Machine Learning Engineer Nanodegree Capstone Project Report - LANL Earthquake Prediction

Praveer Nidamaluri

May 13th, 2019

I. Definition

Project Overview

Earthquakes can cause extreme destruction and loss-of-life. For example, the 2011 March 11th Tohoku earthquake in Japan resulted in $360 billion and almost 20,000 fatalities [1]. The December 26th, 2004 the Sumatra-Andaman earthquake and resulting tsunamis resulted in $15 billion and caused 227,898 fatalities [2]. Considering the disastrous nature of seismic events, any form of earthquake prediction would significantly reduce the human and economic impact. However, no reliable method has been discovered yet.

The Los Alamos National Laboratory (LANL) is now sponsoring a Kaggle competition [3] to further research the applications of machine learning (ML) algorithms in earthquake prediction. The competition provides a vibration dataset from laboratory simulated earthquakes. This project aims to develop a supervised learning regression model that uses the vibration data and predicts the time remaining until an earthquake occurs (time-to-failure). Ideally, in developing the model, insights about useful features from the laboratory dataset will be identified so they can be explored with real earthquakes in future research. The final model will be submitted as an entry to the Kaggle competition.

Problem Statement

The earthquake prediction competition problem will be tackled as follows:

1. Download the dataset from the Kaggle competition website
2. Develop an efficient workflow so that the 9+gb dataset can be worked with within local machine computational constraints (RAM/CPU).
3. Preprocess the raw data into sets of input vibration segments (150,000 samples; the input to the model as defined by the competition) and a single output time until failure value (the regression target variable)
4. Develop a benchmark model.
5. Develop suitable regression model by iteratively performing and refining the following steps:
    a. Develop suitable features from the input vibration segments.
    b. Train, optimize, and compare suitable supervised machine learning regression algorithms
6. Submit the final model to the Kaggle competition

The final solution to the problem will be the optimized regression model, as well as the insights on the features within the input vibrations that are most useful in predicting the laboratory earthquake.

Metrics

The performance of all developed models will be measured by the mean absolute error (MAE) between the actual and the predicted time-to-failure values in the test/validation set. For a set with N datapoints, the performance metric is defined as:

$$MAE(T_{pred}) = \frac{1}{N}\sum_{i=1}^{N} | T_{pred} - T_{actual} |$$

The metric is specified by the Kaggle competition. It is also intuitive, easily quantifiable, and readily available in the Scikit-learn python package [4].

II. Analysis

Data Exploration and Exploratory Visualization

The dataset for the competition is generated from the model laboratory earthquake setup depicted in Figure 1 [3]. In reality, earthquakes occur when two tectonic plates move against each other. The rough interfaces get stuck against each other and start storing energy like a spring. Eventually the friction between the plates becomes unsustainable and the stored energy is released, causing vibrations to spread through the plates. The laboratory set-up models this behavior by forcing a plate to move past two adjacent frictional interfaces. Similar to real earthquakes, the plate will periodically get stuck and then release energy, resulting in a 'labquake'. This can be detected by a sensor that tracks the frictional force in the contact interface. A separate piezoelectric sensor is attached to one of the plates and tracks the vibrations at a sample rate of 4MHz.

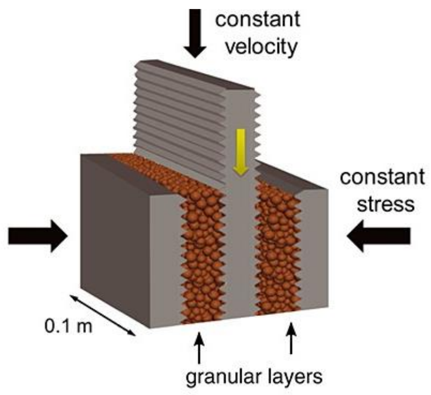

Figure 1: Laboratory earthquake test set-up [3]

The competition dataset provided on the Kaggle website [3] consists of a 9+Gb csv file with two fields: 'acoustic_data' and 'time_to_failure'. The data represents approximately 158s of a continuous lab earthquake experiment. Every 1000th row of the dataset is sampled and plotted in Figure 2: Training dataset sampled at every 1000th row in order to visualize the entire dataset. The key global statistics of the data are subsequently presented in Table 1. Per the table, there are a total of 629.1 million rows in the dataset.

	acoustic_data	time_to_failure
count	6.291455e+08	6.291455e+08
mean	4.519468e+00	5.678292e+00
min	-5.515000e+03	9.550396e-05
max	5.444000e+03	1.610740e+01
std	1.073571e+01	3.672697e+00

