Particle Coordinates Predictions in Resistive Silicon Detectors

Paolo Castellaro, Mehdi Nickzamir
Politecnico di Torino
Student ids: s331568, s323959
paolo.castellaro@studenti.polito.it, mehdi.nickzamir@studenti.polito.it

***Abstract*—comment This project involves the development of a multi-output pipeline for predicting the coordinates (x, y) of a particle passing through a resistive silicon detector (RSD). The pipeline uses the XGBRegressor to analyze signals generated by 12 pads in the detector. The provided dataset contains properties of the signals produced by each pad and the effectiveness of the pipeline is evaluated using the mean Euclidean distance metric.**

I. Problem overview

Our project aims to detect the positions where particles, such as electrons, pass in their trajectories using Resistive Silicon Detector (RSD) sensors. These 2-dimensional sensors, equipped with 12 metallic pads, measure signals when a particle passes through, implying the 2D position (X,Y) of the hit particle. [1] The objective is to develop a prediction pipeline using multi-output regression to analyze these signal features and predict the hit position. This project focuses solely on analyzing signal features predicting hit particle coordinates, without involving data acquisition or other fields. We have access to two datasets.

- **Development set:** database used for the development, containing 385.000 events
- **Evaluation set:** database used for the evaluation, containing 128.500 events without the target coordinates

Each instance in the dataset is characterized by 5 features for each sensor pad, representing the information collected by the pads for each event (see Table I).

TABLE I
Features Of Signals

Feature	Description	Data Type
pmax[i]	the magnitude of the positive peak of the signal, in mV	Float
nehpmax[i]	the magnitude of the negative peak of the signal, in mV	Float
tmax[i]	the delay (in ns) from a reference time when the positive peak of the signal occurs	Float
area[i]	the area under the signal	Float
rms[i]	the root mean square (RMS) value of the signal	Float

While each event provides 18 feature readings, thus a total number of columns of 90, only 12 pads are in the sensor due to hardware constraints. Six of these readings, known to be noise, do not contribute to our predictions. In the evaluation of our models, we use the *mean Euclidean distance* as the metric. The mean Euclidean distance is calculated using the formula shown in Equation 1. We have been provided with a platform for the submission of the predictions made on the evaluation CSV file. This platform computes a score based on our submitted predictions and maintains a leaderboard to track the performance of all submissions.

$$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1}$$

Fig. 1. Mean Euclidean distance formula

II. Proposed approach

A. Preprocessing

Our dataset includes six noise pad readings that do not contribute to the target coordinates, so we must remove them to improve the model's accuracy. Without these extra columns, the model will only focus on the relevant data and make more accurate predictions.

In Resistive Silicon Detectors (RSDs), signal distribution among electrodes is influenced by the proximity of the particle hit. Parameters such as peak amplitude (pmax), time to peak (tmax), and the integral of the signal (area) are tied to the energy deposition and timing of the hit. Pmax and area increase with energy deposited by a particle near a pad, while tmax decreases due to reduced charge carrier travel distance [2]–[4]. However, root mean square (rms) and negative pmax, which measure signal magnitude and minimum value respectively, do not imply proximity in the same way due to their independence from particle hits and specific detector characteristics [5].

Given these observations, pmax, tmax, and area are considered the most significant properties of each signal, as they are most affected by the proximity of the particle hit to a specific pad. Consequently, our noise analysis is primarily based on these parameters. In the process of identifying the columns that contribute noise in our dataset, we can employ three distinct methodologies:

1) Statistical Approach
2) Visualization Approach
3) Brute Force Approach

1) Statistical Approach: In data science, noise refers to random data that does not aid in achieving the desired results, the target coordinates. This phenomenon can be mathematically measured using correlation coefficients. [6] The selection of an appropriate correlation measure is of paramount importance. Pearson correlation, for instance, is adept at assessing linear relationships. However, upon visualizing the first seven pads (considering the presence of six noise pads, it is inevitable to encounter at least one that does not contribute to noise), it becomes apparent that none of them exhibit a linear relationship with either the X or Y coordinates.II-A1 Consequently, we resort to Spearman's correlation. Spearman's

