TRIDENT

A machine learning regression task

Kunz Bryan

June 14, 2025

1 introduction

This machine learning project is a regression task with the goal to predict the initial energy of neutrinos coming from the space. To do so we will use virtual data from the experiment Tropical deep-sea neutrino Telescope (TRIDENT) in the south of China.

Data given is composed of two files. One with spatial and temporal information about sensors, in "features.parquet". The other with initial energy and cumulative length, in "truth.parquet". From cumulative length we create new input data named number of pulses. We will see the difference in prediction by using one input data or the other. The choice of input data have an impact on the model architecture.

Our document begin with an overview about the physical experiment that gives our data. It will be followed by a brief explanation about how a machine learning project is structured. Then it will be the time look at our data. From target to inputs features with a part about number of pulses. In this overlook we will choose which pre-process to apply. It leads us to the choice of models respect to input data. Then a paragraph about the issues encounter, and optimization techniques used. We will conclude by a demonstration of performance of our model.

2 Physical background

Tropical deep-sea neutrino telescope (TRIDENT) is an experiment at the south of China that have the goal of find source of high energy cosmic neutrinos.

The process is the following, a high energy neutrino coming from the space hit the atmosphere then produce a shower of particles, figure 1, form this shower, muons with high energy are produce. They will travel until interact with the ocean water and produce Cherenkov cone, figure 2, this light will be record by sensors.

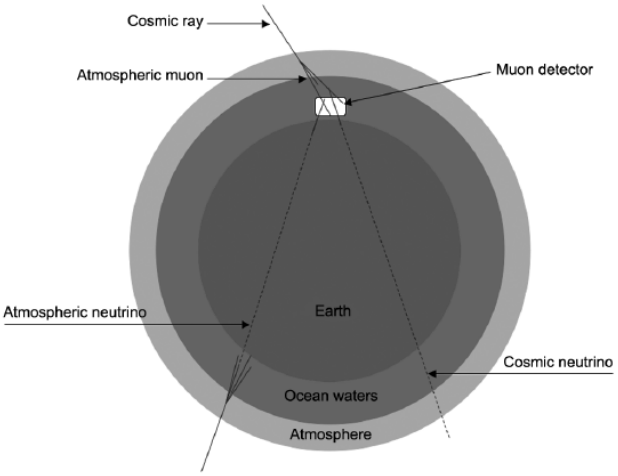

Figure 1: Schema of incoming neutrinos which interact with the atmosphere and production of muons travelling to the detector. Source [1]

The actual measure is an event. Which consist of a sequence given number of sensors excited by the Cherenkov light at a given time. Each sensor is define by a it's position (x, y, z) and a interaction time t.

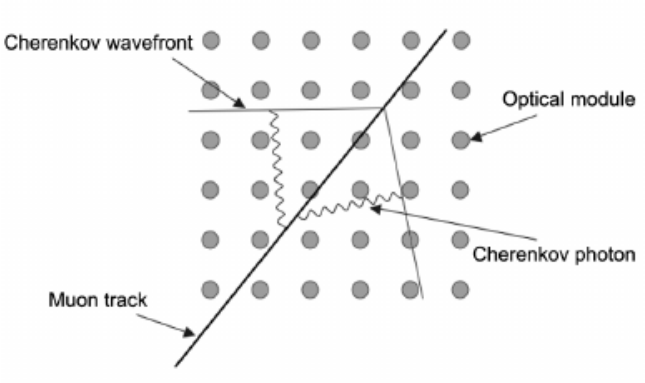

Figure 2: Schema of incoming muon create Cherenkov light by interacting with the water. This light is detect by sensors and allow to reconstruct the initial energy of the neutrino. Source [1]

3 Machine learning Structure

A typical machine learning project is organise as following.

1. Loading data
2. Split data between training dataset, validation dataset and evaluation dataset
3. Pre-process datasets
4. Define a model and train the model on the training dataset, in parallel use validation dataset to see it's performance
5. Use the model on evaluation dataset for it's generalisation

Let see how this structure is apply to our project.

3.1 Trident structure

Here data will be "features.parquet" and "truth.parquet". "features.parquet" have informations of sensors. "truth.parquet" have information about the initial energy and numbers of sensors used for an event. We create a single dataset with sensor features, number of pulses and truth values. The creation of this set implies to pad features to have the same number of features for each events.

The merged dataset is split into three. Training dataset is use for learning while validation dataset is for checking performances of our model. Evaluation dataset is use to test the model in real application.

