving 273 positive and 1056 negative samples.

2.1.2. Run 1 The second run uses a sample length of 10 seconds, and the resolution of data retrieval errors allows us to use the full lists of sample times.

2.1.3. Run 2 The third run returns to the sample length of 4 seconds but continues to use the full lists of sample times as in Run 1. It also adds a feature, permutation entropy, which is adapted from the EntroPy package.[8] It is calculated using

$$H = -\sum_{i=1}^{n!} p(\pi) log_2(p(\pi)) \tag{11}$$

where n is a specified order and p(π) is the probability that a permutation π of order n of the data points occurs. We use an order of 3. The permutation entropy measures complexity by analyzing the amount of repetition of patterns between the data points in a time series.

2.2. Regression

We use scikit-learn,[9] a Python package, to analyze our dataset with machine learning. We apply a regression to develop a linear model that correlates the features in our data to the positive or negative labels and identify those features that contribute most to the model. We want to use a regression algorithm that implements regularization to avoid overfitting the data by using too many channels and features. For this, we choose the Lasso model.

We use the extracted features as the inputs for the Lasso regression. The following procedure is developed based on the previous work[3] and the scikit-learn documentation.[9] First, the features are compiled into an m-by-n matrix $\mathbf{X}$. Each of the n columns contains a feature extracted from one channel, and the total set of columns includes each feature extracted from each channel. Each of the m rows contains features that were extracted from one sample time in the data. We then create an output vector $\mathbf{y}$ with m components containing the "lock loss" and "clean" labels as either +1 or -1. For a given integer c where $1 \leq c \leq m$, the c-th component of $\mathbf{y}$ is the label corresponding to the time from which the c-th row in $\mathbf{X}$ was extracted. The regression then solves

$$\mathbf{Xw} = \mathbf{y} \tag{12}$$

for a vector of coefficients $\mathbf{w}$ that linearly maps $\mathbf{X}$ onto $\mathbf{y}$. Figure 1 visualizes the regression problem.

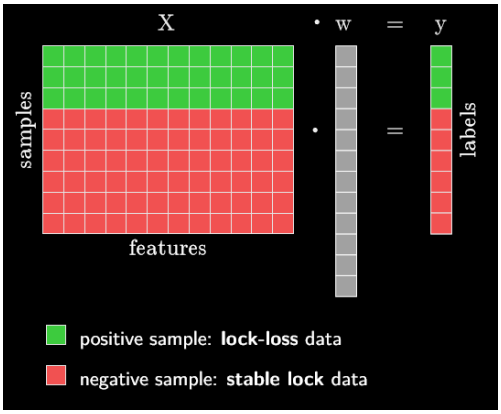


Figure 1: The Lasso regression attempts to solve for $\mathbf{w}$ in this equation. Graphic courtesy of Rollins.[3]

It is important to note that we are working with more features and auxiliary channels than there are sample times in our dataset. Therefore, n is greater than m, and equation 12 is underdetermined. This is why we need to implement regularization, which the Lasso algorithm does by solving equation 12 such that it minimizes

$$\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||_2^2 + \alpha||\mathbf{w}||_1 \tag{13}$$

where α is the regularization coefficient and $||\mathbf{w}||_1$ is the L1-norm of $\mathbf{w}$

$$||\mathbf{w}||_1 = \sum_i |\mathbf{w}_i| \tag{14}$$

The L1-norm is responsible for penalizing the coefficients of $\mathbf{w}$. Using the L1-norm drives some of the coefficients to be exactly zero so that the final linear model does not depend on all the features. Higher α values penalize more coefficients, resulting in models that depend on fewer features.

In order to use Lasso, we pre-process the data using scikit-learn. We split the data such that 70 percent is used to train the model and 30 percent serves as a testing set. We create a function that scales the training set so that it has zero mean and unit variance, and then we apply the same scaling function to the testing set.