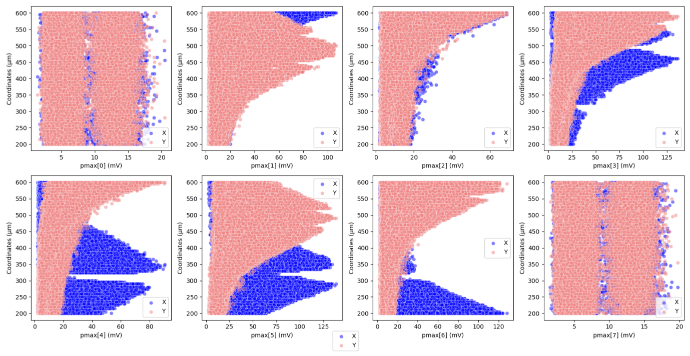

Fig. 2. Figure comparing the coordinates (x, y) with pmax[i] - showing non-linear relationship

rank correlation coefficient, often denoted by the Greek letter , is a non-parametric measure of rank correlation that assesses the strength and direction of monotonic association between two ranked variables. It is considered to be more robust to outliers and non-linear relationships than the Pearson correlation coefficient. [6] 2 To further refine the analysis, a correlation threshold of 0.05 is established for either X, Y, or both. Data that exhibits a correlation value beneath this criterion is inferred to have an insignificant impact on the predictions and may be categorized as noise. In the context of coordinate prediction, features that do not exhibit correlation with either axis may inject unwarranted variance, which could potentially augment the Euclidean distance between the predicted and actual coordinates. Upon applying these criteria to pmax, the indices [0,7,12,15,16,17] are identified as noise. (see Table II)

TABLE II
PMAX COLUMNS WITH PEARSON CORRELATION VALUES LESS THAN 0.05

Feature	X-correlation	Y-correlation
$pmax[0]$	-0.004	0.013
$pmax[7]$	-0.006	0.015
$pmax[12]$	0.002	0.002
$pmax[15]$	0.011	-0.330
$pmax[16]$	0.012	0.004
$pmax[17]$	0.014	-0.002

When the same criteria are applied to tmax, the indices [0,7,8,12,13,15,16,17] emerge as noise. (see Table III)

$$\rho = 1 - \frac{6\sum_{i=1}^{n}(x_i - y_i)^6}{n(n^2-1)}. \qquad (2)$$

Fig. 3. Spearman's Correlation Formula

TABLE III
TMAX COLUMNS WITH PEARSON CORRELATION VALUES LESS THAN 0.05

Feature	X-correlation	Y-correlation
$tmax[0]$	-0.001	0.001
$tmax[7]$	0.001	-0.003
$tmax[8]$	0.217	0.002
$tmax[12]$	-0.002	0.001
$tmax[13]$	-0.061	0.014
$tmax[15]$	0.001	0.093
$tmax[16]$	0.001	0.001
$tmax[17]$	0.001	0.001

And Upon applying the same criteria to area, the indices [0,7,12,15,16,17] are identified as noise.(see Table IV)

TABLE IV
AREA COLUMNS WITH PEARSON CORRELATION VALUES LESS THAN 0.05

Feature	X-correlation	Y-correlation
$area[0]$	-0.002	0.002
$area[7]$	-0.003	0.002
$area[12]$	0.001	-0.001
$area[15]$	0.039	0.018
$area[16]$	0.054	0.007
$area[17]$	0.057	-0.000

If we identify the common elements of these three sets, we arrive at the following result:[0,7,12,15,16,17]

The logic behind this approach is grounded in the principles of set theory and data analysis. By pinpointing the intersection of these sets, we are essentially identifying the common 'noise' columns that all three perspectives agree upon. These common indices may hold significant implications for our analysis as they represent consistent patterns of noise across the sets.

2) Visual Approach: Upon conducting a comprehensive examination of the distributions corresponding to each feature of every sensor, we are presented with the following histogram charts. As previously stated, our analysis will primarily focus on three specific properties: tmax, pmax, and area. An intriguing observation from the visual inspection of the data is the presence of distinct distribution patterns for the indices [0, 7, 12, 15, 16, 17]. (See Figure II-A2; For brevity and optimal visualization, this figure exclusively displays normal columns 4 and 10 as representatives of all other normal columns.) These patterns markedly deviate from the distributions observed for the other indices, indicating a potential difference in the underlying data characteristics.

Furthermore, it is important to highlight that noise typically exhibits a normal distribution. This characteristic pattern of noise can be observed in the distributions of pmax. This observation is crucial as it aids in the identification and subsequent handling of noise within the data, thereby enhancing the accuracy and reliability of our analysis. [7]