To have a model learning well, we have to scale data to have a Gaussian shape. Also we want reduce at it's maximum bias of data to have a learning more fair respect to inputs. This is what we call pre processing. We have explore three types of scaling techniques, "min-max", "standard" and "robust", see more in Data section.

The goal for our model is to learn and not memorise. This issue is call over fitting and it's characterize by a validation loss significantly bigger than the training loss. If the validation loss is significantly under the training loss, it's a singe of over regularise the model. Also we can compute the statistic of performance of our validation dataset, see ,ore in section Results.

Select the model architecture in function of inputs data will affect it's prediction. What is important for our model is the permutation invariant respect to inputs data. That means, if

for same data, shuffling them will change the prediction. If it's permutation invariant, we can use a multi-layer perceptron (MLP), if our model is not permutation invariant, we need to use DeepSet, see more in section model.

4 Data

In this section, we explore data and their representation, the best is to look at statistic of dataset. We want to know which pre-process is the best for each data. Inputs features are in the "features.parquet" file. Initial energy with the respective cumulative length are in the "truth.parquet" file. Cumulative length give the size (number of sensor involve) of a given event, it's important to have this information to be able to determines which feature row is related to an initial energy value.

We want pre-process data to have a model with better performance and stability compare to one using bar data. A key point is to have a model robust respect to outlier, less sensitive to extreme values.

To pre-process data we will try three different scaler and see which is the best:

1. min-max : normalise data to be in range [0,1].
2. standard : normalise data to have a Gaussian shape with a mean at zero and a standard deviation equal to one, best shape for data used in a machine learning project.
3. robust : normalise each features to Gaussian shape, apply independently of the statistic of the dataset. Concretely it will put the median to zero.

4.1 Initial energy state

Truth represent the initial energy of neutrinos. Values are store in "truth.parquet" file. First look at values respect to their position in the file.

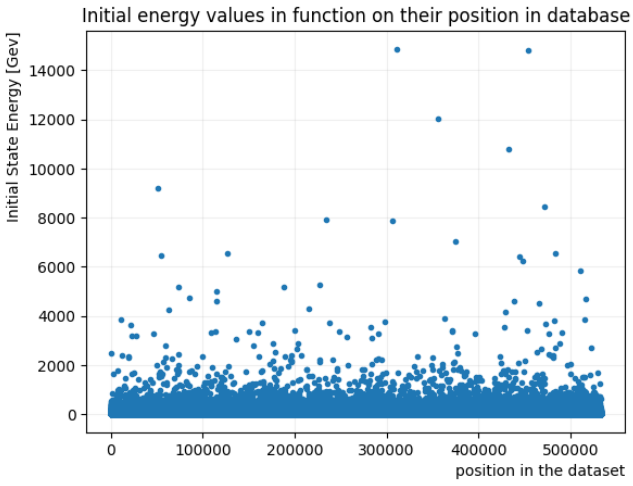

Figure 3: Initial energy state distribution.

We can extract three informations form this picture. First, our dataset is quite small, "only" 50'000 values. Second, the range of energy is huge. Lastly, there is a massive amount of data have low energy. To have a more precise idea about this disparity, look at the plot the count of data within a range of energy, for small values of energy.

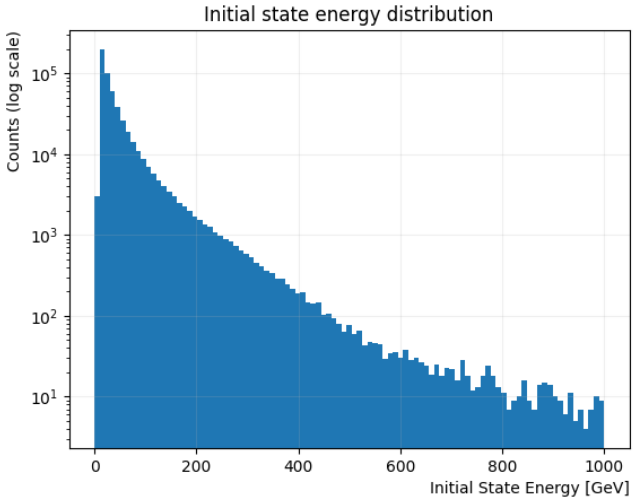

Figure 4: Initial energy state distribution in terms of number of events in small values. Data have been log().

Our outputs data are clearly bias toward low energy values, even we have log() them.

This mean we have to weight the loss function to not give a preferential output. Also we have to find how to pre-process them before compute the weight, to be much more easy to handles.