Table 1: Global statistics of dataset

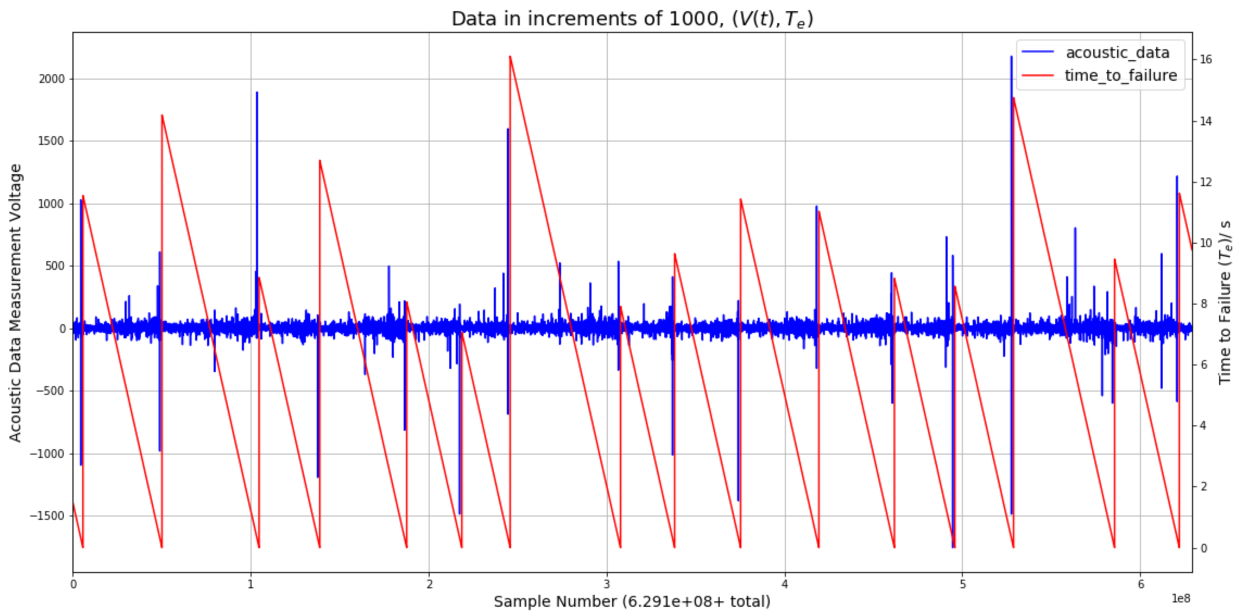

Figure 2: Training dataset sampled at every 1000th row

The red plot in Figure 2: Training dataset sampled at every 1000th row, representing the 'time_to_failure' values, can be seen to hit 0 before spiking multiple times. These represent 16 separate labquakes that occur within the experiment data. The 'time_to_failure' values are the regression targets for the model. The blue plot represents the 'acoustic_data'. Per the competition details, each input 'datapoint' for the solution regression model is a 0.0375s segment of the 'acoustic_data' (150,000 rows). An example of a datapoint is presented in Figure 3.