We have to carefully select the value of α such that we do not end up with too many or too few features upon which our model relies. One of the advantages of using the scikit-learn package for this task is that we can easily repeat the analysis with small tweaks while inputting a minimal amount of manual labor. Once a round of regression is complete, we evaluate its success. We produce receiver operating characteristic (ROC) curves for several rounds of regression with different α values. On the x-axis, these plots show the false positive rate that is calculated by comparing the model predictions for the labels of the testing data to the actual labels of the testing data. Similarly, the true positive rate is on the y-axis.[9] The area under the curve (AUC) for a ROC curve is a score that can be used to determine how well a model performs when applied to a testing set; higher scores indicate better performance. We also visually examine the histograms for the features assigned the highest coefficients by the regression to determine whether they clearly distinguish between lock losses and clean times.

2.3. Clustering

Once we obtain a set of features that have been identified by the Lasso regression as indicators of lock losses, we use the K-means clustering algorithm, available within scikit-learn, to cluster the lock loss samples based on these interesting features.

We choose K-means because it accepts an input for the desired number of clusters. A different algorithm which automatically determines the number of clusters might use a large number of very specific clusters, which would tell us little about common features shared by lock losses. Instead, we use K-means to group our lock losses into three clusters in order to identify causes that underlie broad numbers of lock losses.

For n samples of a dataset x, the algorithm works to minimize the inertia

$$\sum_{i=0}^{n} \min_{\mu_j \in C} (||x_i - \mu_j||^2) \tag{15}$$

where each cluster C is defined by its center μ_j. The algorithm first chooses random initial values for the μ_j of each cluster and assigns each point to the closest center. It then shifts μ_j for each cluster to the average of the points in that cluster, redefining the clusters. Convergence is declared once the amount by which μ_j is shifted reaches a specified tolerance value, which defaults to 0.0001.[9]

To create the input matrix for the clustering, we select only the features pulled out by the Lasso regression with α equal to 0.0261. These features become the columns and the positive samples become the rows, leaving us with a 276 by 153 matrix. Once again, we scale the data to have zero mean and unit variance.

Once we have sorted the lock losses into three clusters, we recreate the histograms of the interesting features except we separately plot the values for the samples assigned to each cluster. We look at these plots to determine whether the clusters are clearly defined in any feature space.

3. Results

Of the 3 runs that we completed, we choose to continue with the extracted features from Run 2 because it uses the most complete dataset and feature set.

After running several rounds of regression with different α values on the data from Run 2, we obtain Figure 2. An α value of 0.0261 yields a model with a perfect ROC AUC score, while a larger value could potentially cause the model to be underfit.

The chosen model assigns nonzero weights to 153 features, of which the complexity from `H1:OMC-FPGA_DTONE_IN1_DQ` (shown in Figure 3a) and the permutation entropy from `H1:OMC-LSC_DITHER_OUT_DQ` (shown in Figure 3b) are among the most significant contributors. Though the model performs well on the testing data, most of the other features it identifies are uninteresting because the histograms do not show a clear separation between the values for the positive and negative samples.

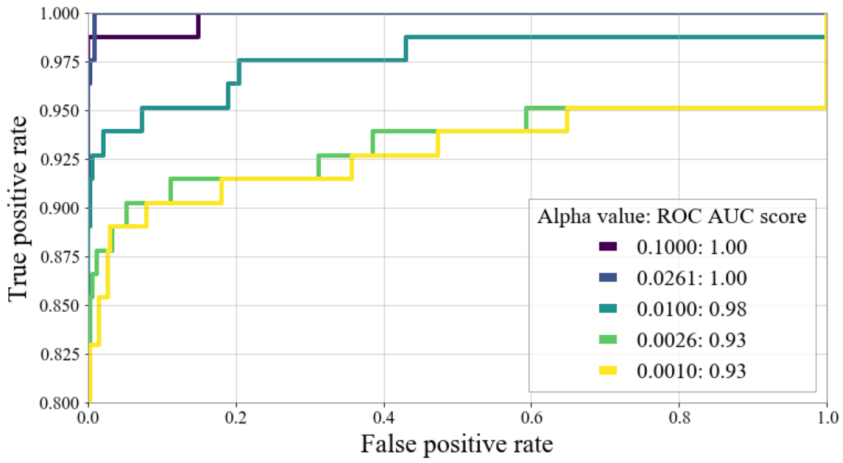
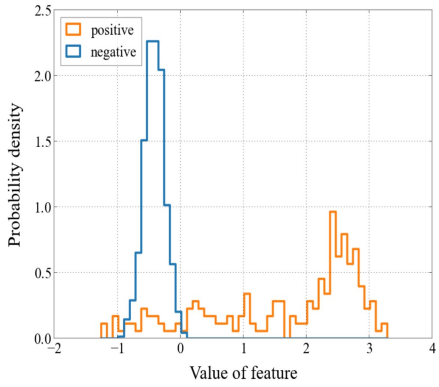
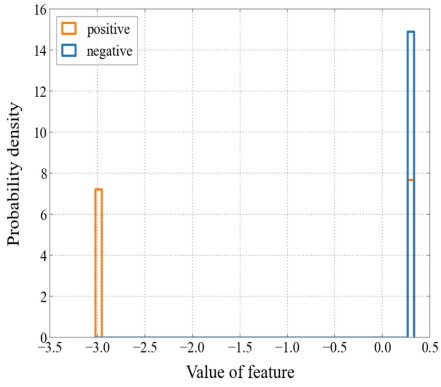


Figure 2: This plot shows the ROC curves calculated for several linear models generated using various α values on the data from Run 2 (2.1.3). The legend lists the α value for each model along with the AUC scores calculated by using the model to predict the labels for the testing dataset.



(a) The spread of complexity values for `H1:OMC-FPGA_DTONE_IN1_DQ`



(b) The spread of permutation entropy values for `H1:OMC-LSC_DITHER_OUT_DQ`

Figure 3: These histograms show the probability densities of the values for the indicated features calculated for the indicated channels. The values themselves are trivial because the data have been pre-processed (**??**), but the clear separations between values for the positive and negative samples make these features interesting.

When we complete the K-means clustering, we are unable to identify any meaningful clusters. The histogram in Figure 4 again shows the complexity values for `H1:OMC-FPGA_DTONE_IN1_DQ` but with the positive samples grouped according to the K-means clustering. The groups are not distinct; they completely overlap. This is true for all of the feature spaces that we examine. Therefore, we are not able to identify any common underlying causes of lock losses.

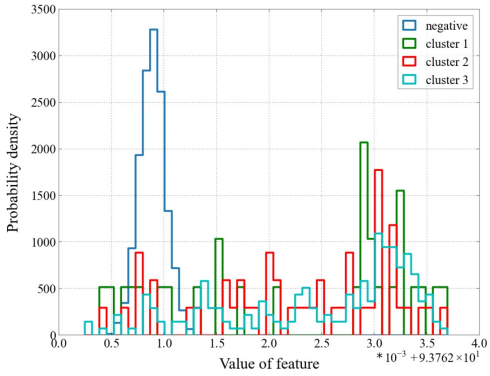
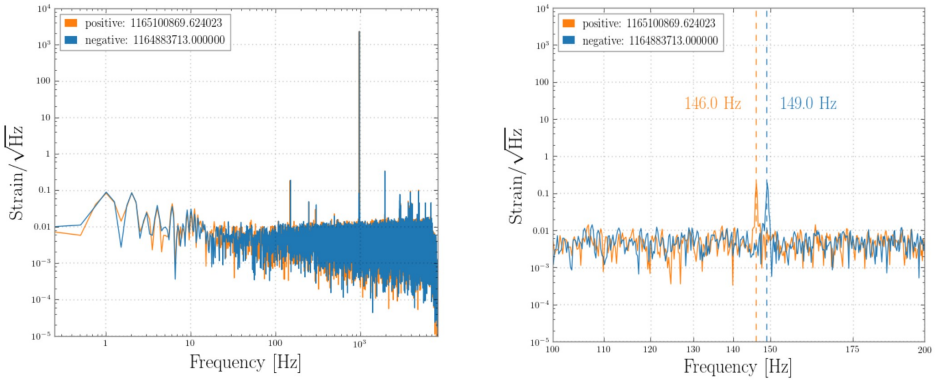


Figure 4: This histogram shows the spread of complexity values as probability densities for `H1:OMC-FPGA_DTONE_IN1_DQ`, as in Figure 3a, except with the positive samples separated into the groups identified by the K-means clustering.