Let have a look at the statistic of our dataset:

Table 1: Statistic of initial energy values, **no** scaler and **no** log.

mean:	47.702
median:	25.899
std:	93.273
min:	10.000
max:	14 859.621

It's telling us what we have already find by looking previous figures. The mean show that a lot of data have low values and the distribution is highly skewed due to the huge difference between min and max. The standard deviation (std) say that there is a large spread, because it's almost twice the mean. All of this indicate that there is extreme outliers data. Also data are not centred.

We have used "log()" function on our data, distribution become less skewed.

Look at effect of different scaler, with log(), then let argue which one to pic:

Table 2: Statistic of initial energy values, **log** + **min-max** scaler.

mean:	0.148
median:	0.124
std:	0.112
min:	0.000
max:	1.000

Table 3: Statistic of initial energy values, **log + standard** scaler.

mean:	0.000
median:	−0.210
std:	1.000
min:	−1.321
max:	7.634

Table 4: Statistic of initial energy values, **log + robust** scaler.

mean:	0.153
median:	0.000
std:	2.727
min:	−0.808
max:	5.705

Min-max scaler will not help because our data are not centred and there is a cluster near zero. Standard scaler is clearly better, but the large difference between min and max, even with a logarithm apply say that there is still outlier effect. We will pic the robust scaler because it is more resistant to outlier values, this will help the model to train and be generalise due to the normal shape.

4.2 Numbers of pulses

Number of pulses is a particular inputs feature, it requires to do few lines of codes (two or three maximum) before obtaining data. This input data come from the cumulative length store in the "truth.parquet" file. We get the number of pulses by calculating the difference of each row of cumulative length data. Cumulative length give the numbers of features (sensor activate) associate to an event.

Now let see see how are these data. Then we will compute it's statistic. As we have done previously, we scale data and argue which one we choose to scale our data.

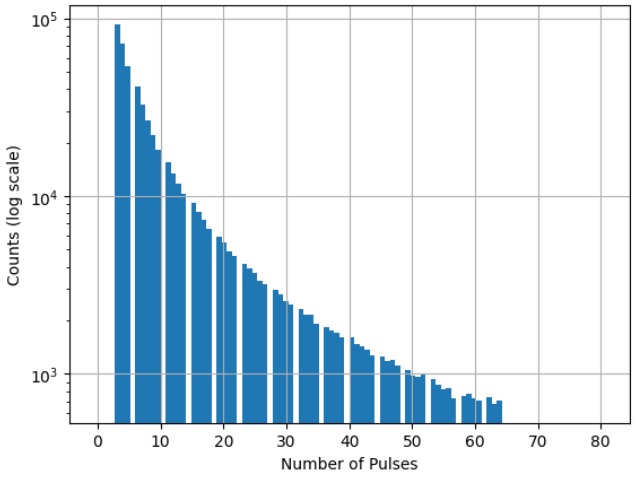

Figure 5: Distribution on the number of inputs. Data have been log().

The biggest event have 64 inputs features associate to it. Event under logarithm, we have a huge bias toward small event and skewed distribution. Look at statistic of our dataset:

Table 5: Statistic of number of pulses values, **no** scaler and **no** log.

mean:	11.370
mdeian:	7.000
std:	11.658
min:	3.000
max:	64.000

This statistic indicate, by a mean bigger than the median, there is a strong right skew. Min, max and std show the range of value is large, also there is fewer large value compare to small values. All values are positive, it will be a problem when using our activation function, rectified linear unit (ReLU), we will be on linear part only. So now look at the statistic under scaler.

Table 6: Statistic of number of pulses values, **log + min-max** scaler.

mean:	0.297
median:	0.249
std:	0.259
min:	0.000
max:	1.000

Table 7: Statistic of number of pulses values, **log + standard** scaler.

mean:	0.000
median:	−0.185
std:	1.000
min:	−1.145
max:	2.716

Table 8: Statistic of number of pulses values, **log + robust** scaler.

mean:	0.130
median:	0.000
std:	0.701
min:	−0.673
max:	2.035

For min-max scaler, we have, like for truth data an cluster near zero. Standard scaler gives a centred and normalize distribution, but, there is still outlier effect (can be see by looking min, max and std). We will choose the robust scaler, because it is less sensitive to outlier and follow a normal distribution with a median centred, and the spread is reasonable.

4.3 Features

Here are the real features relate to a physical entities as discuss in Physical background section. We have see with the number of pulses that events do not have the same numbers of data in it. But what look like features of a random event?

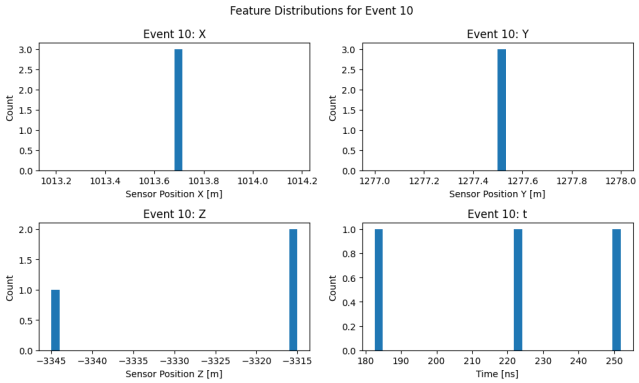

Figure 6: Sensors features associate to the event 10. Each sensor feature is composed of 3 spatial coordinates and 1 temporal feature. Not all event have the same number of sensor activated, but the four features remains.

For an event, we have relatively spread values. What look the shape of features dataset.

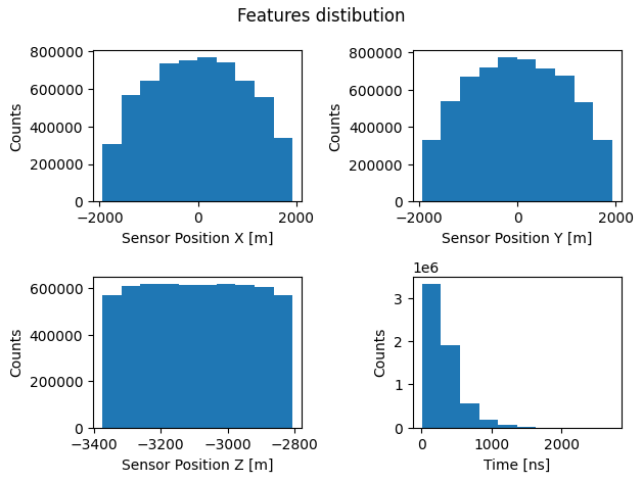

Figure 7: Global distribution of all features, all events.

Data of dataset for "x" and "y" are closer to a normal distribution than the other features. "z" component is essentially flat but experience a strong shift toward negative values due to reference point. This is bad because of our activation function, ReLU which will set to zero our "z" input.

Spacial features give one important indication, with these simulate data, the experiment expect detect particles coming from the x-y plan of the detector.

Temporal feature indicate that the time of an event is generally short. For us temporal data have the same problem as truth and number of pulses, so we will use logarithm on temporal feature.

Table 9: Statistic of features, of all events, **no** scaler.

x_pos		y_pos		z_pos		t, no log()	
mean:	−0.927	mean:	−2.149	mean:	−3090.285	mean:	313.020
median:	1.663	median:	−4.671	median:	−3105.000	median:	252.663
std:	973.550	std:	974.413	std:	170.859	std:	225.900
min:	−1943.353	min:	−1939.758	min:	−3375.000	min:	6.618
max:	1925.376	max:	1932.056	max:	−2805.000	max:	2721.500

Now we look again the effect of the scaler, but we have to do this for each features separately, since their shapes are really different. for t we will also use logarithm.

Table 10: Statistic of features, of all events, **min-max** scaler.

x_pos		y_pos		z_pos		t	
mean:	0.502	mean:	0.500	mean:	0.499	mean:	0.592
median:	0.503	median:	0.500	median:	0.474	median:	0.596
std:	0.252	std:	0.252	std:	0.300	std:	0.121
min:	0.000	min:	0.000	min:	0.000	min:	0.000
max:	1.000	max:	1.000	max:	1.000	max:	1.000

Table 11: Statistic of features, of all events, **standard** scaler.

x_pos		y_pos		z_pos		t	
mean:	0.000	mean:	0.000	mean:	0.000	mean:	0.000
median:	0.003	median:	−0.003	median:	−0.086	median:	0.038
std:	1.000	std:	1.000	std:	1.000	std:	1.000
min:	−1.995	min:	−1.988	min:	−1.666	min:	−4.887
max:	1.979	max:	1.985	max:	1.670	max:	3.372

Table 12: Statistic of features, of all events, **robust** scaler.

x_pos		y_pos		z_pos		t	
mean:	−0.002	mean:	0.002	mean:	0.054	mean:	−0.027
median:	0.000	median:	0.000	median:	0.000	median:	0.000
std:	0.616	std:	0.619	std:	0.633	std:	0.712
min:	−1.231	min:	−1.230	min:	−1.000	min:	−3.504
max:	1.217	max:	1.231	max:	1.111	max:	2.372

As reminder before doing statistic of t we have transform the data with log(). Here min-max scaler look to be a good choice, but over all we prefer a Gaussian shape. We will not pic robust scaler due to the absence of outlier for every scaled distribution, it's a bit over kill to take robust scaler, and due to physical position of sensor and physics, no outlier values will appear. So will take standard scaler.

