Analysis Unidoc

Matthew Segovia masegovia@ucsd.edu
Jun Hwang juh016@ucsd.edu
Ryan Doh rdoh@ucsd.edu
Haotian Zhu haz084@ucsd.edu
Ammie Xie axie@ucsd.edu
Ketki Chakradeo kchakradeo@ucsd.edu
Marco Sanchez mas023@ucsd.edu
Aobo Li aol002@ucsd.edu

Abstract

This project focuses on the development of machine learning models to analyze time series data from the Majorana Demonstrator experiment, which aims to detect neutrinoless double beta decay and identify Majorana neutrinos. The study explores the extraction and utilization of 13 key parameters from raw waveform data to improve classification accuracy. The project aims to design machine learning models that efficiently discriminate between signal and background events, ultimately contributing to the broader understanding of neutrino physics.

Github Repository: https://github.com/matthewsegovia/MajoranaNeutrinoHunt

1 Introduction

The Standard Model of particle physics indicates that fermions possess reciprocal antiparticles of opposite charge; however, in the case of the neutrino, which has no charge, further investigation is warranted. It could be that the neutrino is a Majorana particle, meaning it is its own antiparticle, which could answer puzzling questions such as the matter-antimatter asymmetry in our universe and the neutrino's mass difference when compared to other fermions. A signature of such Majorana neutrinos would come from the rare neutrinoless double beta decay, where the antineutrinos in the two emitted electron-antineutrino pairs annihilate each other and leave the electrons to bear a greater energy. The Majorana Demonstrator experiment aims to spot this specific energy signal utilizing high purity Germanium detectors, but with so many extra particles setting off the detectors (i.e. via background radioactivity or scattered photons), there is a pressing need for machine learning to make data cuts.

The time series data produced by a particle entering the detectors can be analyzed and classified by means of machine learning in order to more efficiently reject these background signals, but different machine learning models can vary with accuracy to the ground truth, which this project aims to help with. During the first quarter of this project, we begin by developing parameters which can be extracted from the time series data, or waveforms. With this, the goal is to propose different models that meet the requirements outlined in the NPML (Neutrino Physics and Machine Learning) instructions included in the Majorana Demonstrator data release notes (1), which stipulate the following: one ML model that will accurately predict datapoint analysis labels given the raw waveform, and preparatory retention of signal-only events for another ML model designed to reconstruct the energy of these events. Completion of this challenge will lead to model comparisons to find the best performance.

2 Parameter Pipeline

A total of 13 parameters were extracted from the waveform data to be trained in our model. These parameters could provide some insight on the various interactions between the variables. These parameters aim to help build a stronger classifier. Additionally, the noise that is present in the measurement of the raw waveforms are not included in this analysis.

2.1 Drift Time

In a germanium detector, ionizing radiation generates electron-hole pairs within the detector material. The movement of these charge carriers across the detector toward the electrodes generates a detectable signal. The drift time (tdrift) is defined as the time taken from the initiation of charge generation to the collection at the detector's point contact. This drift time correlates with the location of the event inside the detector, as interactions that occur further from the point contact result in longer drift times due to the greater distance the charge must travel.

Drift time is extracted from tp0 (given in the dataset) to 99.9% of the peak value. It is typically obtained by interpolating the rising edge of the waveform, allowing for a more accurate measurement, as the resolution of the waveform is often limited by the number of data points in a given time window.

Three different drift times were extracted from the raw waveform data, each measuring at specific points of the increase. As mentioned above, tp0 was extracted at 99.9% of the peak value, additionally tdrift50 and tdrift10 were also extracted at the 50% and 10% points respectively. Each tdrift threshold counts as an individual parameter. It is assumed that the start of the drift times for all waveforms begin at the 1000 time index value.

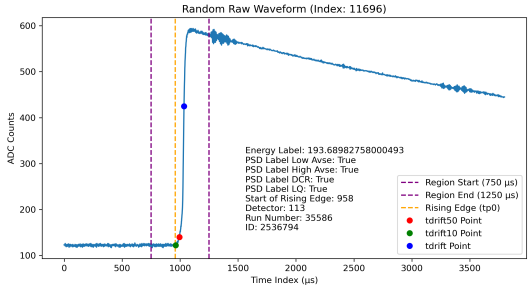

Figure 1: A plot showing the tdrift thresholds

2.2 Late Charge

The LQ80 Parameter is used to measure the amount of energy being collected after 80% of the peak. This is important because multi-site events have extra energy collections that appear in this LQ80 region. In order to extract useful information from our Late Charge parameter we can see the behavior of our late charge in our multi-site versus single-site events. In our multi-site events there is an extra energy collection which causes another spike in our LQ80. This phenomenon is not seen in our single-site events.

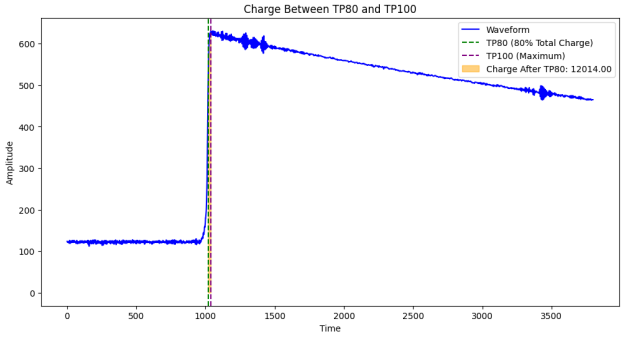

Figure 2: A plot depicting the LQ80 region

2.3 Late Charge Slope (Area Growth Rate)

Using the Growth rate we can see how our multi-site even grows compared to our single site event. To capture this relationship, there was an area created that approximated peak energy from 80% of the charge and subtracted the actual area from 80% of the charge to the peak. This method was done using a summation of the energy points instead of fitting a line in order to keep the small changes in energy present in our growth rate.