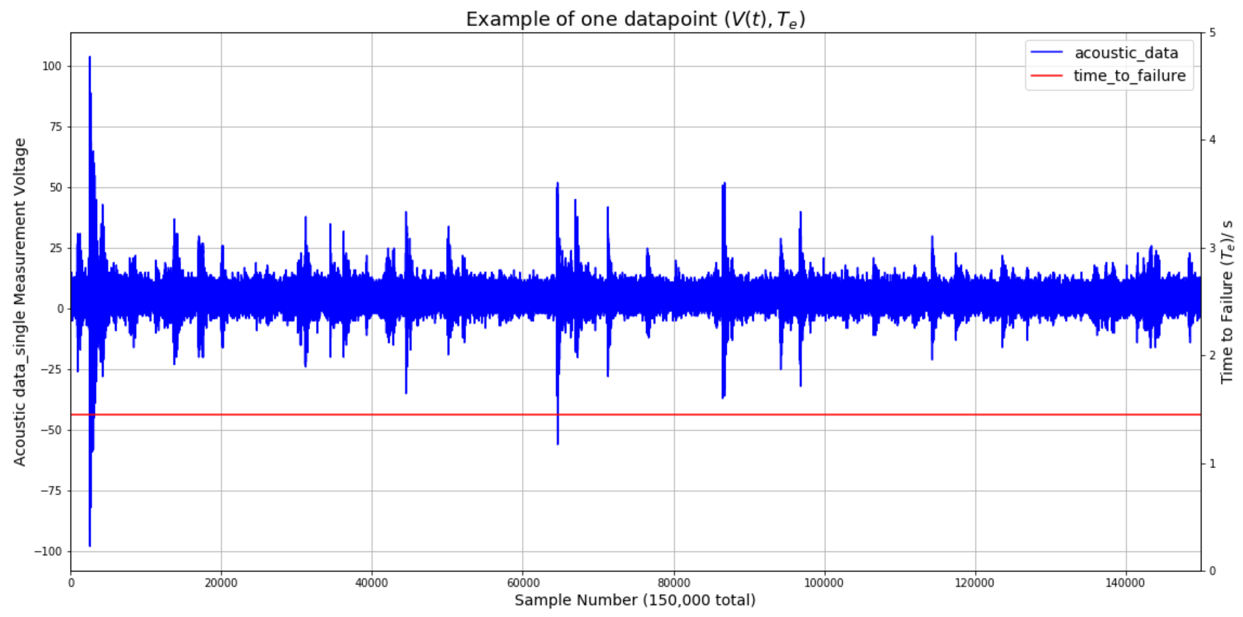

Figure 3: Example of 1 'datapoint'

Exploring the dataset, the following preprocessing observations are made:

1) The input dataset is a stream of continuous raw data. It will have to be converted into a set of input/output datapoints such as the example in Figure 3. Each input is a 0.0375s (150,000 rows) segment of the 'acoustic_data'. The target is a single 'time_to_failure' value from the last row in the 0.0375s segment.

2) The raw data contains 16 labquake spikes. These regions must be excluded when preprocessing the input/output datapoints

3) The input datapoint consists of 150,000 'acoustic_data' values. This is obviously an extremely high number of dimensions. Training a model with the raw values would be computationally expensive, and probably contains excessive amounts of noise. Feature engineering will therefore be a critical component of the path to the optimal solution. The input will have to be processed into a set of features that capture the critical aspects of the 0.0375s time series.

Benchmark

A benchmark model will be made by implementing the machine learning algorithm suggested in previous research [5] from the sponsors of the Kaggle competition. The prior research describes a Random forest regression model trained on simplistic statistic features such as the mean and standard deviation of the input time-series. The output of the model is the estimated time_to_failure. This model represents the current state of the art. It can be recreated for the current competition dataset, and used as a benchmark with which to compare future models. The comparison will be quantified with the mean absolute error calculated on cross-validation or competition test sets.

Using the most basic set of features, an optimized Random Forest Regressor benchmark results in a mean absolute error (MAE) of 2.28 on a test set made from 35% of the input data. The regressor was optimized with the GridSearchCV class in sklearn [4]. The final hyperparameters used for the benchmark were: max_depth = 5; min_samples_leaf = 0.05; n_estimators = 30. The error criterion is equal to the MAE.

The benchmark prediction can be visualized against the full set of available data in Figure 4. The implementation is detailed in the 'Benchmark Model_fset1.ipynb' notebook. The comparison shows that the benchmark is able to roughly predict the trend of the time_to_failure values, but fails at predicting the extreme values.