5 Model

Our task is a regression task, that means we don't have to use output activation layer, like a Sigmoid. Also our loss function should be smooth, we can use MSELoss function which minimise large errors, or L1Loss function which is an mean absolute error (MAE), the purpose is to reduce all error linearly. We have choose a compromise between these two using SmoothL1Loss or Huber loss. All of these loss functions are in torch package. We also use mini-batching to help our model to find a minimum. As optimizer we use AdamW to do weight decay.

We have seen during the lecture two kind of models for this task, first, multilayer perceptron (MLP), second, deep set. We use MLP when our model is permutation invariant under inputs shuffling, and we use deep set if not.

A typical MLP architecture is input layer, activation layer, hidden layer and output layer. Generally there is always the input layer then a sequence of activation layer with hidden layers,

that gives the depth of our model. Before the output layer, we use a last activation layer. First entry of input layer is the number of features for an event. Output layer should return initial energy value, meaning a single value.

For a deep set, we use multiples MLP with a pool stage, this is where do an operation that is permutation invariant (like summing). For deep set we have MLP input, pooling, MLP intermediate and MLP out. We need the intermediate MLP (after pooling) to allow the model to learn and process these interactions, transforming the pool into a useful representation. This MLP intermediate allow also the model to learn more complex pattern.

For number of pulses as input data, we will use a MLP, because clearly data are permutation invariant. For features (from sensor) as input data we will use a deep set architecture since not all data have the same length shuffle them will change prediction.

5.1 Issues and optimization techniques

One of the biggest issue we encounter was over fitting, for both inputs features (number of pulses and sensor features), to reduce this we have add in our model drop-out layer and batch normalisation layer. A drop-out layer will select random a determine fraction of inputs to zero, this force the model to not rely on a single neuron. A batch normalisation layer, set the mean to zero and variance to one between each layer, this accelerate and stabilize the training by reducing internal shift. Also we can play with hyper parameters like learning rate, drop out percentage, depth of the model, changing weight decay, hidden dimension and batch size.

One surprising behaviour that can happen is a validation loss is smaller than the training loss. That usually means we have over regularize our model, to much drop-out or to much batch normalisation. It can be linked to the data distribution, in the sense of our validation dataset is more "easy" than our training dataset.

We have also use early stopping method to stop the training when the validation loss is at it's minimum, also we save the model parameter. That two hyper parameters to our config file.

batch size can play a role in the efficiency of our model. Small batch will add more noise but can help for generalisation. Big batch gives more stable predictions which help for the convergence of the model.

Finally to have the best data to show, we have search the best combination of hyper parameters. Because it is not really efficient to do it manually, we have heard of automatic process like "Optuna" package, "grid search" from "Scikit-learn" package and "Randomize search" from "Scikit-learn" package. We have try Optuna, but due to time, we have stop the procedure and rely on testing manually a set of parameters.

6 Result

For the training part, we have mainly tree outcomes. First is the Loss vs epochs that show the behaviour of the loss function (for the training set and the validation set). The second one is the plot of prediction vs target values, we are interested in predictions within a rang of 5% accuracy from truth values. Finally we plot metrics of the model, which show the performance of our model on validation dataset. What is important is to unscaled truth and prediction before doing the two last plots. Truth and prediction have the same scaling apply if the model have learn well.

For the evaluation part, we just do the plot of prediction vs truth, this plot show the behaviour of our model when it face unseen data.

6.1 Metrics

Here is a summary of metrics and their meaning.

Mean Absolute Error (MAE) : Average of absolute differences between prediction and truth. Measure on average, how far predictions are from targets. Units are the same as target variable.

Mean Squared Error (MSE) : Average of squared differences between prediction and truth. Measure the influence of outliers, by being more sensitive to them. Units are the square of your target variable.

Root Mean Squared Error (RMSE) : Square root of MSE. The measure is the same as MSE but even more sensitive to error. Units are the same as target variable.

$\mathbf{R^2}$: Determination coefficient. Unitless, measures how well predictions explain variance of targets.

R^2	meaning
>1	Something is wrong with data, it can't be possible to be bigger than 1
=1	the model perfectly explain the variance of targets
=0	the model cannot account for fluctuation of targets
<0	really bad, the model have learn nothing

6.2 number of pulses

Here is the result for the inputs number of pulses. Number of pulses is calculated as following, it is the difference between two adjacent row of cumulative length which is in "truth.parquet" file. Input number here is 1. We show one result of our model, the best of many manual try.