In the absence of meaningful clusters, we decide to complete individual follow-up into the two most interesting features that we have identified.

3.1. `H1:OMC-FPGA_DTONE_IN1_DQ` channel

To get a better understanding of the behavior of the `H1:OMC-FPGA_DTONE_IN1_DQ` channel, we plot the ASDs for this channel for different positive and negative samples together. An example of one of these plots is Figure 5a. All of the samples demonstrate peaks at the expected frequencies of 960 Hz and 961 Hz, which cause the 1 Hz beat frequency that makes this a timing diagnostic channel.[10] However, the channels also demonstrate a peak between 130 Hz and 175 Hz that seemingly tends to be lower for the positive samples, as shown in Figure 5b.



(a) The full range of the ASD with the expected large peaks at 960 Hz and 961 Hz

(b) The ASD from 100 Hz to 400 Hz with the largest peak in this range for each sample identified

Figure 5: These plots show the ASD for `H1:OMC-FPGA_DTONE_IN1_DQ` as calculated for a positive sample time, 1165100869.624023, and a negative sample time, 1164883713.000000.

In order to determine whether this peak is indeed shifted to lower frequencies for a majority of the positive samples, we create a histogram displaying the spread of frequencies at which this peak occurs for both the positive and negative samples, as seen in 6. There is no clear shift between the positive and negative sample sets. We are therefore unable to characterize the difference in the data for this channel between lock loss and clean times.

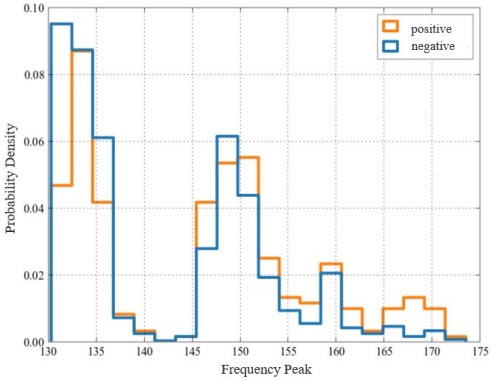
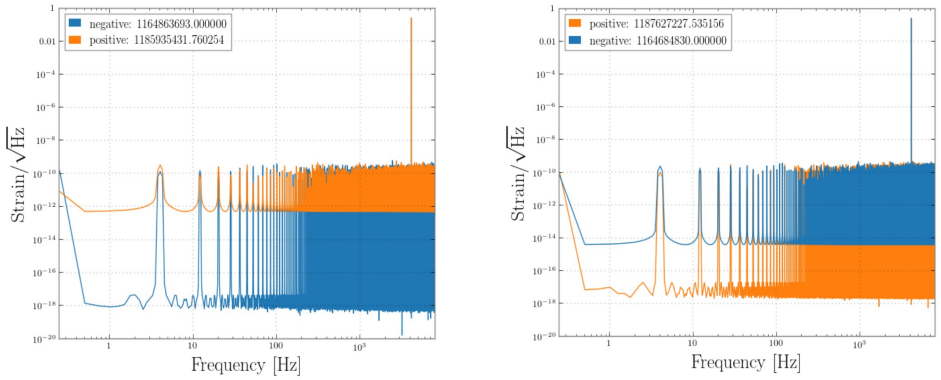


Figure 6: This histogram shows the probability densities of the frequencies for the largest peak occurring between 130 and 175 Hz in the ASD transformations of the data from `H1:OMC-FPGA_DTONE_IN1_DQ`.