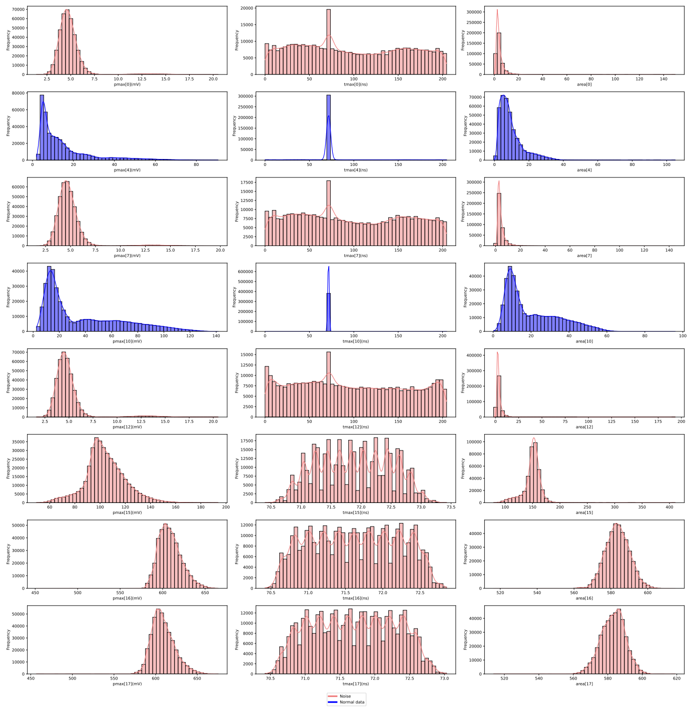
Fig. 4. Distribution of pmax, tmax, and area for pads 0,4,7,10,15,16,17

3) Brute Force Approach: In this approach, we examined all possible combinations of the given set, specifically focusing on subsets of 6 elements out of the total 18. Which results in 18564 combinations. The combination formula is as follows:

$$C(n,k) = \frac{n!}{k!(n-k)!} \tag{3}$$

Fig. 5. Combination Formula

where n is the total number of elements, k is the number of elements in each subset, and ! denotes the factorial operation.

The objective was to ascertain which particular combination when excluded, would yield the most optimal outcome. Upon conducting this exhaustive examination, we observed that the exclusion of the subset [0,7,12,15,16,17] led to the most favorable result.

Through the application of three distinct methodologies, we have identified columns [0,7,12,15,16,17] as noise columns in the dataset. To improve our model's accuracy, we will remove these columns from the dataset.

The final preprocessing step in our project was data normalization, crucial for the XGBoost, a gradient-boosting algorithm. It ensures all features are on the same scale, preventing any single feature from unduly influencing the model due to its range. This is particularly important for XGBoost, which is sensitive to the scale of input features. We conducted evaluations using both MinMax normalization and Standardization methods and found similar results from both. Nevertheless, we selected Standardization owing to its enhanced robustness and capacity to manage outliers.

It is worth mentioning that since XGBoost is robust to multicollinearity, it does not require removal of collinear columns. Other feature selection/engineering methods like PDA, LDA, and feature interaction didn't significantly improve our model's performance and, hence were not used in the final model.

B. Model selection

We conducted tests on various regression algorithms, with a particular emphasis on Random Forest Tree regression. However, we discovered a superior and faster algorithm that yielded improved results, despite not being native to the sklearn toolkit. Consequently, we opted to use XGBoost [8] [9], an algorithm developed recently, in 2016, as a research project at the University of Washington by Tianqi Chen and Carlos Guestrin, and won several Kaggle competitions. XGBoost (eXtreme Gradient Boosting) is an optimized implementation of the gradient boosting trees algorithm. This algorithm involves sequentially combining weak learners (typically trees with only 1 split) in such a way that each new tree corrects the errors of the previous one, it can be summarized as follows: it fits a single decision tree and then evaluates it using a loss function (we used squared loss in our model), it creates a second tree, which should reduce the loss compared to the first tree alone, therefore we want to find the direction in which the loss decreases the fastest, which is mathematically given by the gradient (descent) of the loss function with respect to the output of the previous model. For any step m, gradient boosting trees produce a model so that ensemble(F) at step m ($F(m)$) is equal to:

$$F(m) = F(m-1) + \eta * (-\frac{\partial L}{\partial F(m-1)}) \tag{4}$$

Fig. 6. Ensemble at step m

Where η is simply the learning rate, which we want to choose in order that we do not "walk too far" in any direction and not so low to avoid the model taking too long to converge to the right answer. XGBoost is an optimized version of Gradient Boosting Trees because it also implements::

- Lasso (L1) and Ridge (L2) regularization techniques to prevent overfitting, useful especially for large datasets with a highly dimensional feature space.
- Built-in capabilities to handle missing values, reducing the need for extensive data preprocessing
- Built-in parallelization and tree pruning, which gives faster and exceptional performances
- Built-in cross-validation method at each iteration
- The possibility of running the algorithm on NVIDIA GPUs thanks to the CUDA toolkit (which we installed in order to run the algorithm on an NVIDIA RTX 2060)