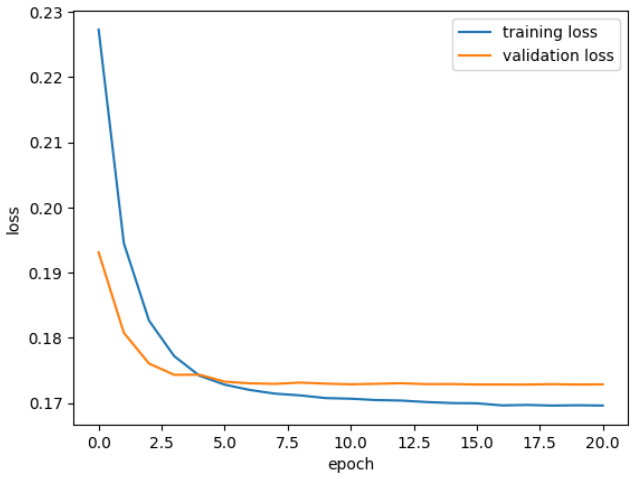

Figure 8: Plot of losses function over the epochs.

Figure 8 show that our training and validation have been stable during the training, also we see that our validation reach quite fast a local minimum, and never moved form it. Probably we should play with the learning rate to find a way to escape this minimum, during other try we didn't have pass the ground of 0.1700 for the validation loss. We can see also that we have

too much normalize our model, but without this little normalization, the model is heavily over fitting, so it appear to be a good compromise. Now look at real performance of our model.

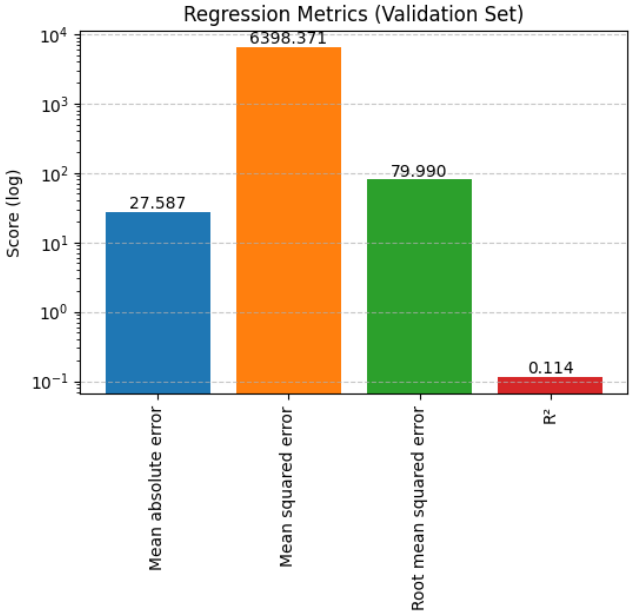

Figure 9: Metrics plot of of our validation set. Values are log.

Here we can already see that our model will predict pretty badly out target due to the huge errors bars for MAE and RMSE, and small determination coefficient. Let have a look to the prediction for the validation dataset.

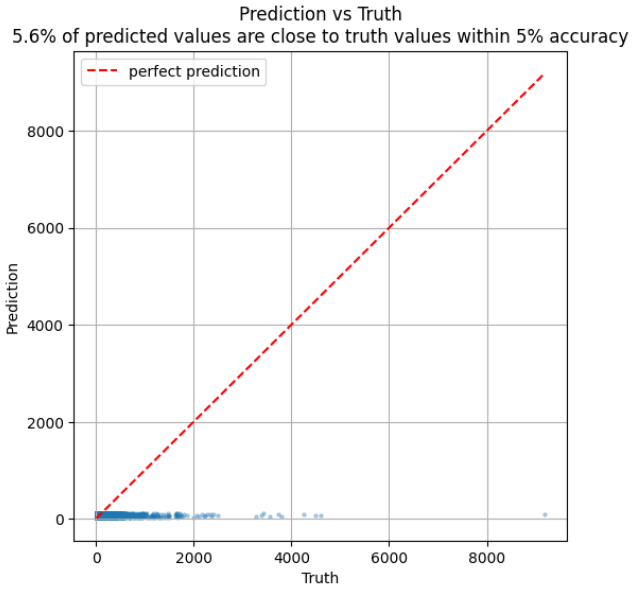

Figure 10: Prediction vs Target plot, for validation dataset.