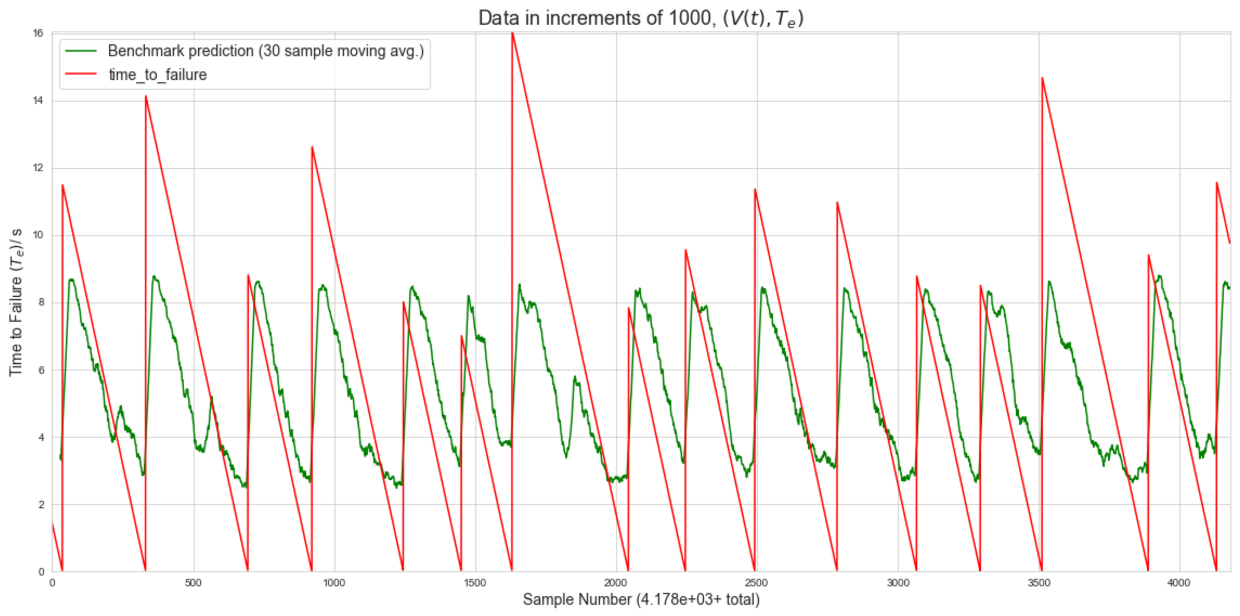

Figure 4: Actual time_to_failure vs Benchmark prediction

Algorithms and Techniques

The problem will be tackled in two stages: exploring the optimal features, and then choosing the optimal supervised learning regression algorithm. The following strategy will be employed:

1) Feature engineering – the following feature engineering strategies will be implemented and compared with the benchmark random forest regression algorithm.

   a. Use the statistical features suggested by past domain research [5] including: the mean, standard deviation, maximum, minimum, skew, kurtosis, and first and third quartiles.

   b. Calculate the critical frequencies and amplitudes from a Fourier transformation of the input time series. This is a common signal processing method used in earthquake engineering.

   c. Explore additional features such as the derivative/integral of the acoustic data. Combine all generated features at once.

2) Algorithm choice – the following algorithms will be implemented and compared using the features with the most importance.

   a. Linear regression
   Basic linear regression is the simplest machine learning algorithm, and should be the first method to attempt unless a more complicated nonlinear relationship is immediately apparent. The algorithm assumes that the 'time_to_failure' value (T_e) can be approximated by a hyperplane in the feature space:

$$f(x) = \mathrm{w} \cdot \mathrm{x} + b$$

   where x is the feature vector, n is the total number of features, w is a weight vector, and b is similar to the y-intercept in the equation of a line. The model is fit to the data using gradient descent (or another iterative complex minimization algorithm). The cost function that is minimized can be defined by the mean squared error or the mean absolute error. Additional terms can be added to the cost function for regularization. In Ridge regression, L2 regularization is added; i.e. the sum of the squares of each w component is added to the cost function. In Lasso regression, L1 regularization is added; i.e. the sum of the absolute values of each w component is added to the cost function. Regularization serves to reduce model complexity that may result in overfitting.

   The main benefit of linear regression is that the simplicity of the formulation makes the final result intuitive. It becomes easy to extrapolate model findings to actual physical relationships in the model domain. In addition, the linear model coefficients indicate the importance of the various features (provided the features are on the same scale). The simplicity of the model also guarantees inexpensive model fitting for any size of dataset.

   However, the simplicity of linear regression also limits its scope of application. If the actual pattern being modeled is not linear or locally linear, linear models will result in significant MAE values.

   Considering the significant advantages of the algorithm, the linearity of the labquake prediction problem will first be gauged by fitting a linear regression model.

   b. Support Vector Regression with a radial basis function kernels [7]
   If linear regression fails, the support vector regression (SVR) algorithm gives access to flexible nonlinear representations with the 'rbf' kernel.

   SVR is a regression method that assumes a regression model of the form:

$$f(x) = \langle w, x \rangle + b$$

   where x is the feature vector of the problem, and w and b are a set of weights to be determined. A unique feature of SVR is that the method attempts to find the model, f, such that all data points are estimated within a tolerance, ε. The weights are determined by formulating a convex optimization problem with the following cost function applied to the training data:

$$minimize \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

The cost function is subject to the following constraints:

$$subject\ to \begin{cases} y_i - f(x) \le \epsilon + \xi_i \\ f(x) - y_i \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

x and y in the above equations are the feature vector and target regression variable in the training data. ε is the size of the tolerance for the error (y – f(x), or f(x) – y). ξ and ξ* are slack variables. It may be impossible to find a function that can approximate all values of y within a margin of ε. Therefore, to make the minimization problem feasible, the slack variables are added to the constraints and the cost function. All datapoints outside the ε-margin around the function are penalized. The penalty is proportional to the C parameter.

The described SVR formulation presents a linear model. However, SVR can be applied to nonlinear functions by making the weights, w, a function of x, and using a nonlinear dot product (the kernel trick). In the nonlinear formulation the final function is described as:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\, k(x_i, x) + b$$

where, the α values are lagrange multipliers that are intermediate results from the optimization problem. In the case of radial basis functions, the kernel is defined by the gaussian function:

$$k(x_i, x) = \exp(-\gamma\, \|x_i - x\|^2)$$

γ is another hyperparameter that determines the width of the gaussian function, and is useful to prevent overfitting.

The main advantage of SVR is that the kernel trick allows very nonlinear and high-dimensional relationships to be well approximated. This makes it particularly applicable for the labquake prediction problem, where there are potentially many features.

However, the flexibility in modeling highly nonlinear relationships can also be a disadvantage as it becomes easy to overfit, especially with the rbf kernel. Therefore, the SVR algorithm must be carefully applied with learning curves and adequate tuning of the C and γ hyperparameters. Another disadvantage is that the algorithm results are difficult to interpret as feature importance details are unavailable. Therefore, insight on key features may have to be gained by selectively using features and comparing the final performance.

c. Gradient boosting ensembles (Catboost)

The gradient tree boosting method works in an iterative process. In the first iteration, a decision tree regressor is fit to the data. In the next iteration, a new tree is formed such that the total error from the addition of the old tree and new tree is minimized. Subsequent iterations continue this process; a new tree is made such that the sum of the previous ensemble with the new tree minimizes the total regression error. The hyperparameters in this process include the total number of iterations (equal to the number of trees), the learning rate, and the decision tree hyperparameters such as the maximum depth.

Gradient tree boosting is advantageous as it allows the modeling of highly nonlinear relationships, while also giving access to feature importance information. The method therefore overcomes the limitations of linear regression (linear problems only) and SVR (no feature importance details). The algorithm is also stable against outliers. This would be useful given the high degree of noise in the data (Figure 2).

The main drawback of gradient tree boosting in this context is the propensity for overfitting. However, this can be addressed by tuning the number of iterations and the maximum depth of the weak learner decision trees.

Since past research in [5] found random forests to be amenable to the earthquake prediction problem, gradient tree boosting should theoretically provide even better performance. The algorithm implementation from Yandex (known as Catboost, [6]) is a popular implementation of gradient tree boosting that has seen widespread recent use in Kaggle competitions. The implementation also has an easily accessible API that is similar to sklearn classes. Catboost will therefore be applied to this problem.

III. Methodology

Data Preprocessing

The preprocessing stage is performed in the 'Data Exploration for Project Proposal.ipynb' and 'Benchmark Model_fset{x}.ipynb' (x = 1, 3, or 4) notebooks with the 'gen_features.py' module. The following steps are performed:

1) The input csv file is 9+ Gb in size. To be able to read in and manipulate the data, the file is split into smaller 112mb csv 'chunks'. This is shown in the 'Data Exploration for Project Proposal' notebook.

2) In order to read and manipulate data from each training chunk without encountering memory errors, the multiprocessing module was used. A separate Python process is called every time a training chunk csv file is read. After the operation is complete, the process is ended and all memory in use is recovered. This workflow is used for all operations, such as generating new sets of features, or reading in data for visualizations.

3) Features are generated with the 'gen_dim_reduced_dataset' and 'gen_features' functions in the 'gen_features.py' module. When features are generated, the range in the 'time_to_failure' values are also tracked. Anytime a spike is detected, the particular set of data is discarded. Ie. This ensures the 16 labquakes are avoided.

4) An additional hyperparameter for the overall project is the 'overlap_ratio' keyword added to the 'gen_dim_reduced_dataset' function. The function is used to iterate through the raw 'acoustic_data' and generate features. The default method is to generate features from 150,000 rows of the data (the size of each 0.0375s segment), and then move on to the next 150,000 rows. This gives approximately 4000 total datapoints for training. However, the 'overlap_ratio' setting allows the generation of more datapoints from the raw_data by allowing overlaps in successive sets of the 150,000 rows. This allows more datapoints for training, but may also result in overfitting to the existing data.

After generating the features, new csv files with the feature and time_to_failure data are stored in the 'Features' folder of the solution.

Implementation and Refinement

In implementing the various features and algorithms, the following steps were performed and documented in jupyter notebooks:

1) The features were read in from the generated csv files from the preprocessing stage above.
2) A StandardScaler from the sklearn package [4] was applied to the data if needed by the algorithm.
3) The data was split into training and testing data in a 65:35 ratio with the sklearn train_test_split method.
4) KFold cross validation is used with the training data with the MAE metric to initially gauge the applicability of the model.
5) The algorithm is further refined with the GridSearchCV and validation_curve methods form sklearn. These helped identify the optimal hyperparameters for each algorithm.
6) Learning_curves were plotted as needed using the sklearn implementation. These were used to track the training and cross-validation error, and used to minimize overfitting.
7) Finally, the optimal model after the above tests was applied to the 35% test data.

The various algorithm and feature implementations were compared with each other using the cross-validation and test set scores. The refinement of the solution is achieved by exploring different features and algorithms. The solution is further refined with the GridSearchCV, validation, and learning curve model evaluation and optimization process.