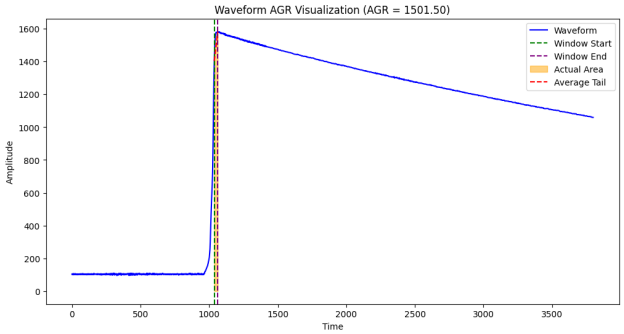

Figure 3: A plot showing the Area Growth Rate

2.4 Second Derivative Inflection Points

The next parameter, our Second Derivative inflection points, finds the amount of inflection points from 80% of our charge to the peak. This is because multi-site events have more concavity changes than single-site events. This was done by using NumPy gradient twice to find the second derivative, and then using NumPy sign and diff and filtering to the elements that had 2 and -2 which would be points where our gradient changed from positive to negative and vice versa.

2.5 Rising Edge Slope

The Rising Edge slope parameter is a way for us to measure the slope of the charge that was recorded. This is important because in our multi-site events there is an extra charge being deposited. This would mean that our slope would be lower in our multi-site events and due to this we can calculate the slope from our initial charge to the peak and use it as a parameter for our model. We can carry this out by fitting a line through linear regression to our rising edge and calculating the slope.

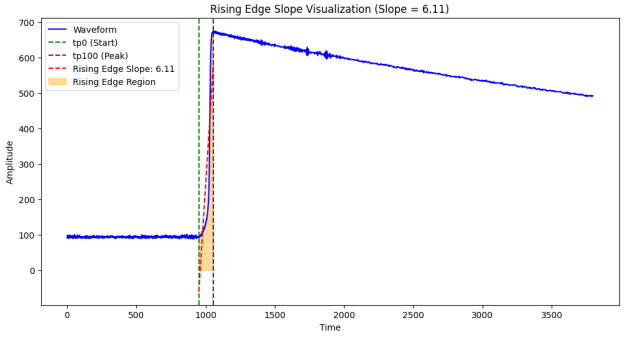

Figure 4: A plot showing the Rising Edge Slope region

2.6 Rising Edge Asymmetry

The rising edge asymmetry parameter describes how tilted in a direction the rising edge of the signal is, or in other words how skewed the rising edge is. This is important because it can help distinguish between MSE and SSE. MSE will usually have a lower skew meaning it is more symmetrical compared to SSE. When fitting to a MSE under-fitting can hide the extra bump and would make

the charge look like a SSE. Another thing to look out for when calculating rising edge asymmetry is if the curve is distorted beyond being comparable to a half-Gaussian, a half-Gaussian curve will have to be fit to be able to calculate the skewness. This can be dangerous since under-fitting or over-fitting should be avoided. In Python:

```
import numpy as np
from scipy import interpolate
from scipy.stats import skew

end = np.where(waveform == np.max(waveform))[0][0]
interp_range = np.arange(tp0, end + 1, 1)
interp = interpolate.interp1d(np.arange(waveform.shape[0]), waveform, kind = '
    linear')
interp_vals = interp(interp_range)
skewness = skew(interp_vals)
```

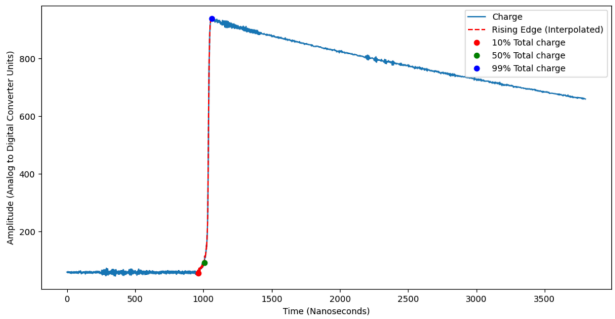

Figure 5: This rising edge is a single site event with a REA of 1.316147266130899

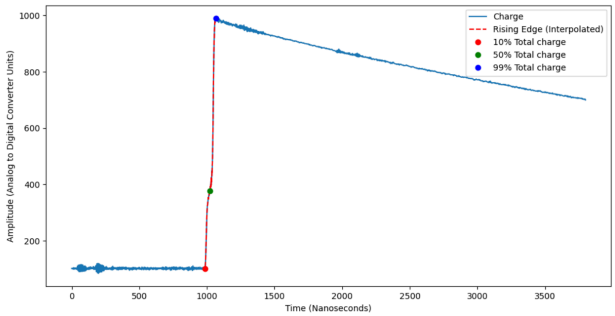

Figure 6: Comparatively this rising edge is a multi-site event and has a REA of 0.7821869431061642

2.7 Current Amplitude

The current amplitude of a single waveform is the peak rate of charge collection, defined as I = dq/dt which means current amplitude is the derivative of charge. The reason we need that parameter is it helps distinguish between different types of events such as SSE and MSE in our particle detection project, where SSE typically shows a higher amplitude as opposed to MSE.

Normalization is performed on the interpolated waveforms to eliminate energy dependence before applying a sliding window derivative function. In many signal processing or particle detection applications, energy dependence can influence measured values, leading to inconsistencies in parameters

across events of different energies. Finalize the amplitude of a waveform by finding the maximum value of that waveform's derivatives.

2.8 Energy Peak

After the particle hits the detector, the energy reaches its peak. A dramatic spike is visible in the energy graph. This is typically the maximum ADC count. The height of this peak correlates with the energy deposited by the particle in the detector, which is why it's used as a measure of the particle's energy.

The peak energy is extracted by locating the index of the maximum value in the waveform using the np.argmax() function, which shows the highest amplitude reached by the signal.

In the parameter extraction code, it shows up as follows:

```
import numpy as np
peak_index = np.argmax(waveform)
peak_value = waveform[peak_index]
```

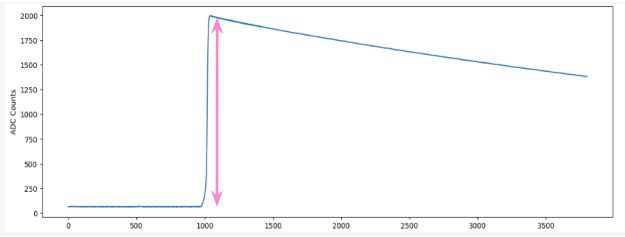

Figure 7: A plot showing the height of a waveform's ADC count

2.9 Tail Slope

The tail slope is the rate of charge collection over the length of the waveform's tail. It indicates how quickly charge dissipates in the detector after the initial interaction. Events where charge deposition occurs near the surface of the detector cause a delayed charge collection and result in a slower decay rate within the tail, so the slope of the event is less negative. Before the slope is extracted, the waveform has to be corrected for negative slope that could be induced by amplification circuits. The tail slope is calculated by fitting a linear model to the final segment or the "tail" of the waveform, giving us the slope which represents the decay rate of the signal. Using the linregress package from scipy, the slope calculation appears like this in the code:

```
from scipy.stats import linregress
tail_segment = waveform[-500:]
time = np.arange(len(tail_segment))
slope, intercept, _, _, _ = linregress(time, tail_segment)
```

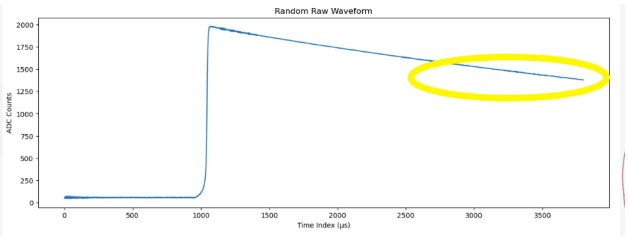

Figure 8: The highlighted section is a fragment of the tail

2.10 Delayed Charge Recovery

The DCR parameter helps to distinguish surface alpha background events by assessing the rate of area growth in the tail slope region. It quantifies the arrival of charges that are delayed by one nanosecond or more after the initial charge collection. Additionally, DCR is responsive to events where charges become temporarily trapped in potential wells within the detector before being released. This parameter is extracted by calculating the area above the tail slope to the peak of the waveform rise.

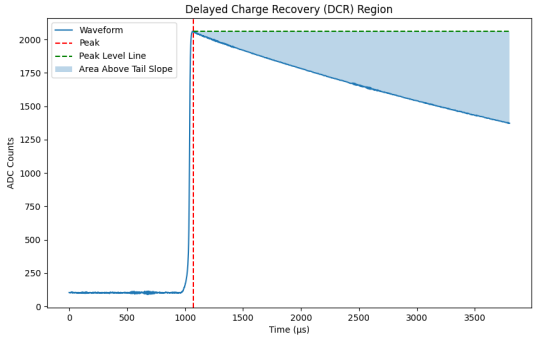

Figure 9: A plot showing the DCR region of a random waveform

2.11 Fourier Transform and Low Frequency Power Ratio

The Fourier Transform is a mathematical operation that transforms a time-domain signal into its frequency-domain representation. It decomposes a complex signal into a sum of sines and cosine components at different frequencies. Fourier Transform is useful for distinguishing between single-site events (SSE) and multi-site events (MSE) within detector data because of the nature of charge distribution within the detector, as most of the energy from the particle interaction in SSE is deposited at a single point whereas it is deposited in multiple locations in MSE. For this experiment, the Fourier transformed waveform is normalized to enhance the results.

To capture the differences between SSE and MSE, Low Frequency Power Ratio (LFPR) is used, quantifying how much of the signal's energy is concentrated in the low-frequency threshold by the total power spectrum of the Fourier transformed waveform. This ratio provides a measure of the waveform's "smoothness". SSE waveforms tend to have a higher concentration of energy at lower frequencies because their energy distribution is concentrated at lower frequencies, resulting in high LFPR. In contrast, MSE waveforms have a more complex distribution of energy from interactions at multiple sites. This broader distribution leads to a lower LFPR.