3.2. `H1:OMC-LSC_DITHER_OUT_DQ` channel

Based on Figure 3b, we want to characterize the difference between positive samples for which the permutation entropy is high and positive samples for which the permutation entropy is low. We first plot the ASDs for a positive sample and a negative sample with similar, high permutation entropy values, as seen in Figure 7a. The noise in the positive sample exceeds that in the negative sample by approximately 6 orders of magnitude. We then plot the ASDs for a positive sample with low permutation entropy and a different negative sample with high permutation entropy, as seen in Figure 7b. The noise in this positive sample is now almost 6 orders of magnitude lower than that in the previous positive sample, but the noise in this negative sample is approximately 4 orders of magnitude higher than that in the previous negative sample. We are unable to identify any patterns in the noise levels for this channel that could account for the permutation entropy distribution seen in Figure 3b.



(a) The ASD for a positive sample time, 1185935431.760254, and a negative sample time, 1164863693.000000, for which the permutation entropy values are similar.

(b) The ASD for a positive sample time, 1187627227.535156, and a negative sample time, 1164684830.000000, for which the permutation entropy values are different.

Figure 7: These plots show the ASDs for `H1:OMC-LSC_DITHER_OUT_DQ` calculated for two different sets of positive and negative sample times

4. Conclusion

We are able to develop and refine a feature extraction process that is capable of reducing large amounts of auxiliary channel data into a manageable dataset. We are also able to run a Lasso regression on that dataset, but we are met with limited success in identifying features that are predictive of lock loss. We believe that we have not chosen the optimal features to test, and future work on this project would largely be focused on finding features that better characterize the differences between lock loss data and clean data. It is possible that the features chosen here are too simplistic to capture the nature of lock loss data or that the frequency domain data would be more useful than the time series. It would also be useful to further prune down the channel list in order to make the matrix less underdetermined. Once a more appropriate set of interesting features can be identified, there will be a need to refine the clustering algorithm to determine the appropriate number of clusters as well as a more automated way to locate the features that are common among samples in a cluster.

We are still able to follow up on features from 2 channels that show differences between the data they record directly preceding lock losses and the data they record during stable observing times. However, we are unable to determine the underlying causes that relate these features to lock losses.

All code can be found in our git repository:
https://git.ligo.org/jameson.rollins/locklasso.

5. Acknowledgements

I would like to thank Jamie Rollins for his guidance and mentorship throughout this project and for helping me develop my skills as a researcher. I acknowledge support from the LIGO SURF program at Caltech and the National Science Foundation. Computing resources for this project were provided by the LIGO Data Grid.

6. References

[1] Evans, M., et al. (2002). Lock acquisition of a gravitational-wave interferometer. *Optics Letters, 27(8).*

[2] Rollins, J. (2015, June 16). Advanced LIGO Guardian Documentation, Release 1447. https://dcc.ligo.org/public/0120/T1500292/001/AdvancedLIGOGuardian.pdf

[3] Rollins, J. (2017, July 20). Machine learning for lock loss analysis. https://dcc.ligo.org/public/0144/G1701409/001/main.pdf.

[4] Documentation of tsfresh. (2019). Retrieved from https://tsfresh.readthedocs.io/en/latest/

[5] Documentation of numpy. (2019). Retrieved from https://docs.scipy.org/doc/numpy/reference/

[6] Documentation of scipy. (2019). Retrieved from https://docs.scipy.org/doc/scipy/reference/

[7] Zwillinger, D. and Kokoska, S. (2000). CRC Standard Probability and Statistics Tables and Formulae. Chapman Hall: New York. 2000.

[8] Documentation of EntroPy. (2019). Retrieved from https://raphaelvallat.com/entropy/build/html/index.html

[9] Documentation of scikit-learn 0.20.3. (2018). Retrieved from https://scikit-learn.org/stable/documentation.html

[10] Márka, Z. (2017, Jan. 18). DuoTone Timing Witness Signal during O2. https://dcc.ligo.org/LIGO-T1700024