As we see in figure 10, predictions are really poor for validation data. All previous model run with number of pulses as inputs predict correctly about 5% of within an accuracy of 5% of target data. We imagine the best possible prediction with this input we can reach 7% within an accuracy of 5% for the validation dataset. Now let look the behaviour of our model with unseen data.

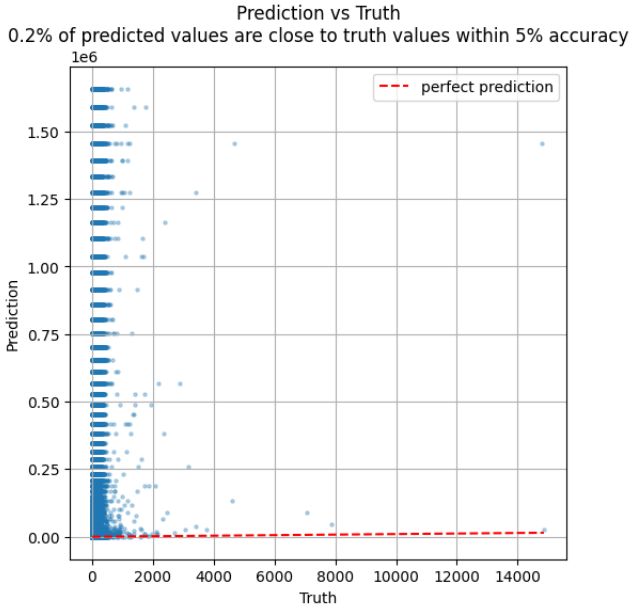

Figure 11: Prediction vs Target plot, for evaluation dataset.

As we see it's even worth than with validation dataset.

To have better prediction, we can try to tune more our hyper parameters. We can check again the pre-processing, just in case we have done something wrong. Explore different loss function. Explore different optimizer. Finally we can try to merge number of pulses and features to have a bigger input dataset, which can help the model. All last four propositions require to modify the code, while hyper parameter tuning is just a modification in config file, so can be done automatically..

6.3 sensor features

Here is the result for the inputs sensor features, input numbers here is 4. We show one result of our model, the best of many manual try. First look at losses over the epochs.

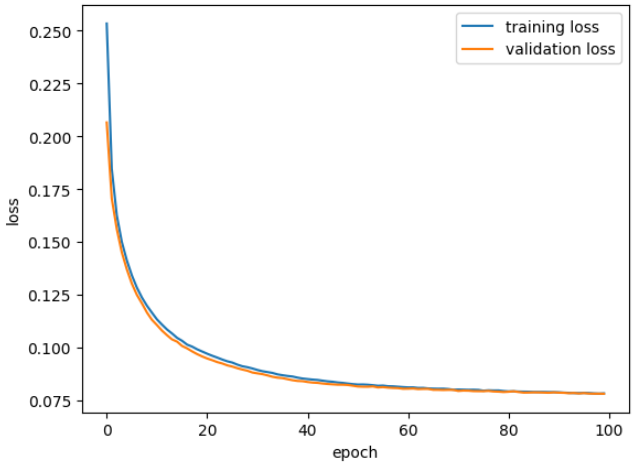

Figure 12: Plot of losses function over the epochs.

Compare to number of pulses, here the loss goes deeper, that indicate our model have learn better with these features. Also we see there is now over fitting which is great. Let see the metrics plot.

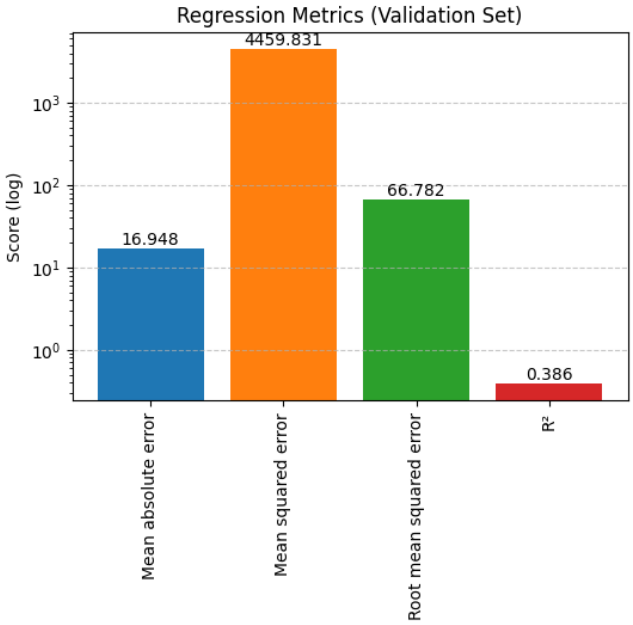

Figure 13: Metrics plot of of our validation set. Values are log.

Metrics plot effectively show that the model have higher performance, compare to the one with number of pulses. But it is still not a model that work as expected, we have still a huge MAE and RMSE. We turn our attention to the prediction plot to see the concrete benefit of using this inputs with the corresponding model.

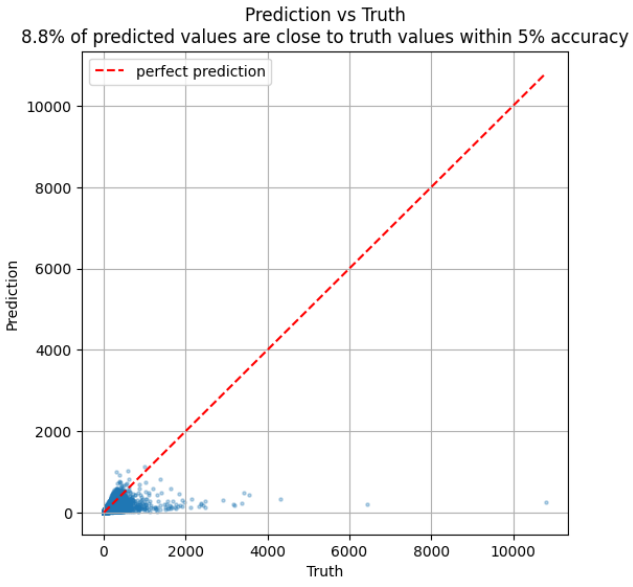

Figure 14: Prediction vs Target plot, for validation dataset.

Here clearly the model show how bad it is, but still better than the previous, but still bad with a cluster of data close to zero. One mistake we have probably done is with this input was with pre-processing. We have pre-process the entire four features whith different scaling, but probably a smarter choice is to pre-process each event individually. Now look at the generalization of the model.

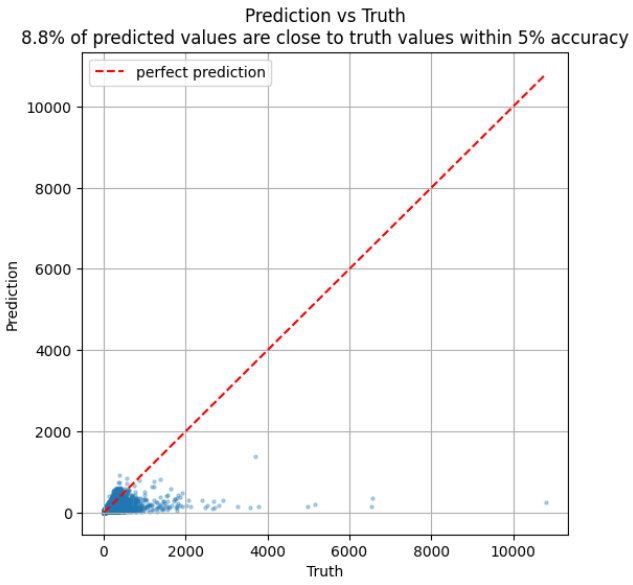

Figure 15: Prediction vs Target plot, for evaluation dataset.

Compare to the previous one, or model is consistent, in a sense of the percentage of valid prediction for the evaluation dataset is the same as the one for validation dataset. We still need to increase the performance of the model, we have to explore same ameliorations as for number of pulses, with the suggest discuss above.

7 conclusion

To conclude our work, we have see how do to a machine learning project. What are the types of inputs we uses and what is the target of our task.

Look at statistic of data allow us to choose which scaler and transformation we have to apply respect to data considered. After pre-process the choice of the model play an important role for robustness and generalisation of our task.

This work have take a lot of time, because of over fitting and poor prediction. In the quest of searching the best set of hyper parameters, we have discover automatise package to do this, but at the end we have explore hyper parameters space by hand due to time.

We have finish our work by show performance of our model with are emphasize on the metric part and plot of prediction vs target. We have seen that the number of pulses gives poor prediction for validation dataset and even worth prediction for evaluation dataset as compared with sensor features which is better, but still not acceptable to be use in real world.

To address these issues we can fine tune more hyper parameters, look at loss function, look at the pre-process of data to see if any error are still here, and finally extend our inputs data by merging number of pulses to sensor features.

References

[1] Migneco E, Antonio Capone, and Piattelli P. Underwater laboratories for astroparticle physics and deep sea science. *Annals of Geophysics*, 49, 12 2009.