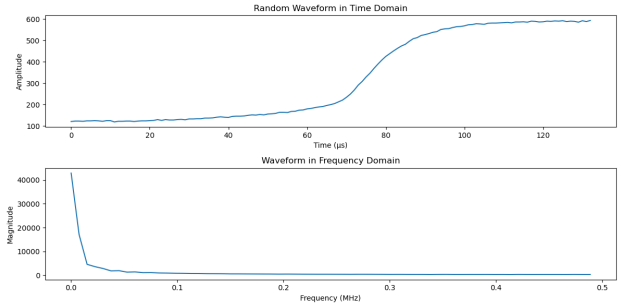

Figure 10: A plot showing the Fourier Transformation

3 Model Building

3.1 Classification Group

3.1.1 XGBoost

For the XGBoost model, our features had already been created and in order to use these features the first step taken was to build our baseline model. But, there was a problem with our true labels. In our data, about 98% of our labels were true and only 2% were false, this would lead to a horrible imbalance in our predictions. In order to combat this, we used imblearn to oversample our false labels and make sure that our model was trained on a fair balance of false and true labels. Next, we were able to create a model to fit the training data and tested it on our testing data. The main problem we ran into was the accuracy of our false labels and in order to fix this there was hyperparameter tuning to best train our model to predict these values. Overall, our model is still being worked on, but it achieved much better accuracy on both the true positives groups. Using feature importance, we see that most of the features have equal importance, with tdrift50 and tdrift10 being the least important.

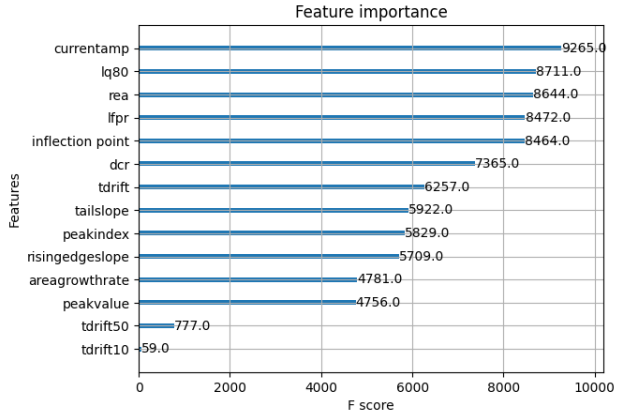

Figure 11: XGBoost Feature Importance

When we analyzed our XGBoost model using the ROC curve, it had an area under the curve of 0.71 showing that it is a well performing model that has room for further improvements.

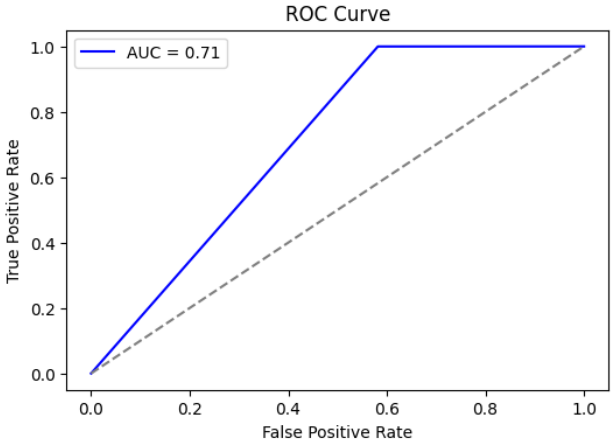

Figure 12: XGBoost ROC Curve

3.1.2 Random Forest Classifier

For the random forest model, the processed dataset was used. This dataset had the specific features that were extracted using the parameter extraction script. The random forest was used to classify the lq label. Similar to the other models, the imblearn distribution was used to create an equal balance of data that was true and false. Once this data balance was achieved, the model was initially tested and resulted in a 91% accuracy score.

Next, a grid search was employed to find the best set of hyperparameters. This also showed which features were most important when creating the classifier.

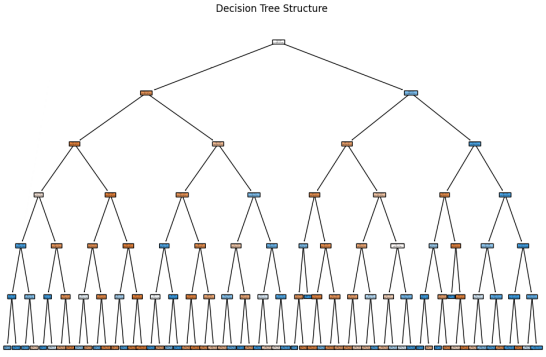

Figure 13: Decision Tree Structure

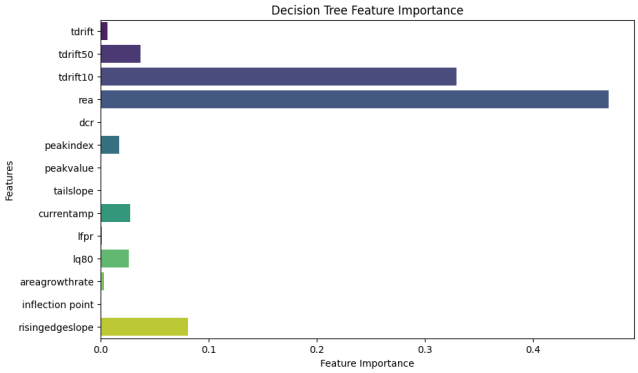

Figure 14: Decision Tree Feature Importance

After hyperparameter testing, the minimum samples and maximum tree depth parameters were modified to make the accuracy increase to a 97% score. A final confusion matrix can be seen in Figure 15. Finally, the cross validation score distribution is also plotted in Figure 16. The range of scores is from 90 to 92, indicating that the model is stable and doesn't overfit to the data when it is tested.

