Electron Localization by Boosted Gradient Regression

Fabrizio Marsico, Kevin Elezi
Politecnico di Torino, Italy
{s319359, s316685}@studenti.polito.it
link to the project

Abstract

In this report, we propose a possible method to address the problem of predicting the positions where particles pass in their trajectories. To later help our algorithm, we tried to develop a feature engineering technique where we hypothetically identified the location of pads sensitive to electron trajectories. With these locations then we mapped the areas of where these particles could pass, assigning various scores, making sure to identify a sub area, simulating the game sea battle. We after show that the proposed approach, based on cat-boosting[4], performs better than the random forest[2] used as baseline.

1. Problem Overview

The objective is to develop a data science pipeline for predicting the position of particle events in a Resistive Silicon Detector (RSD) used in particle physics. The RSD sensor, with a 2-dimensional surface, employs 12 snowflake-shaped metallic pads to measure signals when a particle passes through. Each **event** is characterized by signals from these pads, and the goal is to predict the corresponding (x, y) coordinates of the particle's passage on the sensor's surface. This event is therefore our representative of data and will be featured by the various measures for each pad in the RSD. The project aims to establish a predictive model based on the signal characteristics from each pad for accurate localization of particle events in the RSD. The data set provided consists of two different parts:

- A development set of 385,500 events characterised by comma-separated values such as the (x, y) coordinates to be predicted for each event.
- An evaluation set of 128,500 events with the same structure of the development set, except for the (x, y) coordinates.

By studying the original development set we observed 90 features, without any kind of missing or duplicated value. Furthermore all the type of the values of all the features are float64. This can lead to few or even null actions in data preparation operations. Through all these pad features[6], we should figure out at which of these possible (x, y) positions (see Figure 1), the event, that is the particle moving, occurs.

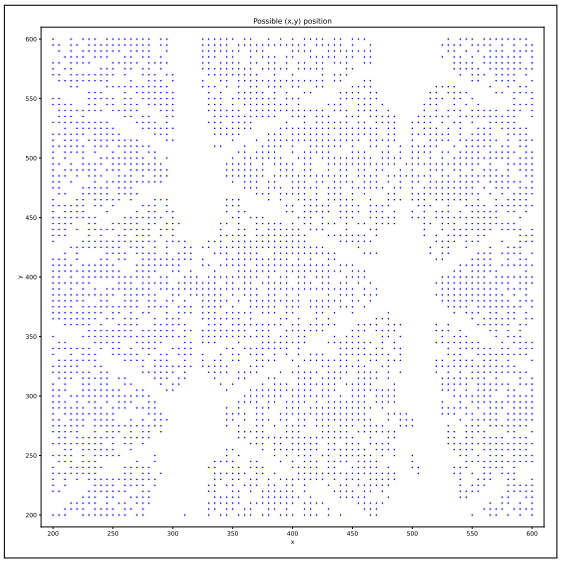

Figure 1. All possible positions that can be undertaken by our events in the dataset.

Although we know that our RSD holds 12 pads, our dataset instead shows us 18 pads. We therefore have 6 **noisy pads** (although in reality these fake, pads are not!) that mislead our analysis. It will then be up to us to be able to identify which of these noisy pads[7] are and eliminate them.

2. Preprocessing

2.1. Noise Elimination

We first focus on the data preparation steps needed to better understand what features may cause noise. By mapping the magnitude of the positive peak of the signal $pmax_i$

for each pad i available in the features, we can easily identify which ones are likely to disturb our analysis, here's the result:

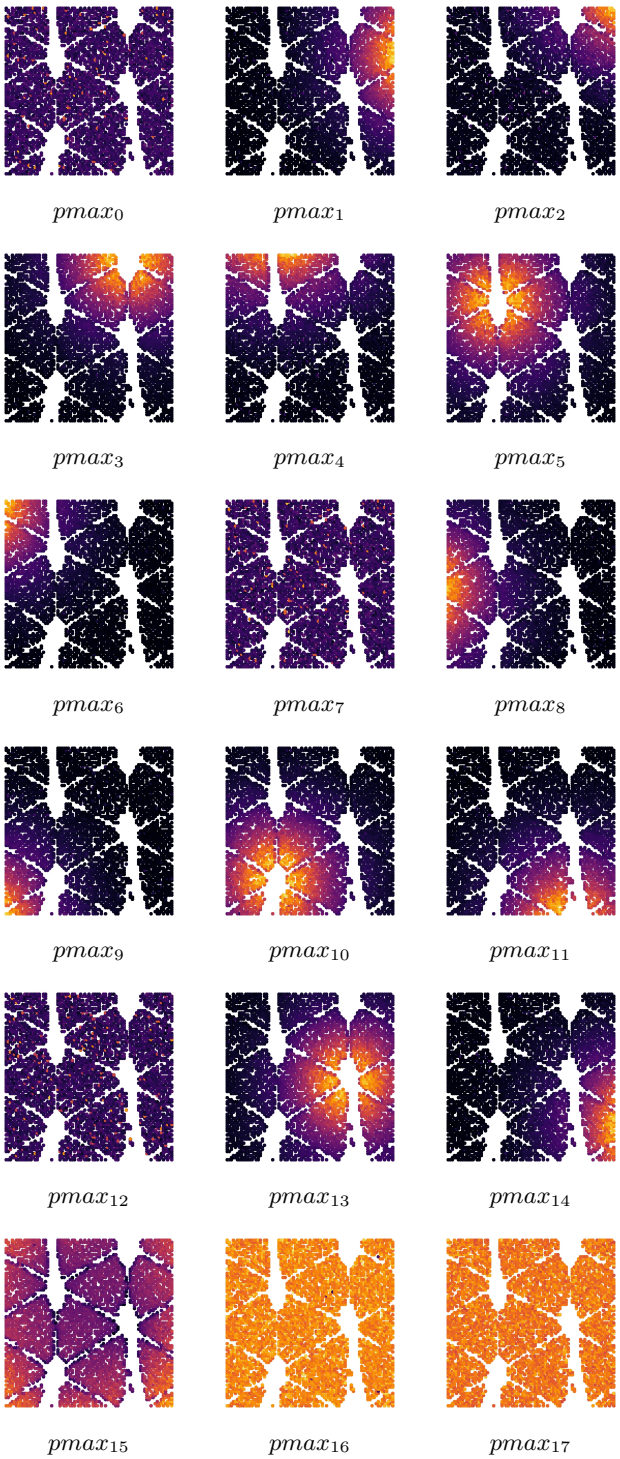

Figure 2. events for each $p_{\max[i]}$

As we can see in Figure 2, the features of 'pads' :

$$[0 - 7 - 12 - 15 - 16 - 17]$$

are to be ignored and will therefore be eliminated. We would like to point out that except for the 'pad' 15 all the other noisy pads are quite homogeneous in terms of the distribution of the magnitude of the peak $pmax$.

A point to note about the features 15, especially with regard to $pmax$, is that, if we look closely at the graph, we can see that despite being a noisy pad, it can suggest whether the event is close to the low end of the RSD. Thus helping us with pad prediction $[14, 11, 9]$. At first sight of this detail we therefore decided to keep this noisy pad with the hope that it would help us in the detection of events located in the four corners of the RSD. Unfortunately, however, during our processing we identified that keeping it did not help. In the end therefore we removed it like all other noisy pads.

2.2. Pads Battleship

The feature engineering work employed in this analysis was to try to identify the coordinates of the pads within the RSD (2.3). Once we obtained these points in space, we determined the neighborhood of each pad, and finally assign a score, in the form of new features, based pad with the maximum signal received, reminiscent of the game of sea battle (2.4). All this work was done to give a clue to the regression model on which area of space the event passed.

2.3. Pads coordinates

To find the hypothetical coordinates of the pads we ran this algorithm :

Algorithm 1 Pad's location algorithm

for each pad i **do**
  while $n_{samples} < 50$ **do**
    Compute the top value of $pmax[i]$ and find the corresponding row-index j.
    Obtain the coordinates (x_j,y_j).
    Append the coordinates (x_j,y_j) to $xcollector_i$.
    Delete the row with index j.
  end while
  Compute the average of $xcollector_i$ to get the coordinates (x_i,y_i).
end for
return $padslocation$

In a nutshell, for each pad_i we grouped all the positions of the top 50 events that were being revealed by that pad_i, thus the most intense events. Once all the points were collected, we calculated the average of the positions.

The approximate coordinates of the pads are :

Pad	Coordinates
1	(599.7, 486.8)
2	(595.9, 597.0)
3	(464.8, 588.6)
4	(341.9, 599.5)
5	(308.9, 492.3)
6	(201.1, 589.2)
8	(200.6, 388.3)
9	(203.6, 201.6)
10	(318.6, 282.6)
11	(466.8, 210.8)
13	(478.5, 371.2)
14	(599.7, 281.2)

Table 1. Pad Coordinates

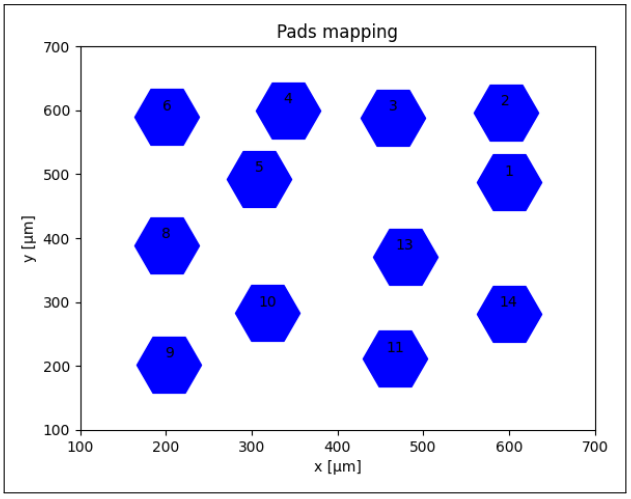

Figure 3. The plot shows the pads distribution using the founded coordinates

As shown in Table 1 and in Figure 3 these are the best aproximated coordinate, knowing that 100% could never be reached as not all measurements are maximum at those points.

We would like to specify that the positions we found are indicative. As is well noted, they do not follow a precise geometry (not like the RSD's one instead). This reason is given by the fact that not all events located next to a pad_i, are necessarily the most intense ones for this pad. We verified that for certain events, despite the fact that they are located significantly close to a pad, they give greater value of $pmax$ for a pad next to it. For this reason, we decided to continue with our feature engineering process by focusing not only on the pad with the greatest value of $pmax$ but also on neighboring pads. Neighboring pads that without this localization work we would not have been able to track down.

2.4. Pads Neighborhood

We then proceeded by defining, for each pad, the neighbouring pads: each n element, where $n \in \mathbb{N}_i$ is the set of the whole neighborhood for the pad i.

In order to create new features that can help us later on, we want to check the level of $pmax$ around the peak pad of each event and call it the p pad. We then check that for each $pmax_p$ (which would then be the pad p reading the maximum value of an event), all n $\in \mathbb{N}_p$ neighbouring pads have a value that exceeds a certain threshold:

$$\mathrm{pmax}_{n,p} > \text{threshold}$$

where $\mathrm{pmax}_{n,p}$ is the value that , a neighbor pad n of the pad p, reads.
We noticed than, by sampling 10.000 events for each pad p where the reading $pmax_p$ was maximum, that each neighbor pad n has :

$$pmax_{n,p} > 19mV$$

After a short test, this assumption was true for earch n.
The result of the various neighborhoods is as follows:

Table 2. Neighborhood Dictionary

Node	Neighbors
1	[2, 3, 13, 14]
2	[1, 3]
3	[1, 2, 4, 5, 13]
4	[3, 5, 6]
5	[3, 4, 6, 8, 10, 13]
6	[4, 5, 8]
8	[6, 5, 10, 9]
9	[8, 10]
10	[9, 8, 5, 13, 11]
11	[10, 13, 14]
13	[1, 3, 5, 10, 11, 14]
14	[1, 13, 11]

To summarise, we are trying to create new features that indicate the pad p where $pmax[p]$ of the event is maximum , and consequently all adjacent $n \in \mathbb{N}_p$ pads as well. This can subsequently lead to better 'geo-localisation' by drawing an imaginary area and narrowing the field.
The new features will be called :

$$[pad_1, pad_2, pad_3, pad_4, pad_5, pad_6$$

$$pad_8, pad_9, pad_{10}, pad_{11}, pad_{13}, pad_{14}]$$

and

$$pad[i] = \begin{cases} 10 & \text{if pad} = p \\ 1 & \text{if pad} = n \\ 0 & \text{else} \end{cases}$$

In this way we will give the highest value which is 10 to the p pads i.e. those to which the event is most likely to be close. The value of 1 on the other hand is assigned to pads n close to pad p. We thus manage to create a system similar to the battle ship game where the closer you get to the goal the more points you can get. Here is a visualization example for $p = 10$ and $p = 3$:

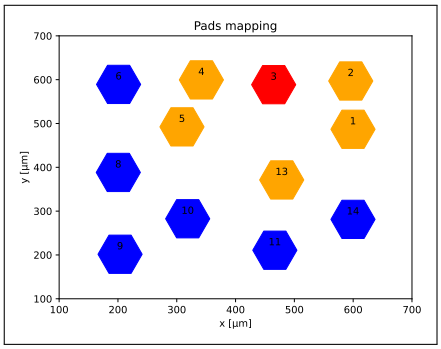

Figure 4. Battleship with $p = 3$

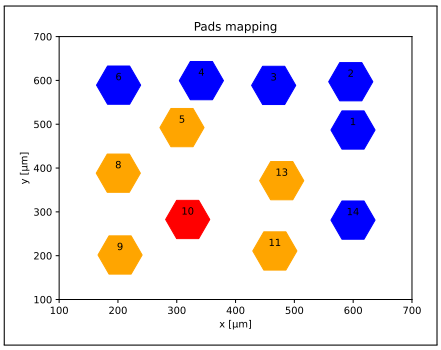

Figure 5. Battleship with $p = 10$

2.5. Cat-Boost and Hyperparameters-tuning

The **DecisionTrees**[9] seemed to us to be the tailor-made model for this type of problem. However, its performance did not get us over the threshold of the score to pass. The most intuitive solution is therefore to use the **RandomForest**[2]. The problem with the latter, however, is, given the number of data and features, that it is too slow[8]. Staying with the regression due to the decision tree[9], we found a really convenient middle way: **CatBoost**[4].
CatBoost[4] is a high-performance, open-source gradient boosting library designed for machine learning tasks, particularly in the field of tabular data. It is known for its efficient handling of categorical features: for that reason also we implemented that 'Battleship' feature engineering, to create categorical features that were absent in the original dataset.

It also utilizes a variant of gradient boosting algorithms; furthermore is optimized for speed and memory efficiency, making it suitable for large datasets, like in this case. This model allows us to create several **DecisionTrees** by optimising the choice of nodes each time (thus decreasing the error) via **gradient boosting**.

We have omitted the various pattern selection attempts (DecisionTree RandomForest[2]) so as not to bore the reader and unnecessarily lengthen the notebook.
The tuning of hyperparameters was configured through the following methodologies :

- **GridSearch**[5], a hyper-parameter optimization technique used in machine learning to systematically explore a predefined set of values for a given model. The idea is to create a grid of hyper-parameter combinations and evaluate the model's performance for each combination, very easy to implement.