The benchmark model was first trained with three different sets of features in the 'benchmark Model_fset1.ipynb', 'benchmark Model_fset3.ipynb', and 'benchmark Model_fset4.ipynb' notebooks. The number of training points were also increased by allowing overlaps in the 0.0375s segments used to generate the features (described in the Data Preprocessing section). After following the above steps, the feature set with the derivative/integral data of the acoustic_data lead to the best cross-validation and test scores with the benchmark random forest regressor.

The default hyperparameter settings (100 estimators) gave a cross-validation MAE score of 2.172 with a training score of 0.84, a clear case of overfitting. The model was optimized with a grid search across values of 40 and 80 for n_estimators, and 4 and 6 for max_depth. The optimal model was 40 estimators with a maximum depth of 6. The best cross-validation MAE score increased to 2.146, with a training score of 2.05. The model is now more generalized. The best test score was 2.139.

The most important features were analyzed with the feature_importances_ attribute of the random forest regressor class from sklearn. They were found to be the standard deviation of the numerical derivative of the acoustic data, and the kurtosis of the acoustic data.

After experimenting with feature engineering and the random forest regressor, other ML algorithms were explored. Linear regression was first implemented in the 'Linear Regression_fset3.ipynb' notebook. The sklearn Ridge (L2 regularization) and Lasso (L1 regularization) classes were used. The CV scores and tests scores were significantly lower than the benchmark model. The best cross-validation MAE score was 2.660. Linear models are clearly insufficient for the earthquake prediction problem, and additional parameter tuning was not attempted.

The Support Vector Regression method was implemented with the sklearn SVR class in the 'SVR_fset4.ipynb' notebook. The 'rbf' kernel was used to capture the nonlinearities in the data. The default SVR with the 'rbf' kernel produced a CV score of 2.238. The C and gamma hyperparameters were tuned with a grid search over C values of 1, 4, 7, and 10; and gamma values of 0.1, 0.2, and 0.5. The optimal parameters are a C value of 4, and gamma value of 0.2. The CV score improved to 2.175, and a final test score of 2.152 was attained. Therefore, the model performance was slightly less successful than the random forest regression models.

Finally, the Catboost algorithm [6] was implemented in the 'Catboost_fset5.ipynb' notebook. All generated features were used for this implementation. The raw model with the loss function set to MAE attained a CV score of 2.147. The final optimized model attained a CV score of 2.040, and a test score of 2.029. The most significant hyperparameter is the number of iterations. This is equivalent to the number of weak learner decision trees in the ensemble. A number of iterations of 15000 lead to the best cross-validation score without excessive overfitting. The Catboost model was clearly better than the benchmark model as well as the other attempted algorithms.

IV. Results

Model Evaluation and Validation

The final model is a Catboost Regressor trained on all of the developed features. This model lead to the highest MAE score in the test set (2.029). The robustness of the model is proven by a 5-fold cross-validation analysis. The final mean MAE score was 2.040, with a standard deviation of 0.045. The fact that the standard deviation was under 3% of the mean, shows that despite perturbations in the input data, the chosen model produced similar results.

Justification

The 'time_to_failure' prediction results from the final model (blue), the benchmark (green), and the actual values (red) are plotted and compared in Figure 5. The benchmark was earlier stated to capture the general trend of the 'time_to_failure' values, but not able to capture the extremes. The plots in Figure 5 show that the final model is consistently between the green benchmark and red target values for 'time_to_failure'. The final model is therefore definitely better at predicting the time until the next labquake than the benchmark. The predictions capture the general trend, but are also closer to the extremes. Nevertheless, the final solution is still not able to capture most of the extremes from the 'time_to_failure' values. For example, the blue lines never go below 1.5s in the 'time_to_failure' plots.

The model was also submitted to the Kaggle competition. The achieved MAE score on the unknown earthquake data is 1.630, which improves on the 1.734 achieved by the benchmark. For comparison, the current top score on the leaderboard is 1.282.

Overall, the solution is not considered significant enough to have 'solved' the problem. Instead, the presented model is seen as a reasonable first attempt at the Kaggle competition. Further feature exploration and model tuning will be required to improve model performance.

V. Conclusion

Free-Form Visualization

Figure 5 presents a comparison of the benchmark, final model, and actual data values for the time remaining until the next labquake in the provided data. As reported in the Results subsection, the final solution performs better than the benchmark, but is still unable to adequately capture the labquake phenomena.

A further aim of the project was to identify the key feature importances that could be used to predict earthquakes. These are extracted from the Catboost model feature_importances_ attribute, and plotted in a bar chart in Figure 6. The three most important features of the acoustic data were found to be the kurtosis, standard deviation of the first derivative, and the mean.

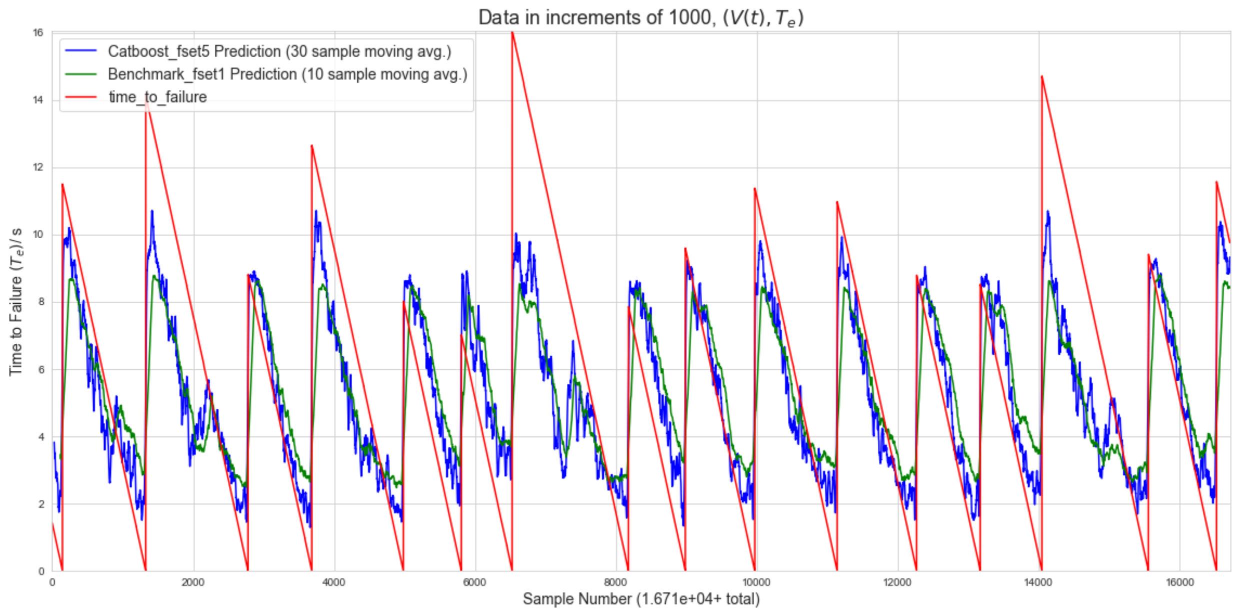

Figure 5: Comparison of Final Model, Benchmark, and actual time_to_failure values

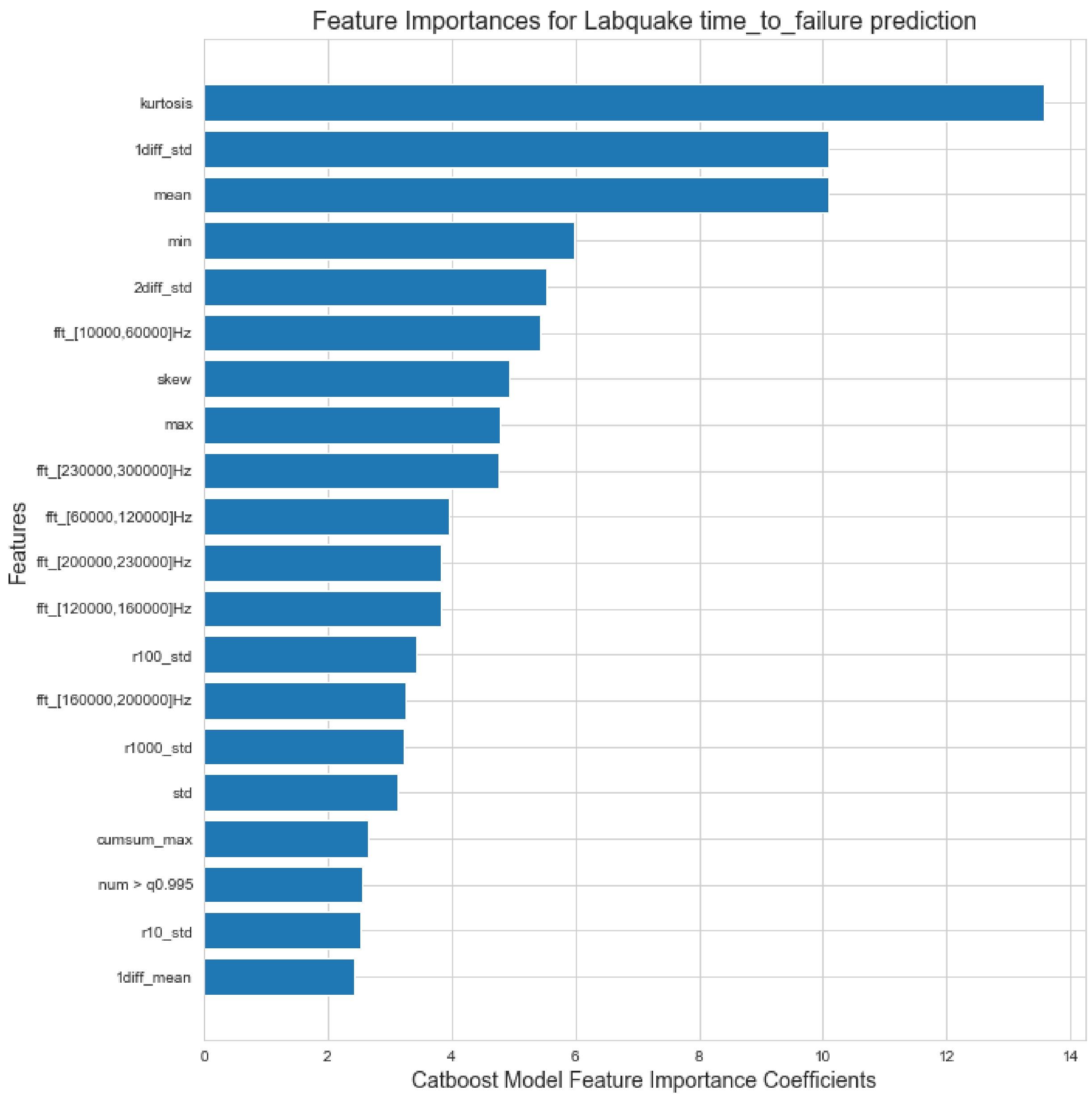

Figure 6: Final Model Feature Importances

Reflection

The process used for the project is detailed in the Problem Statement section and is repeated below:

1. Develop an efficient workflow so that the 9+gb dataset can be worked with within local machine computational constraints (RAM/CPU).

2. Preprocess the raw data into sets of input vibration segments (150,000 samples; the input to the model as defined by the competition) and a single output time until failure value (the regression target variable)
3. Develop a benchmark model.
4. Develop suitable features from the input vibration segments.
5. Train, optimize, and compare suitable supervised machine learning regression algorithms

Steps 1 and 4 were the most difficult and interesting aspects of the project. This was the first project I worked on with this large a dataset. I had to learn how to work with the cpu and memory constraints. Working with the data ultimately became a lot easier after setting up a workflow with the Python multiprocessing library.

Step 4 was difficult because there was no way to know what type of features may be useful. A lot of time was therefore spent trying different types of feature engineering schemes to see if they are useful. I also learnt that no matter how powerful the machine learning algorithms are, without appropriate features, the final predictive potential is significantly affected.

Ultimately, the final model did not meet my expectations for the problem since it was not a sufficient improvement from the benchmark. However, it is an acceptable first attempt at the competition. Moreover, although the model is not applicable for a general setting, the critical features identified (Figure 6) can definitely be further explored for earthquake prediction.

Improvement

The main area of improvement for the project is probably feature engineering. The fact that multiple types of models (random forest regressor, linear models, SVR, Catboost) all resulted in similar final performance (Figure 5), suggests that significant improvements are more likely to occur from improving the training features, than the ML algorithm.

For feature engineering, additional methods to explore include wavelet decomposition, and possibly some form of principal component analysis. Both of these methods have been successfully used on time-series before. In addition, the final model in the project implementation used all the explored features. However, to make the solution more efficient, features with low importance could be trimmed. This would reduce the 'noise' from unimportant features affecting the model and causing overfitting. For example, the sklearn feature engineering package could be used.

Finally, a neural network based approach could also be explored to improve the problem. Deep learning methods are known to capture nonlinearities better than other ML algorithms.

VI. References

[1] "Earthquake, Tsunami, Meltdown – The Triple Disaster's Impact on Japan, Impact on the World", Ferris, Solis, March 11, 2013, https://tinyurl.com/m2mwbbb/.

[2] *"Indian Ocean Tsunami – Economic Aspects"*. indianoceantsunami.web.unc.edu.

[3] https://www.kaggle.com/c/LANL-Earthquake-Prediction#evaluation

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

[5] "Machine Learning Predicts Laboratory Earthquakes", Rouet-Leduc, Hulbert et al, 30 August 2017, https://doi.org/10.1002/2017GL074677.

[6] Anna Veronika Dorogush, Vasily Ershov, Andrey Gulin, "Catboost: gradient boosting with categorical features support", https://catboost.ai/.

[7] "A Tutorial on Support Vector Regression", Alex J. Smola, Bernhard Schölkopf - Statistics and Computing archive Volume 14 Issue 3, August 2004, p. 199-222.