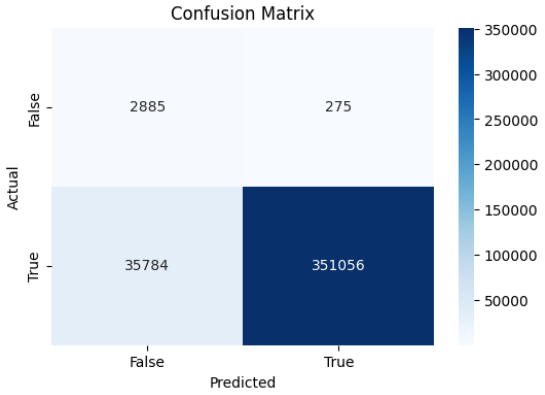

Figure 15: Decision Tree Confusion Matrix

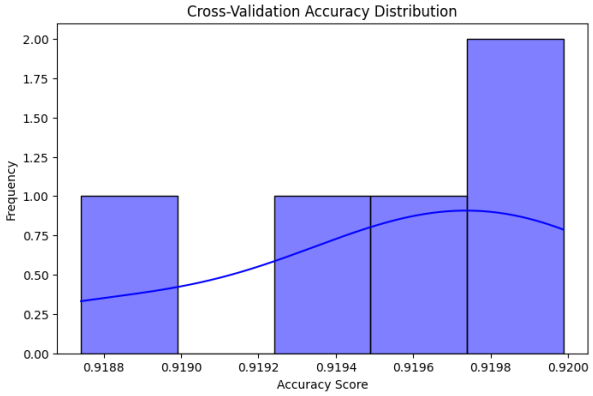

Figure 16: Decision Tree Accuracy Distribution

3.1.3 LightGBM

The classification of lowavse in the Majorana Demonstrator dataset is a critical task to distinguish signal events from background noise. To improve the model's performance, I used the LightGBM classifier, leveraging its efficiency and ability to handle high-dimensional data. I addressed the significant class imbalance between signal and background events by applying SMOTE (Synthetic Minority Over-sampling Technique) to balance the training data, ensuring the model was exposed to more representative examples of the minority class. Furthermore, I conducted hyperparameter tuning using RandomizedSearchCV, optimizing key parameters such as num_leaves, min_child_samples, and colsample_bytree to enhance the model's generalization and prevent overfitting. As a result, I was able to achieve an accuracy of 0.969921.

To evaluate the model's performance, I generated key visualizations, including a Feature Importance Plot, ROC Curve with AUC, SHAP Value Summary Plot, and a Confusion Matrix. The feature importance plot revealed that tdrift, lq80, and currentamp were among the top influential features, aligning with the physical understanding of the experiment, as they are directly related to energy deposits and detector behavior. The ROC Curve indicated a high AUC score of 0.99, suggesting strong model performance in distinguishing between signal and background.

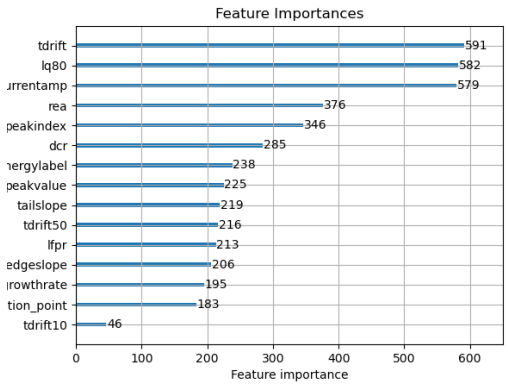

Figure 17: LightGBM Feature Importance

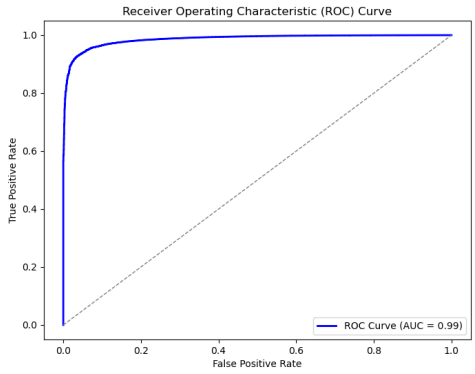

Figure 18: LightGBM ROC Curve

The SHAP plot provided further insight into how individual features impacted the model's predic-

tions, confirming that features such as tdrift and lq80 had the most significant contributions. The thickness of the SHAP summary plot points indicates the density of data points at that SHAP value, with thicker regions representing a higher concentration of instances where the feature had a similar impact on the model's prediction. Lastly, the confusion matrix showed a high number of true positives and a relatively low number of false negatives, indicating that the model effectively identified rare lowavse events while maintaining a reasonable level of precision.