- **RandomSearch**[1][3], another method that instead of exhaustively searching through all possible combinations of hyper-parameter values (as done in grid search[5]), random search[1] randomly samples a predefined number of combinations; also is computationally less expensive than grid search[5] and often performs well in practice, especially when the hyperparameter space is high-dimensional.

3. Results

We will now compare the results obtained in the testset with the two model, using the hyperparameter obtained with the proposed metodology:

- The best combination of parameters found for the Grid Search is :

  {'depth': 8, 'iterations': 1500, 'learning_rate': 0.15},

  with an Euclidean Distance of 0.01863

- The configuration producing the best result for the Random Search is :

  {'learning_rate': 0.1, 'iterations': 1900, 'depth': 8}

  scoring 0.01848 in Euclidean Distance.

The results are also reported in Table 3.

approach	Euclidean Distance	time (minutes)
Grid Search	0.01863	5'28"
Random Search	0.01848	15'34"

Table 3. Here we can observe the values reported by our approaches. Notice that to boost our search time we opted for the convenient library catboost-gpu paired with a GeForce RTX 3080.

4. Conclusions

In conclusion, we can say that through a DataScience pipeline we were able to obtain a model that can identify the position of particles in a more than accurate manor. Given the large dataset (fortunately already clean) with a large number of features, we were able to identify the correct techniques to handle the high dimensionality.

We would like to point out that in this paper we talk a lot and only about the $pmax$ feature. We actually sensed that perhaps the number of features could be reduced considerably. For this very reason we also tested the analysis only with $(pmax, negpmax)$ for each pad_i. The result was definitely sufficient even in that case, but since it did not outscore the case with all features, we decided not to show that part of the analysis to bore the reader.

To best support our chosen regression model (CatBoost), we performed feature engineering operations to solidify the performance of the model, called by us, *Battleship*. This introduction helped the training phase to categorize the abstract area where the particle was moving. In this way we provided support for CatBoost, which welcomes categorical features.

We would like to talk about our considerations of the *Battleship* operation : working on the existing features we noticed how in fact, more or less, all the information is already available within the original dataset. This is a general discussion, for any dataset. Thinking back, the information about which pad the event is most intense about is already present in the dataset. Thinking about all the possible cases of features in any dataset, we still find it hard to come up with new features, which distort the meaning already present! So we believe that feature engineering operations only serve to solidify the result, which is obviously not to be underestimated, at least in this case.

As for the hyper-tuning part, we have that random search outperforms gridsearch, even if it requires a considerable effort in the hyperparameters tuning step (as reported in Table 3, random approach necessitates three times the duration.). Cat-boost effectively handling our implemented categorical data without encoding and managing large amount of data was important for the quality of our results. These results confirm that even such a large dataset with lots of features can be managed in reasonable amount of time thanks to CatBoost.

References

[1] Bergstra, J., Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281–305.

[2] Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization. In Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS 2011), 2546–2554.

[3] Snoek, J., Larochelle, H., Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS 2012), 2951–2959

[4] CatBoost : CatBoost Documentation

[5] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS 2018), 6639–6649.

[6] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. This classic paper by Yann LeCun and others discusses the application of convolutional neural networks to document recognition. It touches upon the importance of padding in CNNs.

[7] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y. (2016). Deep Learning (Vol. 1). MIT press Cambridge. "Deep Learning" is a comprehensive book that covers various aspects of deep learning, including convolutional neural networks. It provides insights into the use of padding and its significance in the design of CNN architectures.

[8] A Gradient Boosting approach to Regression with Qualitative Predictors and Textual Data (2021). DSL Report noname. Politecnico di Torino.

[9] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in Advances in Neural Information Processing Systems (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, pp. 3146–3154, Curran Associates, Inc., 2017.