C. Hyperparameters tuning

For hyperparameter optimization [10], we chose to utilize the classic gridsearch approach (with a verbosity level of 2, in order to be able to visualize additional data during the grid search), but not on the entire dataset, rather on a sample (1 percent) because we have made several attempts to try to perform this search on the entire dataset (or even on 1 tenth) resulting in various crashes and computer freezes. Concerning our model, we focused on tuning various hyperparameters, specifically:

TABLE V
HYPERPARAMETERS SELECTED

Hyperparameter	**Description**
`Eta`	Learning rate
`Gamma`	Minimum loss reduction to make a split
`Max_depth`	Maximum depth of a tree
`Lambda`	L2 regularization term on weight
`Alpha`	L1 regularization term on weight
`Colsampleby-tree`	Subsample ratio of columns for each tree
`Colsampleby-level`	Subsample ratio of columns for each level
`Colsampleby-node`	Subsample ratio of columns for each node
`n-estimators`	maximum number of boosting trees

We must say that this approach did not give us the best results when the values where applied to the entire dataset, therefore we manually tuned some of them in order to get the best results.
To assess the performance of different configurations, we employed the R^2 as the evaluation metric, given our observation, as expected, that higher R^2 corresponded to better scores on the public leaderboard.

Another noteworthy parameter we considered is num-parallel-tree, although we did not conduct a formal tuning process. Simply put, increasing the number of trees leads to improved results at the expense of escalating computational costs. For instance, we ran the model with 1 (default value), 2, 4 and 16 consistently obtaining better results but with progressively longer execution times, as can be seen in the following table:

TABLE VI
CORRELATION BETWEEN NUMBER OF TREES, TIME AND SCORE

Num. Trees	**Time [min]**	**Score**
1	4	4.608
2	8	4.509
4	15	4.470
16	59	4.401

III. RESULTS

We obtained the best results on the public leaderboard, as expected, with XGBoost using the biggest number of trees that we tried (16) and the following configuration of parameters that can be seen in Table VII
The final score on the public leaderboard with this configuration was: 4.401

TABLE VII
FINAL PARAMETERS

Hyperparameter	**Value**
`Colsample-bytree`	0.7
`Colsample-bylevel`	0.8
`Colsample-bynode`	0.8
`Max-depth`	15
`Min-child-weight`	10
`Alpha`	1
`Lambda`	0.01
`Num-parallel-tree`	16
`Gamma`	4
`Eta`	0.06
`n-estimators`	1000

IV. DISCUSSION

While the proposed pipeline demonstrates competitive results, there remains room for improvement. The following considerations are noteworthy:

- Conducting a more extensive gridsearch, considering all parameters and the full dataset, could potentially enhance the score. However, such an approach comes with a considerable computational burden; our search took precisely 7481 seconds, more than 2 hours. If hardware capability allows it, the research should be done on the entire dataset to avoid manually adjusting the parameters.
- Increasing the number of parallel trees in the model has the potential to improve results but at an increased computational cost.
- Efforts were made to engage in feature engineering, exploring different combinations of features to achieve better results. Despite a slight improvement in the development dataset score (accompanied by an increase in computational time, given the desire for nearly 100 features), this enhancement did not manifest in the evaluation set score.

In conclusion, while we take pride in achieving a notable and satisfactory outcome, we acknowledge the existence of avenues for improvement. Apart from the aforementioned considerations, exploring alternative models, such as Deep Neural Networks, could be a valuable next step in pursuit of an even higher score.

REFERENCES

[1] R. Cartiglia, “Resistive silicon detectors,” 2024.
[2] E. Optics, “Basic principles of silicon detectors,” 2022. https://www.edmundoptics.com/knowledge-center/application-notes/optics/basic-principles-of-silicon-detectors/ [Accessed: (Use the date of access)].
[3] H. Photonics, “What is an sipm and how does it work?,” 2022. https://www.hamamatsu.com/eu/en/product/type/S13360-3050CS/index.html [Accessed: (Use the date of access)].
[4] F. Zimmermann, *Particle Detectors and Applications*. CRC Press, 2001.
[5] J. Igba, K. Alemzadeh, C. Durugbo, and E. T. Eiriksson, “Analysing rms and peak values of vibration signals for condition monitoring of wind turbine gearboxes,” *Renewable Energy*.
[6] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013.
[7] C. Grupen, *Data Analysis of Particle Physics Experiments*. Springer, Berlin, Heidelberg, 2016.

[8] "How xgboost works." https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html.
[9] "Xgboost algorithm." https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html.
[10] P. on Kaggle.com, "A guide on xgboost hyperparameters tuning," 2022. https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning.