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," but 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.

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 end of each, and recombined such that there is near perfect destructive interference. Passing gravitational waves distort spacetime, creating a phase difference between the two beams and affecting the interference pattern. In order for one of these detectors to be sensitive enough to detect the minute phase shifts caused by gravitational waves, it must be in a stable state known as "lock." 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] The commissioners who operate the detectors must use the automated lock acquisition system, Guardian,[2] to work through several stages of the lock acquisition process before an interferometer is stable and data collection can begin.

The loss of lock leads to a significant amount of interferometer downtime as the commissioners work to realign the different components. 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 state of the interferometer, and we can potentially use machine learning techniques to identify features in these channels that signal the approach of a lock loss.

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 prevent such events in the future.

In order to do this, we first have to identify the features which are most likely to predict lock loss. This is crucial, because selection of features that are not correlated to loss of lock does not yield meaningful results. We then have to 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 have to 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 similar lock loss events. 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 to reach its end goal of better understanding lock losses so that we can reduce lost time during future data collection.

2. Methods

2.1. Work Completed

The three main steps in this project are feature extraction, regression with regularization, and clustering, but we first must compile the dataset. We limit our analysis to 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 (Appendix A). We identify positive sample times at which a lock loss occurred while the interferometer was in its stable observing mode during the second LIGO observing run, O2. We then select negative sample times from O2 in which the detector was stable and observing. The length of our samples is a parameter which can be tuned between different iterations of our analysis, called "runs" (Appendix A).

To begin our first run, we look for features in the data samples that we believe to 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, and the results would not be meaningful. The solution to this problem is to perform calculations on the samples that reduce the amount of data points while preserving the characteristics that 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, with channel sample rates in the kilohertz range, in order to significantly reduce the amount of data, the frequency bins must be large, which may hide interesting features in the data. Instead, we perform various calculations that return single values to extract the features for our runs (Appendix A).

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. Ideally, certain features 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.

We compile the extracted features into a large matrix in which the columns are the features and the rows are the samples, and we generate a vector indicating whether each row contains data from a positive or negative sample (Appendix B). We use scikit-learn,[4] 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 (Appendix B). It builds upon the ordinary least-squares linear regression, but it reduces the number of features used to create the model by driving coefficients of features that do not contribute significantly to zero. The degree to which it does this is controlled by the α parameter, and 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 need to evaluate its success. We can produce receiver operating characteristic (ROC) curves for several rounds of regression with different α values, as shown in Figure 1. On the x-axis, these plots show the false positive rate that is calculated by comparing the model predictions to the testing data, and the true positive rate is on the y-axis.[4] 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 can also visually examine the histograms for the features assigned the highest coefficients by the regression. Figure 2 shows an example of a histogram for a feature that clearly distinguishes between lock loss and clean times: the complexity of the data recorded by the H1:OMC-FPGA_DTONE_IN1_DQ channel.

For the data from Run 2, an α value of 0.0261 is just large enough to yield a model with a perfect ROC AUC score, as seen in Figure 1. This model assigns nonzero weights to 153 features, of which H1:OMC-FPGA_DTONE_IN1_DQ from Figure 2 is one of the most significant contributors.

2.2. Work To Be Done

We now have a set of features that have been identified as indicators of lock losses. We will use one of the clustering algorithms available within scikit-learn to group these data. Previously, the mean shift clustering algorithm has been used, but we can

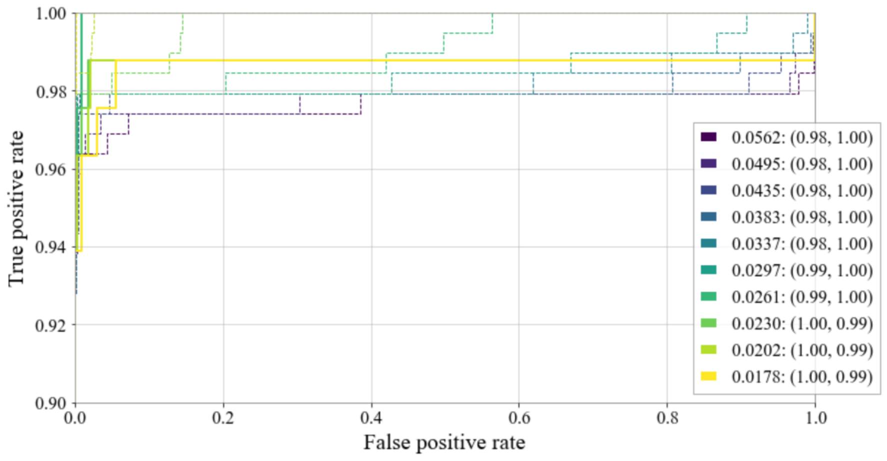

Figure 1: This plot shows the ROC curves calculated for several linear models generated using various α values on the data from Run 2 (Appendix A). The dotted lines are the curves for the training dataset and the solid lines are those for the testing dataset. The legend lists the α value for each model along with the AUC scores for the training and testing set.