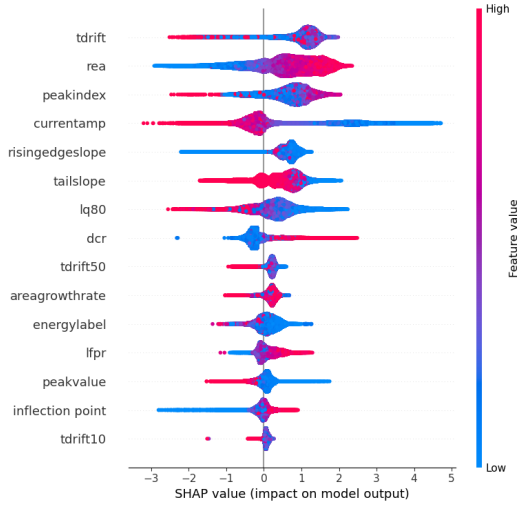

Figure 19: LightGBM SHAP plot

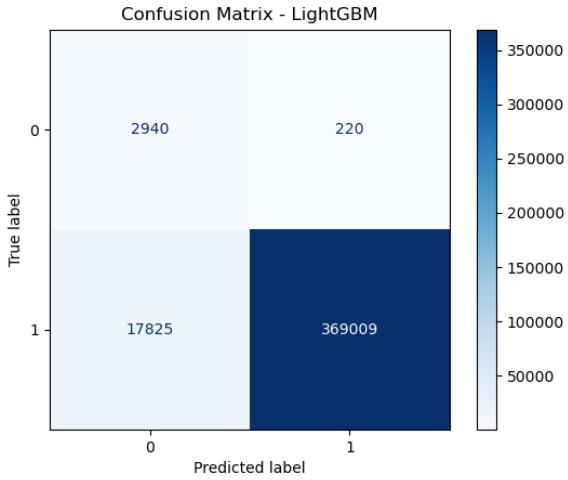

Figure 20: LightGBM Confusion Matrix

3.1.4 CatBoost

The CatBoost model was trained to classify the high_avse label using processed data. Since approximately 98% of the data was labeled as true for high_avse, SMOTE was applied to balance the dataset by generating synthetic samples for the minority class. The default model configuration achieved an accuracy of 96.5%. To improve performance, hyperparameter tuning was performed using RandomizedSearchCV, along with feature importance analysis and cross-validation to check for overfitting. These optimizations increased the model's accuracy to 99.4

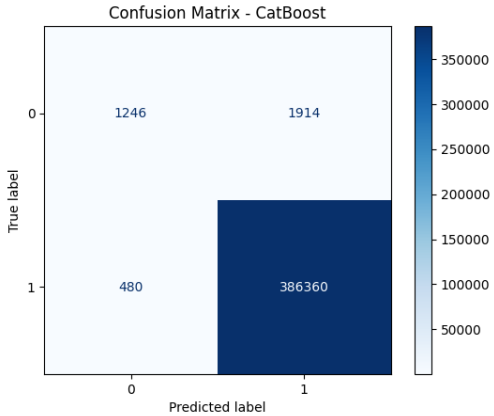

Figure 21: CatBoost Confusion Matrix

3.1.5 SVM Classifier

The Support Vector Machine (SVM) model was initially used to classify the high_avse label with processed data. However, due to the large dataset size, the computation time for a standard SVM with an RBF kernel was too long, making it impractical for efficient training and evaluation. To address this, LinearSVC was used instead, significantly reducing training time while maintaining strong performance. The dataset was first balanced using SMOTE from the imbalanced-learn library to handle class imbalance. With default settings, the model achieved an accuracy of 87.7%. Further improvements were made by applying 5-fold stratified cross-validation, hyperparameter tuning using RandomizedSearchCV and GridSearchCV, and optimizing the regularization strength, loss function, and tolerance. These optimizations increased the model's accuracy to 92%, as demonstrated by the ROC curve, which achieved an area under the curve (AUC) of 0.92.

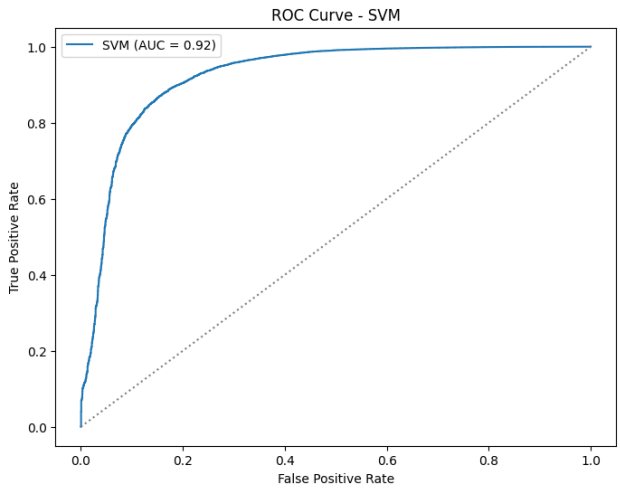

Figure 22: SVM ROC Curve

3.2 Regression Group

To begin our investigation of regression models, we first performed an exploratory data analysis on the processed dataset. This analysis involved examining the distributions of the independent variables and the target variable to gain a deeper understanding of the processed data. The target variable, Energy Label, is a continuous measure representing the energy level of individual waveforms in kiloelectronvolts (keV). Through our analysis, we identified strong linear relationships between specific parameters and the Energy Label. Based on this finding, we determined that a simple linear regression model would be the most appropriate choice for our baseline model.

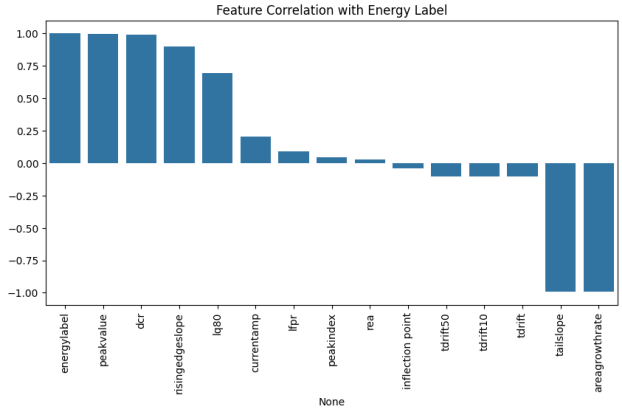

Figure 23: A plot depicting the correlation between parameters and Energy Label

3.2.1 Baseline Model (Linear Regression)

Before training our model, we standardized the features to ensure consistency across the dataset. All models will have the features standardized prior to training to reduce variance between models. Furthermore, certain features, such as tdrift50 and tdrift10, were removed due to high multicollinearity. This decision was based on the fact that tdrift50 and tdrift10 are proportional to tdrift by a constant, making them redundant in the analysis. The baseline model was trained using 5-fold cross-validation to enhance its generalizability across different datasets. This technique helps mitigate overfitting and ensures that the model performs well on unseen data. For evaluation, we selected mean squared error (MSE) as our validation metric because of its robustness in penalizing larger errors more heavily. By minimizing MSE, the model prioritizes accuracy in predicting energy levels, making it a reliable measure of performance. The baseline model achieved an MSE of 4706.4220. Although interpreting MSE in isolation can be challenging, it serves as a benchmark for comparison against more complex models, with the objective of achieving a lower MSE. To determine the optimal number of training parameters, the baseline model was evaluated using a loop that tested parameter counts ranging from 1 to 12. The results indicated that the lowest MSE was obtained when all 12 parameters were utilized, suggesting that incorporating the full set of features improves model performance.

Figure 24: Baseline model performance compared to number of parameters

3.2.2 Ridge Regression

A total of four different models were trained on this dataset, the first being a Ridge regression model. This model closely resembles linear regression, but includes a regularization term to enhance robustness and prevent overfitting. To optimize model performance, grid search was employed to perform cross-validation and hyper-parameter tuning. The best performing regularizer, which minimized the MSE, was determined to be 1e-05. However, despite this optimization, the lowest achieved MSE remained identical to that of the baseline model, indicating that Ridge regression did not provide an improvement in predictive performance.

3.2.3 SVM Regression

The second model developed was a Support Vector Machine (SVM) regression model, which is inherently more complex than the preceding models. However, after conducting cross-validation and hyperparameter tuning, the resulting MSE was 4759.4716, exceeding that of the baseline model. This outcome suggests potential overfitting, indicating that the SVM model may not be well suited for our predictive task.

3.2.4 Random Forest Regressor

Furthermore, the data set was trained using a random forest regression model, a robust model known for its resilience to overfitting, a challenge encountered with the SVM model. Additionally, Random Forest models exhibit versatility and adaptability to various types of relationships within the data. After optimizing for minimal MSE, the model achieved an MSE of 1860.8112, marking a substantial improvement over previous models. Hyper-parameter tuning primarily focused on adjusting the number of trees in the model. Figure 25 illustrates the training and testing MSE across different tree counts, demonstrating that increasing the number of trees leads to a reduction in MSE. The optimal number of trees, resulting in the MSE mentioned above, was determined to be 128.

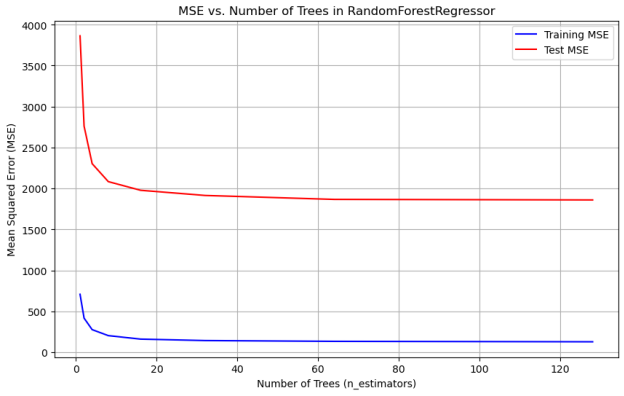

Figure 25: Training and Testing MSE for Random Forest Regressor

3.2.5 Neural Networks

The final model trained and evaluated was a Neural Network with three hidden layers: input -> 64 -> 32 -> 16 -> output, implemented using the PyTorch. It is crucial to note that null values

were explicitly excluded from this analysis, as their presence could compromise the integrity of the algorithm and ruin the whole NN model. Due to computational constraints, the number of epochs was limited to 30; however, in an optimal setting, increasing this to 100 would likely yield improved performance. The Rectified Linear Unit (ReLU) activation function was selected for the neural network due to its versatility and computational efficiency. Following training, the model achieved a minimized MSE of 1507—the lowest MSE among all models tested. This result suggests that the Neural Network model outperforms the other approaches in terms of predictive accuracy. While further training iterations and more extensive hyperparameter tuning could improve model performance—some models were constrained by computational time—these findings highlight the potential of deep learning for our predictive task of predicting the energy label of a waveform.

As we can see here, the wide-spreading residual points in Figure 26 indicate that Neural Networks cannot predict some of the points well at specific actual energy values. One of the reasons might be that we are not able to know the true dependent labels in the test set that can only be predicted by classification. Which could result in absences of information about multi-site events and such physics phenomena that could change the true energy(dependent variable). However, although the Predicted vs Actual graph of the Neural Networks model might seem worse than the one of the Random Forest Regressor, the overall performance of the Neural Networks model still has the lowest MSE among all the models as demonstrated in Figure 27.

