COSI-149B Term Project "LANL Earthquake Prediction"
By: MyongJoon Chang, Mark McAvoy, Eurey Noguchi

Introduction

For the term project, Kaggle's LANL Earthquake Prediction competition, the goal was to forecast when an earthquake would be happening using previous acoustic data. The training data is a single stream of continuous experimental data within each segment. Along with this acoustic data we have received a time to failure variable, which represented how long it will take until the next laboratory earthquake.

Data Input Feature Extraction

However, we felt that only having a single acoustic data was hindering in our ability to make predictions, therefore, using the original single acoustic data, we extrapolated a bigger range of data points. First, we largely split our data into the original data along with denoised data.

The real challenge with denoising data is realizing that the original acoustic data from the artificially inserted data. The main reason that acoustic data would contain noise is in the method of collecting our data. In seismology, to collect acoustic data, artificially generated signals called impulse signals are created to interact with the Earth's actual seismic signal which is then collected. Now, after we have collected this data, cleaning up the data is where the important part is at. This helps us localize the acoustic data that would have a higher correlation with the actual prediction that we are trying to make.

Another big change to the original data that was include was using FFT. After research we found that FFT or fast Fourier Transformation was a very common practice method on acoustic information. Using FFT we could convert the original acoustic signal into components of different frequencies. The largest benefit of using FFT was not that it actually predicted anomalies in our data but rather helped us predict what the "natural" periodic signals in our data should be. Since FFT helps us extrapolate a steady component frequency, finding a 'blip' or anomaly and preferably similar frequency activities before such a 'blip' occurs this would help us find a similar pattern in earthquake prediction.

After we have generated two sets of data, the original along with the cleaned, we next applied feature extraction. Feature extraction was essentially taking the original acoustic data along with the denoised data and extrapolating related or highlighted data. Using multiple kernels which had extrapolated data and collecting all of the data input into one input was our main goal. Some of

the extrapolated data were, STA (Short Term Average), LTA (Long Term Average), k-Nearest Neighbors, Mean, Standard Deviation, Max, Min, Geometric and Harmonic means etc.

We felt that this additional amount of data will allow our program to have more information therefore make a more accurate prediction for time to failure. One of the most interesting public information that we found was using complex analysis. Splitting the statistics on real and imaginary parts allows us to do more manipulation than before also allowing for us to complex and real analysis separately for much more manipulation and feature extraction.

The main benefit in using complex analysis was on an application called Complex seismic trace. Using complex analysis helps us isolate and create a unique set of data points most importantly a data point called instantaneous amplitude, a commonly used information point in acoustic analysis.

In conclusion regarding input data was that data manipulation was the bulk of the work behind acoustic data prediction. After testing different combinations of input data we could tell that depending on the combination of input data used our accuracy would fluctuate. This helped us realize that not only was feature extraction important however it is also quite important to know which feature are useful and which features would only be adding noise.

Failed Attempts

Our first approach to this competition was to observe the data and think about our possible solutions. In our first jupyter notebook “check_data.ipynb”, we plotted graphs using the acoustic_data and time_to_failure variables. We noticed how there were smaller peaks before the actual earthquake and how the acoustic_data for the actual earthquake was not recorded since it was too large. This gave us some insights into how we could generate some features such as comparing the time it takes for the next earthquake for the training of the model. Using some of our previous code from project 2, we further trained a RNN with LSTM as a baseline, which in the end, we decided to not use since it was not performing very well.

Once we built an intuition on how the data was represented for this project, our next step was to generate new features from the training data. As mentioned, these included simple features such as mean, standard deviation, minimum, and maximum values. We further incorporated features such as the quantile, trend, and rolling features reading and learning from public kernels that they had been effective.

With these new added features, we decided to train some models. Realizing the RNN was not the best solution to approach this problem, we started to look into various gradient boosting

machines and regression algorithms. The ones we attempted to use included LightGBM, XGBoost, CatBoost, KernelRidge, and Support Vector Regression. For each of these models, we referenced some of the other public kernels for some parameters that we can use. Using these parameters, we found out that most of the models have a MAE in the similar range. Unsure of how to proceed, from research, we found some common techniques on Kaggle competitions for stacking and blending predictions. Submitting both of our results, it seemed like the results from using blending performed better. This technique ranked us 1257, the top 37% of the leaderboard.

Wanting to improve our score, our intuition was to add more features. In addition to the previously added features, we generated few more features including, but not limited to, values such as “peak-to-peak” and “moments” from the scipy package. Running “feature-generation.ipynb”, this generated our new set of features. Furthermore, we decided to tune some of the parameters that were used for the training of our models to lower the MAE. With the newly generated features and tunes parameters, we ran “earthquake-prediction.ipynb”. Meeting our expectations, this indeed did improve our score ranking us 1015 with a score of 1.481. However, not knowing how to improve, we shifted towards our current approach.

Understanding our Main Prediction Model, Reproduction of Prediction

After all of the feature extraction has happened, our main machine learning model uses a LightGBM to make the predictions. Simply put a LightGBM is a gradient boosting framework that uses a tree based learning algorithm. The main benefit of the LightGBM was that the tree grows horizontally over conventional machine learning trees. Growth of a LightGBM happens when it drops the leaf with the max-delta loss to grow. A couple of warning that we found while researching LightGBM was that this model was quite prone to overfitting. Thus it would only make sense to use the feature extraction previously mentioned to help limit the chance of overfitting.

To reproduce our result we first need to do the feature extraction that had been explained in our previous parts of the this paper. In our zip folder we have included the Jupyter Notebook which does the extrapolation, feature_extraction.ipynb.
Note: Everything was ran on Kaggle Kernel. Therefore, to run locally, you must change the input directory structure. (Change from “../input/…” to “input/…”)

To get the final predictions that can be used in our submission run Final_project_main.ipynb using the extracted data points as entry.