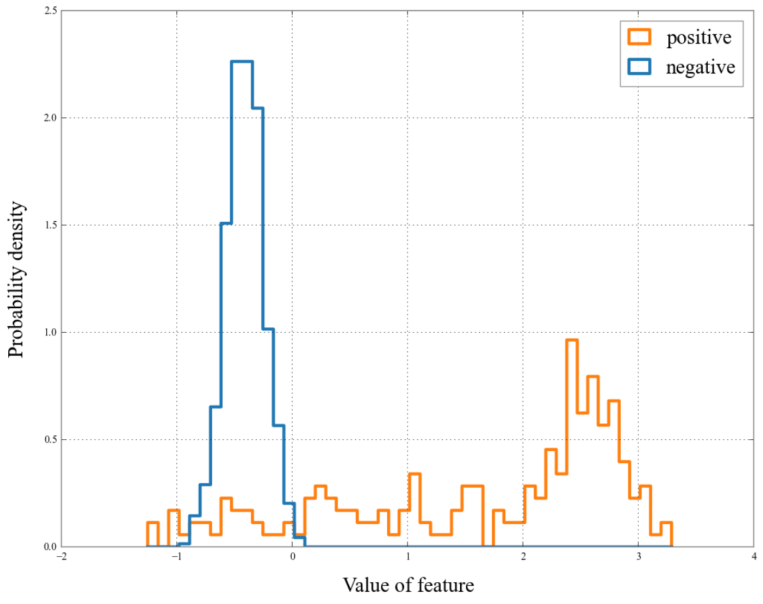

Figure 2: This histogram shows the spread of complexity values calculated for the data recorded by the H1:OMC-FPGA_DTONE_IN1_DQ channel for the positive and negative samples. The values here are trivial because the data have been pre-processed, but the clear separation between the values for the positive and negative samples makes this feature interesting. This feature was identified by doing a lasso regression with α equal to 0.0307 on the features extracted during Run 2 (Appendix A).

test several, as with the regression. We will manually examine the features used to cluster the samples to determine whether they are actually similar or not and to try to identify an underlying cause of each group. A good test of how well our methods have worked is whether or not we recover a group of samples that are earthquake-induced lock losses. We may need to closely study the channels and control loops in the interferometer to understand how they are related and how specific problems arise. We may also need to repeat some or all of the steps in the analysis to tune parameters such as the sample length and the α value.

3. Conclusion

Currently, we have completed several runs of feature extraction on the selected dataset. We have performed the regression with a range of alpha values on each run and identified an interesting set of features. We are now working on clustering them into groups of related lock losses that share underlying causes.

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

4. 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 scikit-learn 0.20.3. (2018). Retrieved from https://scikit-learn.org/stable/documentation.html
[5] Documentation of tsfresh. (2019). Retrieved from https://tsfresh.readthedocs.io/en/latest/
[6] Documentation of EntroPy. (2019). Retrieved from https://raphaelvallat.com/entropy/build/html/index.html

Appendix A. Run Details

The 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. From this list, the calibration channels are removed, as well as 23 channels that started recording in the middle of O2 and 1024 channels that are identified to contain completely flat data during our sample times.

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 subtracting a buffer of 0.25 seconds. The buffer is included to account for the delay between when the interferometer actually loses lock and when Guardian identifies the change in state. We do not want to include the actual lock loss in our analysis because we are working to find features that can predict a lock loss in advance. There are 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 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 observing mode. There are 1086 negative sample times.

Appendix A.1. Run 0

The first run uses a sample length of 4 seconds, meaning that, for each sample time, data are extracted 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. The following extracted features are adapted from the documentation of the tsfresh feature extraction package:[5]

- Complexity
- Absolute energy
- Mean value
- Standard deviation
- Variance
- Kurtosis
- Skewness
- Absolute sum of changes
- Mean absolute change

We also calculate another feature which is the difference between the maximum and minimum values for a given sample. We initially attempt to include sample entropy and approximate entropy in our feature extraction but find them to be too time-consuming.

Appendix A.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. We extract the same features as in Run 0.

Appendix A.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.[6]

Appendix B. Lasso

The main algorithm that we use is Lasso regression. The following description is based on the previous work[3] and the scikit-learn documentation.[4] We use our extracted features as our inputs, and they are compiled into an m-by-n matrix **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 **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 **y** is the label corresponding to the time from which the c-th row in **X** was extracted. The regression then solves

$$\mathbf{X}\mathbf{w} = \mathbf{y} \tag{B.1}$$

for a vector of coefficients **w** that linearly maps **X** onto **y**. Figure B1 visualizes the regression problem.

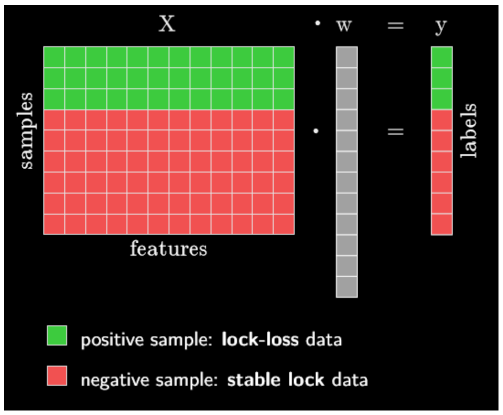
X
• w
=
y
samples
features
labels
positive sample: lock-loss data
negative sample: stable lock data

Figure B1: The Lasso regression attempts to solve for **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 B.1 is underdetermined. This is why we need to implement regularization, which the Lasso algorithm does by solving equation B.1 such that it minimizes

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

where $||\mathbf{w}||_1$ is the 1-norm of **w**

$$||\mathbf{w}||_1 = \sum_i |\mathbf{w}_i| \tag{B.3}$$

and α is the regularization coefficient responsible for penalizing the coefficients of those features that do not contribute significantly to the model. A higher α value drives more coefficients in $\mathbf{w}$ to be zero so that the final linear equation depends on less features.

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