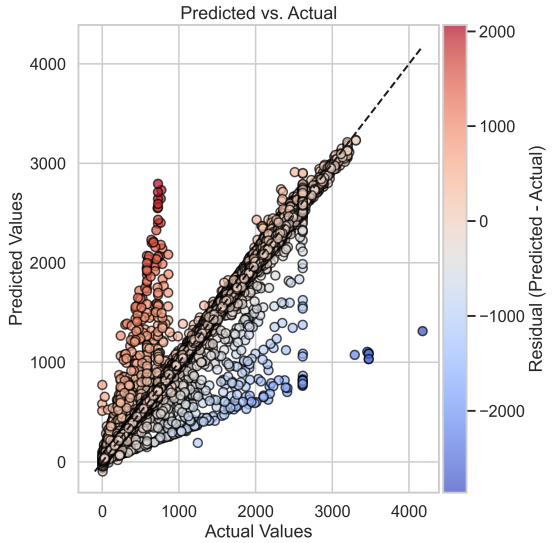

Figure 26: Neural Network Predicted vs Actual

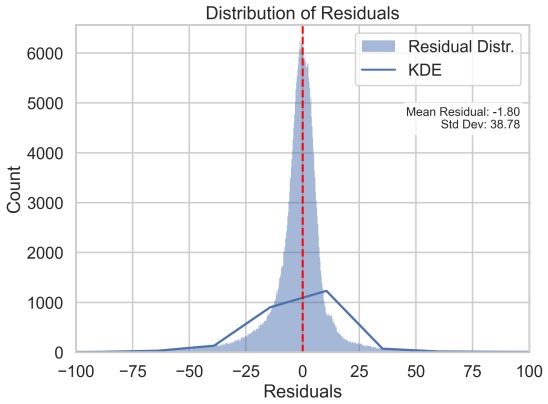

Figure 27: Neural Network Distribution of Residuals

3.2.6 Conclusion

Figure 28 is a violin plot with a y-axis ranging from -75 to +75 that shows the distribution of residuals among the models we used: Neural Network (NN), Random Forest (RF), a Baseline Model, Support Vector Machine (SVM), and Ridge Regression. If we only look at the zoomed figure along with the probability mass table, RF and SVM have relatively slenderer violins around zero whereas NN has the best residual distribution and a higher probability mass between +75 and - 75, meaning they might have better predictive accuracy. Whereas the others appear narrower than NN so they have moderate-value residuals distributed. However, a distribution with narrower center could also indicate under-fitting, where the model doesn't perform well to capture the true variability in the

data , but we can rule out this assumption since all the models have desired R2 scores that are above 95%.

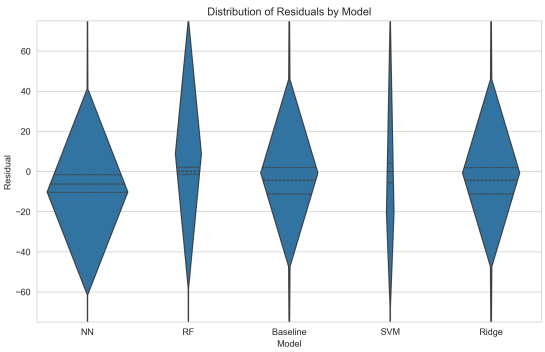

Figure 28: Regression Models Violin Plot

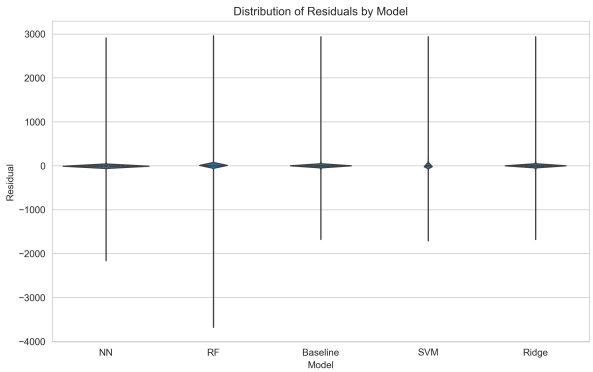

Figure 29: Regression Models Distribution of Residuals

The following table presents the MSE performance as well as probability mass of each model used in the analysis.

Model	MSE	Probability Mass
Baseline	4706.4220	0.9796
Ridge	4706.4220	0.9796
SVM	4759.4716	0.9774
Random Forest Regressor	1860.8112	0.9878
Neural Networks	1507.1353	0.9896

Table 1: MSE performance and Probability Mass of all regression models

Most importantly, by comparing the two plots and the table at once, we can observe that although NN has a relatively similar range of residuals compared to the others(and RF has an even larger one), the probability mass table shows that only a small portion of the residuals are outliers. Despite ridge and baseline looks better than RF in the zoomed plot, we also need to consider the residuals outside of +75 and -75 which is hard to see but well explained by the probability mass table. Therefore, in terms of variance explained, consistency and MSE, NN will be our best overall model.

References

[1] Collaboration, M. (2023b). Majorana Demonstrator data release for AI/ML applications [Dataset]. In Zenodo (CERN European Organization for Nuclear Research). https://doi.org/10.5281/zenodo.8257026